Python Programming

en.wikibooks.org

December 27, 2015

Contents

1 Overview

Python[1] is a high-level[2], structured[3], open-source[4] programming language that can be used for a wide variety of programming tasks. Python was created by Guido Van Rossum in the early 1990s, its following has grown steadily and interest is increased markedly in the last few years or so. It is named after Monty Python's Flying Circus comedy program.

Python[5] is used extensively for system administration (many vital components of Linux[6] Distributions are written in it), also its a great language to teach programming to novice. NASA has used Python for its software systems and has adopted it as the standard scripting language for its Integrated Planning System. Python is also extensively used by Google to implement many components of its Web Crawler and Search Engine & Yahoo! for managing its discussion groups.

Python within itself is an interpreted programming language that is automatically compiled into bytecode before execution (the bytecode is then normally saved to disk, just as automatically, so that compilation need not happen again until and unless the source gets changed). It is also a dynamically typed language that includes (but does not require one to use) object oriented features and constructs.

The most unusual aspect of Python is that whitespace is significant; instead of block delimiters (braces → "{}" in the C family of languages), indentation is used to indicate where blocks begin and end.

For example, the following Python code can be interactively typed at an interpreter prompt, display the famous "Hello World!" on the user screen:

```
 >>> print "Hello World!"
Hello World!
```

Another great Python feature is its availability for all Platforms. Python can run on Microsoft Windows, Macintosh & all Linux distributions with ease. This makes the programs very portable, as any program written for one Platform can easily be used at another.

Python provides a powerful assortment of built-in types (e.g., lists, dictionaries and strings), a number of built-in functions, and a few constructs, mostly statements. For example, loop constructs that can iterate over items in a collection instead of being limited to a simple range of integer values. Python also comes with a powerful standard library[7], which includes

1 https://en.wikibooks.org/wiki/Python
2 https://en.wikibooks.org/wiki/Computer%20programming%2FHighlevel
3 https://en.wikibooks.org/wiki/Computer%20programming%2FStructured%20programming
4 https://en.wikibooks.org/wiki/Open%20Source
5 https://en.wikibooks.org/wiki/Python
6 https://en.wikibooks.org/wiki/Linux
7 https://en.wikibooks.org/wiki/Python%20Programming%2FStandard%20Library

hundreds of modules to provide routines for a wide variety of services including regular expressions[8] and TCP/IP sessions.

Python is used and supported by a large Python Community[9] that exists on the Internet. The mailing lists and news groups[10] like the tutor list[11] actively support and help new python programmers. While they discourage doing homework for you, they are quite helpful and are populated by the authors of many of the Python textbooks currently available on the market.

Note:
Python 2 vs Python 3: Several years ago, the Python developers made the decision to come up with a major new version of Python. Initially called "Python 3000", this became the 3.*x* series of versions of Python. What was radical about this was that the new version is **backward-incompatible** with Python 2.*x* : certain old features (like the handling of Unicode strings) were deemed to be too unwieldy or broken to be worth carrying forward. Instead, new, cleaner ways of achieving the same things were added.

8 Chapter 22 on page 131
9 `http://www.python.org/community/index.html`
10 `http://www.python.org/community/lists.html`
11 `http://mail.python.org/mailman/listinfo/tutor`

2 Getting Python

In order to program in Python you need the Python interpreter. If it is not already installed or if the version you are using is obsolete, you will need to obtain and install Python using the methods below:

2.1 Python 2 vs Python 3

In 2008, a new version of Python (version 3) was published that was not entirely backward compatible. Developers were asked to switch to the new version as soon as possible but many of the common external modules are not yet (as of Aug 2010) available for Python 3. There is a program called *2to3* to convert the source code of a Python 2 program to the source code of a Python 3 program. Consider this fact before you start working with Python.

2.2 Installing Python in Windows

Go to the Python Homepage[1] or the ActiveState website[2] and get the proper version for your platform. Download it, read the instructions and get it installed.

In order to run Python from the command line, you will need to have the python directory in your PATH. Alternatively, you could use an Integrated Development Environment (IDE) for Python like DrPython`http://drpython.sourceforge.net/`, eric`http://www.die-offenbachs.de/eric/index.html`, PyScripter`http://mmm-experts.com/Products.aspx?ProductID=4`, or Python's own IDLE[3] (which ships with every version of Python since 2.3).

The PATH variable can be modified from the Window's System control panel. To add the PATH in Windows 7 :

1. Go to Start.
2. Right click on computer.
3. Click on properties.
4. Click on 'Advanced System Settings'
5. Click on 'Environmental Variables'.
6. In the system variables select Path and edit it, by appending a ';' (without quote) and adding 'C:\python27'(without quote).

1 `http://www.python.org/download/`
2 `http://activestate.com`
3 `https://en.wikipedia.org/wiki/IDLE_%28Python%29`

If you prefer having a temporary environment, you can create a new command prompt short-cut that automatically executes the following statement:

```
PATH %PATH%;c:\python27
```

If you downloaded a different version (such as Python 3.1), change the ”27” for the version of Python you have (27 is 2.7.x, the current version of Python 2.)

2.2.1 Cygwin

By default, the Cygwin installer for Windows does not include Python in the downloads. However, it can be selected from the list of packages.

2.3 Installing Python on Mac

Users on Apple Mac OS X will find that it already ships with Python 2.3 (OS X 10.4 Tiger) or Python 2.6.1 (OS X Snow Leopard), but if you want the more recent version head to Python Download Page[4] follow the instruction on the page and in the installers. As a bonus you will also install the Python IDE.

2.4 Installing Python on Unix environments

Python is available as a package for some Linux distributions. In some cases, the distribution CD will contain the python package for installation, while other distributions require downloading the source code and using the compilation scripts.

2.4.1 Gentoo GNU/Linux

Gentoo is an example of a distribution that installs Python by default - the package system *Portage* depends on Python.

2.4.2 Ubuntu GNU/Linux

Users of Ubuntu will notice that Python comes installed by default, only it sometimes is not the latest version. If you would like to update it, click here[5].

4 `http://www.python.org/download/mac`
5 `http://appnr.com/install/python`

2.4.3 Arch GNU/Linux

Arch does not install python by default, but is easily available for installation through the package manager to pacman. As root (or using sudo if you've installed and configured it), type:

```
$ pacman -Syu python
```

This will be update package databases and install python. Other versions can be built from source from the Arch User Repository.

2.4.4 Source code installations

Some platforms do not have a version of Python installed, and do not have pre-compiled binaries. In these cases, you will need to download the source code from the official site[6]. Once the download is complete, you will need to unpack the compressed archive into a folder.

To build Python, simply run the configure script (requires the Bash shell) and compile using make.

2.4.5 Other Distributions

Python, which is also referred to as CPython[7], is written in the C Programming[8] language. The C source code is generally portable, that means CPython can run on various platforms. More precisely, CPython can be made available on all platforms that provide a compiler to translate the C source code to binary code for that platform.

Apart from CPython there are also other implementations that run on top of a virtual machine. For example, on Java's JRE (Java Runtime Environment) or Microsoft's .NET CLR (Common Language Runtime). Both can access and use the libraries available on their platform. Specifically, they make use of reflection[9] that allows complete inspection and use of all classes and objects for their very technology.

Python Implementations (Platforms)

Environment	**Description**	**Get From**
Jython	Java Version of Python	Jython[10]
IronPython	C# Version of Python	IronPython[11]

6 `http://www.python.org/download/`
7 `https://en.wikibooks.org/wiki/CPython`
8 `https://en.wikibooks.org/wiki/C%20Programming`
9 `http://en.wikipedia.org/wiki/Reflection_(computer_programming)`
10 `http://www.jython.org`
11 `http://www.ironpython.net`

2.4.6 Integrated Development Environments (IDE)

CPython ships with IDLE; however, IDLE is not considered user-friendly.[12] For Linux, KDevelop and Spyder are popular. For Windows, PyScripter is free, quick to install, and comes included with PortablePython[13].

Some Integrated Development Environments (IDEs) for Python

Environment	**Description**	**Get From**
KDevelop	Cross Language IDE for KDE	KDevelop[14]
ActivePython	Highly Flexible, Pythonwin IDE	ActivePython[15]
Anjuta	IDE Linux/Unix	Anjuta[16]
Pythonwin	Windows Oriented Environment	Pythonwin[17]
PyScripter	Free Windows IDE (portable)	PyScripter[18]
VisualWx	Free GUI Builder	VisualWx[19]
Spyder	Free cross-platform IDE	Spyder[20]
Eclipse (PyDev plugin)	Open Source IDE	Eclipse[21]

The Python official wiki has a complete list of IDEs[22].

There are several commercial IDEs such as Komodo, BlackAdder, Code Crusader, Code Forge, and PyCharm. However, for beginners learning to program, purchasing a commercial IDE is unnecessary.

2.5 Keeping Up to Date

Python has a very active community and the language itself is evolving continuously. Make sure to check python.org[23] for recent releases and relevant tools. The website is an invaluable asset.

12 The Things I Hate About IDLE That I Wish Someone Would Fix ^{http://inventwithpython.com/blog/2011/11/29/the-things-i-hate-about-idle-that-i-wish-someone-would-fix/} .
13 http://www.portablepython.com/
14 http://www.kdevelop.org
15 http://www.activestate.com/
16 http://anjuta.sf.net/
17 http://www.python.org/windows/
18 http://code.google.com/p/pyscripter/
19 http://visualwx.altervista.org
20 http://code.google.com/p/spyderlib/
21 http://www.eclipse.org
22 http://wiki.python.org/moin/IntegratedDevelopmentEnvironments
23 http://www.python.org

Public Python-related mailing lists are hosted at mail.python.org[24]. Two examples of such mailing lists are the **Python-announce-list** to keep up with newly released third party-modules or software for Python and the general discussion list **Python-list** . These lists are mirrored to the Usenet newsgroups **comp.lang.python.announce** & **comp.lang.python** .

2.6 Notes

24 `http://mail.python.org`

3 Interactive mode

Python has two basic modes: normal and interactive. The normal mode is the mode where the scripted and finished `.py` files are run in the Python interpreter. Interactive mode is a command line shell which gives immediate feedback for each statement, while running previously fed statements in active memory. As new lines are fed into the interpreter, the fed program is evaluated both in part and in whole.

To start interactive mode, simply type ”python” without any arguments. This is a good way to play around and try variations on syntax. Python should print something like this:

```
$ python
Python 3.0b3 (r30b3:66303, Sep  8 2008, 14:01:02) [MSC v.1500 32 bit (Intel)]
on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

(If Python doesn't run, make sure your path is set correctly. See Getting Python[1].)

The `>>>` is Python's way of telling you that you are in interactive mode. In interactive mode what you type is immediately run. Try typing `1+1` in. Python will respond with `2` . Interactive mode allows you to test out and see what Python will do. If you ever feel the need to play with new Python statements, go into interactive mode and try them out.

A sample interactive session:

```
>>> 5
5
>>> print (5*7)
35
>>> "hello" * 4
'hellohellohellohello'
>>> "hello".__class__
<type 'str'>
```

However, you need to be careful in the interactive environment to avoid confusion. For example, the following is a valid Python script:

```
if 1:
  print("True")
print("Done")
```

If you try to enter this as written in the interactive environment, you might be surprised by the result:

1 Chapter 2 on page 5

```
>>> if 1:
...   print(”True”)
... print(”Done”)
  File ”<stdin>”, line 3
    print(”Done”)
        ^
SyntaxError: invalid syntax
```

What the interpreter is saying is that the indentation of the second print was unexpected. You should have entered a blank line to end the first (i.e., ”if”) statement, before you started writing the next print statement. For example, you should have entered the statements as though they were written:

```
if 1:
  print("True")

print("Done")
```

Which would have resulted in the following:

```
>>> if 1:
...   print(”True”)
...
True
>>> print(”Done”)
Done
>>>
```

3.0.1 Interactive mode

Instead of Python exiting when the program is finished, you can use the -i flag to start an interactive session. This can be **very** useful for debugging and prototyping.

```
python -i hello.py
```

4 Creating Python programs

Welcome to Python! This tutorial will show you how to start writing programs.

Python programs are nothing more than text files, and they may be edited with a standard text editor[1] program.[2] What text editor you use will probably depend on your operating system: any text editor can create Python programs. However, it is easier to use a text editor that includes Python syntax highlighting[3].

4.1 Hello, World!

The first program that beginning programmers usually write is the "w:Hello, World!" program[4]. This program simply outputs the phrase "Hello, World!" then terminates itself. Let's write "Hello, World!" in Python!

Open up your text editor and create a new file called `hello.py` containing just this line (you can copy-paste if you want):

```
print('Hello, world!')
```

This program uses the `print` function, which simply outputs its parameters to the terminal. By default, `print` appends a `newline` character to its output, which simply moves the cursor to the next line.

Note:
In Python 2.x, print is a statement rather than a function. As such, it can be used without parentheses, in which case it prints everything until the end of the line and accepts a standalone comma after the final item on the line to indicate a multi-line statement. In Python 3.x, print is a proper function expecting its arguments inside parentheses. Using print with parentheses (as above) is compatible with Python 2.x and using this style ensures version-independence.

Now that you've written your first program, let's run it in Python! This process differs slightly depending on your operating system.

1 `https://en.wikipedia.org/wiki/Text%20editor`
2 Sometimes, Python programs are distributed in compiled form. We won't have to worry about that for quite a while.
3 `https://en.wikipedia.org/wiki/Syntax%20highlighting`
4 `https://en.wikipedia.org/wiki/Hello%2C%20World%21%22%20program`

4.1.1 Windows

- Create a folder on your computer to use for your Python programs, such as `C:\pythonpractice` , and save your `hello.py` program in that folder.
- In the Start menu, select "Run...", and type in `cmd` . This will cause the Windows terminal to open.
- Type `cd \pythonpractice` to **c** hange **d** irectory to your `pythonpractice` folder, and hit Enter.
- Type `hello.py` to run your program!

If it didn't work, make sure your PATH contains the python directory. See Getting Python[5].

4.1.2 Mac

- Create a folder on your computer to use for your Python programs. A good suggestion would be to name it `pythonpractice` and place it in your Home folder (the one that contains folders for Documents, Movies, Music, Pictures, etc). Save your `hello.py` program into this folder.
- Open the Applications folder, go into the Utilities folder, and open the Terminal program.
- Type `cd pythonpractice` to **c** hange **d** irectory to your `pythonpractice` folder, and hit Enter.
- Type `python ./hello.py` to run your program!

Note:
If you have both Python 2 and Python 3 installed (Your machine comes with a version of Python 2 but you can install Python 3[a] as well), you should run `python3 hello.py`

a https://www.python.org/downloads/

4.1.3 Linux

- Create a folder on your computer to use for your Python programs, such as `~/pythonpractice` , and save your `hello.py` program in that folder..
- Open up the terminal program. In KDE, open the main menu and select "Run Command..." to open Konsole. In GNOME, open the main menu, open the Applications folder, open the Accessories folder, and select Terminal.
- Type `cd ~/pythonpractice` to **c** hange **d** irectory to your `pythonpractice` folder, and hit Enter.
- Type `python ./hello.py` to run your program!

Note:
If you have both Python version 2.6.1 and Python 3.0 installed (Very possible if you are using Ubuntu, and ran **sudo apt-get install python3** to have python3 installed), you should run `python3 hello.py`

5 Chapter 2 on page 5

4.1.4 Linux (advanced)

- Create a folder on your computer to use for your Python programs, such as `~/python-practice` .
- Open up your favorite text editor and create a new file called `hello.py` containing just the following 2 lines (you can copy-paste if you want):[6]

7

```
#! /usr/bin/python
print('Hello, world!')
```

Note:
If you have both python version 2.6.1 and version 3.0 installed (Very possible if you are using a debian or debian-based(*buntu, Mint, ...) distro, and ran **sudo apt-get install python3** to have python3 installed), use

```
! /usr/bin/python3
print('Hello, world!')
```

- save your `hello.py` program in the `~/pythonpractice` folder.
- Open up the terminal program. In KDE, open the main menu and select "Run Command..." to open Konsole. In GNOME, open the main menu, open the Applications folder, open the Accessories folder, and select Terminal.
- Type `cd ~/pythonpractice` to **c** hange **d** irectory to your `pythonpractice` folder, and hit Enter.
- Type `chmod a+x hello.py` to tell Linux that it is an executable program.
- Type `./hello.py` to run your program!
- In addition, you can also use `ln -s hello.py /usr/bin/hello` to make a **s** ymbolic **l in** k `hello.py` to `/usr/bin` under the name `hello` , then run it by simply executing `hello` .

Note that this mainly should be done for complete, compiled programs, if you have a script that you made and use frequently, then it might be a good idea to put it somewhere in your home directory and put a link to it in /usr/bin. If you want a playground, a good idea is to invoke `mkdir ~/.local/bin` and then put scripts in there. To make ~/.local/bin content executable the same way /usr/bin does type `$PATH = $PATH:~/local/bin` (you can add this line into you're shell rc file for exemple ~/.bashrc)

6
7 A Quick Introduction to Unix/My First Shell Script ^{`https://en.wikibooks.org/wiki/A%20Quick%20Introduction%20to%20Unix%2FMy%20First%20Shell%20Script`} explains what a hash bang line does.

Note:
File extensions aren't necessary in UNIX-like file-systems. To linux, hello.py means the exact same thing as hello.txt, hello.mp3, or just hello. Linux mostly uses the contents of the file to determine what type it is.

```
johndoe@linuxbox   $ file /usr/bin/hello
/usr/bin/hello: Python script, ASCII text executable
```

4.1.5 Result

The program should print:

```
Hello, world!
```

Congratulations! You're well on your way to becoming a Python programmer.

4.2 Exercises

1. Modify the `hello.py` program to say hello to someone from your family or your friends (or to Ada Lovelace[8]).
2. Change the program so that after the greeting, it asks, ”How did you get here?”.
3. Re-write the original program to use two `print` statements: one for ”Hello” and one for ”world”. The program should still only print out on one line.

Solutions[9]

4.3 Notes

8 https://en.wikipedia.org/wiki/Ada%20Lovelace
9 https://en.wikibooks.org/wiki/Python%20Programming%2FCreating%20Python%20programs%2FSolutions

5 Basic syntax

There are five fundamental concepts in Python[1].

5.0.1 Case Sensitivity

All variables are case-sensitive. Python treats 'number' and 'Number' as separate, unrelated entities.

5.0.2 Spaces and tabs don't mix

Because whitespace is significant, remember that spaces and tabs don't mix, so use only one or the other when indenting your programs. A common error is to mix them. While they may look the same in editor, the interpreter will read them differently and it will result in either an error or unexpected behavior. Most decent text editors can be configured to let tab key emit spaces instead.

Python's Style Guideline described that the preferred way is using 4 spaces.

Tips: If you invoked python from the command-line, you can give -t or -tt argument to python to make python issue a warning or error on inconsistent tab usage.

```
pythonprogrammer@wikibook: $ python -tt myscript.py
```

This will issue an error if you have mixed spaces and tabs.

5.0.3 Objects

In Python, like all object oriented languages, there are aggregations of code and data called Objects, which typically represent the pieces in a conceptual model of a system.

Objects in Python are created (i.e., instantiated) from templates called Classes[2] (which are covered later, as much of the language can be used without understanding classes). They have "attributes", which represent the various pieces of code and data which comprise the object. To access attributes, one writes the name of the object followed by a period (henceforth called a dot), followed by the name of the attribute.

1 https://en.wikibooks.org/wiki/Python%20Programming
2 Chapter 19 on page 99

An example is the 'upper' attribute of strings, which refers to the code that returns a copy of the string in which all the letters are uppercase. To get to this, it is necessary to have a way to refer to the object (in the following example, the way is the literal string that constructs the object).

```
'bob'.upper
```

Code attributes are called "methods". So in this example, upper is a method of 'bob' (as it is of all strings). To execute the code in a method, use a matched pair of parentheses surrounding a comma separated list of whatever arguments the method accepts (upper doesn't accept any arguments). So to find an uppercase version of the string 'bob', one could use the following:

```
'bob'.upper()
```

5.0.4 Scope

In a large system, it is important that one piece of code does not affect another in difficult to predict ways. One of the simplest ways to further this goal is to prevent one programmer's choice of names from preventing another from choosing that name. Because of this, the concept of scope was invented. A scope is a "region" of code in which a name can be used and outside of which the name cannot be easily accessed. There are two ways of delimiting regions in Python: with functions or with modules. They each have different ways of accessing the useful data that was produced within the scope from outside the scope. With functions, that way is to return the data. The way to access names from other modules lead us to another concept.

5.0.5 Namespaces

It would be possible to teach Python without the concept of namespaces because they are so similar to attributes, which we have already mentioned, but the concept of namespaces is one that transcends any particular programming language, and so it is important to teach. To begin with, there is a built-in function **dir()** that can be used to help one understand the concept of namespaces. When you first start the Python interpreter (i.e., in interactive mode), you can list the objects in the current (or default) namespace using this function.

```
Python 2.3.4 (#53, Oct 18 2004, 20:35:07) [MSC v.1200 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> dir()
['__builtins__', '__doc__', '__name__']
```

This function can also be used to show the names available within a module namespace. To demonstrate this, first we can use the **type()** function to show what __builtins__ is:

```
>>> type(__builtins__)
<type 'module'>
```

Since it is a module, we can list the names within the __builtins__ namespace, again using the **dir()** function (note the complete list of names has been abbreviated):

```
>>> dir(__builtins__)
['ArithmeticError', ... 'copyright', 'credits', ... 'help', ... 'license', ...
 'zip']
>>>
```

Namespaces are a simple concept. A namespace is a place in which a name resides. Each name within a namespace is distinct from names outside of the namespace. This layering of namespaces is called scope. A name is placed within a namespace when that name is given a value. For example:

```
>>> dir()
['__builtins__', '__doc__', '__name__']
>>> name = "Bob"
>>> import math
>>> dir()
['__builtins__', '__doc__', '__name__', 'math', 'name']
```

Note that I was able to add the "name" variable to the namespace using a simple assignment statement. The import statement was used to add the "math" name to the current namespace. To see what math is, we can simply:

```
>>> math
<module 'math' (built-in)>
```

Since it is a module, it also has a namespace. To display the names within this namespace, we:

```
>>> dir(math)
['__doc__', '__name__', 'acos', 'asin', 'atan', 'atan2', 'ceil', 'cos', 'cosh',
 'degrees', 'e',
'exp', 'fabs', 'floor', 'fmod', 'frexp', 'hypot', 'ldexp', 'log', 'log10',
 'modf', 'pi', 'pow',
'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh']
>>>
```

If you look closely, you will notice that both the default namespace, and the math module namespace have a '__name__' object. The fact that each layer can contain an object with the same name is what scope is all about. To access objects inside a namespace, simply use the name of the module, followed by a dot, followed by the name of the object. This allow us to differentiate between the `__name__` object within the current namespace, and that of the object with the same name within the `math` module. For example:

```
>>> print (__name__)
__main__
>>> print (math.__name__)
math
>>> print (math.__doc__)
This module is always available.  It provides access to the
mathematical functions defined by the C standard.
>>> math.pi
3.1415926535897931
```

6 Data types

Data types determine whether an object can do something, or whether it just would not make sense. Other programming languages often determine whether an operation makes sense for an object by making sure the object can never be stored somewhere where the operation will be performed on the object (this type system[1] is called static typing). Python does not do that. Instead it stores the type of an object with the object, and checks when the operation is performed whether that operation makes sense for that object (this is called dynamic typing).

Built-in Data types
Python's built-in (or standard) data types can be grouped into several classes. Sticking to the hierarchy scheme used in the official Python documentation these are **numeric types, sequences, sets and mappings** (and a few more not discussed further here). Some of the types are only available in certain versions of the language as noted below.

- boolean: the type of the built-in values `True` and `False` . Useful in conditional expressions, and anywhere else you want to represent the truth or falsity of some condition. Mostly interchangeable with the integers 1 and 0. In fact, conditional expressions will accept values of any type, treating special ones like boolean `False` , integer 0 and the empty string `""` as equivalent to `False` , and all other values as equivalent to `True` . But for safety's sake, it is best to only use boolean values in these places.

Numeric types:

- int: Integers; equivalent to C longs in Python 2.x, non-limited length in Python 3.x
- long: Long integers of non-limited length; exists only in Python 2.x
- float: Floating-Point numbers, equivalent to C doubles
- complex: Complex Numbers

Sequences:

- str: String; represented as a sequence of 8-bit characters in Python 2.x, but as a sequence of Unicode characters (in the range of U+0000 - U+10FFFF) in Python 3.x
- byte: a sequence of integers in the range of 0-255; only available in Python 3.x
- byte array: like bytes, but mutable (see below); only available in Python 3.x
- list
- tuple

Sets:

- set: an unordered collection of unique objects; available as a standard type since Python 2.6

1 `https://en.wikipedia.org/wiki/Type_system%23Type%20checking`

- frozen set: like set, but immutable (see below); available as a standard type since Python 2.6

Mappings:

- dict: Python dictionaries, also called hashmaps or associative arrays, which means that an element of the list is associated with a definition, rather like a Map in Java[2]

Some others, such as type and callables

Mutable vs Immutable Objects
In general, data types in Python can be distinguished based on whether objects of the type are mutable or immutable. The content of objects of immutable types cannot be changed after they are created.

Some *immutable* types:
- int, float, long, complex
- str
- bytes
- tuple
- frozen set

Some *mutable* types:
- byte array
- list
- set
- dict

Only mutable objects support methods that change the object in place, such as reassignment of a sequence slice, which will work for lists, but raise an error for tuples and strings.

It is important to understand that variables in Python are really just references to objects in memory. If you assign an object to a variable as below

```
a = 1
s = 'abc'
l = ['a string', 456, ('a', 'tuple', 'inside', 'a', 'list')]
```

all you really do is make this variable (`a` , `s` , or `l`) point to the object (`1` , `'abc'` , `['a string', 456, ('a', 'tuple', 'inside', 'a', 'list')]`), which is kept somewhere in memory, as a convenient way of accessing it. If you reassign a variable as below

```
a = 7
s = 'xyz'
l = ['a simpler list', 99, 10]
```

you make the variable point to a different object (newly created ones in our examples). As stated above, only mutable objects can be changed in place (`l[0] = 1` is ok in our example, but `s[0] = 'a'` raises an error). This becomes tricky, when an operation is not explicitly asking for a change to happen in place, as is the case for the `+=` (increment) operator, for example. When used on an immutable object (as in `a += 1` or in `s += 'qwertz'`), Python will silently create a new object and make the variable point to it. However, when used on a mutable object (as in `l += [1,2,3]`), the object pointed to by the variable will be

2 https://en.wikibooks.org/wiki/Java

changed in place. While in most situations, you do not have to know about this different behavior, it is of relevance when several variables are pointing to the same object. In our example, assume you set `p = s` and `m = l` , then `s += 'etc'` and `l += [9,8,7]` . This will change `s` and leave `p` unaffected, but will change both `m` and `l` since both point to the same list object. Python's built-in `id()` function, which returns a unique object identifier for a given variable name, can be used to trace what is happening under the hood.

Typically, this behavior of Python causes confusion in functions. As an illustration, consider this code:

```
def append_to_sequence (myseq):
    myseq += (9,9,9)
    return myseq

t=(1,2,3)      # tuples are immutable
l=[1,2,3]      # lists are mutable

u=append_to_sequence(t)
m=append_to_sequence(l)

print('t = ', t)
print('u = ', u)
print('l = ', l)
print('m = ', m)
```

This will give the (usually unintended) output:

```
t = (1, 2, 3)
u = (1, 2, 3, 9, 9, 9)
l = [1, 2, 3, 9, 9, 9]
m = [1, 2, 3, 9, 9, 9]
```

`myseq` is a local variable of the `append_to_sequence` function, but when this function gets called, `myseq` will nevertheless point to the same object as the variable that we pass in (`t` or `l` in our example). If that object is immutable (like a tuple), there is no problem. The += operator will cause the creation of a new tuple, and `myseq` will be set to point to it. However, if we pass in a reference to a mutable object, that object will be manipulated in place (so `myseq` and `l` , in our case, end up pointing to the same list object).

Links:

- 3.1. Objects, values and types[3], The Python Language Reference, docs.python.org
- 5.6.4. Mutable Sequence Types[4], The Python Standard Library, docs.python.org

Creating Objects of Defined Types

Literal integers can be entered in three ways:

- decimal numbers can be entered directly
- hexadecimal numbers can be entered by prepending a 0x or 0X (0xff is hex FF, or 255 in decimal)

3 `http://docs.python.org/2/reference/datamodel.html#objects-values-and-types`
4 `http://docs.python.org/2/library/stdtypes.html#typesseq-mutable`

- the format of octal literals depends on the version of Python:
  - Python 2.x: octals can be entered by prepending a 0 (0732 is octal 732, or 474 in decimal)
  - Python 3.x: octals can be entered by prepending a 0o or 0O (0o732 is octal 732, or 474 in decimal)

Floating point numbers can be entered directly.

Long integers are entered either directly (1234567891011121314151617181920 is a long integer) or by appending an L (0L is a long integer). Computations involving short integers that overflow are automatically turned into long integers.

Complex numbers are entered by adding a real number and an imaginary one, which is entered by appending a j (i.e. 10+5j is a complex number. So is 10j). Note that j by itself does not constitute a number. If this is desired, use 1j.

Strings can be either single or triple quoted strings. The difference is in the starting and ending delimiters, and in that single quoted strings cannot span more than one line. Single quoted strings are entered by entering either a single quote (') or a double quote (") followed by its match. So therefore

```
'foo' works, and
"moo" works as well,
     but
'bar" does not work, and
"baz' does not work either.
"quux'' is right out.
```

Triple quoted strings are like single quoted strings, but can span more than one line. Their starting and ending delimiters must also match. They are entered with three consecutive single or double quotes, so

```
'''foo''' works, and
"""moo""" works as well,
     but
'"'bar'"' does not work, and
"""baz''' does not work either.
'"'quux"'" is right out.
```

Tuples are entered in parentheses, with commas between the entries:

```
(10, 'Mary had a little lamb')
```

Also, the parenthesis can be left out when it's not ambiguous to do so:

```
10, 'whose fleece was as white as snow'
```

Note that one-element tuples can be entered by surrounding the entry with parentheses and adding a comma like so:

```
('this is a stupid tuple',)
```

Lists are similar, but with brackets:

```
['abc', 1,2,3]
```

Dicts are created by surrounding with curly braces a list of key/value pairs separated from each other by a colon and from the other entries with commas:

```
{ 'hello': 'world', 'weight': 'African or European?' }
```

Any of these composite types can contain any other, to any depth:

```
((((((((('bob',),['Mary', 'had', 'a', 'little', 'lamb']), { 'hello' : 'world' }
 ),),),),),),),)
```

6.1 Null object

The Python analogue of null pointer known from other programming languages is *None* . *None* is not a null pointer or a null reference but an actual object of which there is only one instance. One of the uses of *None* is in default argument values of functions, for which see ../Functions#Default_Argument_Values[5]. Comparisons to *None* are usually made using *is* rather than ==.

Testing for None and assignment:

```
if item is None:
  ...
  another = None

if not item is None:
  ...

if item is not None: # Also possible
  ...
```

Using None in a default argument value:

```
def log(message, type = None):
  ...
```

Links:

- 4. Built-in Constants[6], docs.python.org
- 3.11.7 The Null Object[7], docs.python.org

6.2 Exercises

1. Write a program that instantiates a single object, adds [1,2] to the object, and returns the result.
   a) Find an object that returns an output of the same length (if one exists?).

5 Chapter 14.1.1 on page 74
6 http://docs.python.org/2/library/constants.html?highlight=none#None
7 http://docs.python.org/release/2.5.2/lib/bltin-null-object.html

  b) Find an object that returns an output length 2 greater than it started.
  c) Find an object that causes an error.
2. Find two data types X and Y such that X = X + Y will cause an error, but X += Y will not.

7 Numbers

Python 2.x supports 4 numeric types - int, long, float and complex. Of these, the long type has been dropped in Python 3.x - the int type is now of unlimited length by default. You don't have to specify what type of variable you want; Python does that automatically.

- *Int:* The basic integer type in python, equivalent to the hardware 'c long' for the platform you are using in Python 2.x, unlimited in length in Python 3.x.
- *Long:* Integer type with unlimited length. In python 2.2 and later, Ints are automatically turned into long ints when they overflow. Dropped since Python 3.0, use int type instead.
- *Float:* This is a binary floating point number. Longs and Ints are automatically converted to floats when a float is used in an expression, and with the true-division `/` operator.
- *Complex:* This is a complex number consisting of two floats. Complex literals are written as a + bj where a and b are floating-point numbers denoting the real and imaginary parts respectively.

In general, the number types are automatically 'up cast' in this order:

Int → Long → Float → Complex. The farther to the right you go, the higher the precedence.

```
>>> x = 5
>>> type(x)
<type 'int'>
>>> x = 187687654564658970978909869576453
>>> type(x)
<type 'long'>
>>> x = 1.34763
>>> type(x)
<type 'float'>
>>> x = 5 + 2j
>>> type(x)
<type 'complex'>
```

The result of divisions is somewhat confusing. In Python 2.x, using the / operator on two integers will return another integer, using floor division. For example, `5/2` will give you 2. You have to specify one of the operands as a float to get true division, e.g. `5/2.` or `5./2` (the dot specifies you want to work with float) will yield 2.5. Starting with Python 2.2 this behavior can be changed to true division by the future division statement `from __future__ import division` . In Python 3.x, the result of using the / operator is always true division (you can ask for floor division explicitly by using the // operator since Python 2.2).

This illustrates the behavior of the / operator in Python 2.2+:

```
>>> 5/2
2
>>> 5/2.
2.5
>>> 5./2
2.5
>>> from __future__ import division
>>> 5/2
2.5
>>> 5//2
2
```

8 Strings

8.1 String operations

8.1.1 Equality

Two strings are equal if they have *exactly* the same contents, meaning that they are both the same length and each character has a one-to-one positional correspondence. Many other languages compare strings by identity instead; that is, two strings are considered equal only if they occupy the same space in memory. Python uses the `is` operator[1] to test the identity of strings and any two objects in general.

Examples:

```
>>> a = 'hello'; b = 'hello' # Assign 'hello' to a and b.
>>> a == b                   # check for equality
True
>>> a == 'hello'             #
True
>>> a == "hello"             # (choice of delimiter is unimportant)
True
>>> a == 'hello '            # (extra space)
False
>>> a == 'Hello'             # (wrong case)
False
```

8.1.2 Numerical

There are two quasi-numerical operations which can be done on strings -- addition and multiplication. String addition is just another name for concatenation. String multiplication is repetitive addition, or concatenation. So:

```
>>> c = 'a'
>>> c + 'b'
'ab'
>>> c * 5
'aaaaa'
```

8.1.3 Containment

There is a simple operator 'in' that returns True if the first operand is contained in the second. This also works on substrings

1 Chapter 12.7 on page 63

```
>>> x = 'hello'
>>> y = 'ell'
>>> x in y
False
>>> y in x
True
```

Note that 'print x in y' would have also returned the same value.

8.1.4 Indexing and Slicing

Much like arrays in other languages, the individual characters in a string can be accessed by an integer representing its position in the string. The first character in string s would be s[0] and the nth character would be at s[n-1].

```
>>> s = "Xanadu"
>>> s[1]
'a'
```

Unlike arrays in other languages, Python also indexes the arrays backwards, using negative numbers. The last character has index -1, the second to last character has index -2, and so on.

```
>>> s[-4]
'n'
```

We can also use "slices" to access a substring of s. s[a:b] will give us a string starting with s[a] and ending with s[b-1].

```
>>> s[1:4]
'ana'
```

None of these are assignable.

```
>>> print s
>>> s[0] = 'J'
Traceback (most recent call last):
File "<stdin>", line 1, in ?
TypeError: object does not support item assignment
>>> s[1:3] = "up"
Traceback (most recent call last):
File "<stdin>", line 1, in ?
TypeError: object does not support slice assignment
>>> print s
```

Outputs (assuming the errors were suppressed):

```
Xanadu
Xanadu
```

Another feature of slices is that if the beginning or end is left empty, it will default to the first or last index, depending on context:

```
>>> s[2:]
'nadu'
```

```
>>> s[:3]
'Xan'
>>> s[:]
'Xanadu'
```

You can also use negative numbers in slices:

```
>>> print s[-2:]
'du'
```

To understand slices, it's easiest not to count the elements themselves. It is a bit like counting not on your fingers, but in the spaces between them. The list is indexed like this:

```
Element:     1     2     3     4
Index:    0     1     2     3     4
         -4    -3    -2    -1
```

So, when we ask for the [1:3] slice, that means we start at index 1, and end at index 3, and take everything in between them. If you are used to indexes in C or Java, this can be a bit disconcerting until you get used to it.

8.2 String constants

String constants can be found in the standard string module such as; either single or double quotes may be used to delimit string constants.

8.3 String methods

There are a number of methods or built-in string functions:

- **capitalize**
- **center**
- **count**
- decode
- encode
- endswith
- **expandtabs**
- **find**
- **index**
- **isalnum**
- **isalpha**
- **isdigit**
- **islower**
- **isspace**
- **istitle**
- **isupper**
- **join**
- **ljust**

- **lower**
- **lstrip**
- **replace**
- **rfind**
- **rindex**
- **rjust**
- **rstrip**
- **split**
- **splitlines**
- startswith
- **strip**
- **swapcase**
- **title**
- translate
- **upper**
- zfill

Only emphasized items will be covered.

8.3.1 is*

isalnum(), isalpha(), isdigit(), islower(), isupper(), isspace(), and istitle() fit into this category.

The length of the string object being compared must be at least 1, or the is* methods will return False. In other words, a string object of len(string) == 0, is considered ”empty”, or False.

- **isalnum** returns True if the string is entirely composed of alphabetic and/or numeric characters (i.e. no punctuation).
- **isalpha** and **isdigit** work similarly for alphabetic characters or numeric characters only.
- **isspace** returns True if the string is composed entirely of whitespace.
- **islower** , **isupper** , and **istitle** return True if the string is in lowercase, uppercase, or titlecase respectively. Uncased characters are ”allowed”, such as digits, but there must be at least one cased character in the string object in order to return True. Titlecase means the first cased character of each word is uppercase, and any immediately following cased characters are lowercase. Curiously, 'Y2K'.istitle() returns True. That is because uppercase characters can only follow uncased characters. Likewise, lowercase characters can only follow uppercase or lowercase characters. Hint: whitespace is uncased.

Example:

```
>>> '2YK'.istitle()
False
>>> 'Y2K'.istitle()
True
>>> '2Y K'.istitle()
True
```

8.3.2 Title, Upper, Lower, Swapcase, Capitalize

Returns the string converted to title case, upper case, lower case, inverts case, or capitalizes, respectively.

The **title** method capitalizes the first letter of each word in the string (and makes the rest lower case). Words are identified as substrings of alphabetic characters that are separated by non-alphabetic characters, such as digits, or whitespace. This can lead to some unexpected behavior. For example, the string ”x1x” will be converted to ”X1X” instead of ”X1x”.

The **swapcase** method makes all uppercase letters lowercase and vice versa.

The **capitalize** method is like title except that it considers the entire string to be a word. (i.e. it makes the first character upper case and the rest lower case)

Example:

```
s = 'Hello, wOrLD'
print s              # 'Hello, wOrLD'
print s.title()      # 'Hello, World'
print s.swapcase()   # 'hELLO, WoRld'
print s.upper()      # 'HELLO, WORLD'
print s.lower()      # 'hello, world'
print s.capitalize() # 'Hello, world'
```

Keywords: to lower case, to upper case, lcase, ucase, downcase, upcase.

8.3.3 count

Returns the number of the specified substrings in the string. i.e.

```
>>> s = 'Hello, world'
>>> s.count('o') # print the number of 'o's in 'Hello, World' (2)
2
```

Hint: .count() is case-sensitive, so this example will only count the number of lowercase letter ’o’s. For example, if you ran:

```
>>> s = 'HELLO, WORLD'
>>> s.count('o') # print the number of lowercase 'o's in 'HELLO, WORLD' (0)
0
```

8.3.4 strip, rstrip, lstrip

Returns a copy of the string with the leading (lstrip) and trailing (rstrip) whitespace removed. strip removes both.

```
>>> s = '\t Hello, world\n\t '
>>> print s
        Hello, world

>>> print s.strip()
Hello, world
>>> print s.lstrip()
Hello, world
```

```
        # ends here
>>> print s.rstrip()
        Hello, world
```

Note the leading and trailing tabs and newlines.

Strip methods can also be used to remove other types of characters.

```
import string
s = 'www.wikibooks.org'
print s
print s.strip('w')                # Removes all w's from outside
print s.strip(string.lowercase)   # Removes all lowercase letters from outside
print s.strip(string.printable)   # Removes all printable characters
```

Outputs:

```
www.wikibooks.org
.wikibooks.org
.wikibooks.
```

Note that string.lowercase and string.printable require an import string statement

8.3.5 ljust, rjust, center

left, right or center justifies a string into a given field size (the rest is padded with spaces).

```
>>> s = 'foo'
>>> s
'foo'
>>> s.ljust(7)
'foo    '
>>> s.rjust(7)
'    foo'
>>> s.center(7)
'  foo  '
```

8.3.6 join

Joins together the given sequence with the string as separator:

```
>>> seq = ['1', '2', '3', '4', '5']
>>> ' '.join(seq)
'1 2 3 4 5'
>>> '+'.join(seq)
'1+2+3+4+5'
```

map may be helpful here: (it converts numbers in seq into strings)

```
>>> seq = [1,2,3,4,5]
>>> ' '.join(map(str, seq))
'1 2 3 4 5'
```

now arbitrary objects may be in seq instead of just strings.

8.3.7 find, index, rfind, rindex

The find and index methods return the index of the first found occurrence of the given subsequence. If it is not found, find returns -1 but index raises a ValueError. rfind and rindex are the same as find and index except that they search through the string from right to left (i.e. they find the last occurrence)

```
>>> s = 'Hello, world'
>>> s.find('l')
2
>>> s[s.index('l'):]
'llo, world'
>>> s.rfind('l')
10
>>> s[:s.rindex('l')]
'Hello, wor'
>>> s[s.index('l'):s.rindex('l')]
'llo, wor'
```

Because Python strings accept negative subscripts, index is probably better used in situations like the one shown because using find instead would yield an unintended value.

8.3.8 replace

Replace works just like it sounds. It returns a copy of the string with all occurrences of the first parameter replaced with the second parameter.

```
>>> 'Hello, world'.replace('o', 'X')
'HellX, wXrld'
```

Or, using variable assignment:

```
string = 'Hello, world'
newString = string.replace('o', 'X')
print string
print newString
```

Outputs:

```
Hello, world
HellX, wXrld
```

Notice, the original variable (`string`) remains unchanged after the call to `replace` .

8.3.9 expandtabs

Replaces tabs with the appropriate number of spaces (default number of spaces per tab = 8; this can be changed by passing the tab size as an argument).

```
s = 'abcdefg\tabc\ta'
print s
print len(s)
t = s.expandtabs()
```

```
print t
print len(t)
```

Outputs:

```
abcdefg abc     a
13
abcdefg abc     a
17
```

Notice how (although these both look the same) the second string (t) has a different length because each tab is represented by spaces not tab characters.

To use a tab size of 4 instead of 8:

```
v = s.expandtabs(4)
print v
print len(v)
```

Outputs:

```
abcdefg abc a
13
```

Please note each tab is not always counted as eight spaces. Rather a tab ”pushes” the count to the next multiple of eight. For example:

```
s = '\t\t'
print s.expandtabs().replace(' ', '*')
print len(s.expandtabs())
```

Output:

```
****************
16
```

```
s = 'abc\tabc\tabc'
print s.expandtabs().replace(' ', '*')
print len(s.expandtabs())
```

Outputs:

```
abc*****abc*****abc
19
```

8.3.10 split, splitlines

The **split** method returns a list of the words in the string. It can take a separator argument to use instead of whitespace.

```
>>> s = 'Hello, world'
>>> s.split()
['Hello,', 'world']
>>> s.split('l')
['He', '', 'o, wor', 'd']
```

Note that in neither case is the separator included in the split strings, but empty strings are allowed.

The **splitlines** method breaks a multiline string into many single line strings. It is analogous to split('\n') (but accepts '\r' and '\r\n' as delimiters as well) except that if the string ends in a newline character, **splitlines** ignores that final character (see example).

```
>>> s = """
... One line
... Two lines
... Red lines
... Blue lines
... Green lines
... """
>>> s.split('\n')
['', 'One line', 'Two lines', 'Red lines', 'Blue lines', 'Green lines', '']
>>> s.splitlines()
['', 'One line', 'Two lines', 'Red lines', 'Blue lines', 'Green lines']
```

8.4 Exercises

1. Write a program that takes a string, (1) capitalizes the first letter, (2) creates a list containing each word, and (3) searches for the last occurrence of ”a” in the first word.
2. Run the program on the string ”Bananas are yellow.”
3. Write a program that replaces all instances of ”one” with ”one (1)”. For this exercise capitalization does not matter, so it should treat ”one”, ”One”, and ”oNE” identically.
4. Run the program on the string ”One banana was brown, but one was green.”

8.5 External links

- ”String Methods” chapter[2] -- python.org
- Python documentation of ”string” module[3] -- python.org

2 http://docs.python.org/2/library/stdtypes.html?highlight=rstrip#string-methods
3 http://docs.python.org/2/library/string.html

9 Lists

A list in Python is an ordered group of items (or *elements*). It is a very general structure, and list elements don't have to be of the same type: you can put numbers, letters, strings and nested lists all on the same list.

9.1 Overview

Lists in Python at a glance:

```
list1 = []                      # A new empty list
list2 = [1, 2, 3, "cat"]        # A new non-empty list with mixed item types
list1.append("cat")             # Add a single member, at the end of the list
list1.extend(["dog", "mouse"])  # Add several members
if "cat" in list1:              # Membership test
  list1.remove("cat")           # Remove AKA delete
#list1.remove("elephant") - throws an error
for item in list1:              # Iteration AKA for each item
  print item
print "Item count:", len(list1) # Length AKA size AKA item count
list3 = [6, 7, 8, 9]
for i in range(0, len(list3)):  # Read-write iteration AKA for each item
  list3[i] += 1                 # Item access AKA element access by index
isempty = len(list3) == 0       # Test for emptiness
set1 = set(["cat", "dog"])      # Initialize set from a list
list4 = list(set1)              # Get a list from a set
list5 = list4[:]                # A shallow list copy
list4equal5 = list4==list5      # True: same by value
list4refEqual5 = list4 is list5 # False: not same by reference
list6 = list4[:]
del list6[:]                    # Clear AKA empty AKA erase
print list1, list2, list3, list4, list5, list6, list4equal5, list4refEqual5
print list3[1:3], list3[1:], list3[:2] # Slices
print max(list3 ), min(list3 ), sum(list3) # Aggregates
```

9.2 List creation

There are two different ways to make a list in Python. The first is through assignment ("statically"), the second is using list comprehensions ("actively").

9.2.1 Plain creation

To make a static list of items, write them between square brackets. For example:

```
[ 1,2,3,"This is a list",'c',Donkey("kong") ]
```

Observations:

1. The list contains items of different data types: integer, string, and Donkey class.
2. Objects can be created 'on the fly' and added to lists. The last item is a new instance of Donkey class.

Creation of a new list whose members are constructed from non-literal expressions:

```
a = 2
b = 3
myList = [a+b, b+a, len(["a","b"])]
```

9.2.2 List comprehensions

See also Tips and Tricks[1]

Using list comprehension, you describe the process using which the list should be created. To do that, the list is broken into two pieces. The first is a picture of what each element will look like, and the second is what you do to get it.

For instance, let's say we have a list of words:

```
listOfWords = ["this","is","a","list","of","words"]
```

To take the first letter of each word and make a list out of it using list comprehension, we can do this:

```
>>> listOfWords = ["this","is","a","list","of","words"]
>>> items = [ word[0] for word in listOfWords ]
>>> print items
['t', 'i', 'a', 'l', 'o', 'w']
```

List comprehension supports more than one for statement. It will evaluate the items in all of the objects sequentially and will loop over the shorter objects if one object is longer than the rest.

```
>>> item = [x+y for x in 'cat' for y in 'pot']
>>> print item
['cp', 'co', 'ct', 'ap', 'ao', 'at', 'tp', 'to', 'tt']
```

List comprehension supports an if statement, to only include members into the list that fulfill a certain condition:

```
>>> print [x+y for x in 'cat' for y in 'pot']
['cp', 'co', 'ct', 'ap', 'ao', 'at', 'tp', 'to', 'tt']
>>> print [x+y for x in 'cat' for y in 'pot' if x != 't' and y != 'o' ]
['cp', 'ct', 'ap', 'at']
>>> print [x+y for x in 'cat' for y in 'pot' if x != 't' or y != 'o' ]
['cp', 'co', 'ct', 'ap', 'ao', 'at', 'tp', 'tt']
```

In version 2.x, Python's list comprehension does not define a scope. Any variables that are bound in an evaluation remain bound to whatever they were last bound to when the evaluation was completed. In version 3.x Python's list comprehension uses local variables:

1 https://en.wikibooks.org/wiki/Python%20Programming%2FTips_and_Tricks%23List_comprehension_and_generators

```
>>> print x, y                        #Input to python version 2
r t                                   #Output using python 2

>>> print x, y                        #Input to python version 3
NameError: name 'x' is not defined    #Python 3 returns an error because x and
 y were not leaked
```

This is exactly the same as if the comprehension had been expanded into an explicitly-nested group of one or more 'for' statements and 0 or more 'if' statements.

9.2.3 List creation shortcuts

You can initialize a list to a size, with an initial value for each element:

```
>>> zeros=[0]*5
>>> print zeros
[0, 0, 0, 0, 0]
```

This works for any data type:

```
>>> foos=['foo']*3
>>> print foos
['foo', 'foo', 'foo']
```

But there is a caveat. When building a new list by multiplying, Python copies each item by reference. This poses a problem for mutable items, for instance in a multidimensional array where each element is itself a list. You'd guess that the easy way to generate a two dimensional array would be:

```
listoflists=[ [0]*4 ] *5
```

and this works, but probably doesn't do what you expect:

```
>>> listoflists=[ [0]*4 ] *5
>>> print listoflists
[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
>>> listoflists[0][2]=1
>>> print listoflists
[[0, 0, 1, 0], [0, 0, 1, 0], [0, 0, 1, 0], [0, 0, 1, 0], [0, 0, 1, 0]]
```

What's happening here is that Python is using the same reference to the inner list as the elements of the outer list. Another way of looking at this issue is to examine how Python sees the above definition:

```
>>> innerlist=[0]*4
>>> listoflists=[innerlist]*5
>>> print listoflists
[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
>>> innerlist[2]=1
>>> print listoflists
[[0, 0, 1, 0], [0, 0, 1, 0], [0, 0, 1, 0], [0, 0, 1, 0], [0, 0, 1, 0]]
```

Assuming the above effect is not what you intend, one way around this issue is to use list comprehensions:

```
>>> listoflists=[[0]*4 for i in range(5)]
>>> print listoflists
[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
>>> listoflists[0][2]=1
>>> print listoflists
[[0, 0, 1, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
```

9.3 List Attributes

To find the length of a list use the built in len() method.

```
>>> len([1,2,3])
3
>>> a = [1,2,3,4]
>>> len( a )
4
```

9.4 Combining lists

Lists can be combined in several ways. The easiest is just to 'add' them. For instance:

```
>>> [1,2] + [3,4]
[1, 2, 3, 4]
```

Another way to combine lists is with **extend** . If you need to combine lists inside of a lambda, **extend** is the way to go.

```
>>> a = [1,2,3]
>>> b = [4,5,6]
>>> a.extend(b)
>>> print a
[1, 2, 3, 4, 5, 6]
```

The other way to append a value to a list is to use **append** . For example:

```
>>> p=[1,2]
>>> p.append([3,4])
>>> p
[1, 2, [3, 4]]
>>> # or
>>> print p
[1, 2, [3, 4]]
```

However, [3,4] is an element of the list, and not part of the list. **append** always adds one element only to the end of a list. So if the intention was to concatenate two lists, always use **extend** .

9.5 Getting pieces of lists (slices)

9.5.1 Continuous slices

Like strings[2], lists can be indexed and sliced.

```
>>> list = [2, 4, "usurp", 9.0,"n"]
>>> list[2]
'usurp'
>>> list[3:]
[9.0, 'n']
```

Much like the slice of a string is a substring, the slice of a list is a list. However, lists differ from strings in that we can assign new values to the items in a list.

```
>>> list[1] = 17
>>> list
[2, 17, 'usurp', 9.0,'n']
```

We can even assign new values to slices of the lists, which don't even have to be the same length

```
>>> list[1:4] = ["opportunistic", "elk"]
>>> list
[2, 'opportunistic', 'elk', 'n']
```

It's even possible to append things onto the end of lists by assigning to an empty slice:

```
>>> list[:0] = [3.14,2.71]
>>> list
[3.14, 2.71, 2, 'opportunistic', 'elk', 'n']
```

You can also completely change contents of a list:

```
>>> list[:] = ['new', 'list', 'contents']
>>> list
['new', 'list', 'contents']
```

On the right-hand side of assignment statement can be any iterable type:

```
>>> list[:2] = ('element',('t',),[])
>>> list
['element', ('t',), [], 'contents']
```

With slicing you can create copy of list because slice returns a new list:

```
>>> original = [1, 'element', []]
>>> list_copy = original[:]
>>> list_copy
[1, 'element', []]
>>> list_copy.append('new element')
>>> list_copy
[1, 'element', [], 'new element']
>>> original
[1, 'element', []]
```

2 Chapter 8 on page 29

but this is shallow copy and contains references to elements from original list, so be careful with mutable types:

```
>>> list_copy[2].append('something')
>>> original
[1, 'element', ['something']]
```

9.5.2 Non-Continuous slices

It is also possible to get non-continuous parts of an array. If one wanted to get every n-th occurrence of a list, one would use the :: operator. The syntax is a:b:n where a and b are the start and end of the slice to be operated upon.

```
>>> list = [i for i in range(10) ]
>>> list
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list[::2]
[0, 2, 4, 6, 8]
>>> list[1:7:2]
[1, 3, 5]
```

9.6 Comparing lists

Lists can be compared for equality.

```
>>> [1,2] == [1,2]
True
>>> [1,2] == [3,4]
False
```

Lists can be compared using a less-than operator, which uses lexicographical order:

```
>>> [1,2] < [2,1]
True
>>> [2,2] < [2,1]
False
>>> ["a","b"] < ["b","a"]
True
```

9.7 Sorting lists

Sorting lists is easy with a sort method.

```
>>> list = [2, 3, 1, 'a', 'b']
>>> list.sort()
>>> list
[1, 2, 3, 'a', 'b']
```

Note that the list is sorted in place, and the sort() method returns **None** to emphasize this side effect.

If you use Python 2.4 or higher there are some more sort parameters:

sort(cmp,key,reverse)

cmp : method to be used for sorting key : function to be executed with key element. List is sorted by return-value of the function reverse : sort(reverse=True) or sort(reverse=False)

Python also includes a sorted() function.

```
>>> list = [5, 2, 3, 'q', 'p']
>>> sorted(list)
[2, 3, 5, 'p', 'q']
>>> list
[5, 2, 3, 'q', 'p']
```

Note that unlike the sort() method, sorted(list) does not sort the list in place, but instead returns the sorted list. The sorted() function, like the sort() method also accepts the reverse parameter.

9.8 Iteration

Iteration over lists:

Read-only iteration over a list, AKA for each element of the list:

```
list1 = [1, 2, 3, 4]
for item in list1:
  print item
```

Writable iteration over a list:

```
list1 = [1, 2, 3, 4]
for i in range(0, len(list1)):
  list1[i]+=1 # Modify the item at an index as you see fit
print list
```

From a number to a number with a step:

```
for i in range(1, 13+1, 3): # For i=1 to 13 step 3
  print i
for i in range(10, 5-1, -1): # For i=10 to 5 step -1
  print i
```

For each element of a list satisfying a condition (filtering):

```
for item in list:
  if not condition(item):
    continue
  print item
```

See also ../Loops#For_Loops[3].

3 https://en.wikibooks.org/wiki/..%2FLoops%23For_Loops

9.9 Removing

Removing aka deleting an item at an index (see also #pop(i)[4]):

```
list = [1, 2, 3, 4]
list.pop() # Remove the last item
list.pop(0) # Remove the first item , which is the item at index 0
print list

list = [1, 2, 3, 4]
del list[1] # Remove the 2nd element; an alternative to list.pop(1)
print list
```

Removing an element by value:

```
list = ["a", "a", "b"]
list.remove("a") # Removes only the 1st occurrence of "a"
print list
```

Keeping only items in a list satisfying a condition, and thus removing the items that do not satisfy it:

```
list = [1, 2, 3, 4]
newlist = [item for item in list if item >2]
print newlist
```

This uses a list comprehension[5].

9.10 Aggregates

There are some built-in functions for arithmetic aggregates over lists. These include minimum, maximum, and sum:

```
list = [1, 2, 3, 4]
print max(list), min(list), sum(list)
average = sum(list) / float(len(list)) # Provided the list is non-empty
# The float above ensures the division is a float one rather than integer one.
print average
```

The max and min functions also apply to lists of strings, returning maximum and minimum with respect to alphabetical order:

```
list = ["aa", "ab"]
print max(list), min(list) # Prints "ab aa"
```

9.11 Copying

Copying AKA cloning of lists:

Making a shallow copy:

4 Chapter 9.13.2 on page 48
5 Chapter 9.2.2 on page 40

```
list1= [1, 'element']
list2 = list1[:] # Copy using "[:]"
list2[0] = 2 # Only affects list2, not list1
print list1[0] # Displays 1

# By contrast
list1 = [1, 'element']
list2 = list1
list2[0] = 2 # Modifies the original list
print list1[0] # Displays 2
```

The above does not make a deep copy, which has the following consequence:

```
list1 = [1, [2, 3]] # Notice the second item being a nested list
list2 = list1[:] # A shallow copy
list2[1][0] = 4 # Modifies the 2nd item of list1 as well
print list1[1][0] # Displays 4 rather than 2
```

Making a deep copy:

```
import copy
list1 = [1, [2, 3]] # Notice the second item being a nested list
list2 = copy.deepcopy(list1) # A deep copy
list2[1][0] = 4 # Leaves the 2nd item of list1 unmodified
print list1[1][0] # Displays 2
```

See also #Continuous slices[6].

Links:

- 8.17. copy — Shallow and deep copy operations[7] at docs.python.org

9.12 Clearing

Clearing a list:

```
del list1[:] # Clear a list
list1 = []   # Not really clear but rather assign to a new empty list
```

Clearing using a proper approach makes a difference when the list is passed as an argument:

```
def workingClear(ilist):
  del ilist[:]
def brokenClear(ilist):
  ilist = [] # Lets ilist point to a new list, losing the reference to the
 argument list
list1=[1, 2]; workingClear(list1); print list1
list1=[1, 2]; brokenClear(list1); print list1
```

Keywords: emptying a list, erasing a list, clear a list, empty a list, erase a list.

6 Chapter 9.5.1 on page 43
7 http://docs.python.org/2/library/copy.html

9.13 List methods

9.13.1 append(x)

Add item *x* onto the end of the list.

```
>>> list = [1, 2, 3]
>>> list.append(4)
>>> list
[1, 2, 3, 4]
```

See pop(i)[8]

9.13.2 pop(i)

Remove the item in the list at the index *i* and return it. If *i* is not given, remove the the last item in the list and return it.

```
>>> list = [1, 2, 3, 4]
>>> a = list.pop(0)
>>> list
[2, 3, 4]
>>> a
1
>>> b = list.pop()
>>>list
[2, 3]
>>> b
4
```

9.14 operators

9.14.1 in

The operator 'in' is used for two purposes; either to iterate over every item in a list in a for loop, or to check if a value is in a list returning true or false.

```
>>> list = [1, 2, 3, 4]
>>> if 3 in list:
>>>    ....
>>> l = [0, 1, 2, 3, 4]
>>> 3 in l
True
>>> 18 in l
False
>>>for x in l:
>>>    print x
0
1
2
3
```

8 Chapter 9.13.2 on page 48

4

9.15 Subclassing

In a modern version of Python [which one?], there is a class called 'list'. You can make your own subclass of it, and determine list behaviour which is different from the default standard.

9.16 Exercises

1. Use a list comprehension to construct the list ['ab', 'ac', 'ad', 'bb', 'bc', 'bd'].
2. Use a slice on the above list to construct the list ['ab', 'ad', 'bc'].
3. Use a list comprehension to construct the list ['1a', '2a', '3a', '4a'].
4. Simultaneously remove the element '2a' from the above list and print it.
5. Copy the above list and add '2a' back into the list such that the original is still missing it.
6. Use a list comprehension to construct the list ['abe', 'abf', 'ace', 'acf', 'ade', 'adf', 'bbe', 'bbf', 'bce', 'bcf', 'bde', 'bdf']

9.17 External links

- Python documentation, chapter "Sequence Types"[9] -- python.org
- Python Tutorial, chapter "Lists"[10] -- python.org

}}

9 `http://docs.python.org/2/library/stdtypes.html?highlight=rstrip#sequence-types-str-unicode-list-tuple-byt`
10 `http://docs.python.org/2/tutorial/introduction.html#lists`

10 Dictionaries

A dictionary in Python is a collection of unordered values accessed by key rather than by index. The keys have to be hashable: integers, floating point numbers, strings, tuples, and frozensets are hashable, while lists, dictionaries, and sets other than frozensets are not. Dictionaries were available as early as in Python 1.4.

10.1 Overview

Dictionaries in Python at a glance:

```
dict1 = {}                      # Create an empty dictionary
dict2 = dict()                  # Create an empty dictionary 2
dict2 = {"r": 34, "i": 56}      # Initialize to non-empty value
dict3 = dict([("r", 34), ("i", 56)]) # Init from a list of tuples
dict4 = dict(r=34, i=56)        # Initialize to non-empty value 3
dict1["temperature"] = 32       # Assign value to a key
if "temperature" in dict1:      # Membership test of a key AKA key exists
  del dict1["temperature"]      # Delete AKA remove
equalbyvalue = dict2 == dict3
itemcount2 = len(dict2)         # Length AKA size AKA item count
isempty2 = len(dict2) == 0      # Emptiness test
for key in dict2:               # Iterate via keys
  print key, dict2[key]         # Print key and the associated value
  dict2[key] += 10              # Modify-access to the key-value pair
for value in dict2.values():    # Iterate via values
  print value
dict5 = {} # {x: dict2[x] + 1 for x in dict2 } # Dictionary comprehension in
 Python 2.7 or later
dict6 = dict2.copy()              # A shallow copy
dict6.update({"i": 60, "j": 30}) # Add or overwrite
dict7 = dict2.copy()
dict7.clear()                   # Clear AKA empty AKA erase
print dict1, dict2, dict3, dict4, dict5, dict6, dict7, equalbyvalue, itemcount2
```

10.2 Dictionary notation

Dictionaries may be created directly or converted from sequences. Dictionaries are enclosed in curly braces, `{}`

```
>>> d = {'city':'Paris', 'age':38, (102,1650,1601):'A matrix coordinate'}
>>> seq = [('city','Paris'), ('age', 38), ((102,1650,1601),'A matrix
 coordinate')]
>>> d
{'city': 'Paris', 'age': 38, (102, 1650, 1601): 'A matrix coordinate'}
>>> dict(seq)
{'city': 'Paris', 'age': 38, (102, 1650, 1601): 'A matrix coordinate'}
>>> d == dict(seq)
True
```

Also, dictionaries can be easily created by zipping two sequences.

```
>>> seq1 = ('a','b','c','d')
>>> seq2 = [1,2,3,4]
>>> d = dict(zip(seq1,seq2))
>>> d
{'a': 1, 'c': 3, 'b': 2, 'd': 4}
```

10.3 Operations on Dictionaries

The operations on dictionaries are somewhat unique. Slicing is not supported, since the items have no intrinsic order.

```
>>> d = {'a':1,'b':2, 'cat':'Fluffers'}
>>> d.keys()
['a', 'b', 'cat']
>>> d.values()
[1, 2, 'Fluffers']
>>> d['a']
1
>>> d['cat'] = 'Mr. Whiskers'
>>> d['cat']
'Mr. Whiskers'
>>> 'cat' in d
True
>>> 'dog' in d
False
```

10.4 Combining two Dictionaries

You can combine two dictionaries by using the update method of the primary dictionary. Note that the update method will merge existing elements if they conflict.

```
>>> d = {'apples': 1, 'oranges': 3, 'pears': 2}
>>> ud = {'pears': 4, 'grapes': 5, 'lemons': 6}
>>> d.update(ud)
>>> d
{'grapes': 5, 'pears': 4, 'lemons': 6, 'apples': 1, 'oranges': 3}
>>>
```

10.5 Deleting from dictionary

```
del dictionaryName[membername]
```

10.6 Exercises

Write a program that:

1. Asks the user for a string, then creates the following dictionary. The values are the letters in the string, with the corresponding key being the place in the string.
2. Replaces the entry whose key is the integer 3, with the value ”Pie”.
3. Asks the user for a string of digits, then prints out the values corresponding to those digits.

10.7 External links

- Python documentation, chapter ”Dictionaries”[1] -- python.org
- Python documentation, The Python Standard Library, 5.8. Mapping Types[2] -- python.org

1 `http://docs.python.org/2/tutorial/datastructures.html#dictionaries`
2 `http://docs.python.org/2/library/stdtypes.html#typesmapping`

11 Sets

Starting with version 2.3, Python comes with an implementation of the mathematical set. Initially this implementation had to be imported from the standard module `set` , but with Python 2.6 the types set and frozenset[1] became built-in types. A set is an unordered collection of objects, unlike sequence objects such as lists and tuples, in which each element is indexed. Sets cannot have duplicate members - a given object appears in a set 0 or 1 times. All members of a set have to be hashable, just like dictionary keys. Integers, floating point numbers, tuples, and strings are hashable; dictionaries, lists, and other sets (except frozensets) are not.

11.0.1 Overview

Sets in Python at a glance:

```
set1 = set()                   # A new empty set
set1.add("cat")                # Add a single member
set1.update(["dog", "mouse"])  # Add several members
if "cat" in set1:              # Membership test
  set1.remove("cat")
#set1.remove("elephant") - throws an error
print set1
for item in set1:              # Iteration AKA for each element
  print item
print "Item count:", len(set1) # Length AKA size AKA item count
isempty = len(set1) == 0       # Test for emptiness
set1 = set(["cat", "dog"])     # Initialize set from a list
set2 = set(["dog", "mouse"])
set3 = set1 & set2             # Intersection
set4 = set1 | set2             # Union
set5 = set1 - set3             # Set difference
set6 = set1 ^ set2             # Symmetric difference
issubset = set1 <= set2        # Subset test
issuperset = set1 >= set2      # Superset test
set7 = set1.copy()             # A shallow copy
set7.remove("cat")
set8 = set1.copy()
set8.clear()                   # Clear AKA empty AKA erase
print set1, set2, set3, set4, set5, set6, set7, set8, issubset, issuperset
```

11.0.2 Constructing Sets

One way to construct sets is by passing any sequential object to the "set" constructor.

```
>>> set([0, 1, 2, 3])
set([0, 1, 2, 3])
```

1 Chapter 11.0.8 on page 59

```
>>> set("obtuse")
set(['b', 'e', 'o', 's', 'u', 't'])
```

We can also add elements to sets one by one, using the "add" function.

```
>>> s = set([12, 26, 54])
>>> s.add(32)
>>> s
set([32, 26, 12, 54])
```

Note that since a set does not contain duplicate elements, if we add one of the members of s to s again, the add function will have no effect. This same behavior occurs in the "update" function, which adds a group of elements to a set.

```
>>> s.update([26, 12, 9, 14])
>>> s
set([32, 9, 12, 14, 54, 26])
```

Note that you can give any type of sequential structure, or even another set, to the update function, regardless of what structure was used to initialize the set.

The set function also provides a copy constructor. However, remember that the copy constructor will copy the set, but not the individual elements.

```
>>> s2 = s.copy()
>>> s2
set([32, 9, 12, 14, 54, 26])
```

11.0.3 Membership Testing

We can check if an object is in the set using the same "in" operator as with sequential data types.

```
>>> 32 in s
True
>>> 6 in s
False
>>> 6 not in s
True
```

We can also test the membership of entire sets. Given two sets S_1 and S_2, we check if S_1 is a subset[2] or a superset of S_2.

```
>>> s.issubset(set([32, 8, 9, 12, 14, -4, 54, 26, 19]))
True
>>> s.issuperset(set([9, 12]))
True
```

Note that "issubset" and "issuperset" can also accept sequential data types as arguments

```
>>> s.issuperset([32, 9])
True
```

2 https://en.wikipedia.org/wiki/Subset

Note that the <= and >= operators also express the issubset and issuperset functions respectively.

```
>>> set([4, 5, 7]) <= set([4, 5, 7, 9])
True
>>> set([9, 12, 15]) >= set([9, 12])
True
```

Like lists, tuples, and string, we can use the "len" function to find the number of items in a set.

11.0.4 Removing Items

There are three functions which remove individual items from a set, called pop, remove, and discard. The first, pop, simply removes an item from the set. Note that there is no defined behavior as to which element it chooses to remove.

```
>>> s = set([1,2,3,4,5,6])
>>> s.pop()
1
>>> s
set([2,3,4,5,6])
```

We also have the "remove" function to remove a specified element.

```
>>> s.remove(3)
>>> s
set([2,4,5,6])
```

However, removing a item which isn't in the set causes an error.

```
>>> s.remove(9)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
KeyError: 9
```

If you wish to avoid this error, use "discard." It has the same functionality as remove, but will simply do nothing if the element isn't in the set

We also have another operation for removing elements from a set, clear, which simply removes all elements from the set.

```
>>> s.clear()
>>> s
set([])
```

11.0.5 Iteration Over Sets

We can also have a loop move over each of the items in a set. However, since sets are unordered, it is undefined which order the iteration will follow.

```
>>> s = set("blerg")
>>> for n in s:
...     print n,
...
r b e l g
```

11.0.6 Set Operations

Python allows us to perform all the standard mathematical set operations, using members of set. Note that each of these set operations has several forms. One of these forms, s1.function(s2) will return another set which is created by "function" applied to S_1 and S_2. The other form, s1.function_update(s2), will change S_1 to be the set created by "function" of S_1 and S_2. Finally, some functions have equivalent special operators. For example, s1 & s2 is equivalent to s1.intersection(s2)

Intersection

Any element which is in both S_1 and S_2 will appear in their intersection[3].

```
>>> s1 = set([4, 6, 9])
>>> s2 = set([1, 6, 8])
>>> s1.intersection(s2)
set([6])
>>> s1 & s2
set([6])
>>> s1.intersection_update(s2)
>>> s1
set([6])
```

Union

The union[4] is the merger of two sets. Any element in S_1 or S_2 will appear in their union.

```
>>> s1 = set([4, 6, 9])
>>> s2 = set([1, 6, 8])
>>> s1.union(s2)
set([1, 4, 6, 8, 9])
>>> s1 | s2
set([1, 4, 6, 8, 9])
```

Note that union's update function is simply "update" above[5].

Symmetric Difference

The symmetric difference[6] of two sets is the set of elements which are in one of either set, but not in both.

3 https://en.wikipedia.org/wiki/intersection_%28set_theory%29
4 https://en.wikipedia.org/wiki/union_%28set_theory%29
5 Chapter 11.0.2 on page 55
6 https://en.wikipedia.org/wiki/symmetric_difference

```
>>> s1 = set([4, 6, 9])
>>> s2 = set([1, 6, 8])
>>> s1.symmetric_difference(s2)
set([8, 1, 4, 9])
>>> s1 ^ s2
set([8, 1, 4, 9])
>>> s1.symmetric_difference_update(s2)
>>> s1
set([8, 1, 4, 9])
```

Set Difference

Python can also find the set difference[7] of S_1 and S_2, which is the elements that are in S_1 but not in S_2.

```
>>> s1 = set([4, 6, 9])
>>> s2 = set([1, 6, 8])
>>> s1.difference(s2)
set([9, 4])
>>> s1 - s2
set([9, 4])
>>> s1.difference_update(s2)
>>> s1
set([9, 4])
```

11.0.7 Multiple sets

Starting with Python 2.6, "union", "intersection", and "difference" can work with multiple input by using the set constructor. For example, using "set.intersection()":

```
>>> s1 = set([3, 6, 7, 9])
>>> s2 = set([6, 7, 9, 10])
>>> s3 = set([7, 9, 10, 11])
>>> set.intersection(s1, s2, s3)
set([9, 7])
```

11.0.8 frozenset

A frozenset is basically the same as a set, except that it is immutable - once it is created, its members cannot be changed. Since they are immutable, they are also hashable, which means that frozensets can be used as members in other sets and as dictionary keys. frozensets have the same functions as normal sets, except none of the functions that change the contents (update, remove, pop, etc.) are available.

```
>>> fs = frozenset([2, 3, 4])
>>> s1 = set([fs, 4, 5, 6])
>>> s1
set([4, frozenset([2, 3, 4]), 6, 5])
>>> fs.intersection(s1)
frozenset([4])
>>> fs.add(6)
```

7 https://en.wikipedia.org/wiki/Complement_%28set_theory%29%23Relative_Complement

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'frozenset' object has no attribute 'add'
```

11.0.9 Exercises

1. Create the set {'cat', 1, 2, 3}, call it s.
2. Create the set {'c', 'a', 't', '1', '2', '3'}.
3. Create the frozen set {'cat', 1, 2, 3}, call it fs.
4. Create a set containing the frozenset fs, it should look like {frozenset({'cat', 2, 3, 1})}.

11.0.10 Reference

- Python Tutorial, section "Data Structures", subsection "Sets"[8] -- python.org
- Python Library Reference on Set Types[9] -- python.org

8 https://docs.python.org/2/tutorial/datastructures.html#sets
9 https://docs.python.org/2/library/stdtypes.html#set-types-set-frozenset

12 Operators

12.1 Basics

Python math works like you would expect.

```
>>> x = 2
>>> y = 3
>>> z = 5
>>> x * y
6
>>> x + y
5
>>> x * y + z
11
>>> (x + y) * z
25
```

Note that Python adheres to the PEMDAS order of operations[1].

12.2 Powers

There is a built in exponentiation operator **, which can take either integers, floating point or complex numbers. This occupies its proper place in the order of operations.

```
>>> 2**8
256
```

12.3 Division and Type Conversion

For Python 2.x, dividing two integers or longs uses integer division, also known as "floor division" (applying the floor function[2] after division. So, for example, 5 / 2 is 2. Using "/" to do division this way is deprecated; if you want floor division, use "//" (available in Python 2.2 and later).

"/" does "true division" for floats and complex numbers; for example, 5.0/2.0 is 2.5.

For Python 3.x, "/" does "true division" for all types.[3][4]

1 `https://en.wikipedia.org/wiki/Order%20of%20operations%20`
2 `https://en.wikipedia.org/wiki/Floor%20function`
3 [`http://www.python.org/doc/2.2.3/whatsnew/node7.html` What's New in Python 2.2
4 PEP 238 -- Changing the Division Operator ^{`http://www.python.org/dev/peps/pep-0238/`}

Dividing by or into a floating point number (there are no fractional types in Python) will cause Python to use true division. To coerce an integer to become a float, 'float()' with the integer as a parameter

```
>>> x = 5
>>> float(x)
5.0
```

This can be generalized for other numeric types: int(), complex(), long().

Beware that due to the limitations of floating point arithmetic[5], rounding errors can cause unexpected results. For example:

```
>>> print 0.6/0.2
3.0
>>> print 0.6//0.2
2.0
```

12.4 Modulo

The modulus (remainder of the division of the two operands, rather than the quotient) can be found using the % operator, or by the divmod builtin function. The divmod function returns a tuple containing the quotient and remainder.

```
>>> 10%7
3
```

12.5 Negation

Unlike some other languages, variables can be negated directly:

```
>>> x = 5
>>> -x
-5
```

12.6 Comparison

Numbers, strings and other types can be compared for equality/inequality and ordering:

```
>>> 2 == 3
False
>>> 3 == 3
True
>>> 2 < 3
True
```

5 https://en.wikipedia.org/wiki/floating%20point

```
>>> "a" < "aa"
True
```

12.7 Identity

The operators `is` and `is not` test for object identity: `x is y` is true if and only if x and y are references to the same object in memory. `x is not y` yields the inverse truth value. Note that an identity test is more stringent than an equality test since two distinct objects may have the same value.

```
>>> [1,2,3] == [1,2,3]
True
>>> [1,2,3] is [1,2,3]
False
```

For the built-in immutable data types[6] (like int, str and tuple) Python uses caching mechanisms to improve performance, i.e., the interpreter may decide to reuse an existing immutable object instead of generating a new one with the same value. The details of object caching are subject to changes between different Python versions and are not guaranteed to be system-independent, so identity checks on immutable objects like `'hello' is 'hello'` , `(1,2,3) is (1,2,3)` , `4 is 2**2` may give different results on different machines.

12.8 Augmented Assignment

There is shorthand for assigning the output of an operation to one of the inputs:

```
>>> x = 2
>>> x # 2
2
>>> x *= 3
>>> x # 2 * 3
6
>>> x += 4
>>> x # 2 * 3 + 4
10
>>> x /= 5
>>> x # (2 * 3 + 4) / 5
2
>>> x **= 2
>>> x # ((2 * 3 + 4) / 5) ** 2
4
>>> x %= 3
>>> x # ((2 * 3 + 4) / 5) ** 2 % 3
1

>>> x = 'repeat this  '
>>> x  # repeat this
repeat this
>>> x *= 3  # fill with x repeated three times
>>> x
repeat this  repeat this  repeat this
```

6 Chapter 6 on page 22

12.9 Boolean

or:

```
if a or b:
    do_this
else:
    do_this
```

and:

```
if a and b:
    do_this
else:
    do_this
```

not:

```
if not a:
    do_this
else:
    do_this
```

The order of operations here is: ”not” first, ”and” second, ”or” third. In particular, ”True or True and False or False” becomes ”True or False or False” which is True.

Caution, Boolean operators are valid on things other than Booleans; for instance ”1 and 6” will return 6. Specifically, ”and” returns either the first value considered to be false, or the last value if all are considered true. ”or” returns the first true value, or the last value if all are considered false.

12.10 Exercises

1. Use Python to calculate $2^{2^{2^2}} = 65536$.
2. Use Python to calculate $\frac{(3+2)^4}{7} \approx 89.285$.
3. Use Python to calculate 11111111111111111111+22222222222222222222, but in one line of code with at most 15 characters. (Hint: each of those numbers is 20 digits long, so you have to find some other way to input those numbers)
4. Exactly one of the following expressions evaluates to ”cat”; the other evaluates to ”dog”. Trace the logic to determine which one is which, then check your answer using Python.

```
1 and "cat" or "dog"
0 and "cat" or "dog"
```

12.11 References

13 Flow control

As with most imperative languages, there are three main categories of program control flow:

- loops
- branches
- function calls

Function calls are covered in the next section[1].

Generators and list comprehensions are advanced forms of program control flow, but they are not covered here.

13.0.1 Overview

Control flow in Python at a glance:

```
x = -6                               # Branching
if x > 0:                            # If
  print "Positive"
elif x == 0:                         # Else if AKA elseif
  print "Zero"
else:                                # Else
  print "Negative"
list1 = [100, 200, 300]
for i in list1: print i              # A for loop
for i in range(0, 5): print i        # A for loop from 0 to 4
for i in range(5, 0, -1): print i    # A for loop from 5 to 1
for i in range(0, 5, 2): print i     # A for loop from 0 to 4, step 2
list2 = [(1, 1), (2, 4), (3, 9)]
for x, xsq in list2: print x, xsq    # A for loop with a two-tuple as its
 iterator
l1 = [1, 2]; l2 = ['a', 'b']
for i1, i2 in zip(l1, l2): print i1, i2 # A for loop iterating two lists at
 once.
i = 5
while i > 0:                         # A while loop
  i -= 1
list1 = ["cat", "dog", "mouse"]
i = -1 # -1 if not found
for item in list1:
  i += 1
  if item=="dog":
    break                            # Break; also usable with while loop
print "Index of dog:",i
for i in range(1,6):
  if i <= 4:
    continue                         # Continue; also usable with while loop
  print "Greater than 4:", i
```

1 Chapter 14 on page 73

13.0.2 Loops

In Python, there are two kinds of loops, 'for' loops and 'while' loops.

For loops

A for loop iterates over elements of a sequence (tuple or list). A variable is created to represent the object in the sequence. For example,

```
x = [100,200,300]
for i in x:
      print i
```

This will output

```
100
200
300
```

The `for` loop loops over each of the elements of a list or iterator, assigning the current element to the variable name given. In the example above, each of the elements in `x` is assigned to `i` .

A built-in function called range exists to make creating sequential lists such as the one above easier. The loop above is equivalent to:

```
l = range(100, 301,100)
for i in l:
    print i
```

The next example uses a negative *step* (the third argument for the built-in range function):

```
for i in range(5, 0, -1):
    print i
```

This will output

```
5
4
3
2
1
```

The negative step can be -2:

```
for i in range(10, 0, -2):
    print i
```

This will output

```
10
8
6
4
2
```

For loops can have names for each element of a tuple, if it loops over a sequence of tuples:

```
l = [(1, 1), (2, 4), (3, 9), (4, 16), (5, 25)]
for x, xsquared in l:
    print x, ':', xsquared
```

This will output

```
1 : 1
2 : 4
3 : 9
4 : 16
5 : 25
```

Links:

- 4.2. for Statements[2], The Python Tutorial, docs.python.org
- 4.3. The range() Function[3], The Python Tutorial, docs.python.org

While loops

A while loop repeats a sequence of statements until some condition becomes false. For example:

```
x = 5
while x > 0:
    print x
    x = x - 1
```

Will output:

```
5
4
3
2
1
```

Python's while loops can also have an 'else' clause, which is a block of statements that is executed (once) when the while statement evaluates to false. The break statement inside the while loop will not direct the program flow to the else clause. For example:

```
x = 5
y = x
while y > 0:
```

2 http://docs.python.org/2/tutorial/controlflow.html#for-statements
3 http://docs.python.org/2/tutorial/controlflow.html#the-range-function

```
    print y
    y = y - 1
else:
    print x
```

This will output:

```
5
4
3
2
1
5
```

Unlike some languages, there is no post-condition loop.

Links:

- 3.2. First Steps Towards Programming[4], The Python Tutorial, docs.python.org

Breaking and continuing

Python includes statements to exit a loop (either a for loop or a while loop) prematurely. To exit a loop, use the break statement:

```
x = 5
while x > 0:
    print x
    break
    x -= 1
    print x
```

This will output

```
5
```

The statement to begin the next iteration of the loop without waiting for the end of the current loop is 'continue'.

```
l = [5,6,7]
for x in l:
    continue
    print x
```

This will not produce any output.

Else clause of loops

The else clause of loops will be executed if no break statements are met in the loop.

4 http://docs.python.org/2/tutorial/introduction.html#first-steps-towards-programming

```
l = range(1,100)
for x in l:
    if x == 100:
        print x
        break
    else:
        print x," is not 100"
else:
    print "100 not found in range"
```

Another example of a while loop using the break statement and the else statement:

```
expected_str = "melon"
received_str = "apple"
basket = ["banana", "grapes", "strawberry", "melon", "orange"]
x = 0
step = int(raw_input("Input iteration step: "))

while(received_str != expected_str):
    if(x >= len(basket)): print "No more fruits left on the basket."; break
    received_str = basket[x]
    x += step # Change this to 3 to make the while statement
              # evaluate to false, avoiding the break statement, using the else
 clause.
    if(received_str==basket[2]): print "I hate",basket[2],"!"; break
    if(received_str != expected_str): print "I am waiting for my
 ",expected_str,"."
else:
    print "Finally got what I wanted! my precious ",expected_str,"!"
print "Going back home now !"
```

This will output:

```
Input iteration step: 2
I am waiting for my  melon .
I hate strawberry !
Going back home now !
```

White Space

Python determines where a loop repeats itself by the indentation in the whitespace. Everything that is indented is part of the loop, the next entry that is not indented is not. For example, the code below prints ”1 1 2 1 1 2”

```
for i in [0, 1]:
    for j in ["a","b"]:
        print("1")
    print("2")
```

On the other hand, the code below prints ”1 2 1 2 1 2 1 2”

```
for i in [0, 1]:
    for j in ["a","b"]:
        print("1")
        print("2")
```

13.0.3 Branches

There is basically only one kind of branch in Python, the 'if' statement. The simplest form of the if statement simple executes a block of code only if a given predicate is true, and skips over it if the predicate is false

For instance,

```
>>> x = 10
>>> if x > 0:
...    print "Positive"
...
Positive
>>> if x < 0:
...    print "Negative"
...
```

You can also add "elif" (short for "else if") branches onto the if statement. If the predicate on the first "if" is false, it will test the predicate on the first elif, and run that branch if it's true. If the first elif is false, it tries the second one, and so on. Note, however, that it will stop checking branches as soon as it finds a true predicate, and skip the rest of the if statement. You can also end your if statements with an "else" branch. If none of the other branches are executed, then python will run this branch.

```
>>> x = -6
>>> if x > 0:
...    print "Positive"
... elif x == 0:
...    print "Zero"
... else:
...    print "Negative"
...
'Negative'
```

Links:

- 4.1. if Statements[5], The Python Tutorial, docs.python.org

13.0.4 Conclusion

Any of these loops, branches, and function calls can be nested in any way desired. A loop can loop over a loop, a branch can branch again, and a function can call other functions, or even call itself.

13.1 Exercises

1. Print the numbers from 0 to 1000 (including both 0 and 1000).
2. Print the numbers from 0 to 1000 that are multiples of 5.
3. Print the numbers from 1 to 1000 that are multiples of 5.
4. Use a nested for-loop to prints the 3x3 multiplication table below

5 http://docs.python.org/2/tutorial/controlflow.html#if-statements

```
1 2 3
2 4 6
3 6 9
```

1. Print the 3x3 multiplication table below.

```
  1 2 3
 ------
1|1 2 3
2|2 4 6
3|3 6 9
```

13.2 External links

- 4. More Control Flow Tools[6], The Python Tutorial, docs.python.org

6 http://docs.python.org/2/tutorial/controlflow.html

14 Functions

14.1 Function Calls

A *callable object* is an object that can accept some arguments (also called parameters) and possibly return an object (often a tuple containing multiple objects).

A function is the simplest callable object in Python, but there are others, such as classes[1] or certain class instances.

Defining Functions

A function is defined in Python by the following format:

```
def functionname(arg1, arg2, ...):
    statement1
    statement2
    ...
```

```
>>> def functionname(arg1,arg2):
...     return arg1+arg2
...
>>> t = functionname(24,24) # Result: 48
```

If a function takes no arguments, it must still include the parentheses, but without anything in them:

```
def functionname():
    statement1
    statement2
    ...
```

The arguments in the function definition bind the arguments passed at function invocation (i.e. when the function is called), which are called actual parameters, to the names given when the function is defined, which are called formal parameters. The interior of the function has no knowledge of the names given to the actual parameters; the names of the actual parameters may not even be accessible (they could be inside another function).

A function can 'return' a value, for example:

```
def square(x):
    return x*x
```

1 Chapter 19 on page 99

A function can define variables within the function body, which are considered 'local' to the function. The locals together with the arguments comprise all the variables within the scope of the function. Any names within the function are unbound when the function returns or reaches the end of the function body.

You can **return multiple values** as follows:

```
def first2items(list1):
  return list1[0], list1[1]
a, b = first2items(["Hello", "world", "hi", "universe"])
print a + " " + b
```

Keywords: returning multiple values, multiple return values.

14.1.1 Declaring Arguments

When calling a function that takes some values for further processing, we need to send some values as Function **Arguments** . For example:

```
>>> def find_max(a,b):
   if(a>b):
      print "a is greater than b"
   else:
      print "b is greater than a"
>>> find_max(30,45)  #Here (30,45) are the arguments passing for finding max
 between this two numbers
The ouput will be: 45 is greater than 30
```

Default Argument Values

If any of the formal parameters in the function definition are declared with the format "arg = value," then you will have the option of not specifying a value for those arguments when calling the function. If you do not specify a value, then that parameter will have the default value given when the function executes.

```
>>> def display_message(message, truncate_after=4):
...     print message[:truncate_after]
...
>>> display_message("message")
mess
>>> display_message("message", 6)
messag
```

Links:

- 4.7.1. Default Argument Values[2], The Python Tutorial, docs.python.org

2 http://docs.python.org/2/tutorial/controlflow.html#default-argument-values

Variable-Length Argument Lists

Python allows you to declare two special arguments which allow you to create arbitrary-length argument lists. This means that each time you call the function, you can specify any number of arguments above a certain number.

```
def function(first,second,*remaining):
    statement1
    statement2
    ...
```

When calling the above function, you must provide value for each of the first two arguments. However, since the third parameter is marked with an asterisk, any actual parameters after the first two will be packed into a tuple and bound to ”remaining.”

```
>>> def print_tail(first,*tail):
...     print tail
...
>>> print_tail(1, 5, 2, "omega")
(5, 2, 'omega')
```

If we declare a formal parameter prefixed with *two* asterisks, then it will be bound to a dictionary containing any keyword arguments in the actual parameters which do not correspond to any formal parameters. For example, consider the function:

```
def make_dictionary(max_length=10, **entries):
    return dict([(key, entries[key]) for i, key in enumerate(entries.keys()) if
 i < max_length])
```

If we call this function with any keyword arguments other than max_length, they will be placed in the dictionary ”entries.” If we include the keyword argument of max_length, it will be bound to the formal parameter max_length, as usual.

```
>>> make_dictionary(max_length=2, key1=5, key2=7, key3=9)
{'key3': 9, 'key2': 7}
```

Links:

- 4.7.3. Arbitrary Argument Lists[3], The Python Tutorial, docs.python.org

By Value and by Reference

Objects passed as arguments to functions are passed *by reference* ; they are not being copied around. Thus, passing a large list as an argument does not involve copying all its members to a new location in memory. Note that even integers are objects. However, the distinction of *by value* and *by reference* present in some other programming languages often serves to distinguish whether the passed arguments can be *actually changed* by the called function and whether the *calling function can see the changes* .

Passed objects of *mutable* types such as lists and dictionaries can be changed by the called function and the changes are visible to the calling function. Passed objects of *immutable*

3 `http://docs.python.org/2/tutorial/controlflow.html#arbitrary-argument-lists`

types such as integers and strings cannot be changed by the called function; the calling function can be certain that the called function will not change them. For mutability, see also Data Types[4] chapter.

An example:

```
def appendItem(ilist, item):
  ilist.append(item) # Modifies ilist in a way visible to the caller

def replaceItems(ilist, newcontentlist):
  del ilist[:]                 # Modification visible to the caller
  ilist.extend(newcontentlist) # Modification visible to the caller
  ilist = [5, 6] # No outside effect; lets the local ilist point to a new list
 object,
                # losing the reference to the list object passed as an argument
def clearSet(iset):
  iset.clear()

def tryToTouchAnInteger(iint):
  iint += 1 # No outside effect; lets the local iint to point to a new int
 object,
           # losing the reference to the int object passed as an argument
  print "iint inside:",iint # 4 if iint was 3 on function entry

list1 = [1, 2]
appendItem(list1, 3)
print list1 # [1, 2, 3]
replaceItems(list1, [3, 4])
print list1 # [3, 4]
set1 = set([1, 2])
clearSet(set1 )
print set1 # set([])
int1 = 3
tryToTouchAnInteger(int1)
print int1 # 3
```

14.1.2 Preventing Argument Change

An argument cannot be declared to be constant, not to be changed by the called function. If an argument is of an immutable type, it cannot be changed anyway, but if it is of a mutable type such as list, the calling function is at the mercy of the called function. Thus, if the calling function wants to make sure a passed list does not get changed, it has to pass a copy of the list.

An example:

```
def evilGetLength(ilist):
  length = len(ilist)
  del ilist[:] # Muhaha: clear the list
  return length

list1 = [1, 2]
print evilGetLength(list1) # list1 gets cleared
print list1
list1 = [1, 2]
print evilGetLength(list1[:]) # Pass a copy of list1
print list1
```

4 Chapter 6 on page 22

14.1.3 Calling Functions

A function can be called by appending the arguments in parentheses to the function name, or an empty matched set of parentheses if the function takes no arguments.

```
foo()
square(3)
bar(5, x)
```

A function's return value can be used by assigning it to a variable, like so:

```
x = foo()
y = bar(5,x)
```

As shown above, when calling a function you can specify the parameters by name and you can do so in any order

```
def display_message(message, start=0, end=4):
   print message[start:end]

display_message("message", end=3)
```

This above is valid and start will have the default value of 0. A restriction placed on this is after the first named argument then all arguments after it must also be named. The following is not valid

```
display_message(end=5, start=1, "my message")
```

because the third argument (”my message”) is an unnamed argument.

14.2 Closures

A *closure* is a nested function with an after-return access to the data of the outer function, where the nested function is returned by the outer function as a function object. Thus, even when the outer function has finished its execution after being called, the closure function returned by it can refer to the values of the variables that the outer function had when it defined the closure function.

An example:

```
def adder(outer_argument): # outer function
  def adder_inner(inner_argument): # inner function, nested function
    return outer_argument + inner_argument # Notice outer_argument
  return adder_inner
add5 = adder(5) # a function that adds 5 to its argument
add7 = adder(7) # a function that adds 7 to its argument
print add5(3) # prints 8
print add7(3) # prints 10
```

Closures are possible in Python because functions are *first-class objects* . A function is merely an object of type function. Being an object means it is possible to pass a function object (an uncalled function) around as argument or as return value or to assign another name to the function object. A unique feature that makes closure useful is that the enclosed function may use the names defined in the parent function's scope.

14.3 Lambda Expressions

A lambda is an anonymous (unnamed) function. It is used primarily to write very short functions that are a hassle to define in the normal way. A function like this:

```
>>> def add(a, b):
...    return a + b
...
>>> add(4, 3)
7
```

may also be defined using lambda

```
>>> print (lambda a, b: a + b)(4, 3)
7
```

Lambda is often used as an argument to other functions that expects a function object, such as sorted()'s 'key' argument.

```
>>> sorted([[3, 4], [3, 5], [1, 2], [7, 3]], key=lambda x: x[1])
[[1, 2], [7, 3], [3, 4], [3, 5]]
```

The lambda form is often useful as a closure, such as illustrated in the following example:

```
>>> def attribution(name):
...    return lambda x: x + ' -- ' + name
...
>>> pp = attribution('John')
>>> pp('Dinner is in the fridge')
'Dinner is in the fridge -- John'
```

Note that the lambda function can use the values of variables from the scope[5] in which it was created (like pre and post). This is the essence of closure.

Links:

- 4.7.5. Lambda Expressions[6], The Python Tutorial, docs.python.org

14.3.1 Generator Functions

When discussing loops, you can across the concept of an *iterator* . This yields in turn each element of some sequence, rather than the entire sequence at once, allowing you to deal with sequences much larger than might be able to fit in memory at once.

You can create your own iterators, by defining what is known as a *generator function* . To illustrate the usefulness of this, let us start by considering a simple function to return the *concatenation* of two lists:

```
def concat(a, b) :
    return a + b
#end concat
```

5 Chapter 15 on page 81

6 http://docs.python.org/2/tutorial/controlflow.html#lambda-expressions

```
print concat([5, 4, 3], ["a", "b", "c"])
# prints [5, 4, 3, 'a', 'b', 'c']
```

Imagine wanting to do something like `concat(range(0, 1000000), range(1000000, 2000000))`

That would work, but it would consume a lot of memory.

Consider an alternative definition, which takes two iterators as arguments:

```
def concat(a, b) :
    for i in a :
        yield i
    #end for
    for i in b :
        yield i
    #end b
#end concat
```

Notice the use of the **yield** statement, instead of **return** . We can now use this something like

```
for i in concat(xrange(0, 1000000), xrange(1000000, 2000000))
    print i
#end for
```

and print out an awful lot of numbers, without using a lot of memory at all.

Note:
You can still pass a list or other sequence type wherever Python expects an iterator (like to an argument of your `concat` function); this will still work, and makes it easy not to have to worry about the difference where you don't need to.

14.3.2 External Links

- 4.6. Defining Functions[7], The Python Tutorial, docs.python.org

de:Python unter Linux: Funktionen[8] es:Inmersión en Python/Su primer programa en Python/Declaración de funciones[9] fr:Programmation_Python/Fonction[10] pt:Python/Conceitos básicos/Funções[11]

7 `http://docs.python.org/2/tutorial/controlflow.html#defining-functions`
8 `https://de.wikibooks.org/wiki/Python%20unter%20Linux%3A%20Funktionen`
9 `https://es.wikibooks.org/wiki/Inmersi%C3%B3n%20en%20Python%2FSu%20primer%20programa%20en%20Python%2FDecla`
10 `https://fr.wikibooks.org/wiki/Programmation_Python%2FFonction`
11 `https://pt.wikibooks.org/wiki/Python%2FConceitos%20b%C3%A1sicos%2FFun%C3%A7%C3%B5es`

15 Scoping

15.0.1 Variables

Variables in Python are automatically declared by assignment. Variables are always references to objects, and are never typed. Variables exist only in the current scope or global scope. When they go out of scope, the variables are destroyed, but the objects to which they refer are not (unless the number of references to the object drops to zero).

Scope is delineated by function and class blocks. Both functions and their scopes can be nested. So therefore

```
def foo():
    def bar():
        x = 5 # x is now in scope
        return x + y # y is defined in the enclosing scope later
    y = 10
    return bar() # now that y is defined, bar's scope includes y
```

Now when this code is tested,

```
>>> foo()
15
>>> bar()
Traceback (most recent call last):
  File "<pyshell#26>", line 1, in -toplevel-
    bar()
NameError: name 'bar' is not defined
```

The name 'bar' is not found because a higher scope does not have access to the names lower in the hierarchy.

It is a common pitfall to fail to lookup an attribute (such as a method) of an object (such as a container) referenced by a variable before the variable is assigned the object. In its most common form:

```
>>> for x in range(10):
         y.append(x) # append is an attribute of lists

Traceback (most recent call last):
  File "<pyshell#46>", line 2, in -toplevel-
    y.append(x)
NameError: name 'y' is not defined
```

Here, to correct this problem, one must add y = [] before the for loop.

16 Exceptions

Python handles all errors with exceptions.

An *exception* is a signal that an error or other unusual condition has occurred. There are a number of built-in exceptions, which indicate conditions like reading past the end of a file, or dividing by zero. You can also define your own exceptions.

16.0.1 Raising exceptions

Whenever your program attempts to do something erroneous or meaningless, Python raises exception to such conduct:

```
>>> 1 / 0
Traceback (most recent call last):
    File "<stdin>", line 1, in ?
ZeroDivisionError: integer division or modulo by zero
```

This *traceback* indicates that the ZeroDivisionError exception is being raised. This is a built-in exception -- see below for a list of all the other ones.

16.0.2 Catching exceptions

In order to handle errors, you can set up *exception handling blocks* in your code. The keywords try and except are used to catch exceptions. When an error occurs within the try block, Python looks for a matching except block to handle it. If there is one, execution jumps there.

If you execute this code:

```
try:
    print 1/0
except ZeroDivisionError:
    print "You can't divide by zero, you're silly."
```

Then Python will print this:

You can’t divide by zero, you’re silly.

If you don’t specify an exception type on the except line, it will cheerfully catch all exceptions. This is generally a bad idea in production code, since it means your program will blissfully ignore *unexpected* errors as well as ones which the except block is actually prepared to handle.

Exceptions can propagate up the call stack:

```
def f(x):
    return g(x) + 1

def g(x):
    if x < 0: raise ValueError, "I can't cope with a negative number here."
    else: return 5

try:
    print f(-6)
except ValueError:
    print "That value was invalid."
```

In this code, the print statement calls the function f. That function calls the function g, which will raise an exception of type ValueError. Neither f nor g has a try/except block to handle ValueError. So the exception raised propagates out to the main code, where there *is* an exception-handling block waiting for it. This code prints:

That value was invalid.

Sometimes it is useful to find out exactly what went wrong, or to print the python error text yourself. For example:

```
try:
    the_file = open("the_parrot")
except IOError, (ErrorNumber, ErrorMessage):
    if ErrorNumber == 2: # file not found
        print "Sorry, 'the_parrot' has apparently joined the choir invisible."
    else:
        print "Congratulation! you have managed to trip a #%d error" %
 ErrorNumber
        print ErrorMessage
```

Which of course will print:

Sorry, 'the_parrot' has apparently joined the choir invisible.

Custom Exceptions

Code similar to that seen above can be used to create custom exceptions and pass information along with them. This can be extremely useful when trying to debug complicated projects. Here is how that code would look; first creating the custom exception class:

```
class CustomException(Exception):
    def __init__(self, value):
        self.parameter = value
    def __str__(self):
        return repr(self.parameter)
```

And then using that exception:

```
try:
    raise CustomException("My Useful Error Message")
except CustomException, (instance):
    print "Caught: " + instance.parameter
```

Trying over and over again

16.0.3 Recovering and continuing with finally

Exceptions could lead to a situation where, after raising an exception, the code block where the exception occurred might not be revisited. In some cases this might leave external resources used by the program in an unknown state.

`finally` clause allows programmers to close such resources in case of an exception. Between 2.4 and 2.5 version of python there is change of syntax for `finally` clause.

- Python 2.4

```
try:
    result = None
    try:
        result = x/y
    except ZeroDivisionError:
        print "division by zero!"
    print "result is ", result
finally:
    print "executing finally clause"
```

- Python 2.5

```
try:
    result = x / y
except ZeroDivisionError:
    print "division by zero!"
else:
    print "result is", result
finally:
    print "executing finally clause"
```

16.0.4 Built-in exception classes

All built-in Python exceptions[1]

16.0.5 Exotic uses of exceptions

Exceptions are good for more than just error handling. If you have a complicated piece of code to choose which of several courses of action to take, it can be useful to use exceptions to jump out of the code as soon as the decision can be made. The Python-based mailing list software Mailman does this in deciding how a message should be handled. Using exceptions like this may seem like it's a sort of GOTO -- and indeed it is, but a limited one called an *escape continuation* . Continuations are a powerful functional-programming tool and it can be useful to learn them.

Just as a simple example of how exceptions make programming easier, say you want to add items to a list but you don't want to use "if" statements to initialize the list we could replace this:

1 `http://docs.python.org/library/exceptions.html`

```
if hasattr(self, 'items'):
    self.items.extend(new_items)
else:
    self.items = list(new_items)
```

Using exceptions, we can emphasize the normal program flow—that usually we just extend the list—rather than emphasizing the unusual case:

```
try:
    self.items.extend(new_items)
except AttributeError:
    self.items = list(new_items)
```

17 Input and output

17.1 Input

Note on Python version: The following uses the syntax of Python 2.x. Some of the following is not going to work with Python 3.x.

Python has two functions designed for accepting data directly from the user:

- `input()`
- `raw_input()`

There are also very simple ways of reading a file and, for stricter control over input, reading from stdin if necessary.

17.1.1 raw_input()

raw_input() asks the user for a string of data (ended with a newline), and simply returns the string. It can also take an argument, which is displayed as a prompt before the user enters the data. E.g.

```
print (raw_input('What is your name? '))
```

prints out

```
What is your name? <user input data here>
```

Example: in order to assign the user's name, i.e. string data, to a variable "x" you would type

```
x = raw_input('What is your name?')
```

Once the user inputs his name, e.g. Simon, you can call it as x

```
print ('Your name is ' + x)
```

prints out

```
Your name is Simon
```

Note:
in 3.x "...raw_input() was renamed to input(). That is, the new input() function reads a line from sys.stdin and returns it with the trailing newline stripped. It raises EOFError if the input is terminated prematurely. To get the old behavior of input(), use eval(input())."

17.1.2 input()

input() uses raw_input to read a string of data, and then attempts to evaluate it as if it were a Python program, and then returns the value that results. So entering

```
[1,2,3]
```

would return a list containing those numbers, just as if it were assigned directly in the Python script.

More complicated expressions are possible. For example, if a script says:

```
x = input('What are the first 10 perfect squares? ')
```

it is possible for a user to input:

```
map(lambda x: x*x, range(10))
```

which yields the correct answer in list form. Note that no inputted statement can span more than one line.

input() should not be used for anything but the most trivial program. Turning the strings returned from raw_input() into python types using an idiom such as:

```
x = None
while not x:
    try:
        x = int(raw_input())
    except ValueError:
        print 'Invalid Number'
```

is preferable, as input() uses eval() to turn a literal into a python type. This will allow a malicious person to run arbitrary code from inside your program trivially.

17.1.3 File Input

File Objects

Python includes a built-in file type. Files can be opened by using the file type's constructor:

```
f = file('test.txt', 'r')
```

This means f is open for reading. The first argument is the filename and the second parameter is the mode, which can be 'r', 'w', or 'rw', among some others.

The most common way to read from a file is simply to iterate over the lines of the file:

```
f = open('test.txt', 'r')
for line in f:
    print line[0]
f.close()
```

This will print the first character of each line. Note that a newline is attached to the end of each line read this way.

The newer and better way to read from a file:

```
with open("text.txt", "r") as txt:
    for line in txt:
        print line
```

The advantage is, that the opened file will close itself after reading each line.

Because files are automatically closed when the file object goes out of scope, there is no real need to close them explicitly. So, the loop in the previous code can also be written as:

```
for line in open('test.txt', 'r'):
    print line[0]
```

You can read limited numbers of characters at a time like this:

```
c = f.read(1)
while len(c) > 0:
    if len(c.strip()) > 0: print c,
    c = f.read(1)
```

This will read the characters from f one at a time, and then print them if they're not whitespace.

A file object implicitly contains a marker to represent the current position. If the file marker should be moved back to the beginning, one can either close the file object and reopen it or just move the marker back to the beginning with:

```
f.seek(0)
```

Standard File Objects

Like many other languages, there are built-in file objects representing standard input, output, and error. These are in the sys module and are called stdin, stdout, and stderr. There are also immutable copies of these in __stdin__, __stdout__, and __stderr__. This is for IDLE and other tools in which the standard files have been changed.

You must import the sys module to use the special stdin, stdout, stderr I/O handles.

```
import sys
```

For finer control over input, use sys.stdin.read(). In order to implement the UNIX 'cat' program in Python, you could do something like this:

```
import sys
for line in sys.stdin:
    print line,
```

Note that sys.stdin.read() will read from standard input till EOF. (which is usually Ctrl+D.)

Also important is the sys.argv array. sys.argv is an array that contains the command-line arguments passed to the program.

```
python program.py hello there programmer!
```

This array can be indexed,and the arguments evaluated. In the above example, sys.argv[2] would contain the string ”there”, because the name of the program (”program.py”) is stored in argv[0]. For more complicated command-line argument processing, see the ”argparse” module.

17.2 Output

Note on Python version: The following uses the syntax of Python 2.x. Much of the following is not going to work with Python 3.x. In particular, Python 3.x requires round brackets around arguments to ”print”.

The basic way to do output is the print statement.

```
print 'Hello, world'
```

To print multiple things on the same line separated by spaces, use commas between them, like this:

```
print 'Hello,', 'World'
```

This will print out the following:

```
Hello, World
```

While neither string contained a space, a space was added by the print statement because of the comma between the two objects. Arbitrary data types can be printed this way:

```
print 1,2,0xff,0777,(10+5j),-0.999,map,sys
```

This will output the following:

```
1 2 255 511 (10+5j) -0.999 <built-in function map> <module 'sys' (built-in)>
```

Objects can be printed on the same line without needing to be on the same line if one puts a comma at the end of a print statement:

```
for i in range(10):
    print i,
```

This will output the following:

```
0 1 2 3 4 5 6 7 8 9
```

To end the printed line with a newline, add a print statement without any objects.

```
for i in range(10):
    print i,
print
for i in range(10,20):
    print i,
```

This will output the following:

```
0 1 2 3 4 5 6 7 8 9
10 11 12 13 14 15 16 17 18 19
```

If the bare print statement were not present, the above output would look like:

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
```

You can use similar syntax when writing to a file instead of to standard output, like this:

```
print >> f, 'Hello, world'
```

This will print to any object that implements write(), which includes file objects.

17.2.1 Omitting newlines

To avoid adding spaces and newlines between objects' output with subsequent print statements, you can do one of the following:

Concatenation : Concatenate the string representations of each object, then later print the whole thing at once.

```
print str(1)+str(2)+str(0xff)+str(0777)+str(10+5j)+str(-0.999)+str(map)+str(sys)
```

This will output the following:

```
12255511(10+5j)-0.999<built-in function map><module 'sys' (built-in)>
```

Write function : You can make a shorthand for *sys.stdout.write* and use that for output.

```
import sys
write = sys.stdout.write
write('20')
write('05\n')
```

This will output the following:

```
2005
```

You may need sys.stdout.flush() to get that text on the screen quickly.

17.2.2 Examples

Examples of output with *Python 2.x* :

- print "Hello"
- print "Hello", "world"
  - Separates the two words with a space.
- print "Hello", 34
  - Prints elements of various data types, separating them by a space.
- print "Hello " + 34
  - Throws an error as a result of trying to concatenate a string and an integer.
- print "Hello " + str(34)
  - Uses "+" to concatenate strings, after converting a number to a string.
- print "Hello",
  - Prints "Hello " without a newline, with a space at the end.
- sys.stdout.write("Hello")
  - Prints "Hello" without a newline. Doing "import sys" is a prerequisite. Needs a subsequent "sys.stdout.flush()" in order to display immediately on the user's screen.
- sys.stdout.write("Hello\n")
  - Prints "Hello" with a newline.
- print >> sys.stderr, "An error occurred."
  - Prints to standard error stream.
- sys.stderr.write("Hello\n")
  - Prints to standard error stream.
- sum=2+2; print "The sum: %i" % sum
  - Prints a string that has been formatted with the use of an integer passed as an argument.
- formatted_string = "The sum: %i" % (2+2); print formatted_string
  - Like the previous, just that the formatting happens outside of the print statement.
- print "Float: %6.3f" % 1.23456
  - Outputs "Float: 1.234". The number 3 after the period specifies the number of decimal digits after the period to be displayed, while 6 before the period specifies the total number of characters the displayed number should take, to be padded with spaces if needed.
- print "%s is %i years old" % ("John", 23)
  - Passes two arguments to the formatter.

Examples of output with *Python 3.x* :

- from __future__ import print_function
  - Ensures Python 2.6 and later Python 2.x can use Python 3.x print function.
- print ("Hello", "world")

  - Prints the two words separated with a space. Notice the surrounding brackets, ununsed in Python 2.x.
- print (”Hello world”, end=””)
  - Prints without the ending newline.
- print (”Hello”, ”world”, sep=”-”)
  - Prints the two words separated with a a dash.

17.2.3 File Output

Printing numbers from 1 to 10 to a file, one per line:

```
file1 = open("TestFile.txt","w")
for i in range(1,10+1):
  print >>file1, i
file1.close()
```

With ”w”, the file is opened for writing. With ”>>file”, print sends its output to a file rather than standard output.

Printing numbers from 1 to 10 to a file, separated with a dash:

```
file1 = open("TestFile.txt","w")
for i in range(1,10+1):
  if i>1:
    file1.write("-")
  file1.write(str(i))
file1.close()
```

Opening a file for appending rather than overwriting:

```
file1 = open("TestFile.txt","a")
```

See also ../Files/[1] chapter.

17.3 External Links

- 7. Input and Output[2] in The Python Tutorial, python.org
- 6.6. The print statement[3] in The Python Language Reference, python.org
- 2. Built-in Functions #open[4] in The Python Standard Library at Python Documentation, python.org
- 5. Built-in Types #file.write[5] in The Python Standard Library at Python Documentation, python.org
- 27.1. sys — System-specific parameters and functions[6] in Python Documentation, python org -- mentions sys.stdout, and sys.stderr

1 Chapter 27 on page 149
2 `http://www.python.org/doc/current/tutorial/inputoutput.html`
3 `http://docs.python.org/2/reference/simple_stmts.html#print`
4 `http://docs.python.org/2/library/functions.html#open`
5 `http://docs.python.org/2/library/stdtypes.html?highlight=write#file.write`
6 `http://docs.python.org/2/library/sys.html`

- 2.3.8 File Objects[7] in Python Library Reference, python.org, for ”flush”
- 5.6.2. String Formatting Operations[8] in The Python Standard Library at Python Documentation, python.org -- for ”%i”, ”%s” and similar string formatting
- 7.2.2. The string format operator[9], in Python 2.5 quick reference, nmt.edu, for ”%i”, ”%s” and similar string formatting

7 `http://docs.python.org/release/2.3.5/lib/bltin-file-objects.html`
8 `http://docs.python.org/2/library/stdtypes.html#string-formatting-operations`
9 `http://infohost.nmt.edu/tcc/help/pubs/python25/web/str-format.html`

18 Modules

Modules are a simple way to structure a program. Mostly, there are modules in the standard library and there are other Python files, or directories containing Python files, in the current directory (each of which constitute a module). You can also instruct Python to search other directories for modules by placing their paths in the PYTHONPATH environment variable.

18.1 Importing a Module

Modules in Python are used by importing them. For example,

```
import math
```

This imports the math standard module. All of the functions in that module are namespaced by the module name, i.e.

```
import math
print math.sqrt(10)
```

This is often a nuisance, so other syntaxes are available to simplify this,

```
from string import whitespace
from math import *
from math import sin as SIN
from math import cos as COS
from ftplib import FTP as ftp_connection
print sqrt(10)
```

The first statement means whitespace is added to the current scope (but nothing else is). The second statement means that all the elements in the math namespace is added to the current scope.

Modules can be three different kinds of things:

- Python files
- Shared Objects (under Unix and Linux) with the .so suffix
- DLL's (under Windows) with the .pyd suffix
- directories

Modules are loaded in the order they're found, which is controlled by sys.path. The current directory is always on the path.

Directories should include a file in them called __init__.py, which should probably include the other files in the directory.

Creating a DLL that interfaces with Python is covered in another section.

18.2 Creating a Module

18.2.1 From a File

The easiest way to create a module is by having a file called mymod.py either in a directory recognized by the PYTHONPATH variable or (even easier) in the same directory where you are working. If you have the following file mymod.py

```
class Object1:
        def __init__(self):
                self.name = 'object 1'
```

you can already import this ”module” and create instances of the object *Object1* .

```
import mymod
myobject = mymod.Object1()
from mymod import *
myobject = Object1()
```

18.2.2 From a Directory

It is not feasible for larger projects to keep all classes in a single file. It is often easier to store all files in directories and load all files with one command. Each directory needs to have a `__init__.py` file which contains python commands that are executed upon loading the directory.

Suppose we have two more objects called `Object2` and `Object3` and we want to load all three objects with one command. We then create a directory called *mymod* and we store three files called `Object1.py` , `Object2.py` and `Object3.py` in it. These files would then contain one object per file but this not required (although it adds clarity). We would then write the following `__init__.py` file:

```
from Object1 import *
from Object2 import *
from Object3 import *

__all__ = ["Object1", "Object2", "Object3"]
```

The first three commands tell python what to do when somebody loads the module. The last statement defining __all__ tells python what to do when somebody executes *from mymod import ** . Usually we want to use parts of a module in other parts of a module, e.g. we want to use Object1 in Object2. We can do this easily with an *from . import ** command as the following file *Object2.py* shows:

```
from . import *

class Object2:
        def __init__(self):
                self.name = 'object 2'
                self.otherObject = Object1()
```

We can now start python and import *mymod* as we have in the previous section.

18.3 External links

- Python Documentation[1]

1 `http://docs.python.org/tutorial/modules.html`

19 Classes

Classes are a way of aggregating similar data and functions. A class is basically a scope inside which various code (especially function definitions) is executed, and the locals to this scope become *attributes* of the class, and of any objects constructed by this class. An object constructed by a class is called an *instance* of that class.

19.0.1 Defining a Class

To define a class, use the following format:

```
class ClassName:
    "Here is an explanation about your class"
    pass
```

The capitalization in this class definition is the convention, but is not required by the language. It's usually good to add at least a short explanation of what your class is supposed to do. The pass statement in the code above is just to say to the python interpreter just go on and do nothing. You can remove it as soon as you are adding your first statement.

19.0.2 Instance Construction

The class is a callable object that constructs an instance of the class when called. Let's say we create a class Foo.

```
class Foo:
    "Foo is our new toy."
    pass
```

To construct an instance of the class, Foo, "call" the class object:

```
f = Foo()
```

This constructs an instance of class Foo and creates a reference to it in f.

19.0.3 Class Members

In order to access the member of an instance of a class, use the syntax <class instance>.<member>. It is also possible to access the members of the class definition with <class name>.<member>.

Methods

A method is a function within a class. The first argument (methods must always take at least one argument) is always the instance of the class on which the function is invoked. For example

```
>>> class Foo:
...     def setx(self, x):
...         self.x = x
...     def bar(self):
...         print self.x
```

If this code were executed, nothing would happen, at least until an instance of Foo were constructed, and then bar were called on that instance.

Invoking Methods

Calling a method is much like calling a function, but instead of passing the instance as the first parameter like the list of formal parameters suggests, use the function as an attribute of the instance.

```
>>> f = Foo()
>>> f.setx(5)
>>> f.bar()
```

This will output

```
5
```

It is possible to call the method on an arbitrary object, by using it as an attribute of the defining class instead of an instance of that class, like so:

```
>>> Foo.setx(f,5)
>>> Foo.bar(f)
```

This will have the same output.

Dynamic Class Structure

As shown by the method setx above, the members of a Python class can change during runtime, not just their values, unlike classes in languages like C or Java. We can even delete f.x after running the code above.

```
>>> del f.x
>>> f.bar()
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
  File "<stdin>", line 5, in bar
AttributeError: Foo instance has no attribute 'x'
```

Another effect of this is that we can change the definition of the Foo class during program execution. In the code below, we create a member of the Foo class definition named y. If we then create a new instance of Foo, it will now have this new member.

```
>>> Foo.y = 10
>>> g = Foo()
>>> g.y
10
```

Viewing Class Dictionaries

At the heart of all this is a dictionary[1] that can be accessed by ”vars(ClassName)”

```
>>> vars(g)
{}
```

At first, this output makes no sense. We just saw that g had the member y, so why isn't it in the member dictionary? If you remember, though, we put y in the class definition, Foo, not g.

```
>>> vars(Foo)
{'y': 10, 'bar': <function bar at 0x4d6a3c>, '__module__': '__main__',
 'setx': <function setx at 0x4d6a04>, '__doc__': None}
```

And there we have all the members of the Foo class definition. When Python checks for g.member, it first checks g's vars dictionary for ”member,” then Foo. If we create a new member of g, it will be added to g's dictionary, but not Foo's.

```
>>> g.setx(5)
>>> vars(g)
{'x': 5}
```

Note that if we now assign a value to g.y, we are not assigning that value to Foo.y. Foo.y will still be 10, but g.y will now override Foo.y

```
>>> g.y = 9
>>> vars(g)
{'y': 9, 'x': 5}
>>> vars(Foo)
{'y': 10, 'bar': <function bar at 0x4d6a3c>, '__module__': '__main__',
 'setx': <function setx at 0x4d6a04>, '__doc__': None}
```

Sure enough, if we check the values:

```
>>> g.y
9
>>> Foo.y
10
```

Note that f.y will also be 10, as Python won't find 'y' in vars(f), so it will get the value of 'y' from vars(Foo).

1 Chapter 10 on page 51

Some may have also noticed that the methods in Foo appear in the class dictionary along with the x and y. If you remember from the section on lambda functions[2], we can treat functions just like variables. This means that we can assign methods to a class during runtime in the same way we assigned variables. If you do this, though, remember that if we call a method of a class instance, the first parameter passed to the method will always be the class instance itself.

Changing Class Dictionaries

We can also access the members dictionary of a class using the __dict__ member of the class.

```
>>> g.__dict__
{'y': 9, 'x': 5}
```

If we add, remove, or change key-value pairs from g.__dict__, this has the same effect as if we had made those changes to the members of g.

```
>>> g.__dict__['z'] = -4
>>> g.z
-4
```

19.0.4 New Style Classes

New style classes were introduced in python 2.2. A new-style class is a class that has a built-in as its base, most commonly object. At a low level, a major difference between old and new classes is their type. Old class instances were all of type `instance` . New style class instances will return the same thing as x.__class__ for their type. This puts user defined classes on a level playing field with built-ins. Old/Classic classes are slated to disappear in Python 3. With this in mind all development should use new style classes. New Style classes also add constructs like properties and static methods familiar to Java programmers.

Old/Classic Class

```
>>> class ClassicFoo:
...     def __init__(self):
...         pass
```

New Style Class

```
>>> class NewStyleFoo(object):
...     def __init__(self):
...         pass
```

Properties

Properties are attributes with getter and setter methods.

2 Chapter 14.3 on page 78

```
>>> class SpamWithProperties(object):
...     def __init__(self):
...         self.__egg = "MyEgg"
...     def get_egg(self):
...         return self.__egg
...     def set_egg(self, egg):
...         self.__egg = egg
...     egg = property(get_egg, set_egg)

>>> sp = SpamWithProperties()
>>> sp.egg
'MyEgg'
>>> sp.egg = "Eggs With Spam"
>>> sp.egg
'Eggs With Spam'
>>>
```

and since Python 2.6, with @property decorator

```
>>> class SpamWithProperties(object):
...     def __init__(self):
...         self.__egg = "MyEgg"
...     @property
...     def egg(self):
...         return self.__egg
...     @egg.setter
...     def egg(self, egg):
...         self.__egg = egg
```

Static Methods

Static methods in Python are just like their counterparts in C++ or Java. Static methods have no "self" argument and don't require you to instantiate the class before using them. They can be defined using staticmethod()

```
>>> class StaticSpam(object):
...     def StaticNoSpam():
...         print "You can't have have the spam, spam, eggs and spam without any
 spam... that's disgusting"
...     NoSpam = staticmethod(StaticNoSpam)

>>> StaticSpam.NoSpam()
You can't have have the spam, spam, eggs and spam without any spam... that's
 disgusting
```

They can also be defined using the function decorator @staticmethod.

```
>>> class StaticSpam(object):
...     @staticmethod
...     def StaticNoSpam():
...         print "You can't have have the spam, spam, eggs and spam without any
 spam... that's disgusting"
```

19.0.5 Inheritance

Like all object oriented languages, Python provides for inheritance. Inheritance is a simple concept by which a class can extend the facilities of another class, or in Python's case, multiple other classes. Use the following format for this:

```
class ClassName(superclass1,superclass2,superclass3,...):
    ...
```

The subclass will then have all the members of its superclasses. If a method is defined in the subclass and in the superclass, the member in the subclass will override the one in the superclass. In order to use the method defined in the superclass, it is necessary to call the method as an attribute on the defining class, as in Foo.setx(f,5) above:

```
>>> class Foo:
...     def bar(self):
...         print "I'm doing Foo.bar()"
...     x = 10
...
>>> class Bar(Foo):
...     def bar(self):
...         print "I'm doing Bar.bar()"
...         Foo.bar(self)
...     y = 9
...
>>> g = Bar()
>>> Bar.bar(g)
I'm doing Bar.bar()
I'm doing Foo.bar()
>>> g.y
9
>>> g.x
10
```

Once again, we can see what's going on under the hood by looking at the class dictionaries.

```
>>> vars(g)
{}
>>> vars(Bar)
{'y': 9, '__module__': '__main__', 'bar': <function bar at 0x4d6a04>,
 '__doc__': None}
>>> vars(Foo)
{'x': 10, '__module__': '__main__', 'bar': <function bar at 0x4d6994>,
 '__doc__': None}
```

When we call g.x, it first looks in the vars(g) dictionary, as usual. Also as above, it checks vars(Bar) next, since g is an instance of Bar. However, thanks to inheritance, Python will check vars(Foo) if it doesn't find x in vars(Bar).

19.0.6 Special Methods

There are a number of methods which have reserved names which are used for special purposes like mimicking numerical or container operations, among other things. All of these names begin and end with two underscores. It is convention that methods beginning with a single underscore are 'private' to the scope they are introduced within.

Initialization and Deletion

__init__

One of these purposes is constructing an instance, and the special name for this is '__init__'. __init__() is called before an instance is returned (it is not necessary to return the instance manually). As an example,

```
class A:
    def __init__(self):
        print 'A.__init__()'
a = A()
```

outputs

```
A.__init__()
```

__init__() can take arguments, in which case it is necessary to pass arguments to the class in order to create an instance. For example,

```
class Foo:
    def __init__ (self, printme):
        print printme
foo = Foo('Hi!')
```

outputs

```
Hi!
```

Here is an example showing the difference between using __init__() and not using __init__():

```
class Foo:
    def __init__ (self, x):
         print x
foo = Foo('Hi!')
class Foo2:
    def setx(self, x):
        print x
f = Foo2()
Foo2.setx(f,'Hi!')
```

outputs

```
Hi!
Hi!
```

__del__

Similarly, '__del__' is called when an instance is destroyed; e.g. when it is no longer referenced.

Representation

__str__ Converting an object to a string, as with the print statement or with the str() conversion function, can be overridden by overriding __str__. Usually, __str__ returns a formatted version of the objects content. This will NOT usually be something that can be executed.For example:

```
class Bar:
    def __init__ (self, iamthis):
        self.iamthis = iamthis
    def __str__ (self):
        return self.iamthis
bar = Bar('apple')
print bar
```

outputs apple **__repr__** This function is much like __str__(). If __str__ is not present but this one is, this function's output is used instead for printing.__repr__ is used to return a representation of the object in string form. In general, it can be executed to get back the original object.For example:

```
class Bar:
    def __init__ (self, iamthis):
        self.iamthis = iamthis
    def __repr__(self):
        return "Bar('%s')" % self.iamthis
bar = Bar('apple')
bar
```

outputs (note the difference: now is not necessary to put it inside a print) Bar('apple')

String Representation Override Functions	
Function	**Operator**
__str__	str(A)
__repr__	repr(A)
__unicode__	unicode(x) (2.x only)

Attributes

__setattr__ This is the function which is in charge of setting attributes of a class. It is provided with the name and value of the variables being assigned. Each class, of course, comes with a default __setattr__ which simply sets the value of the variable, but we can override it.

```
>>> class Unchangable:
...    def __setattr__(self, name, value):
...        print "Nice try"
...
>>> u = Unchangable()
>>> u.x = 9
Nice try
>>> u.x
```

Traceback (most recent call last): File "<stdin>", line 1, in ? AttributeError: Unchangable instance has no attribute 'x' **__getattr__** Similar to __setattr__, except this function is called when we try to access a class member, and the default simply returns the value.

```
>>> class HiddenMembers:
...     def __getattr__(self, name):
...         return "You don't get to see " + name
...
>>> h = HiddenMembers()
>>> h.anything
"You don't get to see anything"
```

__delattr__ This function is called to delete an attribute.

```
>>> class Permanent:
...     def __delattr__(self, name):
...         print name, "cannot be deleted"
...
>>> p = Permanent()
>>> p.x = 9
>>> del p.x
x cannot be deleted
>>> p.x
9
```

Attribute Override Functions		
Function	**Indirect form**	**Direct Form**
__getattr__	getattr(A, B)	A.B
__setattr__	setattr(A, B, C)	A.B = C
__delattr__	delattr(A, B)	del A.B

Operator Overloading

Operator overloading allows us to use the built-in Python syntax and operators to call functions which we define.

Binary Operators

Binary Operator Override Functions	
Function	**Operator**
__add__	A + B
__sub__	A - B
__mul__	A * B
__truediv__	A / B
__floordiv__	A // B
__mod__	A % B
__pow__	A ** B
__and__	A & B
__or__	A | B
__xor__	A ˆ B
__eq__	A == B
__ne__	A != B
__gt__	A > B
__lt__	A < B
__ge__	A >= B
__le__	A <= B
__lshift__	A << B
__rshift__	A >> B
__contains__	A in B A not in B

If a class has the __add__ function, we can use the '+' operator to add instances of the class. This will call __add__ with the two instances of the class passed as parameters, and the return value will be the result of the addition.

```
>>> class FakeNumber:
...     n = 5
...     def __add__(A,B):
...         return A.n + B.n
...
>>> c = FakeNumber()
>>> d = FakeNumber()
>>> d.n = 7
>>> c + d
12
```

To override the augmented assignment[3] operators, merely add 'i' in front of the normal binary operator, i.e. for '+=' use '__iadd__' instead of '__add__'. The function will be given one argument, which will be the object on the right side of the augmented assignment operator. The returned value of the function will then be assigned to the object on the left of the operator.

```
>>> c.__imul__ = lambda B: B.n - 6
>>> c *= d
>>> c
1
```

It is important to note that the augmented assignment[4] operators will also use the normal operator functions if the augmented operator function hasn't been set directly. This will work as expected, with "__add__" being called for "+=" and so on.

```
>>> c = FakeNumber()
>>> c += d
>>> c
12
```

3 Chapter 12.8 on page 63
4 Chapter 12.8 on page 63

Unary Operators

Unary operators will be passed simply the instance of the class that they are called on.

```
>>> FakeNumber.__neg__ = lambda A : A.n + 6
>>> -d
13
```

Unary Operator Override Functions	
Function	**Operator**
__pos__	+A
__neg__	-A
__inv__	˜A
__abs__	abs(A)
__len__	len(A)

Item Operators

It is also possible in Python to override the indexing and slicing[5] operators. This allows us to use the class[i] and class[a:b] syntax on our own objects.The simplest form of item operator is __getitem__. This takes as a parameter the instance of the class, then the value of the index.

```
>>> class FakeList:
...     def __getitem__(self,index):
...         return index * 2
...
>>> f = FakeList()
>>> f['a']
'aa'
```

We can also define a function for the syntax associated with assigning a value to an item. The parameters for this function include the value being assigned, in addition to the parameters from __getitem__

```
>>> class FakeList:
...     def __setitem__(self,index,value):
...         self.string = index + " is now " + value
...
>>> f = FakeList()
>>> f['a'] = 'gone'
>>> f.string
'a is now gone'
```

We can do the same thing with slices. Once again, each syntax has a different parameter list associated with it.

```
>>> class FakeList:
...     def __getslice___(self,start,end):
...         return str(start) + " to " + str(end)
...
>>> f = FakeList()
>>> f[1:4]
'1 to 4'
```

Keep in mind that one or both of the start and end parameters can be blank in slice syntax. Here, Python has default value for both the start and the end, as show below.

```
>> f[:]
'0 to 2147483647'
```

Note that the default value for the end of the slice shown here is simply the largest possible signed integer on a 32-bit system, and may vary depending on your system and C compiler.

- __setslice__ has the parameters (self,start,end,value)

We also have operators for deleting items and slices.

- __delitem__ has the parameters (self,index)
- __delslice__ has the parameters (self,start,end)

Item Operator Override Functions	
Function	**Operator**
__getitem__	C[i]
__setitem__	C[i] = v
__delitem__	del C[i]
__getslice__	C[s:e]
__setslice__	C[s:e] = v
__delslice__	del C[s:e]

Other Overrides

Other Override Functions	
Function	**Operator**
__cmp__	cmp(x, y)
__hash__	hash(x)
__nonzero__	bool(x)
__call__	f(x)
__iter__	iter(x)
__reversed__	reversed(x) (2.6+)
__divmod__	divmod(x, y)
__int__	int(x)
__long__	long(x)
__float__	float(x)
__complex__	complex(x)
__hex__	hex(x)
__oct__	oct(x)
__index__	
__copy__	copy.copy(x)
__deepcopy__	copy.deepcopy(x)
__sizeof__	sys.getsizeof(x) (2.6+)
__trunc__	math.trunc(x) (2.6+)
__format__	format(x, ...) (2.6+)

19.0.7 Programming Practices

The flexibility of python classes means that classes can adopt a varied set of behaviors. For the sake of understandability, however, it's best to use many of Python's tools sparingly. Try to declare all methods in the class definition, and always use the <class>.<member> syntax instead of __dict__ whenever possible. Look at classes in C++[6] and Java[7] to see what most programmers will expect from a class.

Encapsulation

Since all python members of a python class are accessible by functions/methods outside the class, there is no way to enforce encapsulation[8] short of overriding __getattr__, __setattr__ and __delattr__. General practice, however, is for the creator of a class or module to simply trust that users will use only the intended interface and avoid limiting access to the workings of the module for the sake of users who do need to access it. When using parts of a class or module other than the intended interface, keep in mind that the those parts may change in later versions of the module, and you may even cause errors or undefined behaviors in the module.since encapsulation is private.

Doc Strings

When defining a class, it is convention to document the class using a string literal at the start of the class definition. This string will then be placed in the __doc__ attribute of the class definition.

```
>>> class Documented:
...     """This is a docstring"""
...     def explode(self):
...         """
...         This method is documented, too! The coder is really serious about
...         making this class usable by others who don't know the code as well
...         as he does.
...
...         """
...         print "boom"
>>> d = Documented()
>>> d.__doc__
'This is a docstring'
```

Docstrings are a very useful way to document your code. Even if you never write a single piece of separate documentation (and let's admit it, doing so is the lowest priority for many coders), including informative docstrings in your classes will go a long way toward making them usable.

Several tools exist for turning the docstrings in Python code into readable API documentation, *e.g.* , EpyDoc[9].

6 https://en.wikibooks.org/wiki/C%2B%2B%20Programming%2FClasses
7 https://en.wikipedia.org/wiki/Class%20%28computer%20science%29%23Java
8 https://en.wikipedia.org/wiki/Information%20Hiding
9 http://epydoc.sourceforge.net/using.html

Don't just stop at documenting the class definition, either. Each method in the class should have its own docstring as well. Note that the docstring for the method *explode* in the example class *Documented* above has a fairly lengthy docstring that spans several lines. Its formatting is in accordance with the style suggestions of Python's creator, Guido van Rossum in PEP 8[10].

Adding methods at runtime

To a class

It is fairly easy to add methods to a class at runtime. Lets assume that we have a class called *Spam* and a function cook. We want to be able to use the function cook on all instances of the class Spam:

```
class Spam:
  def __init__(self):
    self.myeggs = 5

def cook(self):
  print "cooking %s eggs" % self.myeggs

Spam.cook = cook    #add the function to the class Spam
eggs = Spam()       #NOW create a new instance of Spam
eggs.cook()         #and we are ready to cook!
```

This will output

```
cooking 5 eggs
```

To an instance of a class

It is a bit more tricky to add methods to an instance of a class that has already been created. Lets assume again that we have a class called *Spam* and we have already created eggs. But then we notice that we wanted to cook those eggs, but we do not want to create a new instance but rather use the already created one:

```
class Spam:
  def __init__(self):
    self.myeggs = 5

eggs = Spam()

def cook(self):
  print "cooking %s eggs" % self.myeggs

import types
f = types.MethodType(cook, eggs, Spam)
eggs.cook = f
```

10 http://www.python.org/dev/peps/pep-0008/

```
eggs.cook()
```

Now we can cook our eggs and the last statement will output:

```
cooking 5 eggs
```

Using a function

We can also write a function that will make the process of adding methods to an instance of a class easier.

```
def attach_method(fxn, instance, myclass):
  f = types.MethodType(fxn, instance, myclass)
  setattr(instance, fxn.__name__, f)
```

All we now need to do is call the attach_method with the arguments of the function we want to attach, the instance we want to attach it to and the class the instance is derived from. Thus our function call might look like this:

```
attach_method(cook, eggs, Spam)
```

Note that in the function add_method we cannot write `instance.fxn = f` since this would add a function called fxn to the instance.

fr:Programmation Python/Programmation orienté objet[11] pt:Python/Conceitos básicos/Classes[12]

11 https://fr.wikibooks.org/wiki/Programmation%20Python%2FProgrammation%20orient%C3%A9%20objet
12 https://pt.wikibooks.org/wiki/Python%2FConceitos%20b%C3%A1sicos%2FClasses

20 Metaclasses

In Python, classes are themselves objects. Just as other objects are instances of a particular class, classes themselves are instances of a metaclass.

20.0.1 Python3

The Pep 3115[1] defines the changes to python 3 metaclasses. In python3 you have a method __prepare__ that is called in the metaclass to create a dictionary or other class to store the class members.[2] Then there is the __new__ method that is called to create new instances of that class. [3]

20.0.2 Class Factories

The simplest use of Python metaclasses is a class factory. This concept makes use of the fact that class definitions in Python are first-class objects. Such a function can create or modify a class definition, using the same syntax[4] one would normally use in declaring a class definition. Once again, it is useful to use the model of classes as dictionaries[5]. First, let's look at a basic class factory:

```
>>> def StringContainer():
...     # define a class
...     class String:
...         def __init__(self):
...             self.content_string = ""
...         def len(self):
...             return len(self.content_string)
...     # return the class definition
...     return String
...
>>> # create the class definition
... container_class = StringContainer()
>>>
>>> # create an instance of the class
... wrapped_string = container_class()
>>>
>>> # take it for a test drive
... wrapped_string.content_string = 'emu emissary'
>>> wrapped_string.len()
12
```

1 http://www.python.org/dev/peps/pep-3115/
2 http://www.python.org/dev/peps/pep-3115/
3 http://eli.thegreenplace.net/2011/08/14/python-metaclasses-by-example/
4 Chapter 19.0.1 on page 99
5 Chapter 19.0.3 on page 101

Of course, just like any other data in Python, class definitions can also be modified. Any modifications to attributes in a class definition will be seen in any instances of that definition, so long as that instance hasn't overridden the attribute that you're modifying.

```
>>> def DeAbbreviate(sequence_container):
...     sequence_container.length = sequence_container.len
...     del sequence_container.len
...
>>> DeAbbreviate(container_class)
>>> wrapped_string.length()
12
>>> wrapped_string.len()
 Traceback (most recent call last):
   File "<stdin>", line 1, in ?
 AttributeError: String instance has no attribute 'len'
```

You can also delete class definitions, but that will not affect instances of the class.

```
>>> del container_class
>>> wrapped_string2 = container_class()
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
NameError: name 'container_class' is not defined
>>> wrapped_string.length()
12
```

20.0.3 The type Metaclass

The metaclass for all standard Python types is the "type" object.

```
>>> type(object)
<type 'type'>
>>> type(int)
<type 'type'>
>>> type(list)
<type 'type'>
```

Just like list, int and object, "type" is itself a normal Python object, and is itself an instance of a class. In this case, it is in fact an instance of itself.

```
>>> type(type)
<type 'type'>
```

It can be instantiated to create new class objects similarly to the class factory example above by passing the name of the new class, the base classes to inherit from, and a dictionary defining the namespace to use.

For instance, the code:

```
>>> class MyClass(BaseClass):
...     attribute = 42
```

Could also be written as:

```
>>> MyClass = type("MyClass", (BaseClass,), {'attribute' : 42})
```

20.0.4 Metaclasses

It is possible to create a class with a different metaclass than type by setting its __metaclass__ attribute when defining. When this is done, the class, and its subclass will be created using your custom metaclass. For example

```
class CustomMetaclass(type):
    def __init__(cls, name, bases, dct):
        print "Creating class %s using CustomMetaclass" % name
        super(CustomMetaclass, cls).__init__(name, bases, dct)

class BaseClass(object):
    __metaclass__ = CustomMetaclass

class Subclass1(BaseClass):
    pass
```

This will print

```
Creating class BaseClass using CustomMetaclass
Creating class Subclass1 using CustomMetaclass
```

By creating a custom metaclass in this way, it is possible to change how the class is constructed. This allows you to add or remove attributes and methods, register creation of classes and subclasses creation and various other manipulations when the class is created.

20.0.5 More resources

- Wikipedia article on Aspect Oriented Programming[6]
- Unifying types and classes in Python 2.2[7]
- O'Reilly Article on Python Metaclasses[8]

20.0.6 References

6 https://en.wikipedia.org/wiki/Aspect-oriented_programming
7 http://www.python.org/2.2/descrintro.html
8 http://www.onlamp.com/pub/a/python/2003/04/17/metaclasses.html

21 Reflection

A Python script can find out about the type, class, attributes and methods of an object. This is referred to as **reflection** or **introspection** . See also ../Metaclasses/[1].

Reflection-enabling functions include type(), isinstance(), callable(), dir() and getattr().

21.1 Type

The type method enables to find out about the type of an object. The following tests return True:

- type(3) is int
- type('Hello') is str
- type([1, 2]) is list
- type([1, [2, 'Hello']]) is list
- type({'city': 'Paris'}) is dict

21.2 Isinstance

Determines whether an object is an instance of a class.

The following returns True:

- isinstance(3, int)
- isinstance([1, 2], list)

Note that isinstance provides a weaker condition than a comparison using #Type[2].

21.3 Duck typing

Duck typing provides an indirect means of reflection. It is a technique consisting in using an object as if it was of the requested type, while catching exceptions resulting from the object not supporting some of the features of the class or type.

1 Chapter 20 on page 125
2 Chapter 21.1 on page 129

21.4 Callable

For an object, determines whether it can be called. A class can be made callable by providing a __call__() method.

Examples:

- callable(2)
  - Returns False. Ditto for callable("Hello") and callable([1, 2]).
- callable([1,2].pop)
  - Returns True, as pop without "()" returns a function object.
- callable([1,2].pop())
  - Returns False, as [1,2].pop() returns 2 rather than a function object.

21.5 Dir

Returns the list of attributes of an object, which includes methods.

Examples:

- dir(3)
- dir("Hello")
- dir([1, 2])

21.6 Getattr

Returns the value of an attribute of an object, given the attribute name passed as a string.

An example:

- getattr(3, "imag")

The list of attributes of an object can be obtained using #Dir[3].

21.7 External links

- 2. Built-in Functions[4], docs.python.org
- How to determine the variable type in Python?[5], stackoverflow.com
- Differences between isinstance() and type() in python[6], stackoverflow.com
- W:Reflection (computer_programming)#Python[7], Wikipedia
- W:Type introspection#Python[8], Wikipedia

3 Chapter 21.5 on page 130
4 `http://docs.python.org/2/library/functions.html`
5 `http://stackoverflow.com/questions/402504/how-to-determine-the-variable-type-in-python`
6 `http://stackoverflow.com/questions/1549801/differences-between-isinstance-and-type-in-python`
7 `https://en.wikipedia.org/wiki/Reflection%20%28computer_programming%29%23Python`
8 `https://en.wikipedia.org/wiki/Type%20introspection%23Python`

22 Regular Expression

Python includes a module for working with regular expressions on strings. For more information about writing regular expressions and syntax not specific to Python, see the regular expressions[1] wikibook. Python's regular expression syntax is similar to Perl's[2]

To start using regular expressions in your Python scripts, import the "re" module:

```
import re
```

22.1 Overview

Regular expression functions in Python at a glance:

```
import re
if re.search("l+","Hello"):        print 1  # Substring match suffices
if not re.match("ell.","Hello"):   print 2  # The beginning of the string has to
 match
if re.match(".el","Hello"):        print 3
if re.match("he..o","Hello",re.I): print 4  # Case-insensitive match
print re.sub("l+", "l", "Hello")            # Prints "Helo"; replacement AKA
 substitution
print re.sub(r"(.*)\1", r"\1", "HeyHey")    # Prints "Hey"; backreference
for match in re.findall("l+.", "Hello Dolly"):
  print match                               # Prints "llo" and then "lly"
for match in re.findall("e(l+.)", "Hello Dolly"):
  print match                               # Prints "llo"; match picks group 1
matchObj = re.match("(Hello|Hi) (Tom|Thom)","Hello Tom Bombadil")
if matchObj is not None:
  print matchObj.group(0)                   # Prints the whole match
 disregarding groups
  print matchObj.group(1) + matchObj.group(2) # Prints "HelloTom"
```

22.2 Matching and searching

One of the most common uses for regular expressions is extracting a part of a string or testing for the existence of a pattern in a string. Python offers several functions to do this.

The match and search functions do mostly the same thing, except that the match function will only return a result if the pattern matches at the beginning of the string being searched, while search will find a match anywhere in the string.

1 https://en.wikibooks.org/wiki/regular%20expressions
2 https://en.wikibooks.org/wiki/Perl%20Programming%2FRegular%20Expressions%20Reference

```
>>> import re
>>> foo = re.compile(r'foo(.{,5})bar', re.I+re.S)
>>> st1 = 'Foo, Bar, Baz'
>>> st2 = '2. foo is bar'
>>> search1 = foo.search(st1)
>>> search2 = foo.search(st2)
>>> match1 = foo.match(st1)
>>> match2 = foo.match(st2)
```

In this example, match2 will be `None` , because the string `st2` does not start with the given pattern. The other 3 results will be Match objects (see below).

You can also match and search without compiling a regexp:

```
>>> search3 = re.search('oo.*ba', st1, re.I)
```

Here we use the search function of the re module, rather than of the pattern object. For most cases, its best to compile the expression first. Not all of the re module functions support the flags argument and if the expression is used more than once, compiling first is more efficient and leads to cleaner looking code.

The compiled pattern object functions also have parameters for starting and ending the search, to search in a substring of the given string. In the first example in this section, `match2` returns no result because the pattern does not start at the beginning of the string, but if we do:

```
>>> match3 = foo.match(st2, 3)
```

it works, because we tell it to start searching at character number 3 in the string.

What if we want to search for multiple instances of the pattern? Then we have two options. We can use the start and end position parameters of the search and match function in a loop, getting the position to start at from the previous match object (see below) or we can use the findall and finditer functions. The findall function returns a list of matching strings, useful for simple searching. For anything slightly complex, the finditer function should be used. This returns an iterator object, that when used in a loop, yields Match objects. For example:

```
>>> str3 = 'foo, Bar Foo. BAR FoO: bar'
>>> foo.findall(str3)
[', ', '. ', ': ']
>>> for match in foo.finditer(str3):
...     match.group(1)
...
', '
'. '
': '
```

If you're going to be iterating over the results of the search, using the finditer function is almost always a better choice.

22.2.1 Match objects

Match objects are returned by the search and match functions, and include information about the pattern match.

The group function returns a string corresponding to a capture group (part of a regexp wrapped in `()`) of the expression, or if no group number is given, the entire match. Using the `search1` variable we defined above:

```
>>> search1.group()
'Foo, Bar'
>>> search1.group(1)
', '
```

Capture groups can also be given string names using a special syntax and referred to by `matchobj.group('name')` . For simple expressions this is unnecessary, but for more complex expressions it can be very useful.

You can also get the position of a match or a group in a string, using the start and end functions:

```
>>> search1.start()
0
>>> search1.end()
8
>>> search1.start(1)
3
>>> search1.end(1)
5
```

This returns the start and end locations of the entire match, and the start and end of the first (and in this case only) capture group, respectively.

22.3 Replacing

Another use for regular expressions is replacing text in a string. To do this in Python, use the sub function.

sub takes up to 3 arguments: The text to replace with, the text to replace in, and, optionally, the maximum number of substitutions to make. Unlike the matching and searching functions, sub returns a string, consisting of the given text with the substitution(s) made.

```
>>> import re
>>> mystring = 'This string has a q in it'
>>> pattern = re.compile(r'(a[n]? )(\w) ')
>>> newstring = pattern.sub(r"\1'\2' ", mystring)
>>> newstring
"This string has a 'q' in it"
```

This takes any single alphanumeric character (\w in regular expression syntax) preceded by "a" or "an" and wraps in in single quotes. The `\1` and `\2` in the replacement string are backreferences to the 2 capture groups in the expression; these would be group(1) and group(2) on a Match object from a search.

The subn function is similar to sub, except it returns a tuple, consisting of the result string and the number of replacements made. Using the string and expression from before:

```
>>> subresult = pattern.subn(r"\1'\2' ", mystring)
>>> subresult
("This string has a 'q' in it", 1)
```

Replacing without constructing and compiling a pattern object:

```
>>> result = re.sub(r"b.*d","z","abccde")
>>> result
'aze'
```

22.4 Splitting

The split function splits a string based on a given regular expression:

```
>>> import re
>>> mystring = '1. First part 2. Second part 3. Third part'
>>> re.split(r'\d\.', mystring)
['', ' First part ', ' Second part ', ' Third part']
```

22.5 Escaping

The escape function escapes all non-alphanumeric characters in a string. This is useful if you need to take an unknown string that may contain regexp metacharacters like `(` and `.` and create a regular expression from it.

```
>>> re.escape(r'This text (and this) must be escaped with a "\" to use in a
 regexp.')
'This\\ text\\ \\(and\\ this\\)\\ must\\ be\\ escaped\\ with\\ a\\ \\"\\\\\\"\\
 to\\ use\\ in\\ a\\ regexp\\.'
```

22.6 Flags

The different flags use with regular expressions:

Abbreviation	Full name	Description
`re.I`	`re.IGNORECASE`	Makes the regexp case-insensitive[3]
`re.L`	`re.LOCALE`	Makes the behavior of some special sequences (`\w, \W, \b, \B, \s, \S`) dependent on the current locale[4]
`re.M`	`re.MULTILINE`	Makes the `^` and `$` characters match at the beginning and end of each line, rather than just the beginning and end of the string
`re.S`	`re.DOTALL`	Makes the `.` character match every character *including* newlines.
`re.U`	`re.UNICODE`	Makes `\w, \W, \b, \B, \d, \D, \s, \S` dependent on Unicode character properties

3 https://en.wikipedia.org/wiki/case%20sensitivity
4 https://en.wikipedia.org/wiki/locale

Abbreviation	Full name	Description
`re.X`	`re.VERBOSE`	Ignores whitespace except when in a character class or preceded by an non-escaped backslash, and ignores `#` (except when in a character class or preceded by an non-escaped backslash) and everything after it to the end of a line, so it can be used as a comment. This allows for cleaner-looking regexps.

22.7 Pattern objects

If you're going to be using the same regexp more than once in a program, or if you just want to keep the regexps separated somehow, you should create a pattern object, and refer to it later when searching/replacing.

To create a pattern object, use the compile function.

```
import re
foo = re.compile(r'foo(.{,5})bar', re.I+re.S)
```

The first argument is the pattern, which matches the string ”foo”, followed by up to 5 of any character, then the string ”bar”, storing the middle characters to a group, which will be discussed later. The second, optional, argument is the flag or flags to modify the regexp's behavior. The flags themselves are simply variables referring to an integer used by the regular expression engine. In other languages, these would be constants, but Python does not have constants. Some of the regular expression functions do not support adding flags as a parameter when defining the pattern directly in the function, if you need any of the flags, it is best to use the compile function to create a pattern object.

The `r` preceding the expression string indicates that it should be treated as a raw string. This should normally be used when writing regexps, so that backslashes are interpreted literally rather than having to be escaped.

22.8 External links

- Python `re` documentation[5] - Full documentation for the re module, including pattern objects and match objects

fr:Programmation Python/Regex[6]

5 `http://docs.python.org/library/re.html`
6 `https://fr.wikibooks.org/wiki/Programmation%20Python%2FRegex`

23 GUI Programming

There are various GUI toolkits to start with.

23.1 Tkinter

Tkinter, a Python wrapper for Tcl/Tk[1], comes bundled with Python (at least on Win32 platform though it can be installed on Unix/Linux and Mac machines) and provides a cross-platform GUI. It is a relatively simple to learn yet powerful toolkit that provides what appears to be a modest set of widgets. However, because the Tkinter widgets are extensible, many compound widgets can be created rather easily (e.g. combo-box, scrolled panes). Because of its maturity and extensive documentation Tkinter has been designated as the de facto GUI for Python.

To create a very simple Tkinter window frame one only needs the following lines of code:

```
import Tkinter

root = Tkinter.Tk()
root.mainloop()
```

From an object-oriented perspective one can do the following:

```
import Tkinter

class App:
    def __init__(self, master):
        button = Tkinter.Button(master, text="I'm a Button.")
        button.pack()

if __name__ == '__main__':
    root = Tkinter.Tk()
    app = App(root)
    root.mainloop()
```

To learn more about Tkinter visit the following links:

- `http://www.astro.washington.edu/users/rowen/TkinterSummary.html` <- A summary
- `http://infohost.nmt.edu/tcc/help/lang/python/tkinter.html` <- A tutorial
- `http://www.pythonware.com/library/tkinter/introduction/` <- A reference

1 https://en.wikibooks.org/wiki/Programming%3ATcl%20

23.2 PyGTK

See also book PyGTK For GUI Programming[2]

PyGTK[3] provides a convenient wrapper for the GTK+[4] library for use in Python programs, taking care of many of the boring details such as managing memory and type casting. The bare GTK+ toolkit runs on Linux, Windows, and Mac OS X (port in progress), but the more extensive features — when combined with PyORBit and gnome-python — require a GNOME[5] install, and can be used to write full featured GNOME applications.

Home Page[6]

23.3 PyQt

PyQt is a wrapper around the cross-platform Qt C++ toolkit[7]. It has many widgets and support classes[8] supporting SQL, OpenGL, SVG, XML, and advanced graphics capabilities. A PyQt hello world example:

```
from PyQt4.QtCore import *
from PyQt4.QtGui import *

class App(QApplication):
    def __init__(self, argv):
        super(App, self).__init__(argv)
        self.msg = QLabel("Hello, World!")
        self.msg.show()

if __name__ == "__main__":
    import sys
    app = App(sys.argv)
    sys.exit(app.exec_())
```

PyQt[9] is a set of bindings for the cross-platform Qt[10] application framework. PyQt v4 supports Qt4 and PyQt v3 supports Qt3 and earlier.

2 https://en.wikibooks.org/wiki/PyGTK%20For%20GUI%20Programming
3 http://www.pygtk.org/
4 http://www.gtk.org
5 http://www.gnome.org
6 http://www.pygtk.org/
7 http://web.archive.org/web/20060514211039/http://www.trolltech.com/products/qt
8 http://www.riverbankcomputing.com/static/Docs/PyQt4/html/classes.html
9 http://www.riverbankcomputing.co.uk/pyqt/
10 https://en.wikibooks.org/wiki/Qt

23.4 wxPython

Bindings for the cross platform toolkit wxWidgets[11]. WxWidgets is available on Windows, Macintosh, and Unix/Linux.

```
import wx

class test(wx.App):
    def __init__(self):
        wx.App.__init__(self, redirect=False)

    def OnInit(self):
        frame = wx.Frame(None, -1,
                         "Test",
                         pos=(50,50), size=(100,40),
                         style=wx.DEFAULT_FRAME_STYLE)
        button = wx.Button(frame, -1, "Hello World!", (20, 20))
        self.frame = frame
        self.frame.Show()
        return True

if __name__ == '__main__':
        app = test()
        app.MainLoop()
```

- wxPython[12]

23.5 Dabo

Dabo is a full 3-tier application framework. Its UI layer wraps wxPython, and greatly simplifies the syntax.

```
import dabo
dabo.ui.loadUI("wx")

class TestForm(dabo.ui.dForm):
        def afterInit(self):
                self.Caption = "Test"
                self.Position = (50, 50)
                self.Size = (100, 40)
                self.btn = dabo.ui.dButton(self, Caption="Hello World",
                     OnHit=self.onButtonClick)
                self.Sizer.append(self.btn, halign="center", border=20)

        def onButtonClick(self, evt):
                dabo.ui.info("Hello World!")

if __name__ == '__main__':
        app = dabo.ui.dApp()
        app.MainFormClass = TestForm
        app.start()
```

- Dabo[13]

11 http://www.wxwidgets.org/
12 http://wxpython.org/
13 http://dabodev.com/

23.6 pyFltk

pyFltk[14] is a Python wrapper for the FLTK[15], a lightweight cross-platform GUI toolkit. It is very simple to learn and allows for compact user interfaces.

The ”Hello World” example in pyFltk looks like:

```
from fltk import *

window = Fl_Window(100, 100, 200, 90)
button = Fl_Button(9,20,180,50)
button.label("Hello World")
window.end()
window.show()
Fl.run()
```

23.7 Other Toolkits

- PyKDE[16] - Part of the kdebindings package, it provides a python wrapper for the KDE libraries.
- PyXPCOM[17] provides a wrapper around the Mozilla XPCOM[18] component architecture, thereby enabling the use of standalone XUL[19] applications in Python. The XUL toolkit has traditionally been wrapped up in various other parts of XPCOM, but with the advent of libxul and XULRunner[20] this should become more feasible.

fr:Programmation Python/L’interface graphique[21] pt:Python/Programação com GUI[22]

14 http://pyfltk.sourceforge.net/
15 http://www.fltk.org/
16 http://www.riverbankcomputing.co.uk/pykde/index.php
17 http://developer.mozilla.org/en/docs/PyXPCOM
18 http://developer.mozilla.org/en/docs/XPCOM
19 http://developer.mozilla.org/en/docs/XUL
20 http://developer.mozilla.org/en/docs/XULRunner
21 https://fr.wikibooks.org/wiki/Programmation%20Python%2FL%27interface%20graphique
22 https://pt.wikibooks.org/wiki/Python%2FPrograma%C3%A7%C3%A3o%20com%20GUI

24 Authors

24.1 Authors of Python textbook

- Quartz25[1]
- Jesdisciple[2]
- Hannes Röst[3]
- David Ross[4]
- Lawrence D'Oliveiro[5]

1 https://en.wikibooks.org/wiki/User%3AQuartz25
2 https://en.wikibooks.org/wiki/User%3AJesdisciple
3 https://en.wikibooks.org/wiki/User%3AHannes%20R%C3%B6st
4 https://en.wikibooks.org/wiki/User%3AHackbinary
5 https://en.wikibooks.org/wiki/User%3ALdo

25 Game Programming in Python

25.1 3D Game Programming

25.1.1 3D Game Engine with a Python binding

- Irrlicht Enginehttp://irrlicht.sourceforge.net/ (Python binding website: http://pypi.python.org/pypi/pyirrlicht)
- Ogre Engine http://www.ogre3d.org/ (Python binding website: http://www.python-ogre.org/)

Both are very good free open source C++ 3D game Engine with a Python binding.

- CrystalSpace[1] is a free cross-platform software development kit for real-time 3D graphics, with particular focus on games. Crystal Space is accessible from Python in two ways: (1) as a Crystal Space plugin module in which C++ code can call upon Python code, and in which Python code can call upon Crystal Space; (2) as a pure Python module named 'cspace' which one can 'import' from within Python programs. To use the first option, load the 'cspython' plugin as you would load any other Crystal Space plugin, and interact with it via the SCF 'iScript' interface .The second approach allows you to write Crystal Space applications entirely in Python, without any C++ coding. CS Wiki[2]

25.1.2 3D Game Engines written for Python

Engines designed for Python from scratch.

- Blender[3] is an impressive 3D tool with a fully integrated 3D graphics creation suite allowing modeling, animation, rendering, post-production, real-time interactive 3D and game creation and playback with cross-platform compatibility. The 3D game engine uses an embedded python interpreter to make 3D games.
- PySoy[4] is a 3d cloud game engine for Python 3. It was designed for rapid development with an intuitive API that gets new game developers started quickly. The cloud gaming[5] design allows PySoy games to be played on a server without downloading them, greatly reducing the complexity of game distribution. XMPP[6] accounts (such as Jabber

1 http://www.crystalspace3d.org
2 http://en.wikipedia.org/wiki/Crystal_Space
3 http://www.blender.org/
4 http://www.pysoy.org/
5 https://en.wikipedia.org/wiki/Cloud_gaming
6 https://en.wikipedia.org/wiki/XMPP

or GMail) can be used for online gaming identities, chat, and initiating connections to game servers. PySoy is released under the GNU AGPL license[7].

- Soya[8] is a 3D game engine with an easy to understand design. Its written in the Pyrex[9] programming language and uses Cal3d for animation and ODE[10] for physics. Soya is available under the GNU GPL license[11].
- Panda3D[12] is a 3D game engine. It's a library written in C++ with Python bindings. Panda3D is designed in order to support a short learning curve and rapid development. This software is available for free download with source code under the BSD License. The development was started by [Disney]. Now there are many projects made with Panda3D, such as Disney's Pirate's of the Caribbean Online[13], ToonTown[14], Building Virtual World[15], Schell Games[16] and many others. Panda3D supports several features: Procedural Geometry, Animated Texture, Render to texture, Track motion, fog, particle system, and many others.
- CrystalSpace[17] Is a 3D game engine, with a Python bindings, named *PyCrystal[18], view Wikipedia page of *CrystalSpace[19].

25.2 2D Game Programming

- Pygame[20] is a cross platform Python library which wraps SDL[21]. It provides many features like Sprite groups and sound/image loading and easy changing of an objects position. It also provides the programmer access to key and mouse events. A full tutorial can be found in the free book "Making Games with Python & Pygame"[22].
- Phil's Pygame Utilities (PGU)[23] is a collection of tools and libraries that enhance Pygame. Tools include a tile editor and a level editor[24] (tile, isometric, hexagonal). GUI enhancements include full featured GUI, HTML rendering, document layout, and text rendering. The libraries include a sprite and tile engine[25] (tile, isometric, hexagonal), a state engine, a timer, and a high score system. (Beta with last update March, 2007. APIs to be deprecated and isometric and hexagonal support is currently Alpha and subject to change.) [Update 27/02/08 Author indicates he is not currently actively developing this library

7 https://en.wikipedia.org/wiki/GNU_AGPL
8 http://www.soya3d.org/
9 https://en.wikipedia.org/wiki/Pyrex%20programming%20language
10 https://en.wikipedia.org/wiki/Open%20Dynamics%20Engine
11 https://en.wikipedia.org/wiki/GNU_GPL
12 http://www.panda3d.org/
13 http://disney.go.com/pirates/online/
14 http://www.toontown.com/
15 http://www.etc.cmu.edu/bvw
16 http://www.schellgames.com
17 http://www.crystalspace3d.org/
18 http://www.crystalspace3d.org/main/PyCrystal
19 http://en.wikipedia.org/wiki/Crystalspace
20 https://en.wikipedia.org/wiki/Pygame
21 https://en.wikipedia.org/wiki/Simple_DirectMedia_Layer
22 http://inventwithpython.com/pygame
23 http://www.imitationpickles.org/pgu/wiki/index
24 https://en.wikipedia.org/wiki/Level_editor
25 https://en.wikipedia.org/wiki/Tile_engine

and anyone that is willing to develop their own scrolling isometric library offering can use the existing code in PGU to get them started.]

- Pyglet[26] is a cross-platform windowing and multimedia library for Python with no external dependencies or installation requirements. Pyglet provides an object-oriented programming interface for developing games and other visually-rich applications for Windows[27], Mac OS X[28] and Linux[29]. Pyglet allows programs to open multiple windows on multiple screens, draw in those windows with OpenGL, and play back audio and video in most formats. Unlike similar libraries available, pyglet has no external dependencies (such as SDL) and is written entirely in Python. Pyglet is available under a BSD-Style license[30].
- Kivy[31] Kivy is a library for developing multi-touch applications. It is completely cross-platform (Linux/OSX/Win & Android with OpenGL ES2). It comes with native support for many multi-touch input devices, a growing library of multi-touch aware widgets and hardware accelerated OpenGL drawing. Kivy is designed to let you focus on building custom and highly interactive applications as quickly and easily as possible.
- Rabbyt[32] A fast Sprite[33] library for Python with game development in mind. With Rabbyt Anims, even old graphics cards can produce very fast animations of 2,400 or more sprites handling position, rotation, scaling, and color simultaneously.

25.3 See Also

- 10 Lessons Learned [34]- How To Build a Game In A Week From Scratch With No Budget

26 `http://www.pyglet.org/`
27 `https://en.wikipedia.org/wiki/Windows`
28 `https://en.wikipedia.org/wiki/Mac_OS_X`
29 `https://en.wikipedia.org/wiki/Linux`
30 `https://en.wikipedia.org/wiki/BSD_licenses`
31 `http://kivy.org/`
32 `http://arcticpaint.com/projects/rabbyt/`
33 `https://en.wikipedia.org/wiki/Sprite_%28computer_graphics%29`
34 `http://www.gamedev.net/reference/articles/article2259.asp`

26 Sockets

26.1 HTTP Client

Make a very simple HTTP client

```
import socket
s = socket.socket()
s.connect(('localhost', 80))
s.send('GET / HTTP/1.1\nHost:localhost\n\n')
s.recv(40000) # receive 40000 bytes
```

26.2 NTP/Sockets

Connecting to and reading an NTP time server, returning the time as follows

```
ntpps       picoseconds portion of time
ntps        seconds portion of time
ntpms       milliseconds portion of time
ntpt        64-bit ntp time, seconds in upper 32-bits, picoseconds in lower
32-bits
```

27 Files

27.1 File I/O

Read entire file:

```
inputFileText = open("testit.txt", "r").read()
print(inputFileText)
```

In this case the "r" parameter means the file will be opened in read-only mode.

Read certain amount of bytes from a file:

```
inputFileText = open("testit.txt", "r").read(123)
print(inputFileText)
```

When opening a file, one starts reading at the beginning of the file, if one would want more random access to the file, it is possible to use `seek()` to change the current position in a file and `tell()` to get to know the current position in the file. This is illustrated in the following example:

```
>>> f=open("/proc/cpuinfo","r")
>>> f.tell()
0L
>>> f.read(10)
'processor\t'
>>> f.read(10)
': 0\nvendor'
>>> f.tell()
20L
>>> f.seek(10)
>>> f.tell()
10L
>>> f.read(10)
': 0\nvendor'
>>> f.close()
>>> f
<closed file '/proc/cpuinfo', mode 'r' at 0xb7d79770>
```

Here a file is opened, twice ten bytes are read, `tell()` shows that the current offset is at position 20, now `seek()` is used to go back to position 10 (the same position where the second read was started) and ten bytes are read and printed again. And when no more operations on a file are needed the `close()` function is used to close the file we opened.

Read one line at a time:

```
for line in open("testit.txt", "r"):
    print line
```

In this case `readlines()` will return an array containing the individual lines of the file as array entries. Reading a single line can be done using the `readline()` function which returns the current line as a string. This example will output an additional newline between the individual lines of the file, this is because one is read from the file and print introduces another newline.

Write to a file requires the second parameter of `open()` to be "w", this will overwrite the existing contents of the file if it already exists when opening the file:

```
outputFileText = "Here's some text to save in a file"
open("testit.txt", "w").write(outputFileText)
```

Append to a file requires the second parameter of `open()` to be "a" (from append):

```
outputFileText = "Here's some text to add to the existing file."
open("testit.txt", "a").write(outputFileText)
```

Note that this does not add a line break between the existing file content and the string to be added.

27.2 Testing Files

Determine whether path exists:

```
import os
os.path.exists('<path string>')
```

When working on systems such as Microsoft Windows™, the directory separators will conflict with the path string. To get around this, do the following:

```
import os
os.path.exists('C:\\windows\\example\\path')
```

A better way however is to use "raw", or `r` :

```
import os
os.path.exists(r'C:\windows\example\path')
```

But there are some other convenient functions in `os.path` , where `path.code.exists()` only confirms whether or not path exists, there are functions which let you know if the path is a file, a directory, a mount point or a symlink. There is even a function `os.path.realpath()` which reveals the true destination of a symlink:

```
>>> import os
>>> os.path.isfile("/")
False
>>> os.path.isfile("/proc/cpuinfo")
True
>>> os.path.isdir("/")
True
>>> os.path.isdir("/proc/cpuinfo")
False
>>> os.path.ismount("/")
True
>>> os.path.islink("/")
```

```
False
>>> os.path.islink("/vmlinuz")
True
>>> os.path.realpath("/vmlinuz")
'/boot/vmlinuz-2.6.24-21-generic'
```

27.3 Common File Operations

To copy or move a file, use the shutil library.

```
import shutil
shutil.move("originallocation.txt","newlocation.txt")
shutil.copy("original.txt","copy.txt")
```

To perform a recursive copy it is possible to use `copytree()` , to perform a recursive remove it is possible to use `rmtree()`

```
import shutil
shutil.copytree("dir1","dir2")
shutil.rmtree("dir1")
```

To remove an individual file there exists the `remove()` function in the os module:

```
import os
os.remove("file.txt")
```

27.4 Finding Files

Files can be found using *glob* :

```
glob.glob('*.txt') # Finds files in the currect directory ending in dot txt
glob.glob('*\\*.txt') # Finds files in any of the direct subdirectories
                      # of the currect directory ending in dot txt
glob.glob('C:\\Windows\\*.exe')
for fileName in glob.glob('C:\\Windows\\*.exe'):
  print fileName
```

The content of a directory can be listed using *listdir* :

```
filesAndDirectories=os.listdir('.')
for item in filesAndDirectories:
  if os.path.isfile(item) and item.endswith('.txt'):
    print "Text file: " + item
  if os.path.isdir(item):
    print "Directory: " + item
```

Getting a list of all items in a directory, including the nested ones:

```
for root, directories, files in os.walk('/user/Joe Hoe'):
  print "Root: " + root
  for directory in directories:
    print "Directory: " + directory
  for file in files:
    print "File: " + file
```

27.5 Current Directory

Getting current working directory:

```
os.getcwd()
```

Changing current working directory:

```
os.chdir('C:\\')
```

27.6 External Links

- os — Miscellaneous operating system interfaces[1] in Python documentation
- glob — Unix style pathname pattern expansion[2] in Python documentation
- shutil — High-level file operations[3] in Python documentation
- Brief Tour of the Standard Library[4] in The Python Tutorial

1 http://docs.python.org/2/library/os.html
2 http://docs.python.org/2/library/glob.html
3 http://docs.python.org/2/library/shutil.html
4 http://docs.python.org/2/tutorial/stdlib.html

28 Database Programming

28.1 Generic Database Connectivity using ODBC

The Open Database Connectivity[1] (ODBC) API standard allows transparent connections with any database that supports the interface. This includes most popular databases, such as PostgreSQL[2] or Microsoft Access[3]. The strengths of using this interface is that a Python script or module can be used on different databases by only modifying the connection string.

There are four ODBC modules for Python:

1. **PythonWin ODBC Module** : provided by Mark Hammond with the PythonWin[4] package for Microsoft Windows (only). This is a minimal implementation of ODBC, and conforms to Version 1.0 of the Python Database API. Although it is stable, it will likely not be developed any further.[5]
2. **mxODBC** : a commercial Python package (`http://www.egenix.com/products/python/mxODBC/`), which features handling of DateTime objects and prepared statements (using parameters).
3. **pyodbc** : an open-source Python package (`http://code.google.com/p/pyodbc`), which uses only native Python data-types and uses prepared statements for increased performance. The present version supports the Python Database API Specification v2.0.[6]
4. **pypyodbc** : a "pure Python" DBAPI adapter based on the ctypes module, (`https://pypi.python.org/pypi/pypyodbc/1.3.0`) , (`http://code.google.com/p/pypyodbc/`), with a focus on keeping code "Simple - the whole module is implemented in a single script with less than 3000 lines".

28.1.1 pyodbc

An example using the `pyodbc` Python package with a Microsoft Access file (although this database connection could just as easily be a MySQL database):

```
import pyodbc

DBfile = '/data/MSAccess/Music_Library.mdb'
conn = pyodbc.connect('DRIVER={Microsoft Access Driver (*.mdb)};DBQ='+DBfile)
```

1 `https://en.wikipedia.org/wiki/Open%20Database%20Connectivity`
2 `https://en.wikipedia.org/wiki/PostgreSQL`
3 `https://en.wikipedia.org/wiki/Microsoft%20Access`
4 `http://starship.python.net/crew/mhammond/win32/`
5 Hammond, M. Python Programming on Win32 . O'Reilly , , 2000
6 Python Database API Specification v2.0 [7]. Python . Retrieved

```
#use below conn if using with Access 2007, 2010 .accdb file
#conn = pyodbc.connect(r'Driver={Microsoft Access Driver (*.mdb,
 *.accdb)};DBQ='+DBfile)
cursor = conn.cursor()

SQL = 'SELECT Artist, AlbumName FROM RecordCollection ORDER BY Year;'
for row in cursor.execute(SQL): # cursors are iterable
    print row.Artist, row.AlbumName
    # print row # if print row it will return tuple of all fields

cursor.close()
conn.close()
```

Many more features and examples are provided on the pyodbc website.

code create problem shown below. ImportError: DLL load failed: The specified procedure could not be found.

28.2 Postgres connection in Python

-> see Python Programming/Databases[8] code create problem shown below ImportError: DLL load failed: The specified procedure could not be found.

28.3 MySQL connection in Python

-> see Python Programming/Databases[9]

28.4 SQLAlchemy in Action

SQLAlchemy has become the favorite choice for many large Python projects that use databases. A long, updated list of such projects is listed on the SQLAlchemy site. Additionally, a pretty good tutorial can be found there, as well. Along with a thin database wrapper, Elixir, it behaves very similarly to the ORM in Rails, ActiveRecord.

28.5 See also

- Python Programming/Databases[10]

8 https://en.wikibooks.org/wiki/Python%20Programming%2FDatabases
9 https://en.wikibooks.org/wiki/Python%20Programming%2FDatabases
10 https://en.wikibooks.org/wiki/Python%20Programming%2FDatabases

28.6 References

28.7 External links

- SQLAlchemy[11]
- SQLObject[12]
- PEP 249[13] - Python Database API Specification v2.0
- MySQLdb Tutorial[14]
- Database Topic Guide[15] on python.org
- SQLite Tutorial[16]

11 http://www.sqlalchemy.org/
12 http://www.sqlobject.org/
13 http://www.python.org/dev/peps/pep-0249/
14 http://thepythonguru.com/beginner-guide-to-access-mysql-in-python/
15 http://www.python.org/doc/topics/database/
16 http://talkera.org/python/python-database-programming-sqlite-tutorial/

29 Web Page Harvesting

30 Threading

Threading in python is used to run multiple threads (tasks, function calls) at the same time. Note that this does not mean that they are executed on different CPUs. Python threads will NOT make your program faster if it already uses 100 % CPU time. In that case, you probably want to look into parallel programming. If you are interested in parallel programming with python, please see here[1].

Python threads are used in cases where the execution of a task involves some waiting. One example would be interaction with a service hosted on another computer, such as a webserver. Threading allows python to execute other code while waiting; this is easily simulated with the sleep function.

30.1 Examples

30.1.1 A Minimal Example with Function Call

Make a thread that prints numbers from 1-10, waits for 1 sec between:

```
import threading
import time

def loop1_10():
    for i in range(1, 11):
        time.sleep(1)
        print(i)

threading.Thread(target=loop1_10).start()
```

30.1.2 A Minimal Example with Object

```
#!/usr/bin/env python
import threading
import time
from __future__ import print_function

class MyThread(threading.Thread):
    def run(self):
        print("{} started!".format(self.getName()))             # "Thread-x
 started!"
        time.sleep(1)                                     # Pretend to work for
 a second
        print("{} finished!".format(self.getName()))            # "Thread-x
 finished!"
```

1 http://wiki.python.org/moin/ParallelProcessing

```
if __name__ == '__main__':
    for x in range(4):                                     # Four times...
        mythread = MyThread(name = "Thread-{}".format(x + 1))  # ...Instantiate
 a thread and pass a unique ID to it
        mythread.start()                                   # ...Start the thread
        time.sleep(.9)                                     # ...Wait 0.9 seconds
 before starting another
```

This should output:

```
Thread-1 started!
Thread-2 started!
Thread-1 finished!
Thread-3 started!
Thread-2 finished!
Thread-4 started!
Thread-3 finished!
Thread-4 finished!
```

Note: this example appears to crash IDLE in Windows XP (seems to work in IDLE 1.2.4 in Windows XP though)

There seems to be a problem with this, if you replace `sleep(1)` with (2), and change `range(4)` to `range(10)` . Thread-2 finished is the first line before its even started. in WING IDE, Netbeans, Eclipse is fine.

fr:Programmation Python/Les threads[2]

2 https://fr.wikibooks.org/wiki/Programmation%20Python%2FLes%20threads

31 Extending with C

This gives a minimal Example on how to Extend Python with C. Linux is used for building (feel free to extend it for other Platforms). If you have any problems, please report them (e.g. on the dicussion page), I will check back in a while and try to sort them out.

31.1 Using the Python/C API

On an Ubuntu system, you might need to run

```
$ sudo apt-get install python-dev
```

This command installs you the python developement package and ensures that you can use the line `#include <Python.h>` in the C source code. On other systems like openSUSE the needed package calls `python-devel` and can be installed by using `zypper` :

```
$ sudo zypper install python-devel
```

- `https://docs.python.org/2/extending/index.html`
- `https://docs.python.org/2/c-api/index.html`

31.1.1 A minimal example

The minimal example we will create now is very similar in behaviour to the following python snippet:

```
def say_hello(name):
    "Greet somebody."
    print "Hello %s!" % name
```

The C source code (`hellomodule.c`)

```
#include <Python.h>

static PyObject*
say_hello(PyObject* self, PyObject* args)
{
    const char* name;

    if (!PyArg_ParseTuple(args, "s", &name))
        return NULL;

    printf("Hello %s!\n", name);
    Py_RETURN_NONE;
}

static PyMethodDef HelloMethods[] =
{
     {"say_hello", say_hello, METH_VARARGS, "Greet somebody."},
     {NULL, NULL, 0, NULL}
};

PyMODINIT_FUNC
inithello(void)
{
     (void) Py_InitModule("hello", HelloMethods);
}
```

Building the extension module with GCC for Linux

To build our extension module we create the file `setup.py` like:

```
from distutils.core import setup, Extension

module1 = Extension('hello', sources = ['hellomodule.c'])

setup (name = 'PackageName',
        version = '1.0',
        description = 'This is a demo package',
        ext_modules = [module1])
```

Now we can build our module with

```
python setup.py build
```

The module `hello.so` will end up in `build/lib.linux-i686-`*x* `.`*y* .

Building the extension module with GCC for Microsoft Windows

Microsoft Windows users can use MinGW[1] to compile this from `cmd.exe`[2] using a similar method to Linux user, as shown above. Assuming `gcc` is in the `PATH` environment variable, type:

```
python setup.py build -cmingw32
```

The module `hello.pyd` will end up in `build\lib.win32-`*x* `.`*y* , which is a Python Dynamic Module (similar to a `DLL`).

An alternate way of building the module in Windows is to build a DLL. (This method does not need an extension module file). From `cmd.exe` , type:

1 https://en.wikipedia.org/wiki/MinGW
2 https://en.wikipedia.org/wiki/cmd.exe

```
gcc -c  hellomodule.c -I/PythonXY /include
gcc -shared hellomodule.o -L/PythonXY /libs -lpythonXY  -o hello.dll
```

where XY represents the version of Python, such as ”24” for version 2.4.

Building the extension module using Microsoft Visual C++

With VC8 distutils is broken. We will use cl.exe from a command prompt instead:

```
cl /LD hellomodule.c /Ic:\Python24\include c:\Python24\libs\python24.lib
/link/out:hello.dll
```

Using the extension module

Change to the subdirectory where the file ‘hello.so‘ resides. In an interactive python session you can use the module as follows.

```
>>> import hello
>>> hello.say_hello("World")
Hello World!
```

31.1.2 A module for calculating fibonacci numbers

The C source code (fibmodule.c)

```
#include <Python.h>

int
_fib(int n)
{
    if (n < 2)
        return n;
    else
        return _fib(n-1) + _fib(n-2);
}

static PyObject*
fib(PyObject* self, PyObject* args)
{
    int n;

    if (!PyArg_ParseTuple(args, "i", &n))
        return NULL;

    return Py_BuildValue("i", _fib(n));
}

static PyMethodDef FibMethods[] = {
    {"fib", fib, METH_VARARGS, "Calculate the Fibonacci numbers."},
    {NULL, NULL, 0, NULL}
};
```

```
PyMODINIT_FUNC
initfib(void)
{
    (void) Py_InitModule("fib", FibMethods);
}
```

The build script (setup.py)

```
from distutils.core import setup, Extension

module1 = Extension('fib', sources = ['fibmodule.c'])

setup (name = 'PackageName',
        version = '1.0',
        description = 'This is a demo package',
        ext_modules = [module1])
```

How to use it?

```
>>> import fib
>>> fib.fib(10)
55
```

31.2 Using SWIG

Creating the previous example using SWIG is much more straight forward. To follow this path you need to get SWIG[3] up and running first. To install it on an Ubuntu system, you might need to run the following commands

```
$ sudo apt-get install swig
$ sudo apt-get install python-dev
```

After that create two files.

```
/*hellomodule.c*/

#include <stdio.h>

void say_hello(const char* name) {
    printf("Hello %s!\n", name);
}
```

```
/*hello.i*/

%module hello
extern void say_hello(const char* name);
```

Now comes the more difficult part, gluing it all together.

3 http://www.swig.org/

First we need to let SWIG do its work.

```
swig -python hello.i
```

This gives us the files ‘hello.py‘ and ‘hello_wrap.c‘.

The next step is compiling (substitute /usr/include/python2.4/ with the correct path for your setup!).

```
gcc -fpic -c hellomodule.c hello_wrap.c -I/usr/include/python2.4/
```

Now linking and we are done!

```
gcc -shared hellomodule.o hello_wrap.o -o _hello.so
```

The module is used in the following way.

```
>>> import hello
>>> hello.say_hello("World")
Hello World!
```

32 Extending with C++

There are different ways to extend Python:

- In plain C, using Python.h
- Using Swig
- Using Boost.Python, optionally with Py++ preprocessing
- Using Cython.

This page describes Boost.Python[1]. Before the emergence of Cython, it was the most comfortable way of writing C++[2] extension modules.

Boost.Python comes bundled with the Boost C++ Libraries[3]. To install it on an Ubuntu system, you might need to run the following commands

```
$ sudo apt-get install libboost-python-dev
$ sudo apt-get install python-dev
```

32.1 A Hello World Example

32.1.1 The C++ source code (hellomodule.cpp)

```
#include <iostream>

using namespace std;

void say_hello(const char* name) {
    cout << "Hello " <<  name << "!\n";
}

#include <boost/python/module.hpp>
#include <boost/python/def.hpp>
using namespace boost::python;

BOOST_PYTHON_MODULE(hello)
{
    def("say_hello", say_hello);
}
```

1 http://www.boost.org/libs/python/doc/
2 https://en.wikibooks.org/wiki/C%2B%2B
3 http://www.boost.org/

32.1.2 setup.py

```
#!/usr/bin/env python

from distutils.core import setup
from distutils.extension import Extension

setup(name="PackageName",
    ext_modules=[
        Extension("hello", ["hellomodule.cpp"],
        libraries = ["boost_python"])
    ])
```

Now we can build our module with

```
python setup.py build
```

The module 'hello.so' will end up in e.g 'build/lib.linux-i686-2.4'.

32.1.3 Using the extension module

Change to the subdirectory where the file 'hello.so' resides. In an interactive python session you can use the module as follows.

```
>>> import hello
>>> hello.say_hello("World")
Hello World!
```

32.2 An example with CGAL

Some, but not all, functions of the CGAL library have already Python bindings. Here an example is provided for a case without such a binding and how it might be implemented. The example is taken from the CGAL Documentation[4].

```
// test.cpp
using namespace std;

/* PYTHON */
#include <boost/python.hpp>
#include <boost/python/module.hpp>
#include <boost/python/def.hpp>
namespace python = boost::python;

/* CGAL */
#include <CGAL/Cartesian.h>
#include <CGAL/Range_segment_tree_traits.h>
#include <CGAL/Range_tree_k.h>

typedef CGAL::Cartesian<double> K;
typedef CGAL::Range_tree_map_traits_2<K, char> Traits;
```

4 http://www.cgal.org/Manual/3.3/doc_html/cgal_manual/SearchStructures/Chapter_main.html#Subsection_46.5.1

```
typedef CGAL::Range_tree_2<Traits> Range_tree_2_type;

typedef Traits::Key Key;
typedef Traits::Interval Interval;

Range_tree_2_type *Range_tree_2 = new Range_tree_2_type;

void create_tree()   {

  typedef Traits::Key Key;
  typedef Traits::Interval Interval;

  std::vector<Key> InputList, OutputList;
  InputList.push_back(Key(K::Point_2(8,5.1), 'a'));
  InputList.push_back(Key(K::Point_2(1.0,1.1), 'b'));
  InputList.push_back(Key(K::Point_2(3,2.1), 'c'));

  Range_tree_2->make_tree(InputList.begin(),InputList.end());
  Interval win(Interval(K::Point_2(1,2.1),K::Point_2(8.1,8.2)));
  std::cout << "\n Window Query:\n";
  Range_tree_2->window_query(win, std::back_inserter(OutputList));
  std::vector<Key>::iterator current=OutputList.begin();
  while(current!=OutputList.end()){
      std::cout << "  " << (*current).first.x() << "," << (*current).first.y()
           << ":" << (*current).second << std::endl;
      current++;
    }
  std::cout << "\n Done\n";
}

void initcreate_tree() {;}

using namespace boost::python;
BOOST_PYTHON_MODULE(test)
{
    def("create_tree", create_tree, "");
}


// setup.py
#!/usr/bin/env python

from distutils.core import setup
from distutils.extension import Extension

setup(name="PackageName",
    ext_modules=[
        Extension("test", ["test.cpp"],
        libraries = ["boost_python"])
    ])
```

We then compile and run the module as follows:

```
$ python setup.py build
$ cd build/lib*
$ python
>>> import test
>>> test.create_tree()
Window Query:
 3,2.1:c
 8,5.1:a
Done
>>>
```

32.3 Handling Python objects and errors

One can also handle more complex data, e.g. Python objects like lists. The attributes are accessed with the extract function executed on the objects ”attr” function output. We can also throw errors by telling the library that an error has occurred and returning. In the following case, we have written a C++ function called ”afunction” which we want to call. The function takes an integer N and a vector of length N as input, we have to convert the python list to a vector of strings before calling the function.

```
#include <vector>
using namespace std;

void _afunction_wrapper(int N, boost::python::list mapping) {

    int mapping_length = boost::python::extract<int>(mapping.attr("__len__")());
    //Do Error checking, the mapping needs to be at least as long as N
    if (mapping_length < N) {
        PyErr_SetString(PyExc_ValueError,
            "The string mapping must be at least of length N");
        boost::python::throw_error_already_set();
        return;
    }

    vector<string> mystrings(mapping_length);
    for (int i=0; i<mapping_length; i++) {
        mystrings[i] = boost::python::extract<char const *>(mapping[i]);
    }

   //now call our C++ function
   _afunction(N, mystrings);

}

using namespace boost::python;
BOOST_PYTHON_MODULE(c_afunction)
{
    def("afunction", _afunction_wrapper);
}
```

33 Extending with ctypes

ctypes`http://python.net/crew/theller/ctypes/` is a foreign function interface[1] module for Python (included with Python 2.5 and above), which allows you to load in dynamic libraries and call C functions. This is not technically extending Python, but it serves one of the primary reasons for extending Python: to interface with external C code.

33.1 Basics

A library is loaded using the `ctypes.CDLL` function. After you load the library, the functions inside the library are already usable as regular Python calls. For example, if we wanted to forego the standard Python print statement and use the standard C library function, `printf` , you would use this:

```
from ctypes import *
libName = 'libc.so' # If you're on a UNIX-based system
libName = 'msvcrt.dll' # If you're on Windows
libc = CDLL(libName)
libc.printf("Hello, World!\n")
```

Of course, you must use the libName line that matches your operating system, and delete the other. If all goes well, you should see the infamous Hello World string at your console.

33.2 Getting Return Values

ctypes assumes, by default, that any given function's return type is a signed integer of native size. Sometimes you don't want the function to return anything, and other times, you want the function to return other types. Every ctypes function has an attribute called `restype` . When you assign a ctypes class to `restype` , it automatically casts the function's return value to that type.

33.2.1 Common Types

ctypes name	**C type**	**Python type**	**Notes**
None	void	None	the None object
c_bool	C99 _Bool	bool	
c_byte	signed char	int	

1 `https://en.wikipedia.org/wiki/Foreign%20function%20interface`

ctypes name	**C type**	**Python type**	**Notes**
c_char	signed char	str	length of one
c_char_p	char *	str	
c_double	double	float	
c_float	float	float	
c_int	signed int	int	
c_long	signed long	long	
c_longlong	signed long long	long	
c_short	signed short	long	
c_ubyte	unsigned char	int	
c_uint	unsigned int	int	
c_ulong	unsigned long	long	
c_ulonglong	unsigned long long	long	
c_ushort	unsigned short	int	
c_void_p	void *	int	
c_wchar	wchar_t	unicode	length of one
c_wchar_p	wchar_t *	unicode	

34 WSGI web programming

35 WSGI Web Programming

35.1 External Resources

http://docs.python.org/library/wsgiref.html

36 References

36.1 Language reference

The latest documentation for the standard python libraries and modules can always be found at The Python.org documents section[1]

1 `http://www.python.org/doc/`

37 Contributors

Edits	User
1	Aaronchall[1]
1	AdamPro[2]
1	Adeelq[3]
3	Adriatikus[4]
3	Adrignola[5]
1	Ahab1964[6]
1	Ahornedal[7]
4	Albmont[8]
2	Alexander256[9]
1	AlisonW[10]
1	Apeigne~enwikibooks[11]
1	ArrowStomper[12]
50	Artevelde[13]
2	Atcovi[14]
2	Auk[15]
1	Avicennasis[16]
1	Avnerium[17]
1	Az1568[18]
1	Baijum81[19]
1	Beary605~enwikibooks[20]
1	Behnam[21]

1 https://en.wikibooks.org/w/index.php%3ftitle=User:Aaronchall&action=edit&redlink=1
2 https://en.wikibooks.org/w/index.php%3ftitle=User:AdamPro&action=edit&redlink=1
3 https://en.wikibooks.org/w/index.php%3ftitle=User:Adeelq&action=edit&redlink=1
4 https://en.wikibooks.org/wiki/User:Adriatikus
5 https://en.wikibooks.org/wiki/User:Adrignola
6 https://en.wikibooks.org/w/index.php%3ftitle=User:Ahab1964&action=edit&redlink=1
7 https://en.wikibooks.org/w/index.php%3ftitle=User:Ahornedal&action=edit&redlink=1
8 https://en.wikibooks.org/wiki/User:Albmont
9 https://en.wikibooks.org/w/index.php%3ftitle=User:Alexander256&action=edit&redlink=1
10 https://en.wikibooks.org/wiki/User:AlisonW
11 https://en.wikibooks.org/w/index.php%3ftitle=User:Apeigne~enwikibooks&action=edit&redlink=1
12 https://en.wikibooks.org/w/index.php%3ftitle=User:ArrowStomper&action=edit&redlink=1
13 https://en.wikibooks.org/wiki/User:Artevelde
14 https://en.wikibooks.org/wiki/User:Atcovi
15 https://en.wikibooks.org/wiki/User:Auk
16 https://en.wikibooks.org/wiki/User:Avicennasis
17 https://en.wikibooks.org/w/index.php%3ftitle=User:Avnerium&action=edit&redlink=1
18 https://en.wikibooks.org/wiki/User:Az1568
19 https://en.wikibooks.org/wiki/User:Baijum81
20 https://en.wikibooks.org/wiki/User:Beary605~enwikibooks
21 https://en.wikibooks.org/w/index.php%3ftitle=User:Behnam&action=edit&redlink=1

2	Beland[22]
1	Benrolfe[23]
2	Betalpha[24]
3	Bittner[25]
20	BobGibson[26]
1	Boyombo[27]
1	Brian McErlean~enwikibooks[28]
13	CWii[29]
2	CaffeinatedPonderer[30]
1	CaptainSmithers[31]
1	Cburnett[32]
1	Chazz[33]
1	Chesemonkyloma[34]
1	Christian.ego[35]
6	Chuckhoffmann[36]
1	Cic[37]
1	Cladmi[38]
1	Clorox[39]
1	Cogiati[40]
2	Convex~enwikibooks[41]
1	Cosmoscalibur[42]
2	Cribe~enwikibooks[43]
1	Cspurrier[44]
2	DaKrazyJak[45]
1	Daemonax[46]

22 https://en.wikibooks.org/wiki/User:Beland
23 https://en.wikibooks.org/w/index.php%3ftitle=User:Benrolfe&action=edit&redlink=1
24 https://en.wikibooks.org/w/index.php%3ftitle=User:Betalpha&action=edit&redlink=1
25 https://en.wikibooks.org/wiki/User:Bittner
26 https://en.wikibooks.org/w/index.php%3ftitle=User:BobGibson&action=edit&redlink=1
27 https://en.wikibooks.org/w/index.php%3ftitle=User:Boyombo&action=edit&redlink=1
28 https://en.wikibooks.org/w/index.php%3ftitle=User:Brian_McErlean~enwikibooks&action=edit&redlink=1
29 https://en.wikibooks.org/wiki/User:CWii
30 https://en.wikibooks.org/w/index.php%3ftitle=User:CaffeinatedPonderer&action=edit&redlink=1
31 https://en.wikibooks.org/wiki/User:CaptainSmithers
32 https://en.wikibooks.org/wiki/User:Cburnett
33 https://en.wikibooks.org/wiki/User:Chazz
34 https://en.wikibooks.org/w/index.php%3ftitle=User:Chesemonkyloma&action=edit&redlink=1
35 https://en.wikibooks.org/w/index.php%3ftitle=User:Christian.ego&action=edit&redlink=1
36 https://en.wikibooks.org/wiki/User:Chuckhoffmann
37 https://en.wikibooks.org/w/index.php%3ftitle=User:Cic&action=edit&redlink=1
38 https://en.wikibooks.org/w/index.php%3ftitle=User:Cladmi&action=edit&redlink=1
39 https://en.wikibooks.org/w/index.php%3ftitle=User:Clorox&action=edit&redlink=1
40 https://en.wikibooks.org/wiki/User:Cogiati
41 https://en.wikibooks.org/w/index.php%3ftitle=User:Convex~enwikibooks&action=edit&redlink=1
42 https://en.wikibooks.org/w/index.php%3ftitle=User:Cosmoscalibur&action=edit&redlink=1
43 https://en.wikibooks.org/w/index.php%3ftitle=User:Cribe~enwikibooks&action=edit&redlink=1
44 https://en.wikibooks.org/wiki/User:Cspurrier
45 https://en.wikibooks.org/w/index.php%3ftitle=User:DaKrazyJak&action=edit&redlink=1
46 https://en.wikibooks.org/w/index.php%3ftitle=User:Daemonax&action=edit&redlink=1

68	Dan Polansky[47]
1	Danielkhashabi[48]
44	Darklama[49]
2	DavidCary[50]
11	DavidRoss[51]
2	Dbolton[52]
2	Deep shobhit[53]
4	Derbeth[54]
4	Dirk Hünniger[55]
1	DivineAlpha[56]
4	Dragonecc[57]
6	Driscoll~enwikibooks[58]
1	Edleafe[59]
1	EdoDodo[60]
3	ElieDeBrauwer[61]
1	Eric Silva[62]
1	Esquivalience[63]
1	FerranJorba~enwikibooks[64]
8	Fishpi[65]
21	Flarelocke[66]
2	Flowzn[67]
1	Foxj[68]
1	Fry-kun[69]
2	Gasto5[70]
1	Glaisher[71]

47 https://en.wikibooks.org/wiki/User:Dan_Polansky
48 https://en.wikibooks.org/w/index.php%3ftitle=User:Danielkhashabi&action=edit&redlink=1
49 https://en.wikibooks.org/wiki/User:Darklama
50 https://en.wikibooks.org/wiki/User:DavidCary
51 https://en.wikibooks.org/wiki/User:DavidRoss
52 https://en.wikibooks.org/wiki/User:Dbolton
53 https://en.wikibooks.org/w/index.php%3ftitle=User:Deep_shobhit&action=edit&redlink=1
54 https://en.wikibooks.org/wiki/User:Derbeth
55 https://en.wikibooks.org/wiki/User:Dirk_H%25C3%25BCnniger
56 https://en.wikibooks.org/w/index.php%3ftitle=User:DivineAlpha&action=edit&redlink=1
57 https://en.wikibooks.org/wiki/User:Dragonecc
58 https://en.wikibooks.org/w/index.php%3ftitle=User:Driscoll~enwikibooks&action=edit&redlink=1
59 https://en.wikibooks.org/w/index.php%3ftitle=User:Edleafe&action=edit&redlink=1
60 https://en.wikibooks.org/wiki/User:EdoDodo
61 https://en.wikibooks.org/w/index.php%3ftitle=User:ElieDeBrauwer&action=edit&redlink=1
62 https://en.wikibooks.org/w/index.php%3ftitle=User:Eric_Silva&action=edit&redlink=1
63 https://en.wikibooks.org/w/index.php%3ftitle=User:Esquivalience&action=edit&redlink=1
64 https://en.wikibooks.org/w/index.php%3ftitle=User:FerranJorba~enwikibooks&action=edit&redlink=1
65 https://en.wikibooks.org/w/index.php%3ftitle=User:Fishpi&action=edit&redlink=1
66 https://en.wikibooks.org/wiki/User:Flarelocke
67 https://en.wikibooks.org/w/index.php%3ftitle=User:Flowzn&action=edit&redlink=1
68 https://en.wikibooks.org/wiki/User:Foxj
69 https://en.wikibooks.org/w/index.php%3ftitle=User:Fry-kun&action=edit&redlink=1
70 https://en.wikibooks.org/w/index.php%3ftitle=User:Gasto5&action=edit&redlink=1
71 https://en.wikibooks.org/wiki/User:Glaisher

1	Greyweather~enwikibooks[72]
1	Grind24[73]
1	Guanabot~enwikibooks[74]
1	Guanaco[75]
4	Gutworth[76]
45	Hackbinary[77]
4	Hagindaz[78]
27	Hannes Röst[79]
1	Harrybrowne1986[80]
2	Howipepper[81]
17	Hypergeek14[82]
3	IO[83]
2	Imapiekindaguy[84]
5	ImperfectlyInformed[85]
1	Intgr[86]
3	Irvin.sha[87]
12	JackPotte[88]
2	Jakec[89]
17	Jbeyerl[90]
2	Jerf~enwikibooks[91]
1	Jesdisciple[92]
32	Jguk[93]
1	JocelynD[94]
1	JohnL4[95]
1	Jonathan Webley[96]

72 https://en.wikibooks.org/w/index.php%3ftitle=User:Greyweather~enwikibooks&action=edit&redlink=1
73 https://en.wikibooks.org/wiki/User:Grind24
74 https://en.wikibooks.org/wiki/User:Guanabot~enwikibooks
75 https://en.wikibooks.org/wiki/User:Guanaco
76 https://en.wikibooks.org/wiki/User:Gutworth
77 https://en.wikibooks.org/wiki/User:Hackbinary
78 https://en.wikibooks.org/wiki/User:Hagindaz
79 https://en.wikibooks.org/wiki/User:Hannes_R%25C3%25B6st
80 https://en.wikibooks.org/wiki/User:Harrybrowne1986
81 https://en.wikibooks.org/w/index.php%3ftitle=User:Howipepper&action=edit&redlink=1
82 https://en.wikibooks.org/w/index.php%3ftitle=User:Hypergeek14&action=edit&redlink=1
83 https://en.wikibooks.org/w/index.php%3ftitle=User:IO&action=edit&redlink=1
84 https://en.wikibooks.org/w/index.php%3ftitle=User:Imapiekindaguy&action=edit&redlink=1
85 https://en.wikibooks.org/wiki/User:ImperfectlyInformed
86 https://en.wikibooks.org/wiki/User:Intgr
87 https://en.wikibooks.org/w/index.php%3ftitle=User:Irvin.sha&action=edit&redlink=1
88 https://en.wikibooks.org/wiki/User:JackPotte
89 https://en.wikibooks.org/wiki/User:Jakec
90 https://en.wikibooks.org/wiki/User:Jbeyerl
91 https://en.wikibooks.org/w/index.php%3ftitle=User:Jerf~enwikibooks&action=edit&redlink=1
92 https://en.wikibooks.org/wiki/User:Jesdisciple
93 https://en.wikibooks.org/wiki/User:Jguk
94 https://en.wikibooks.org/w/index.php%3ftitle=User:JocelynD&action=edit&redlink=1
95 https://en.wikibooks.org/wiki/User:JohnL4
96 https://en.wikibooks.org/wiki/User:Jonathan_Webley

1	Jonbryan[97]
1	Jperryhouts[98]
1	JuethoBot[99]
2	KarlDubost[100]
1	Kayau[101]
1	Kernigh[102]
1	Ketan Arlulkar[103]
11	LDiracDelta~enwikibooks[104]
7	Ldo[105]
1	Leaderboard[106]
1	Legoktm[107]
1	Lena2289[108]
4	Leopold augustsson[109]
1	Linuxman255[110]
3	Logictheo[111]
1	MMJ~enwikibooks[112]
4	ManWhoFoundPony[113]
1	ManuelGR[114]
5	MarceloAraujo[115]
1	Mathonius[116]
1	Mattzazami[117]
1	Maxim kolosov[118]
1	Mdupont[119]
1	Mh7kJ[120]
4	Microdot[121]

97 `https://en.wikibooks.org/w/index.php%3ftitle=User:Jonbryan&action=edit&redlink=1`
98 `https://en.wikibooks.org/w/index.php%3ftitle=User:Jperryhouts&action=edit&redlink=1`
99 `https://en.wikibooks.org/wiki/User:JuethoBot`
100 `https://en.wikibooks.org/w/index.php%3ftitle=User:KarlDubost&action=edit&redlink=1`
101 `https://en.wikibooks.org/wiki/User:Kayau`
102 `https://en.wikibooks.org/wiki/User:Kernigh`
103 `https://en.wikibooks.org/w/index.php%3ftitle=User:Ketan_Arlulkar&action=edit&redlink=1`
104 `https://en.wikibooks.org/w/index.php%3ftitle=User:LDiracDelta~enwikibooks&action=edit&redlink=1`
105 `https://en.wikibooks.org/wiki/User:Ldo`
106 `https://en.wikibooks.org/w/index.php%3ftitle=User:Leaderboard&action=edit&redlink=1`
107 `https://en.wikibooks.org/wiki/User:Legoktm`
108 `https://en.wikibooks.org/w/index.php%3ftitle=User:Lena2289&action=edit&redlink=1`
109 `https://en.wikibooks.org/w/index.php%3ftitle=User:Leopold_augustsson&action=edit&redlink=1`
110 `https://en.wikibooks.org/w/index.php%3ftitle=User:Linuxman255&action=edit&redlink=1`
111 `https://en.wikibooks.org/wiki/User:Logictheo`
112 `https://en.wikibooks.org/w/index.php%3ftitle=User:MMJ~enwikibooks&action=edit&redlink=1`
113 `https://en.wikibooks.org/w/index.php%3ftitle=User:ManWhoFoundPony&action=edit&redlink=1`
114 `https://en.wikibooks.org/wiki/User:ManuelGR`
115 `https://en.wikibooks.org/w/index.php%3ftitle=User:MarceloAraujo&action=edit&redlink=1`
116 `https://en.wikibooks.org/wiki/User:Mathonius`
117 `https://en.wikibooks.org/w/index.php%3ftitle=User:Mattzazami&action=edit&redlink=1`
118 `https://en.wikibooks.org/w/index.php%3ftitle=User:Maxim_kolosov&action=edit&redlink=1`
119 `https://en.wikibooks.org/wiki/User:Mdupont`
120 `https://en.wikibooks.org/wiki/User:Mh7kJ`
121 `https://en.wikibooks.org/w/index.php%3ftitle=User:Microdot&action=edit&redlink=1`

1	Mithrill2002[122]
1	Monobi[123]
33	Mr.Z-man[124]
2	Mshonle[125]
17	Mwtoews[126]
3	Myururdurmaz[127]
2	N313t3[128]
1	Natuur12[129]
7	Nbarth[130]
3	Nikai[131]
1	Nikhil389[132]
1	NithinBekal[133]
1	Nobelium[134]
1	Offpath[135]
1	Otus[136]
6	Panic2k4[137]
1	Pavlix~enwikibooks[138]
22	Pdilley[139]
1	Perey[140]
1	Peteparke[141]
1	Pingveno[142]
6	Quartz25[143]
15	QuiteUnusual[144]
4	Qwertyus[145]
2	Rdnk~enwikibooks[146]

122 https://en.wikibooks.org/w/index.php%3ftitle=User:Mithrill2002&action=edit&redlink=1
123 https://en.wikibooks.org/wiki/User:Monobi
124 https://en.wikibooks.org/wiki/User:Mr.Z-man
125 https://en.wikibooks.org/wiki/User:Mshonle
126 https://en.wikibooks.org/wiki/User:Mwtoews
127 https://en.wikibooks.org/w/index.php%3ftitle=User:Myururdurmaz&action=edit&redlink=1
128 https://en.wikibooks.org/w/index.php%3ftitle=User:N313t3&action=edit&redlink=1
129 https://en.wikibooks.org/wiki/User:Natuur12
130 https://en.wikibooks.org/wiki/User:Nbarth
131 https://en.wikibooks.org/wiki/User:Nikai
132 https://en.wikibooks.org/w/index.php%3ftitle=User:Nikhil389&action=edit&redlink=1
133 https://en.wikibooks.org/wiki/User:NithinBekal
134 https://en.wikibooks.org/wiki/User:Nobelium
135 https://en.wikibooks.org/w/index.php%3ftitle=User:Offpath&action=edit&redlink=1
136 https://en.wikibooks.org/wiki/User:Otus
137 https://en.wikibooks.org/wiki/User:Panic2k4
138 https://en.wikibooks.org/wiki/User:Pavlix~enwikibooks
139 https://en.wikibooks.org/w/index.php%3ftitle=User:Pdilley&action=edit&redlink=1
140 https://en.wikibooks.org/wiki/User:Perey
141 https://en.wikibooks.org/w/index.php%3ftitle=User:Peteparke&action=edit&redlink=1
142 https://en.wikibooks.org/wiki/User:Pingveno
143 https://en.wikibooks.org/wiki/User:Quartz25
144 https://en.wikibooks.org/wiki/User:QuiteUnusual
145 https://en.wikibooks.org/wiki/User:Qwertyus
146 https://en.wikibooks.org/w/index.php%3ftitle=User:Rdnk~enwikibooks&action=edit&redlink=1

3	Recent Runes[147]
1	Remi0o[148]
31	Remote[149]
3	Richard001[150]
3	Robm351[151]
1	Rotlink[152]
1	Ruy Pugliesi[153]
1	RyanPenner[154]
1	Senobyte[155]
1	Shanmugamp7[156]
15	Sigma 7[157]
4	Singingwolfboy[158]
1	Smalls123456[159]
1	Sol˜enwikibooks[160]
1	StephenFerg[161]
2	Suchenwi[162]
1	Surfer190[163]
3	Syum90[164]
6	Szeeshanalinaqvi[165]
1	Tecky2[166]
1	Tedzzz1[167]
3	The Kid[168]
9	The djinn[169]
18	Thunderbolt16[170]
2	Tobych[171]

147 `https://en.wikibooks.org/wiki/User:Recent_Runes`
148 `https://en.wikibooks.org/wiki/User:Remi0o`
149 `https://en.wikibooks.org/w/index.php%3ftitle=User:Remote&action=edit&redlink=1`
150 `https://en.wikibooks.org/wiki/User:Richard001`
151 `https://en.wikibooks.org/w/index.php%3ftitle=User:Robm351&action=edit&redlink=1`
152 `https://en.wikibooks.org/w/index.php%3ftitle=User:Rotlink&action=edit&redlink=1`
153 `https://en.wikibooks.org/wiki/User:Ruy_Pugliesi`
154 `https://en.wikibooks.org/w/index.php%3ftitle=User:RyanPenner&action=edit&redlink=1`
155 `https://en.wikibooks.org/w/index.php%3ftitle=User:Senobyte&action=edit&redlink=1`
156 `https://en.wikibooks.org/wiki/User:Shanmugamp7`
157 `https://en.wikibooks.org/wiki/User:Sigma_7`
158 `https://en.wikibooks.org/w/index.php%3ftitle=User:Singingwolfboy&action=edit&redlink=1`
159 `https://en.wikibooks.org/w/index.php%3ftitle=User:Smalls123456&action=edit&redlink=1`
160 `https://en.wikibooks.org/w/index.php%3ftitle=User:Sol~enwikibooks&action=edit&redlink=1`
161 `https://en.wikibooks.org/w/index.php%3ftitle=User:StephenFerg&action=edit&redlink=1`
162 `https://en.wikibooks.org/wiki/User:Suchenwi`
163 `https://en.wikibooks.org/w/index.php%3ftitle=User:Surfer190&action=edit&redlink=1`
164 `https://en.wikibooks.org/wiki/User:Syum90`
165 `https://en.wikibooks.org/w/index.php%3ftitle=User:Szeeshanalinaqvi&action=edit&redlink=1`
166 `https://en.wikibooks.org/wiki/User:Tecky2`
167 `https://en.wikibooks.org/w/index.php%3ftitle=User:Tedzzz1&action=edit&redlink=1`
168 `https://en.wikibooks.org/w/index.php%3ftitle=User:The_Kid&action=edit&redlink=1`
169 `https://en.wikibooks.org/wiki/User:The_djinn`
170 `https://en.wikibooks.org/wiki/User:Thunderbolt16`
171 `https://en.wikibooks.org/w/index.php%3ftitle=User:Tobych&action=edit&redlink=1`

2	Tom Morris[172]
1	Treilly[173]
2	Unionhawk[174]
1	Watchduck[175]
24	Webaware[176]
1	Wenhaosparty˜enwikibooks[177]
2	Whym[178]
1	WikiNazi[179]
1	Wilbur.harvey[180]
59	Withinfocus[181]
1	Wolf104[182]
14	Wolma[183]
3	Xania[184]
1	Yasondinalt[185]
20	Yath˜enwikibooks[186]
1	Σ[187]

172 https://en.wikibooks.org/wiki/User:Tom_Morris
173 https://en.wikibooks.org/w/index.php%3ftitle=User:Treilly&action=edit&redlink=1
174 https://en.wikibooks.org/wiki/User:Unionhawk
175 https://en.wikibooks.org/w/index.php%3ftitle=User:Watchduck&action=edit&redlink=1
176 https://en.wikibooks.org/wiki/User:Webaware
177 https://en.wikibooks.org/w/index.php%3ftitle=User:Wenhaosparty~enwikibooks&action=edit&redlink=1
178 https://en.wikibooks.org/wiki/User:Whym
179 https://en.wikibooks.org/w/index.php%3ftitle=User:WikiNazi&action=edit&redlink=1
180 https://en.wikibooks.org/w/index.php%3ftitle=User:Wilbur.harvey&action=edit&redlink=1
181 https://en.wikibooks.org/wiki/User:Withinfocus
182 https://en.wikibooks.org/w/index.php%3ftitle=User:Wolf104&action=edit&redlink=1
183 https://en.wikibooks.org/w/index.php%3ftitle=User:Wolma&action=edit&redlink=1
184 https://en.wikibooks.org/wiki/User:Xania
185 https://en.wikibooks.org/wiki/User:Yasondinalt
186 https://en.wikibooks.org/wiki/User:Yath~enwikibooks
187 https://en.wikibooks.org/wiki/User:%25CE%25A3

List of Figures

- GFDL: Gnu Free Documentation License. `http://www.gnu.org/licenses/fdl.html`
- cc-by-sa-3.0: Creative Commons Attribution ShareAlike 3.0 License. `http://creativecommons.org/licenses/by-sa/3.0/`
- cc-by-sa-2.5: Creative Commons Attribution ShareAlike 2.5 License. `http://creativecommons.org/licenses/by-sa/2.5/`
- cc-by-sa-2.0: Creative Commons Attribution ShareAlike 2.0 License. `http://creativecommons.org/licenses/by-sa/2.0/`
- cc-by-sa-1.0: Creative Commons Attribution ShareAlike 1.0 License. `http://creativecommons.org/licenses/by-sa/1.0/`
- cc-by-2.0: Creative Commons Attribution 2.0 License. `http://creativecommons.org/licenses/by/2.0/`
- cc-by-2.0: Creative Commons Attribution 2.0 License. `http://creativecommons.org/licenses/by/2.0/deed.en`
- cc-by-2.5: Creative Commons Attribution 2.5 License. `http://creativecommons.org/licenses/by/2.5/deed.en`
- cc-by-3.0: Creative Commons Attribution 3.0 License. `http://creativecommons.org/licenses/by/3.0/deed.en`
- GPL: GNU General Public License. `http://www.gnu.org/licenses/gpl-2.0.txt`
- LGPL: GNU Lesser General Public License. `http://www.gnu.org/licenses/lgpl.html`
- PD: This image is in the public domain.
- ATTR: The copyright holder of this file allows anyone to use it for any purpose, provided that the copyright holder is properly attributed. Redistribution, derivative work, commercial use, and all other use is permitted.
- EURO: This is the common (reverse) face of a euro coin. The copyright on the design of the common face of the euro coins belongs to the European Commission. Authorised is reproduction in a format without relief (drawings, paintings, films) provided they are not detrimental to the image of the euro.
- LFK: Lizenz Freie Kunst. `http://artlibre.org/licence/lal/de`
- CFR: Copyright free use.

- EPL: Eclipse Public License. `http://www.eclipse.org/org/documents/epl-v10.php`

Copies of the GPL, the LGPL as well as a GFDL are included in chapter Licenses[188]. Please note that images in the public domain do not require attribution. You may click on the image numbers in the following table to open the webpage of the images in your webbrower.

188 Chapter 38 on page 191

38 Licenses

38.1 GNU GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

Copyright © 2007 Free Software Foundation, Inc. <http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed. Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow. TERMS AND CONDITIONS 0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion. 1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work. 2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary. 3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures. 4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee. 5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

* a) The work must carry prominent notices stating that you modified it, and giving a relevant date. * b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices". * c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it. * d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate. 6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

* a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange. * b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge. * c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b. * d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements. * e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying. 7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

* a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or * b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or * c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or * d) Limiting the use for publicity purposes of names of licensors or authors of the material; or * e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or * f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way. 8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10. 9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so. 10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it. 11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law. 12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program. 13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such. 14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License “or any later version” applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version. 15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION. 16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. 17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the “copyright” line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.> Copyright (C) <year> <name of author>

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

<program> Copyright (C) <year> <name of author> This program comes with ABSOLUTELY NO WARRANTY; for details type ‘show w’. This is free software, and you are welcome to redistribute it under certain conditions; type ‘show c’ for details.

The hypothetical commands ‘show w’ and ‘show c’ should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an “about box”.

You should also get your employer (if you work as a programmer) or school, if any, to sign a “copyright disclaimer” for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see <http://www.gnu.org/licenses/>.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read <http://www.gnu.org/philosophy/why-not-lgpl.html>.

38.2 GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc. <http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed. 0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document ”free” in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of ”copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference. 1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The ”Document”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as ”you”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A ”Modified Version” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A ”Secondary Section” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The ”Invariant Sections” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The ”Cover Texts” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A ”Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not ”Transparent” is called ”Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The ”Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, ”Title Page” means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The ”publisher” means any person or entity that distributes copies of the Document to the public.

A section ”Entitled XYZ” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as ”Acknowledgements”, ”Dedications”, ”Endorsements”, or ”History”.) To ”Preserve the Title” of such a section when you modify the Document means that it remains a section ”Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License. 2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies. 3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document. 4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

* A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission. * B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement. * C. State on the Title page the name of the publisher of the Modified Version, as the publisher. * D. Preserve all the copyright notices of the Document. * E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices. * F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below. * G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice. * H. Include an unaltered copy of this License. * I. Preserve the section Entitled ”History”, Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled ”History” in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence. * J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the ”History” section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission. * K. For any section Entitled ”Acknowledgements” or ”Dedications”, Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein. * L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles. * M. Delete any section Entitled ”Endorsements”. Such a section may not be included in the Modified Version. * N. Do not retitle any existing section to be Entitled ”Endorsements” or to conflict in title with any Invariant Section. * O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled ”Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version. 5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled ”History” in the various original documents, forming one section Entitled ”History”; likewise combine any sections Entitled ”Acknowledgements”, and any sections Entitled ”Dedications”. You must delete all sections Entitled ”Endorsements”. 6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document. 7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an ”aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate. 8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled ”Acknowledgements”, ”Dedications”, or ”History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title. 9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it. 10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License ”or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document. 11. RELICENSING

”Massive Multiauthor Collaboration Site” (or ”MMC Site”) means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A ”Massive Multiauthor Collaboration” (or ”MMC”) contained in the site means any set of copyrightable works thus published on the MMC site.

”CC-BY-SA” means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

”Incorporate” means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is ”eligible for relicensing” if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing. ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (C) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled ”GNU Free Documentation License”.

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the ”with … Texts.” line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

38.3 GNU Lesser General Public License