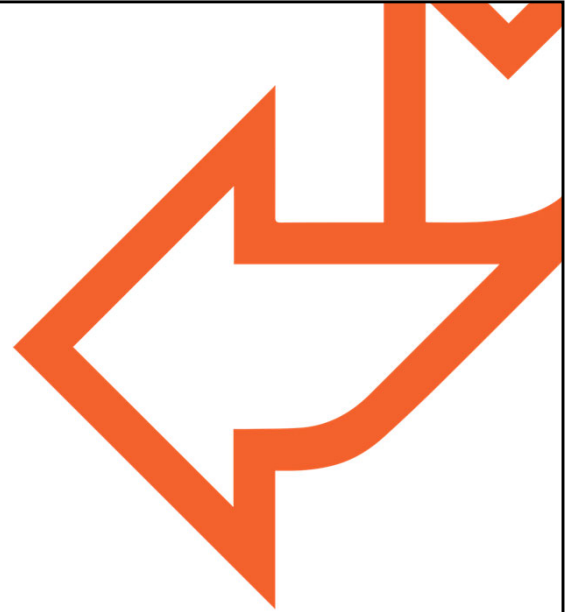




Python 3 Programming

Introduction to Python 3





INTRODUCTION TO PYTHON 3

Contents

- What is Python?
- Why Python?
- Zen of Python
- Python Interactive Session
- IPython and Jupyter
- Python scripts
- Python help
- Example of a Python script
- Modules
- Functions and built-ins

Summary

- Python built-in functions



This chapter gives an introduction to version 3 of the Python language. Python is a general purpose, object oriented, scripting language.

We will only have time to look at basics in this chapter, but that should be enough to get started.

QA What is Python?

Python is an object-oriented scripting language

- First published in 1991 by Guido van Rossum
- Designed as an OOP language from day one
- Does not need knowledge of OO to use

It is powerful

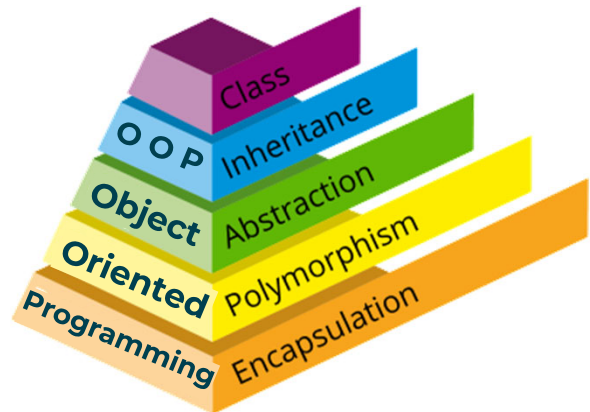
- General purpose, fully functional, rich
- Many extension modules

It is free

- Open source: Python licence is less restrictive than GPL

It is portable

- UNIX, Linux, Windows, OS X, Android, etc...
- Ported to the Java and .NET virtual machines



Python is a scripting language with a clean and Object Oriented (OO) based interface.

Python was invented by Guido van Rossum, however a much wider community has been involved in its development. Although a free and open source, Python version 2 is copyright BeOpen.com and Stichting Mathematisch Centrum Amsterdam. The Python 3 copyright however is owned by the Python Software Foundation, which was formed in 2001. It is not controlled by the Free Software Foundation, and does not contain any GNU code (although there are optional links), although the licence is "GPL compatible".

Selected links:

Python: <http://www.python.org>

Jython: <http://www.jython.org>

Iron Python:

<http://www.codeplex.com/Wiki/View.aspx?ProjectName=IronPython>



WHAT IS PYTHON 3?



Python 3 was released in December 2008

- Also known as Python 3000 or Py3k
- New version of the language

Not backward compatible with Python 2

- `2to3.py` tool distributed from Python 2.6 and 3.*n*

Most language features are the same

- Some detail has changed
- Syntax cleaned up
- Many deprecated features have been removed

4

The first thing everyone notices about Python 3 is that the **print** statement is no longer a, um, statement: it is a built-in function (see later). The effect is that we now must put parentheses around the thing we wish to print. That took a while to get used to after years of missing them out.

Many Python 3 scripts will run on earlier versions of Python 2, particularly on version 2.6 or later. This should be a coincidence rather than a feature and should not be relied on. The same may be said of Python 2 scripts which coincidentally runs on Python 3. From Python 2.6, many Python 3 features have been added to Python 2. Python 2.6 has a `-3` option to warn about differences between that version and Python 3, and there is a conversion utility **`2to3.py`**.

Python 2 development has ceased with its final release (2.7) on July 4th 2010, but Python 2.7 maintenance (bug fixes) will continue for some time.

Python 2 will continue to be used, but the future is Python 3. Guido van Rossum says: "if you're starting a brand-new thing, you should use 3.0". Mostly because of performance problems, 3.0 only had a short life and was replaced by 3.1 within a few months.

The King is dead. Long live the King!

QA Why Python?

The most common version is written in C

- Known as CPython
- Java based version: Jython
- C# (.Net) based version: Iron Python

Python based version: PyPy

- Just-in-time compiler
- Remarkable performance improvements x4
- Fewer threading issues
- Latest version is 3.9.10
- Might be the default Python one day
- Compile it yourself!

Platform Agnostic

- No platform specific functions in the base language
- Greater portability for applications



There are a number of versions of Python, and they all differ in subtle ways. The most common version, and the one assumed for this course, is known as CPython. It is written in C, for speed and portability, and runs on many platforms. You can even download the source code and compile it yourself.

The Java based version, Jython, uses Java native services for many of its features. It is particularly suitable for interfacing with Java systems, since it can use Java classes. Iron Python does a similar task with .Net objects. Neither Jython nor Iron Python currently have Python 3 compatible versions, but they are planned.

All these implementations, including CPython, have developers who talk to each other and exchange ideas, and sometimes even code. A pure Python VM, called PyPy, is in development, yet the other VMs have all gained new insights from it.

One of the features of the Python language is that operating specific features are separated out into a module called `os`.

Nothing in that module is guaranteed to be portable - that is the point of it. Despite that we can never be 100% sure that our scripts will run across platforms without change, filename syntaxes, for example, vary between UNIX, Windows, and Apple filesystems.

QA Why Python?

Power

- Garbage collector
- Native Unicode objects

Supports many component libraries

- GUI libraries such as wxPython and PyQt
- Scientific libraries such as NumPy, DISLIN, SciPy, etc.
- Mature JSON support
- Web template systems such as Jinja2, Mako, etc.
- Web frameworks such as Django, Pyjamas, and Zope
- Relational Databases support
- Libraries may be coded in C or C++



6

Like most modern languages, Python has a garbage collector. This means we do not have to worry too much about tidying up memory (we said "too much", not "at all"). Unicode is the term used to describe multi-byte character sets, i.e. those containing more than 256 characters. Western Europeans may consider these "foreign", but Unicode is required to store and display the Euro currency symbol €.

One of the strengths of community based products like Python is the huge range of add-on libraries, most of them free. They include numeric libraries like NumPy (<http://numpy.scipy.org>), and DISLIN - a library for data visualization (<http://www.mps.mpg.de/dislin>) .

JSON (JavaScript Object Notation) is considered an important part of Python, and is heavily used. It is used with Ajax, and is more or less an alternative to XML. There are several web development frameworks, not just those listed, and many other web interfaces, including at least four to the Twitter API (most use JSON).

RDBMS support includes PostgreSQL, Oracle, DB2, Sybase, SAPDB, Informix, Ingres, MySQL, SQLite, ODBC, and native Python databases buzhug, and SnakeSQL. There have been issues with Microsoft's SQL Server in the past.

The add-ons themselves would not be possible without low-level hooks into the C programming language. Most APIs (Application Program Interfaces) are designed around C, and Python can have libraries written in C. To interface to a 3rd-party product therefore, one "merely" has to write a C-Python wrapper around it.

Zen of Python

Guiding principles

```
IDLE Shell 3.10.6
File Edit Shell Debug Options Window Help
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
>>>
```



...and some entertainment

- pygame – set of libraries for writing games
- Many games use Python as a scripting language
- A fun way to learn

```
>>> import this
```

The Zen of Python, by Tim Peters

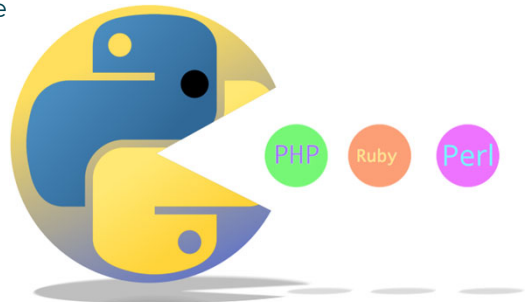
Try it yourself from IDLE.

See also <http://www.pygame.org> "Takes the C++ out of Game Development"

QA Performance downsides

Performance

- Python programs are compiled at runtime into 'byte code'
- Byte code does not compile down to PE or ELF
- Cannot be as fast as well written C
- Worse CPU usage than Java and C# on most benchmarks
- Better memory management



But...

- Roughly equivalent to Perl and Ruby - depending on the benchmark
- Better than PHP, much better than Tcl
- Packages often use compiled extensions
- Google *Unladen Swallow* project improved the base

8

Like many scripting languages, Java, C#, Perl, Ruby, PHP, Python is compiled at runtime into an internal format known as byte-code. It is unrealistic to expect this to run as quickly as C code which has been compiled into a native executable format such as PE COFF (on Windows) or ELF (on UNIX). Despite this, some remarkable benchmarks have been produced where the performance is sometimes comparable. It is worth pointing out that it requires a very good C programmer to produce fast, efficient code which is safe. Achieving safe code in Python is much easier, and quicker. By the way, as we shall see later, the byte-code produced by a Python module is usually saved, so it only has to be compiled once. This byte-code is not portable.

Further information on benchmarks is available at:

<http://shootout.alioth.debian.org/gp4/python.php>.

Iron Python and Jython perform about the same as CPython, however on all these VM implementations there is continual work on improvements and optimisations.

There are a number of projects working on speeding up Python. Psyco was an old Python 2 package that has maintenance issues - it will not run on 64-bit machines, for example. The development

effort on Psyco has moved to PyPy (Python written in Python), which has recently been ported to Python 3. Google's "Unladen Swallow" project was merged into the base, and has improved the base performance ("Unladen Swallow" comes from a memorable line in "Monty Python and the Holy Grail"), see PEP 3146.

QA The community

Python Software Foundation (PSF)

- Holds the Python intellectual property rights
- The starting point: www.python.org

Many newsgroups and forums

- Main download and docs: <https://python.org>
- Main newsgroup for Python users: comp.lang.python
- Web forum: python-forum.org/pythonforum/index.php
- Blog: <http://planet.python.org/>

Python Conference: PyCon

- EuroPython moves around European cities

Python Interest Groups

- U.S.A. - PIGgies



The starting point for anything Python related should be the Python Software Foundation's website. There you will find many links, including downloads and modules.

There are several Python Conferences around the world each year, including a EuroPython. See the PyCon link from python.org.

"Python", and the Python logo, are registered trademarks of the PSF.

personal firewall. On Linux, IDLE requires that Tk is installed.

* IDLE controls on OS X are slightly different:

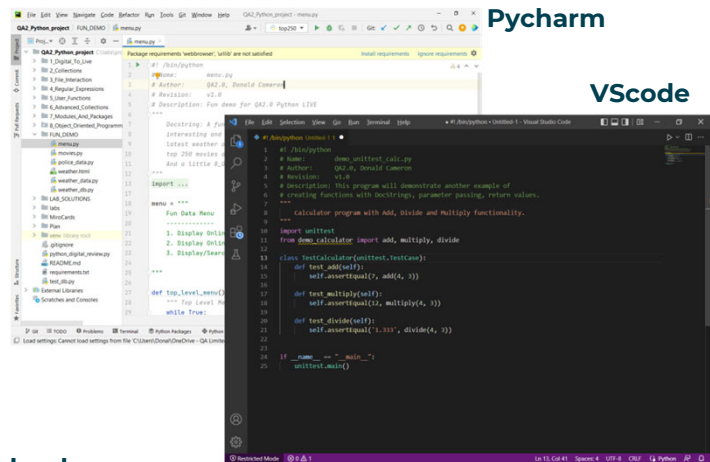
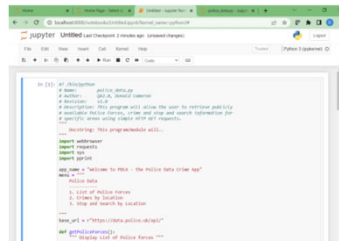
<CTRL>P	Previous statements
<CTRL>N	Next statements

QA IDEs

Integrate Development Environments

- User Friendly GUI
- Built-in text editor
- Built-in debugger
- Integrated console
- Coloured Coded highlighting
- Code Search
- Integrate with Online Repository
- Support CI/CD

Jupyter Notebooks



Integrate Development Environments

- Web-based and shareable
- Supports 100+ languages
- Data Sciences
- Machine Learning

The Python REPL prompt and IDLE are very basic interfaces with limited features.

Even beginner Pythonistas will move quickly onto using more feature rich IDEs such as Pycharm or VSCode. These IDEs provide a more powerful interface, console, editor and debugging features as well as integration with Version Control and online repositories like GitHub and BitBucket.

We even have web-based interfaces such as Jupyter Notebooks which is more specific to the Data Science community.

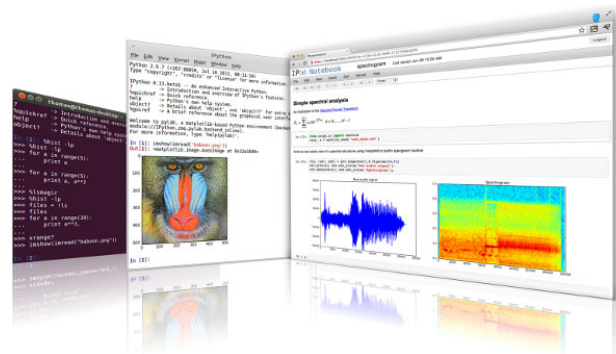
IPython and Jupyter

Interactive computing and data visualisation

- Powerful Interactive Shell
- Kernel for Jupyter
- Interactive data visualisation
- Support for Parallel programming

Jupyter Notebooks

- Fork of IPython in 2014
- Name derived from Julia, Python, and R
- Computational notebook for Literate Programming
- Support for multiple kernels



Source: <https://ipython.org/>

'Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do.' - Donald Knuth

12

IPython or Interactive Python is a command line shell for interactive computing. It offers introspection (can examine the type of an object), rich media support (text, images, graphs etc), shell syntax, tab completion and history.

It also provides a browser based notebook style interface with support for code, text, math expressions and interactive data visualisation such as plots. It is often used with Math and Data Science libraries such as NumPy, SciPy, and matplotlib

In 2014, the Ipython creator Fernando Pérez developed a spin-off project called Project Jupyter. IPython now exists as the kernel and shell for the Jupyter notebook inspired interface. Jupyter is language agnostic and gets its name to the core programming languages it supports including Julia, Python and R.

As such, Jupyter Notebook (formerly IPython Notebook) is a web-based interactive computational environment for creating, executing and visualising data and supports over 100 kernels – IPython being one of them. The programs are written in a Literate

Programming style first introduced by Donald Knuth in 1984

Python scripts

Python command line options

```
python [-options]... [-c cmd|-m mod|script file|-] [script arguments]...
```

Python scripts are compiled into byte-code

- Like Java, Perl, .NET, etc...
- Script files are suffixed .py by convention
- Compiled modules are suffixed .pyc
- Make sure your script names do not conflict with standard modules
- Use Sh-Bang (#!/usr/bin/python) in UNIX and Linux scripts

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

hello.py

```
chmod u+x hello.py  
./hello.py
```

13

Python programs (often called scripts) and modules usually have the file name suffix .py. Python modules (program components) are automatically saved in a semi-compiled format, with the suffix .pyc, to speed loading. The byte-code is automatically updated if the source file has a later timestamp, but is not portable.

For command line options to python, see `python -h`.

When choosing your script names, make sure they do not have the same name as a Python module. If in doubt, using a naming convention, for example by prefixing the script name with a code. We shall discuss modules in more detail later.

Python scripts on UNIX would normally be run by specifying `#!/python path` on the first line. The `#!` method is specific to UNIX, and requires that the script has execute access (`chmod u+x scriptname`). You only need to supply the `#!` line and `chmod` the main script; any imported modules only need read access. The `-c` option is not used much, but can be handy for small jobs. Specifying a dash instead of `-c` or a script file will read standard input.

The `-i` option allows you to run a script but then enter the interactive interpreter. Any functions or variables declared in the

script will be visible to the command-line (following the usual rules of scope).

The `-m` option runs an external module as a script.

You can check the version of Python you are using with `python -V` (uppercase V). This course assumes Python 3.0 or later.

Python help

Online

- <http://docs.python.org>
- <http://docs.python.org/lib>

UNIX/Linux man pages

```
$ man python
```

Python pydocs

```
$ pydoc sys
```

Python Shell – interactive help()

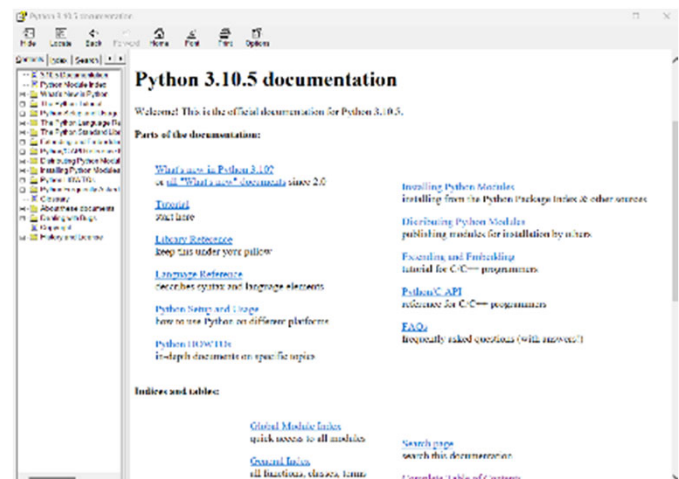
```
Python 3.10.5 Shell
File Edit Shell Debug Options Window Help
Python 3.10.5 Shell - C:\Python310\python.exe
>>>help()
Welcome to Python 3.10's help utility!
If this is your first time using Python, you should definitely check out
the tutorial on the internet at https://docs.python.org/3.10/tutorial/.

Enter the name of any module, keyword, or topic to get help on writing
Python programs and using Python modules. To quit this help utility and
return to the interpreter, just type "quit".

To get a list of available modules, keywords, symbols, or topics, type
"modules", "keywords", "symbols", or "topics". Each module also comes
with a one-line summary of what it does; to list the modules whose name
or summary contain a given string such as "spam", type "modules spam".

help> |
```

IDLE – Press <F1> for help



Python documentation is available online from the site shown, for example <http://www.python.org/doc/current/lib/modindex.html> is the online module reference. <http://docs.python.org/3.0/> is the Python 3 language reference.

IDLE has the documentation built-in, and on UNIX/Linux systems the man pages for python come bundled (try `man -k python` for extensions).

On UNIX/Linux the pydoc tool is usually available as a script, so just call `pydoc keyword` from the shell, for example:

```
pydoc sys
```

On Windows cmd.exe use the `pydoc.py` script in the Lib folder, for example:

```
C:\Python32\Lib\pydoc.py sys
```

There is also a useful TK GUI version usually bundled with the Windows release (in the Tools folder), but you may have to setup a shortcut to that.

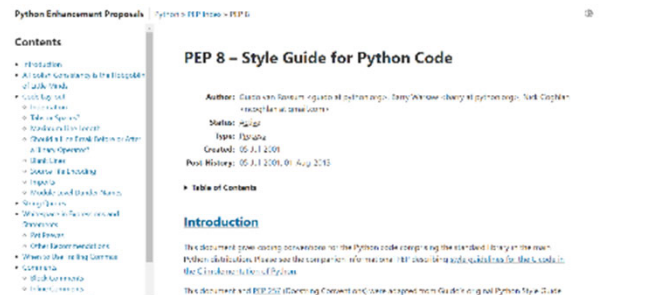
QA PEP – Python Enhancement Proposals

Language design document

- Evolution
- Information
- New features

PEP 8 – Style guide for Python

- <https://peps.python.org/pep-0008/>
- Code layout
- Readability
- Consistency
- Indentation – four spaces!
- Dos and don'ts



'Code is read much more often than it is written.'
- Guido Van Rossum

Python Enhancement Proposals, or PEPs, are an explanation of the rational of a particular feature of Python, and are often useful to aid understanding.

The (large) index is here: <http://www.python.org/dev/peps/>, but is also available in the Python 3 documentation, and there are many links to PEPs within the help text.

One of the most important PEP documents is PEP 8. It is the Style Guide for Python code and details best practice guidelines in layout, readability, consistency and general Do's and Don'ts when writing Python code. It should be READ and REFERENCED often.

🔗 Example of a Python program

```
#!/usr/bin/python

# Example Python script

import sys

argc = len(sys.argv)

if argc > 1:
    print('Too many args')
else:
    where = 'World'
    print("Hello", where)

print('Goodbye from ' +
      sys.argv[0])
```

Can enclose string literals in either " or '

#!/ line for UNIX/Linux (ignored on Windows)

Comment line

Load an external module

Variable assignment and function call

Condition is terminate by a colon :
Limits within a conditional are by
consistent indentation

print inserts a space between parameters

Statements are terminated by a new-line
unless inside brackets. *Note the + used to
join strings*

16

When a Python script is run, the code is first compiled then executed - much the same way as Java, C#, and many other languages. The compilation phase is very fast and there is (usually) no discernible delay.

All the lines shown are optional, and right now we do not show the whole story – for example, functions and global variables are omitted.

The first line indicates to UNIX the name of the Python interpreter. It is preceded by a #, which also marks the start of a comment, so that on non-UNIX platforms the line is seen as a comment.

The **import** statement introduces an external Python module. You will usually find this statement near the beginning of the script.

There are a many external modules available, this particular one, sys, gives us details of the runtime environment. More on modules later.

A variable assignment is shown - we do not formally declare the variable. The if condition is terminated by a colon and membership is by indentation. This also applies to other conditions such as while loops.

We are using **print**, and this is the reason for the py3 icon. Prior to Python 3, the parentheses were not required around the argument list (we won't mention that again). It writes text to the standard output stream (STDOUT), which is usually the terminal screen, and prints a new-line by default.

Statements are usually terminated by a new-line, but there are a few alternatives. A new-line which appears inside brackets (parentheses, square brackets or braces) is just considered to be

another white-space character. A new-line may be 'escaped' by placing a \ in front. If you must, several statements may be placed on the same line terminated by semi-colons.

Modules

Most Python programs load other Python code

- Standard Library modules bundled with Python
- Downloaded extensions, or modules written locally

```
>>> import sys
>>> print(sys.platform)
```

Find and compile 'sys'
Must specify the module name

```
>>> from sys import *
>>> print(platform)
```

Find and compile 'sys', and import all
names to our namespace

Examples:

- Operating system specific code - os
- Interface to the runtime environment - sys
- Scientific libraries like NumPy, DISLIN, SciPy, and many others
- Python's built-in functions are in the 'built-ins' module
- Automatically imported

```
>>> help("builtins")
```

17

Modules are files of Python code compiled and run as part of the main program. In fact, even the main program is really a module - called 'main'. Each module has its own namespace, which means that variables and functions created within them will not be confused with others of the same name in different modules. Modules are bundled with Python, and often used. A list of those issued with the system can be found in the online help. Modules can also be downloaded, usually from <http://pypi.python.org/pypi> (note the side-bar for modules specifically for Python 3). Python usually finds its modules among directories listed in the environment variable PYTHONPATH, or in Python's lib directory. Help text for modules can be obtained in Idle by typing help, then modules keyword, where keyword is part (or all) of the module name. Alternatively, use help('module-name'). We will be looking into writing our own modules later...

QA Functions and built-ins

A function is a named block of program code

- It can be passed values, which might be altered
- It can return a value
- We can write our own functions

```
lhs = function_name(arg1, arg2, ...)
```

Python includes many functions built into the compiler

- Called (remarkably) 'built-ins' and viewed as a module
- Always available - no need to import
- Examples: print(), len(), str(), list(), and set()
- See the on-line documentation
 - The Python Standard Library > Built-in Functions
 - Summary list at the end of this chapter

18

Functions are blocks of code provided to carry out a specific task. There are many supplied with Python, which adds greatly to the functionality of the language. A list is given after the chapter summary, but use the online documentation for details. The difference between built-in functions and those in a module is that they do not require any external file to be included - they are always available in your program, and are very fast. In Python 2.6 the Python 3.0, built-ins were available in a module called `future_built-ins`.



SUMMARY



Python is a free fully functional language

- Extensive on-line documentation

Python Scripts can be run:

- From scripts
- Interactively
- Or from an IDE

Python syntax is different!

Python syntax requires good indentation!

- And is intentional!

Using external modules is commonplace


- Load a module by using `import`

Built-in functions are:

- fast, always available, and useful!



PYTHON BUILT-IN FUNCTIONS (1)



```
abs(x)
all(iterable)
aiter(iterable)
any(iterable)
anext(iterable)
ascii(object)
bin(x)
bool([x])
breakpoint(function)
bytearray([arg[, encoding[, errors]]])
bytes([arg[, encoding[, errors]]])
callable(object)
chr(i)
classmethod(function)
compile(source, filename,
        mode[, flags[, dont_inherit]])
complex([real[, imag]])
delattr(object, name)
dict([arg])
```

```
dir([object])
divmod(a, b)
enumerate(iterable[, start=0])
eval(expression[, globals[, locals]])
exec(object[, globals[, locals]])
filter(function, iterable)
float([x])
format(value[, format_spec])
frozenset([iterable])
getattr(object, name[, default])
globals()
hasattr(object, name)
hash(object)
help([object])
hex(x)
id(object)
input([prompt])
int([number | string[, radix]])
isinstance(object, classinfo)
```

This slide, and the one which follows, is meant for reference - you are not expected to remember these! You will probably only regularly use about a quarter of them anyhow - not many experienced Python programmers could produce this list from memory.

This is not a substitute for the main online documentation.



PYTHON BUILT-IN FUNCTIONS (2)

```
issubclass(class, classinfo)
iter(o[, sentinel])
len(s)
list([iterable])
locals()
map(function, iterable, ...)
max(iterable[, args...], *[, key])
memoryview(obj)
min(iterable[, args...], *[, key])
next(iterator[, default])
object()
oct(x)
open(file[, mode='r'[, buffering=None
    [, encoding=None[, errors=None
    [, newline=None[, closefd=True]]]])
ord(c)
pow(x, y[, z])
```

21

This slide, and the one which follows, is meant for reference - you are not expected to remember these! You will probably only regularly use about a quarter of them anyhow - not many experienced Python programmers could produce this list from memory.

This is not a substitute for the main online documentation.