Python programming: An overview

Ali Dariush (Institute of Astronomy, University of Cambridge)

Cambridge International School (August 2016)

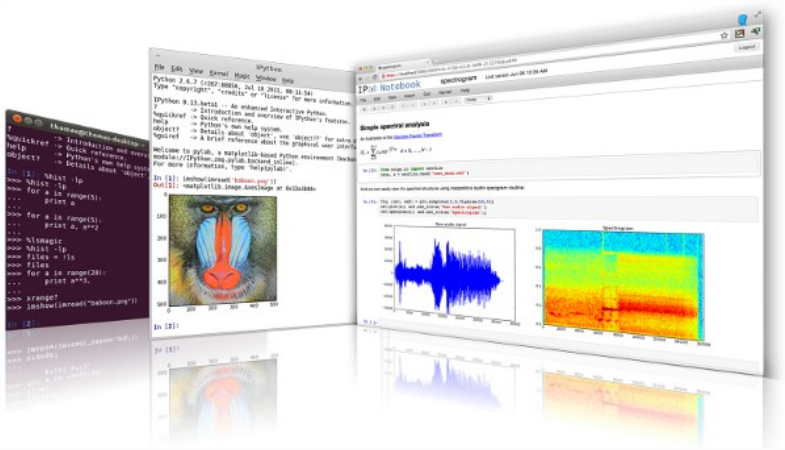
Computer program

A computer program (computer code) is a detailed set of instructions that tells a computer what to do with the data which is stored on a computer.

Python is a scripting language (not a compiled language)

Python environments

- IPython
- IPython notebook
- Anaconda
- Komodo Edit (or any other text editor)



python™ Web Resources

Python software foundation

https://www.python.org/doc/

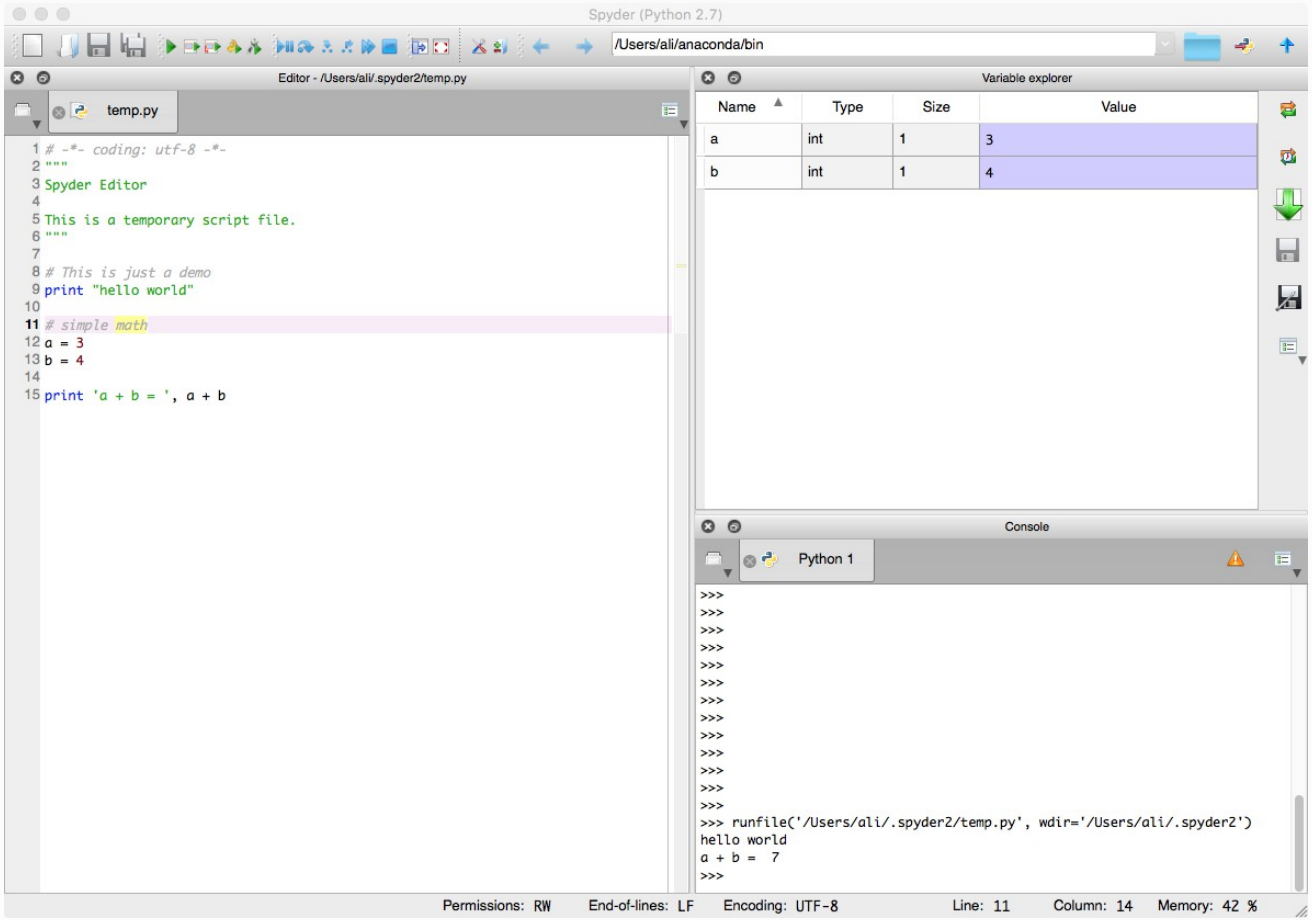
Tutorial points (almost everything including Python)

http://www.tutorialspoint.com

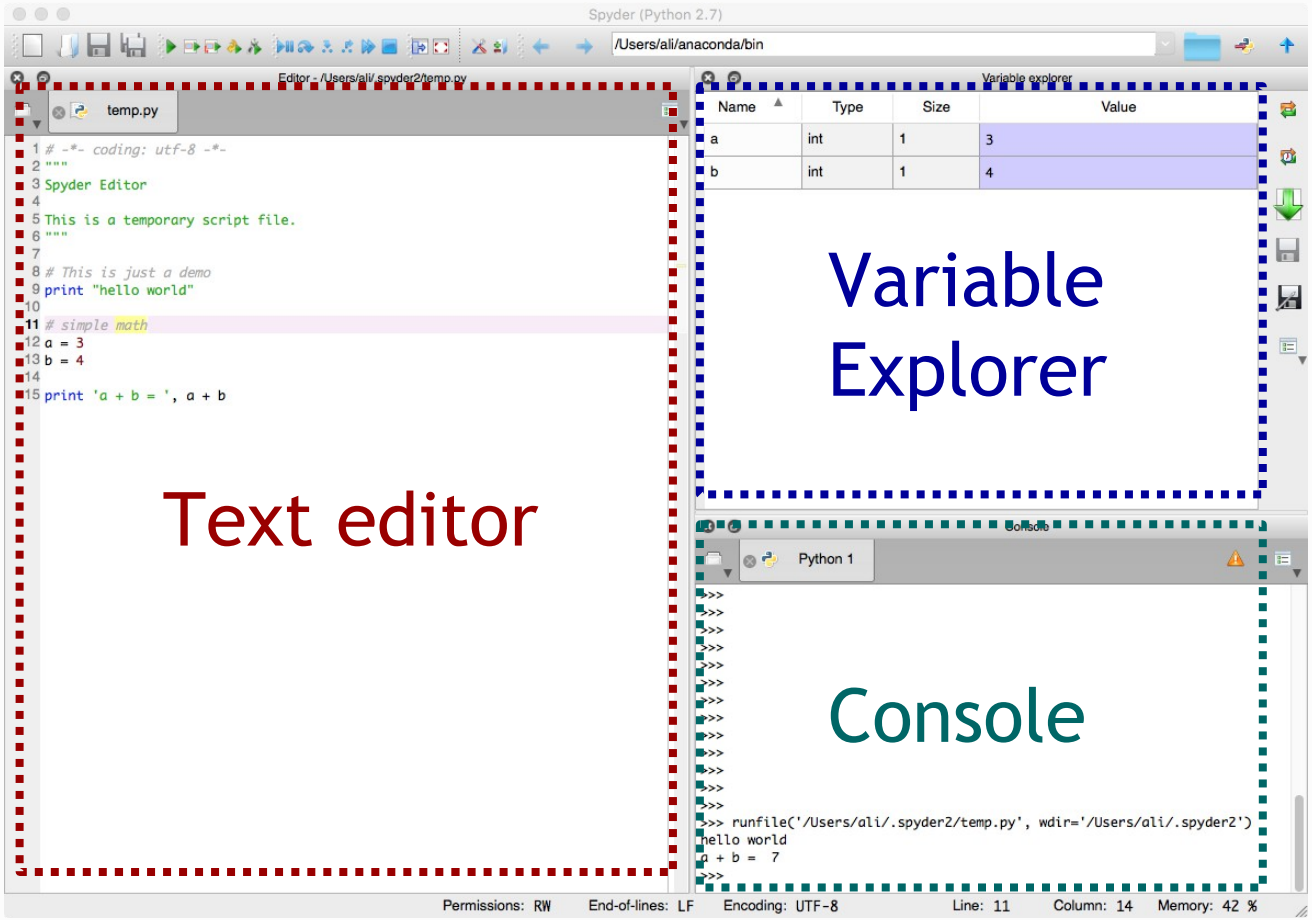
Stack overflow (Q&A supported by Python user community)

http://stackoverflow.com

Anaconda



Anaconda



Math

Comparison operators

==	Equal to
!=	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

Practice:

```
>>> 8 < 13          True
>>> 2 <= 1          False
>>> 13 > 12         True
>>> 12 != 13        True
>>> False < True    True
```

Math

Operators

Addition	+
Subtraction	-
Division	/
Multiplication	*

Practice:

```
>>> 4 + 12
>>> 15 - 3
>>> 9 + 6 - 15 + 12
>>> 2 * 15
>>> 16 / 4
>>> 15 // 4
>>> 16.0 / 4.0
>>> 15.0 / 4.0
>>> 15.0 // 4.0
```

Variables

&

Data types

Variables & Data types

Variables

- You can assign a name (variable) to a value (with a specific data type) once, but keep the result to use later.
- You can keep the same name for a variable, but change the value.

Data types (example)

`"Cambridge"`	string	
`345`	integer	
`3.14`	float	
`True`	boolean	▫ more details in next slides
`[1,2,3.4,"film"]`	list	▫ more details in next slides

Python tells us about types using the type() function:

```
>>> name = 'ali'
>>> a = 4
>>> b = 6.5
>>> print type(name) , type(a) , type(b)
<type 'str'> <type 'int'> <type 'float'>
```

String

String operators

+	Concatenation
*	Multiplication

Practice:

```
>>> ali
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'ali' is not defined
>>> 'ali'
'ali'
>>> "ali"
'ali'
>>> 'ali' + '@Cambridge'
'ali@Cambridge'
>>> " ali "*4
' ali  ali  ali  ali '
```

Data type: List

List

A list is a sequence of objects

```
>>> FootballTeams = [“Wales”, “Iceland”, “Brazil”, “Germany”]
>>> WorldRank = [65, 89, 5, 2]
```

Guess the output of the following commands:

```
>>> type(FootballTeams)
```

```
>>> type(WorldRank)
```

Data type: List

List

A list is a sequence of objects

```
>>> FootballTeams = [“Wales”, “Iceland”, “Brazil”, “Germany”]
>>> WorldRank = [65, 89, 5, 2]
```

Guess the output of the following commands:

```
>>> type(FootballTeams)
<type 'list'>

>>> type(WorldRank)
<type 'list'>
```

Data type: List

List

Index: Where an item is in the list

```
>>> Beatles = [“John”, “Paul”, “George”, “Ringo”]
>>> Beatles[0]
‘John‘
```

```
[“John”,     “Paul”,      “George”,    “Ringo”]
0            1            2            3
```

Python always starts at zero!

Data type: Booleans

Booleans

Q: What happens when we type Boolean values in the interpreter?

When the words 'True' and 'False' begin with upper case letters, Python knows to treat them like Booleans instead of strings or integers.

Try this:

```
>>> True
>>> False


>>> true
>>> false
>>> type(True)
>>> type("True")
```

Data type: Booleans

Booleans(and , or, not)

bool1	bool1	and	or
True	True	True	True
True	False	False	True
False	False	False	False

not

You can use the word `not` to reverse the answer that Python gives:

Any expression that is `True` can become `False`:

```
>>> 1==1
True

>>> not 1==1
False

>>> not True
False
```

Data type: Booleans

Booleans(and , or, not)

bool1	bool1	and	or
True	True	True	True
True	False	False	True
False	False	False	False

Try this:

```
>>> True and True
>>> True and False
>>> False and False

>>> True or True
>>> False or True
>>> False or False

>>> not True and True
>>> not True or True
```

Logic

&

Loops

Logic

`if` Statement

Making decisions:

`if` a *condition* is met:

perform an action

Example:

"If you're Tired, let's rest."

"If you like Football, let's play!"

Try this:

```
>>> Tired = True
>>> if Tired: print "Let's have a rest"
...
Let's have a rest

>>> game = 'Basketball'
>>> if game != 'Football': print "I do not want to play this game!"
...
I do not want to play this game!
```

Logic

`if` Statement

Adding a choice:

Adding a choice in our code
with the `else` clause:

"If you're hungry, let's eat lunch. Or else we can eat in an hour."

"If you like Frisbee, let's play! Or else we can play rugby."

Try this:

```
>>> city = "Cambridge"
>>> if city == "Oxford": print "county is Oxfordshire"
... else: print "county is Cambridgeshire"
...
county is Cambridgeshire
```

Logic

`if` Statement

Adding many choices:

Adding more choices in our code with the `elif` clause:

“If you’re hungry, let’s eat lunch. Or else we can eat in an hour. Or else we Can go home, or else ...”

Example

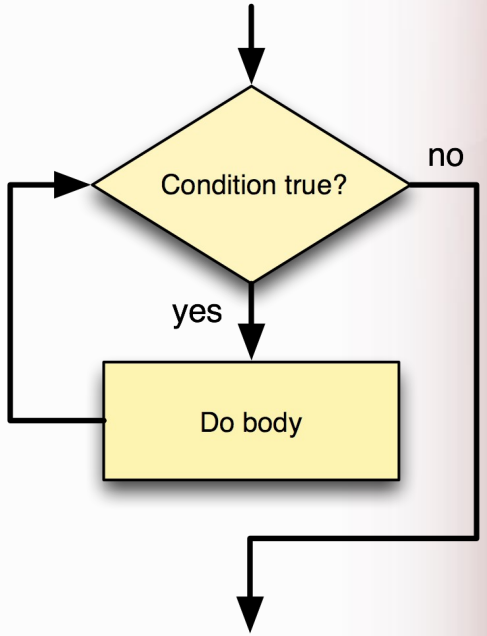
```
>>> if name == “Sara”
print “Hi Sara!”
  elif name == “Mary”:
print “Hi Mary!”
  else:
print “Who are you ?”
```

Loops

Loops

Loops are chunks of code that repeat a task over and over again.

- Counting loops repeat a certain number of times.
- Conditional loops keep going until a certain thing happens (or as long as some condition is `True`).



There are two types of loops in Python:
`for` and `while` loops

Loops

Loops (`for`)

Counting loops repeat a certain number of times – they keep going until they get to the end of a count.

```
>>> for mynum in [1, 2, 3, 4, 5]:
    print "Hello", mynum
Hello 1
Hello 2
Hello 3
Hello 4
Hello 5
```

The `for` keyword is used to create this kind of loop, so it is usually just called a for loop.

Loops

Loops (`while`)

Conditional loops repeat until something happens (or as long as some condition is `True`).

```
>>> count = 0
>>> while (count < 4):
    print 'The count is:', count
    count = count + 1
The count is: 0
The count is: 1
The count is: 2
The count is: 3
```

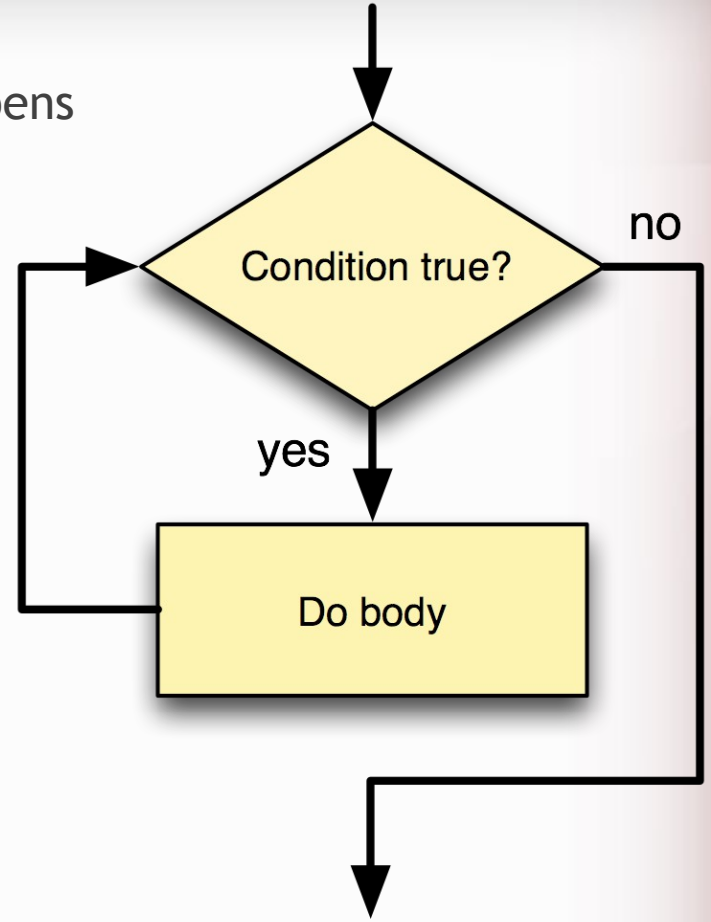
The `while` keyword is used to create this kind of loop, so it is usually just called a **while loop**.

Loops

Loops (`while`)

Conditional loops repeat until something happens (or as long as some condition is `True`).

```
>>> count = 0
>>> while (count < 4):
    print 'The count is:', count
    count = count + 1
The count is: 0
The count is: 1
The count is: 2
The count is: 3
```



The `while` keyword is used to create this kind of loop, so it is usually just called a **while loop**.

Algorithm

&

Functions

Algorithm & Functions

Algorithm

A set of instructions in order to perform a task or solve a problem.

How to make a cup of tea?

Get a flavour of tea bag.

Get a kettle.

Get a tea-pot.

Get a pot of water.

Make sure the kettle is plugged in…

…and on, and on, and on.

But to a human, it's just "make a cup of tea".

Algorithm & Functions

functions

Functions are just a concise way to group instructions into a bundle.

What it's like in our minds:

"Make a cup of tea." → bundle

In Python, you could say it like this:

```
make_tea(tea_bag, tea_pot, tea_cup, water, kettle)
```

function name

function parameters

How to define a function in Python?

- Functions are defined using `def`.
- Functions are called using parentheses `()`.
- Functions take *parameters* (inputs) and return *results* (outputs) using `return` keyword.
- `print` displays information, but does not give a *value*.
- `return` gives a *value* to the caller.

Algorithm & Functions

functions

Functions are just a concise way to group instructions into a bundle.

What it's like in our minds:

"Make a cup of tea." → bundle

In Python, you could say it like this:

```
make_tea(tea_bag, tea_pot, tea_cup, water, kettle)
```

function name — function parameters

Example

```
>>> def calculate_sum(value1, value2):
...     return value1 + value2
...
>>> calculate_sum( 85 , 95)
180
```

How to define a function in Python?

- Functions are defined using `def`.
- Functions are called using parentheses `()`.
- Functions take *parameters* (inputs) and return *results* (outputs) using `return` keyword.
- `print` displays information, but does not give a *value*.
- `return` gives a *value* to the caller.