Python Bootcamp
Basics II

PhD Bruno C. Quint
bquint@ctio.noao.edu
Resident Astronomer at SOAR Telescope

Table of Contents

Loops and control

if/elif/else

```
fruit = ‘banana’
if fruit is ‘apple’:
    eat_it()
elif fruit is ‘orange’:
    make_a_juice()
else:
    leave_it()
```

More at
https://docs.python.org/2/tutorial/controlflow.html

Loops and control

Python relies on identation,

so DON'T MESS UP!

if/elif/else

```
fruit = 'banana'
if fruit is 'apple':
....eat_it()
elif fruit is 'orange':
....make_a_juice()
else:
....leave_it()
```

More at
https://docs.python.org/2/tutorial/controlflow.html

Loops and control

Python relies on identation,
so DON'T MESS UP!

PEP8 HIGHLY recommends you to use 4 spaces. And NEVER mix spaces and tabs. For more informations, read the PEP-8.

if/elif/else

```
fruit = 'banana'
if fruit is 'apple':
....eat_it()
elif fruit is 'orange':
.....make_a_juice()
else:
....leave_it()
```

More at
https://docs.python.org/2/tutorial/controlflow.html

Loops and control

Python relies on identation,
so DON'T MESS UP!

PEP8 HIGHLY recommends you to use 4 spaces. And NEVER mix spaces and tabs. For more informations, read the PEP-8.

if/elif/else

```
fruit = 'banana'
if fruit == 'apple':
....eat_it()
elif fruit == 'orange':
....make_a_juice()
else:
....leave_it()
```

More at
https://docs.python.org/2/tutorial/controlflow.html

Loops and control

Python relies on identation,
so DON'T MESS UP!

for

List, tuples, arrays,
matrixes, dictionaries

```
my_list = ['a', 'b' 'c']
for my_item in my_list:
....print my_item
a
b
c
```

More at
https://docs.python.org/2/tutorial/controlflow.html

Loops and control

Python relies on identation,
so DON'T MESS UP!

for

List, tuples, arrays,
matrixes, dictionaries

```
my_list = ['a', 'b' 'c']
for my_item in my_list:
....print my_item
a
b
c
```

More at
https://docs.python.org/2/tutorial/controlflow.html

Loops and control

Python relies on identation,
so DON'T MESS UP!

for

List, tuples, arrays,
matrixes, dictionaries

Using indexes

```
my_list = ['a', 'b' 'c']

for index in range(len(my_list)):
....print index, my_item[index]

0 a
1 b
2 c
```

More at https://docs.python.org/2/tutorial/controlflow.html

Loops and control

Python relies on identation,
so DON'T MESS UP!

for

len(x) → return the number of elements of x.

List, tuples, arrays, matrixes, dictionaries

Using indexes

```
my_list = ['a', 'b' 'c']

for index in range(len(my_list)):
....print index, my_item[index]

0 a
1 b
2 c
```

More at https://docs.python.org/2/tutorial/controlflow.html

Loops and control

Python relies on identation,
so DON'T MESS UP!

for

len(x) → return the number of elements of x.

List, tuples, arrays, matrixes, dictionaries

Using indexes

```
my_list = ['a', 'b' 'c']

for index in range(3):
....print index, my_item[index]

0 a
1 b
2 c
```

More at https://docs.python.org/2/tutorial/controlflow.html

Loops and control

Python relies on identation,

so DON'T MESS UP!

for

range(n) → return a list with n integers starting at 0.

List, tuples, arrays, matrixes, dictionaries

Using indexes

```
my_list = ['a', 'b' 'c']

for index in range(3):
....print index, my_item[index]

0 a
1 b
2 c
```

More at https://docs.python.org/2/tutorial/controlflow.html

Loops and control

Python relies on identation,
so DON'T MESS UP!

for

range(n) → return a list with n integers starting at 0.

List, tuples, arrays, matrixes, dictionaries

Using indexes

```
my_list = ['a', 'b' 'c']

for index in [0, 1, 2]:
....print index, my_item[index]

0 a
1 b
2 c
```

More at https://docs.python.org/2/tutorial/controlflow.html

Loops and control

Python relies on identation,

so DON'T MESS UP!

While

```
while condition:
....do_something()
```

More at https://docs.python.org/2/tutorial/controlflow.html

Loops and control

Python relies on identation,

so DON'T MESS UP!

While

```
while condition:
....do_something()
```

foo.py

```
01 n_interested = 5
02
03 while n_interested < 0:
04 ....print('#Success :D')
05 ....n_interested = n_interested - 1
06
07 print('#Fail :(')
```

More at https://docs.python.org/2/tutorial/controlflow.html

Loops and control

Python relies on identation,

so DON'T MESS UP!

While

```
while condition:
....do_something()
```

foo.py

```
01 n_interested = 5
02
03 while n_interested
04 ....print('#Success
05 ....n_interested = n_interested - 1
06
07 print('#Fail :(')
```

```
$ python foo.py
#Success :D
#Success :D
#Success :D
#Success :D
#Success :D
#Fail :(
```

More at https://docs.python.org/2/tutorial/controlflow.html

Methods Defining your own

Python relies on identation,

so DON'T MESS UP!

Define a method

```
>>> def my_method(x, y):
>>> ....""""Add here some description"""
>>> ....k = 2 * x - y
>>> ....return k
```

x and **y** are required parameters.

More on methods (Functions): Here

Methods Defining your own

Python relies on identation,

so DON'T MESS UP!

Define a method

```
>>> def my_method(x, y):
>>> ....""""Add here some description"""
>>> ....k = 2 * x - y
>>> ....return k
```

```
>>> my_method(2, 4)
0
```

More on methods (Functions): Here

Methods Defining your own

Python relies on identation,

so DON'T MESS UP!

Watch out namespaces!

```
>>> x = 2
>>> def my_method(y):
>>> ....""""Some description"""
>>> ....k = 2 * x - y
>>> ....return k
>>>
>>> my_method(4)
0
```

More on methods (Functions): Here

Methods Defining your own

Python relies on identation,

so DON'T MESS UP!

Watch out namespaces!

```
>>> x = 2
>>> def my_method(x, y):
>>> ....""""Some description"""
>>> ....k = 2 * x - y
>>> ....return k
>>>
>>> my_method(3, 4)
2
```

More on methods (Functions): Here

Methods Defining your own

Python relies on identation,

so DON'T MESS UP!

Use **lambda**!

```
>>> x = 2
>>> my_method = lambda x, y: 2 * x - y
>>>
>>> my_method(3, 4)
2
```

More on methods (Functions): Here

Methods Defining your own

Python relies on identation,

so DON'T MESS UP!

Re-using your functions

```
01 def my_method(x, y):
02 ....“““Some description”””
03 ....k = 2 * x - y
04 ....return k
05 print my_method(3, 4)
```

foo.py

More on methods (Functions): Here

Methods Defining your own

Python relies on identation,

so DON'T MESS UP!

Re-using your functions

```
01 def my_method(x, y):
02 ....“““Some description”””
03 ....k = 2 * x - y
04 ....return k
05 print my_method(3, 4)
```

foo.py

```
01 import foo
02 foo.my_method(5, 2)
```

use.py

More on methods (Functions): Here

Methods Defining your own

Python relies on identation,

so DON'T MESS UP!

Re-using your functions

```
01 def my_method(x, y):
02 ....""""Some description"""
03 ....k = 2 * x - y
04 ....return k
05 print my_method(3, 4)
```

foo.py

```
01 import foo
02 foo.my_method(5, 2)
```

use.py

```
$ python use.py
2
8
```

More on methods (Functions): Here

Methods Defining your own

Python relies on identation,

so DON'T MESS UP!

Re-using your functions

```
01 def my_method(x, y):
02 ....“““Some description”””
03 ....k = 2 * x - y
04 ....return k
05 if __name__ == ‘__main__’:
06 ....print my_method(3, 4)
```

foo.py

```
01 import foo
02 foo.my_method(5, 2)
```

use.py

More on methods (Functions): Here

Methods Defining your own

Python relies on identation,

so DON'T MESS UP!

Re-using your functions

foo.py

```
01 def my_method(x, y):
02 ....“““Some description”””
03 ....k = 2 * x - y
04 ....return k
05 if __name__ == ‘__main__’:
06 ....print my_method(3, 4)
```

use.py

```
01 import foo
02 foo.my_method(5, 2)
```

```
$ python use.py
8
```

More on methods (Functions): Here

Methods Defining your own

Python relies on identation,

so DON'T MESS UP!

Positional Arguments

```
def my_method(x, y, z=2, w=3):
    """Add here some description"""
    k = 2 * x - y / z + w
    return k
```

More on methods (Functions): Here

Methods Defining your own

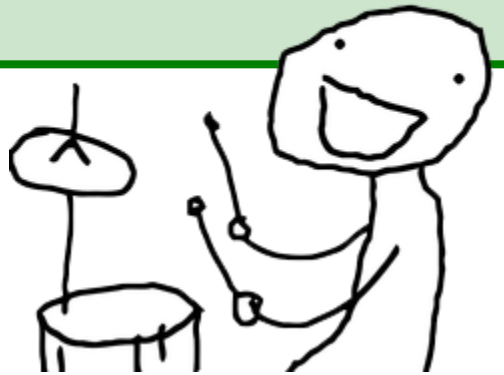
Python relies on identation,

so DON'T MESS UP!

Positional Arguments

```
def my_method(x, y, z=2, w=3):
    """Add here some description"""
    k = 2 * x - y / z + w
    return k
```

BA DUM TSSS



Positional arguments, their position matters.

More on methods (Functions): Here

Methods Defining your own

Python relies on identation,

so DON'T MESS UP!

Key Arguments

```
def my_method(x, y, **kwargs):
    """Add here some description"""
    k = 2 * x + y / kwargs['z']
    return k
```

**kwargs indicated that you may enter a dictionary as parameter to your function or a couple of optional parameters with no order.

Check out this link.

Methods Defining your own

Python relies on identation,

so DON'T MESS UP!

Key Arguments

```
def my_method(x, y, **options):
    """Add here some description"""
    k = 2 * x + y / options['z']
    return k
```

**kwargs indicated that you may enter a dictionary as parameter to your function or a couple of optional parameters with no order.

Check out this link.

Methods Using them

Built-in methods (or Functions)

```
>>> len('Hello World!')
12
>>> range(5)
[0,1,2,3,4]
>>> type('Aloha!')
<type 'string'>
>>> abs(-5.3)
5.3
```

Check all the build-in functions at
https://docs.python.org/2/library/functions.html

Methods Using them

Using methods from libs

```
>>> import math
>>> math.sqrt(9)
3
```

For more, check:
https://docs.python.org/2/tutorial/modules.html

Methods Using them

Using methods from libs

```
>>> import math
>>> math.sqrt(9)
3
>>> import math as m
>>> m.sqrt(9)
3
```

For more, check:
https://docs.python.org/2/tutorial/modules.html

Methods Using them

Using methods from libs

```
>>> from math import sqrt, log
>>> sqrt(9)
3
>>> log(10)
1
```

For more, check:
https://docs.python.org/2/tutorial/modules.html

Methods Using them

Using methods from libs

```
>>> from math import *
>>> sqrt(9)
3
>>> log(10)
1
```

For more, check:
https://docs.python.org/2/tutorial/modules.html

Methods Using them

Using methods from libs

```
>>> from math import sqrt as msqrt
>>> msqrt(9)
3
```

For more, check:
https://docs.python.org/2/tutorial/modules.html

Methods Using them

Using methods from objects

```
>>> s = 'Hello World!'
>>> s.lower()
'hello world!'
>>> s.isdigit()
False
```

For more, check:
https://docs.python.org/2/tutorial/modules.html

Gathering information

The built-in function **help()**

```
>>> def double(x):
>>> ....“““Doubles the value of x”””
>>> ....return 2 * x
>>>
```

Help on built-in function **help()** here.

Gathering information

The built-in function **help()**

```
>>> help(double_value)
    Help on function double_value:

double_value(x)
    Return the double of x
```

Help on built-in function **help()** here.

Gathering information

The built-in function **help()**

```
>>> y = 3.
>>> help(y)
Help on float object:

    class float(object)
 |  float(x) -> floating point number
 |
 |  Convert a string or number to a floating point
 |  number, if possible.
...
```

Help on built-in function **help()** here.

Gathering information

The built-in function **help()**

```
>>> y = 3.
>>> help(y)
```

```
>>> z = 3
>>> help(z)
```

```
>>> s = 'a string'
>>> help(s)
```

Help on built-in function **help()** here.

Gathering information

The built-in function **dir()**

```
>>> y = 3.
>>> dir(y)
['__abs__',
 '__add__',
  ...,
 'is_integer',
 'real']
```

Help on built-in function dir() here.

Gathering information

The built-in function **dir()**

```
>>> y = 3.
>>> dir(y)
['__abs__',
 '__add__',
  ...,
 'is_integer',
 'real']
```

```
>>> y = 3
>>> dir(y)
['__abs__',
 '__add__',
  ...,
 'real',
 'to_bytes']
```

Help on built-in function dir() here.

Gathering information

The built-in function **dir()**

```
>>> y = 3.
>>> dir(y)
['__abs__',
 '__add__',
 ...,
 'is_integer',
 'real']
```

._variable

is semiprivate and meant just for convention

.__variable

is considered superprivate and gets namemangled to prevent accidental access

.__variable__

is typically reserved for builtin methods or variables

Help on built-in function dir() here.

Python Vs. iPython

	Python	**iPython**
Colors	No...	Yes!
Auto-complete	No...	Yes!
For help?	Only with **help()**	**help()** or **?**
For attributes?	Only with **dir()**	**dir()** or . and TAB
Another features?	None	Magic Functions, PyLab, Notebook, etc

Questions?