



Python Bootcamp

An(other) Python Introduction II

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



Table of Contents

- What is Python?
- What will you need?
- Python as a terminal
- Python as a script
- Types of variables
- Loops and Control
- Using Methods and Libs
- Gathering Information



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

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

if/elif/else

Python relies
on indentation, so

DON'T MESS UP!

```
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

if/elif/else

PEP8 **HIGHLY** recommends you to use 4 spaces. And **NEVER** mix spaces and tabs. For more informations, read the [PEP-8](https://www.python.org/dev/peps/pep-0008/).



```
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

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

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

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

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

for

List, tuples, arrays,
matrixes, dictionaries

Using indexes

```
>>> my_list = ['a', 'b', 'c']
>>>
>>> for index in range(len(my_list)):
    ....print index, my_list[index]
    ....
0 a
1 b
2 c
```

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



Loops and control

for List, tuples, arrays,
matrixes, dictionaries

Using indexes

```
>>> my_list = ['a', 'b', 'c']
>>>
>>> for index in range(len(my_list)):
    .... print index, my_list[index]
    ....
0 a
1 b
2 c
```

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

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



Loops and control

for List, tuples, arrays,
matrixes, dictionaries

Using indexes

```
>>> my_list = ['a', 'b', 'c']
>>>
>>> for index in range(3):
    ....print index, my_list[index]
    ....
0 a
1 b
2 c
```

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

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



Loops and control

for List, tuples, arrays,
matrixes, dictionaries

Using indexes

```
>>> my_list = ['a', 'b', 'c']
>>>
>>> for index in range(3):
    ....print index, my_list[index]
    ....
0 a
1 b
2 c
```

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

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



Loops and control

for List, tuples, arrays,
matrixes, dictionaries

Using indexes

```
>>> my_list = ['a', 'b', 'c']
>>>
>>> for index in [0, 1, 2]:
    ....print index, my_list[index]
    ....

0 a
1 b
2 c
```

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

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



Loops and control

while

```
>>> while some_condition_is_true:  
    ....do_something()
```

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



Loops and control

while

test_while.py

```
>>> n_interested = 5
>>>
>>> while n_interested < 0:
...     print(" #Success :D ")
...     n_interested = n_interested - 1
...
>>> print(" #Fail :( ")
```

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



Loops and control

while

```
>>> n_interested = 5
>>>
>>> while n_interested > 0:
>>>     ....print(" #Success :D")
>>>     ....n_interested -= 1
>>>     ....
>>> print(" #Fail :( ")
```

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

Define a method

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

More on methods (Functions): [Here](#)



Methods

Defining your own

Define a method

x and **y** are **required** parameters.

```
>>> 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

Define a method **x** and **y** are now **keyword** parameters.

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

More on methods (Functions): [Here](#)



Methods

Defining your own

Define a method **z** and **w** are now **default** parameters.

```
>>> def my_method(x, y, z=1, w=0):  
.....    """Add here some description"""  
.....    k = 2 * x - y / z + w  
.....    return k  
.....  
>>> my_method(y=4, x=2)  
0
```

More on methods (Functions): [Here](#)



Methods

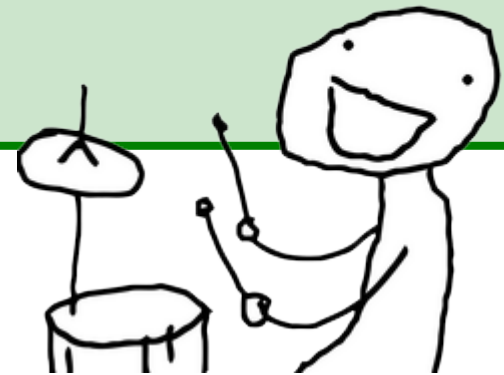
Defining your own

Define a method **z** and **w** are now **default** parameters.

```
>>> def my_method(x, y, z=1, w=0):  
.....    """Add here some description"""  
.....    k = 2 * x - y / z + w  
.....    return k  
.....  
>>> my_method(y=4, x=2)  
0
```

BA DUM TSSS

Default parameters have **default** values.



More on methods (Functions): [Here](#)



Methods

Defining your own

Define a method **z** and **w** are now **default** parameters.

```
>>> def my_method(x, y, z=1, w=0):  
..... """Add here some description"""  
..... k = 2 * x - y / z + w  
..... return k  
.....  
>>> my_method(y=4, x=2, w=5)  
5
```

More on methods (Functions): [Here](#)



Methods

Defining your own

Watch out namespaces!

```
>>>
>>> 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

Watch out namespaces!

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

More on methods (Functions): [Here](#)



Methods

Defining your own

Watch out namespaces!

```
>>>
>>> def my_method(x, y):
...     """Add here some description"""
...     k = 2 * x - y
...     return k
...
>>> my_method(4)
NameError: global name 'x' is not defined
```

More on methods (Functions): [Here](#)



Methods

Defining your own

Watch out namespaces!

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

More on methods (Functions): [Here](#)



Methods

Defining your own

Watch out namespaces!

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

More on methods (Functions): [Here](#)



Methods

Defining your own

Watch out namespaces!

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

More on methods (Functions): [Here](#)



Methods

Defining your own

Watch out namespaces!

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

More on methods (Functions): [Here](#)



Methods

Defining your own

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

Re-using your functions

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

foo.py

More on methods (Functions): [Here](#)



Methods

Defining your own

Re-using your functions

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

Re-using your functions

```
01 def my_method(x, y):  
02     """Some description"""  
03     k = 2 * x - y  
04     return k  
05  
06 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

Re-using your functions

```
01 def my_method(x, y):  
02     """Some description"""  
03     k = 2 * x - y  
04     return k  
05  
06  
07 print my_method(3, 4)
```

foo.py

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

use.py

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

More on methods (Functions): [Here](#)



Methods

Defining your own

Re-using your functions

```
01 def my_method(x, y):  
02     """Some description"""  
03     k = 2 * x - y  
04     return k  
05  
06 if __name__ == '__main__':  
07     print my_method(3, 4)
```

foo.py

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

use.py

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

More on methods (Functions): [Here](#)



Methods

Defining your own

Re-using your functions

```
01 def my_method(x, y):  
02     """Some description"""  
03     k = 2 * x - y  
04     return k  
05  
06 if __name__ == '__main__':  
07     print my_method(3, 4)
```

foo.py

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

use.py

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

More on methods (Functions): [Here](#)



Methods Using yours

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 yours

Using methods from libs

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

For more, check:

<https://docs.python.org/2/tutorial/modules.html>



Methods Using yours

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 yours

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 yours

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 yours

Using methods from libs

```
>>> from math import *
>>> from numpy import *
>>> sqrt(9)
3
>>> x = array([1, 4, 9])
>>> sqrt(x)
[1, 2, 3]
```

For more, check:

<https://docs.python.org/2/tutorial/modules.html>



Methods Using yours

Using methods from libs

```
>>> from numpy import *
>>> from math import *
>>> sqrt(9)
3
>>> x = array([1, 4, 9])
>>> sqrt(x)
```

TypeError: only length-1 arrays can be converted to Python scalars

For more, check:

<https://docs.python.org/2/tutorial/modules.html>



Methods Using yours

Using methods from libs

```
>>> from math import sqrt as msqrt
>>> from numpy import sqrt as nsqrt
>>> msqrt(9)
3.
>>> nsqrt([1, 4, 9])
[1., 2., 3.]
```

For more, check:

<https://docs.python.org/2/tutorial/modules.html>



Methods Using yours

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](#).



>>> **y = 3.**

Help on float object:

■ ■ ■

13/02/2017



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](#).



Questions?

