Python Tutorial

A Gentle Introduction 1

Yann Tambouret

Scientific Computing and Visualization
Information Services & Technology
Boston University
111 Cummington St.
yannpaul@bu.edu

October 2012

This Tutorial

- This is for non-programmers.
- The first half is very gentle.
- The second half is more in depth.

This Tutorial

- This is for non-programmers.
- The first half is very gentle.
- The second half is more in depth.
- If you have any programming experience, feel free to entertain yourself at codingbat.com/python

This Tutorial

- This is for non-programmers.
- The first half is very gentle.
- The second half is more in depth.
- If you have any programming experience, feel free to entertain yourself at codingbat.com/python
- Get ready to type... this is definitely interactive.

Python Overview

- Python is named after the BBC show "Monty Python's Flying Circus"

Python Overview

- Python is named after the BBC show "Monty Python's Flying Circus"
- We will focus on Python 2 today.

Python Overview

- Python is named after the BBC show "Monty Python's Flying Circus"
- We will focus on Python 2 today.
- Python on Katana and on Windows or Mac

Python Overview

- Python is named after the BBC show ”Monty Python’s Flying Circus”
- We will focus on Python 2 today.
- Python on Katana and on Windows or Mac
- This tutorial borrows largely from a tutorial by the Boston Python Group

boston python workshop

Python is Interpreted

- Python can be run interactively.

Python is Interpreted

- Python can be run interactively.
- Code ⇒ execution is almost instant; No explicit compilation step required.
- This allows for a faster development process

Python is Interpreted

- Python can be run interactively.
- Code ⇒ execution is almost instant; No explicit compilation step required.
- This allows for a faster development process
- The final product is usually more resource intensive, and as a side effect slower then comparable C/Fortran code.

Python is Interactive

Practice running python, type **python** in the terminal, hit Enter:

```
% python
Python 2.7 (#1, Feb 28 2010, 00:02:06)
Type "help", "copyright", "credits" or "license" for
>>>
```

- The '>>>' is a prompt asking for the next python line of code.
- Sometimes you'll see '...' as a prompt too.
- To exit, type `exit()` and Enter
  Try it!

Numbers 1

Start python (type python, then Enter), and try typing the following Addition:

```
2 + 2
1.5 + 2.25
```

Subtraction:

```
4 - 2
100 - .5
0 - 2
```

Multiplication:

```
2 * 3
```

Numbers 1 - Output

```
>>> 2 + 2
4
>>> 1.5 + 2.25
3.75
>>> 4 - 2
2
>>> 100 - .5
99.5
>>> 0 - 2
-2
>>> 2 * 3
6
```

Numbers 2

Division:

```
4 / 2
1 / 2
1.0 / 2
3/4 + 1/4
3.0/4 + 1.0/4
3.0/4.0 + 1.0/4.0
```

Numbers 2 - Output

```
>>> 4/2
2
>>> 1/2
0
>>> 1.0/2
0.5
>>> 3/4 + 1/4
0
>>> 3.0/4 + 1.0/4
1.0
>>> 3.0/4.0 + 1.0/4.0
1.0
```

type()

type() is a function that tells you what data type Python thinks something is. Try:

```
type(1)
type(1.0)
```

type()

type() is a function that tells you what data type Python thinks something is. Try:

```
type(1)
type(1.0)
```

results in:

```
>>> type(1)
<type ’int’>
>>> type(1.0)
<type ’float’>
```

type()

type() is a function that tells you what data type Python thinks something is. Try:

```
type(1)
type(1.0)
```

results in:

```
>>> type(1)
<type 'int'>
>>> type(1.0)
<type 'float'>
```

type() is a *function*, it takes one *argument* and it prints some info to the screen. We will talk more about functions in a bit, and create our own.

Tip

- Press the up arrow a few times in the terminal.
- The Python **Interpreter** saves a history of what you type.
- Pressing up allows you to access previous lines.
- Hit return and you re-run a command.

Variables 1

- Python variables can be made of any data type.
- Giving a name to some value is called **assignment**.
- Variable names cannot have spaces, and they need to start with a letter.

Try typing:

```
type(4)
x = 4
x
type(x)
2 * x
```

Variables 1 - output

and we get:

```
>>> type(4)
<type ’int’>
>>> x = 4
>>> x
4
>>> type(x)
<type ’int’>
>>> 2 * x
8
```

Note on Output

Just typing a value and the interpreter spits it back out at you. If you assign 4 to a variable, nothing is printed.

```
>>> 4
4
>>> x = 4
```

Variables 2

Reassignment is possible:

```
>>> x = 4
>>> x
4
>>> x = 5
>>> x
5
```

Variables 2

Reassignment is possible:

```
>>> x = 4
>>> x
4
>>> x = 5
>>> x
5
```

And order of operations is as you might expect:

```
>>> x = 3
>>> y = 4
>>> 2 * x - 1 * y
2
>>> (2*x) - (1*y)
2
```

Strings 1

Strings are surrounded by quotes:

```
"Hello"
"Python, I'm your #1 fan!"
```

And you can still use **type()** to check things out:

```
type("Hello")
type(1)
type("1")
```

Strings 1 - Output

```
>>> "Hello"
’Hello’
>>> "Python, I’m your #1 fan!"
"Python, I’m your #1 fan!"
>>> type("Hello")
<type ’str’>
>>> type(1)
<type ’int’>
>>> type("1")
<type ’str’>
```

Strings 2

Strings can be combined (concatenated):

```
"Hello" + "World"
```

Strings 2

Strings can be combined (concatenated):

```
"Hello" + "World"
```

And you can formally print strings with the **print** command:

```
print "Hello" + "World"
```

Strings 2 - Output

```
>>> "Hello" + "World"
'HelloWorld'
>>> print "Hello" + "World"
HelloWorld
```

The effect is the same, but there's a subtle difference of missing quotes.

print will become important soon, when we start writing scripts...

A Note about Errors

What happens when you type:

```
z
"Hello" + 1
```

A Note about Errors - Output

```
>>> z
Traceback (most recent call last):
  File "<console>", line 1, in <module>
NameError: name ’z’ is not defined
>>> "Hello" + 1
Traceback (most recent call last):
  File "<console>", line 1, in <module>
TypeError: cannot concatenate ’str’ and ’int’ objects
```

A Note about Errors - Output

```
>>> z
Traceback (most recent call last):
  File "<console>", line 1, in <module>
NameError: name 'z' is not defined
>>> "Hello" + 1
Traceback (most recent call last):
  File "<console>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int' objects
```

A **traceback** occurs:

- **TypeError** is the error that occurs
- **cannot concatenate 'str' and 'int' objects** is the 'helpful' message
- and every thing from Traceback to the error tells you where it happened

Strings 3

Single or Double quotes are OK:

```
print ’Hello’
print "Hello"
```

But be careful to escape extra quotes:

```
print ’I’m a happy camper’
print "I’m a happy camper"
print ’I\’m a happy camper’
```

And you can *multiply* strings by an integer:

```
h = "Happy"
b = "Birthday"
print (h + b) * 10
```

Strings 3 - Output

```
>>> print 'Hello'
Hello
>>> print "Hello"
Hello
>>> print 'I'm a happy camper'
  File "<console>", line 1
    print 'I'm a happy camper'
             ^
SyntaxError: invalid syntax
>>> print 'I\'m a happy camper'
I'm a happy camper
>>> print "I'm a happy camper"
I'm a happy camper
>>> h = "Happy"
>>> b = "Birthday"
>>> print (h + b) * 10
HappyBirthdayHappyBirthdayHappyBirthdayHappyBirthdayH
```

End of Part 1

Nobel Laureates 1

- The Python prompt is great for quick tasks: math, short bits of code, and testing.
- For bigger projects, it's easier to store code in a file.
- One such example can be found in `examples\nobel.py`

Nobel Laureates 2

1. Exit python: `exit()` then hit return
   go to examples directory: `% cd examples`
2. Run the script: `python nobel.py`
3. Open 'nobel.py' (e.g. using Emacs, gedit, vi)
   and answer these questions:

Nobel Laureates 2

1. Exit python: `exit()` then hit return
   go to examples directory: `% cd examples`
2. Run the script: `python nobel.py`
3. Open 'nobel.py' (e.g. using Emacs, gedit, vi) and answer these questions:

1. How do you comment code in Python?
2. How do you print a newline?
3. How do you print a multi-line string so that whitespace is preserved?

Booleans 1

A Boolean type is a type with two values: True/False.

Try typing the following:

```
True
type(True)
False
type(False)
```

```
>>> True
True
>>> type(True)
<type 'bool'>
>>> False
False
>>> type(False)
<type 'bool'>
```

Booleans 2a

You can also compare values, to see if they're equal:

```
0  ==  0
0  ==  1
```

Booleans 2b

You can also compare values, to see if they're equal:

```
>>> 0 == 0
True
>>> 0 == 1
False
```

== (equal equal) is for equality test

= (equal) is for *assignment*

Be careful! This can lead to bugs!

Booleans 3

You can do other comparisons: != means not equal

```
"a" != "a"
"a" != "A"
```

Others are just like math class:

```
1 > 0
2 >= 3
-1 < 0
.5 <= 1
```

Booleans 3 - Output

```
>>> "a" != "a"
False
>>> "a" != "A"
True
>>> 1 > 0
True
>>> 2 >= 3
False
>>> -1 < 0
True
>>> .5 <= 1
True
```

Booleans 4

You can see if something is *in* something else:

```python
"H" in "Hello"
"X" in "Hello"
```

or *not*:

```python
"a" not in "abcde"
"Perl" not in "Python Tutorial"
```

Booleans 4 - Output

```
>>> "H" in "Hello"
True
>>> "X" in "Hello"
False
>>> "a" not in "abcde"
False
>>> "Perl" not in "Python Tutorial"
True
```

Flow Control 1

You can use *Booleans* to decide if some code should be executed:

```
if 6 > 5:
    print "Six is greater than five!"
```

This is a multi-line piece of code:

1. `if 6 > 5:`
2. Enter
3. 4 spaces
4. `print "Six is greater than five!"`
5. Enter
6. Enter again...

Flow Control 2

The "..." is a special prompt; Python realizes this is a **code block**.
Final enter is to signify the end of **code block**.

Flow Control 2

The "..." is a special prompt; Python realizes this is a **code block**.

Final enter is to signify the end of **code block**.

```
>>> if 6 > 5:
...      print "Six is greater than five!"
...
Six is greater than five!
```

What's going on here?

if looks for a *Boolean*, and if it is true,

the **code block** is executed.

Flow Control 2

The "..." is a special prompt; Python realizes this is a **code block**.
Final enter is to signify the end of **code block**.

```
>>> if 6 > 5:
...      print "Six is greater than five!"
...
Six is greater than five!
```

What's going on here?
if looks for a *Boolean*, and if it is true,
the **code block** is executed.
`6 > 5` is `True`
so the next line is executed.

Flow Control 3

Now what will happened?

```
if 0 > 2:
    print "Zero is greater than two!"
if "banana" in "bananarama":
    print "I miss the 80s."
```

Flow Control 3 - Output

```
>>> if 0 > 2:
...     print "Zero is greater than two!"
...
>>> if "banana" in "bananarama":
...     print "I miss the 80s"
...
I miss the 80s
```

Indentation, what's up with that?

- If you've programmed in other languages, this indentation thing might seem weird.
- Python prides itself as an easy-to-read language, and indentation makes it easy to read **code blocks**.
- So Python requires indentation over if/end-if, begin-block/end-block organization.

Indentation - example

```
# this looks like other languages,
# but I use a comment to organize
if 1 == 1:
    print "Everything is going to be OK!"
    if 10 < 0:
        print "or is it?"
    #end if
    print "Inside first code block!"
#end if
```

Don't use `#end if`, just keep it in your mind if it gets confusing...

Flow Control 4

More control over choices `if` and `else`:

```
sister_age = 15
brother_age = 12
if sister_age > brother_age:
    print "sister is older"
else:
    print "brother is older"
```

- `else` **block** needs to be correctly indented too.
- `else` gets executed if *Boolean* is `False`.
- You don't *shouldn't* hit Enter twice between if code block and `else` statement.

Compound Conditionals 1

- `and` and `or` allow you to combine tests.
- `and`: `True` only if both are `True`
- `or`: `True` if **at least one** is `True`

Try these:

```
1 > 0 and 1 < 2
1 < 2 and "x" in "abc"
"a" in "hello" or "e" in "hello"
1 <= 0 or "a" not in "abc"
```

Compound Conditionals 1 - Output

```
>>> 1 > 0 and 1 < 2
True
>>> 1 < 2 and "x" in "abc"
False
>>> "a" in "hello" or "e" in "hello"
True
>>> 1 <= 0 or "a" not in "abc"
False
```

Compound Conditionals 2

Try this:

```
temperature = 32
if temperature > 60 and temperature < 75:
    print "It's nice and cozy in here!"
else:
    print "Too extreme for me."
```

Compound Conditionals 2 - Output

```
>>> temperature = 32
>>> if temperature > 60 and temperature < 75:
...     print "It's nice and cozy in here!"
... else:
...     print "Too extreme for me."
...
Too extreme for me.
```

Compound Conditions 3

And try this:

```
hour = 11
if hour < 7 or hour > 23:
    print "Go away!"
    print "I'm sleeping!"
else:
    print "Welcome to the cheese shop!"
    print "Can I interest you in some choice gouda?"
```

Compound Conditions 3 - Output

```
>>> hour = 11
>>> if hour < 7 or hour > 23:
...     print "Go away!"
...     print "I'm sleeping!"
... else:
...     print "Welcome to the cheese shop!"
...     print "Can I interest you in some choice goud
...
Welcome to the cheese shop!
Can I interest you in some choice gouda?
```

Flow Control 5

There's also `elif`:

```
sister_age = 15
brother_age = 12
if sister_age > brother_age:
    print "sister is older"
elif sister_age == brother_age:
    print "sister and brother are the same age"
else:
    print "brother is older"
```

Flow Control 5 - Output

```
>>> sister_age = 15
>>> brother_age = 12
>>> if sister_age > brother_age:
...     print "sister is older"
... elif sister_age == brother_age:
...     print "sister and brother are the same age"
... else:
...     print "brother is older"
...
sister is older
```

`else` is not required at the end,
just like in the first `if` example.

Functions

Remember `type()`? Functions ...

- do some useful work,
- let us re-use code without having to retype it,
- can take some input, and optionally `return` a value.

Functions

Remember `type()`? Functions ...

- do some useful work,
- let us re-use code without having to retype it,
- can take some input, and optionally `return` a value.

You call a function by using its name, followed by its **arguments** in parenthesis:

```python
length = len("Mississippi")
```

This assigns the number of characters in the string ”Mississippi” to the variable `length`.

Functions: Step 1

Write the function signature, how it will be called:

1. `def`, Tells Python you're defining a function.
2. Then a space, and the function's name.
3. Then an open parenthesis.
4. Then a comma-separated list of **parameters**
5. Then a closing parenthesis.
6. And finally a colon, ':'.

```python
def myFunction():
```

or

```python
def myFunction(myList, myInteger):
```

Functions: Step 2

Do something (useful):

- Underneath the function signature you do some work.
- This code must be indented, just like `if`/`else` blocks.
- This tells python that it's part of the function.
- You can use variables passed as **parameters** just like you used variables before

```
def add(x, y):
    result = x + y
```

Functions: Step 3

- Return something (if you want to).
- `return` tells python to return a result.

```
def add(x, y):
    result = x + y
    return result
```

or shorter....

```
def add(x, y):
    return x + y
```

In Python you can return anything: strings, booleans ... even other functions!

Functions: Step 3

Once `return` is called, the work in the function ends:

```
def absoluteValue(number):
    if number < 0:
        return number * -1
    return number
```

Functions: Step 3

Once `return` is called, the work in the function ends:

```python
def absoluteValue(number):
    if number < 0:
        return number * -1
    return number
```

This code have also been written like:

```python
def absoluteValue(number):
    if number < 0:
        return number * -1
    else:
        return number
```

Functions: Step 4

Use them! Again and again and again....

```python
def add(x, y):
    return x + y

result = add(1234, 5678)
print result
result = add(-1.5, .5)
print result
```

Functions: Step 4

Use them! Again and again and again....

```python
def add(x, y):
    return x + y

result = add(1234, 5678)
print result
result = add(-1.5, .5)
print result
```

Keep in mind, functions don't have to return something, but they usually do.

End of Part 2

Thanks!

Fill out the survey please!

Resources

- Like this tutorial: `https://openhatch.org/wiki/Boston_Python_Workshop_6/Friday`
- A good place to practice: `http://codingbat.com/python`
- Much more detail: `http://docs.python.org/tutorial/`