Refreshing Your Knowledge: Python Fundamentals for This Course

Xavier Morera

HELPING DEVELOPERS UNDERSTAND SEARCH & BIG DATA

@xmorera www.xaviermorera.com

History of Python

Conceived in the late 1980's

- Guido van Rossum

Implementation began in 1989

- First release in 1991
- Version 2 in 2000
- Version 3 in 2008

Adopted widely

About Python

Multi-paradigm programming language

Interpreted language

- REPL

Dynamic typing

Many libraries and built-in functions

Functional programming

Functional Programming

Paradigm

Break a problem into a set of functions

- No internal state
- Same input, same output

Opposite of OOP

Expressions, not statements

Philosophy: Zen of Python

PEP 20

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.

...

```
>>> import this
```

Running Python Code

Two ways of executing code

Run from terminal

```
# python test.py
```

```
# cat test.py
def tell_running():
    print "I am a running application in Python"
if __name__ == "__main__":
    tell_running()

# tell_running
```

Run from Terminal

Sample program

A function and scope

Execute and see the result

Running Python Code

Two ways of executing code

Run from terminal

```
# python test.py
```

Use the interactive shell

```
# python
>>>
```

Read Evaluate Print Loop

REPL

Work interactively

- Take inputs
- Evaluate
- Return result to user
- Keep going

Great for testing and prototyping

```
def tell_running():
    print "I am a running application in Python"
tell_running()

# tell_running
# exit()
```

REPL

Declared variables and functions remain in memory

History available

Ctrl + l to clean your screen, but you can scroll up

A Few Things to Note on the REPL

Immediate feedback from your expressions

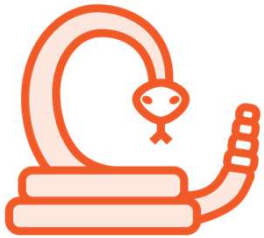
- Including errors

Powerful, but only for specific purposes

Does not include type ahead

No interface with your system

- No path completion



The Basics Required to Write Python Code

```
>>> message = "This is a physical line"
>>> message = "These are two
>>> message = "This is a logical \
line"
```

A Python Program

Made up of multiple logical lines

- Ends with NEWLINE

Each logical line can be made up of multiple physical lines

- Separate with \

```
# This is a single line comment
# This is not a good \
comment
""" This is a
multiline comment
"""
```

Comments

Single line

`#`

Multiline

`"""`

```
valid = "This is a valid variable"
1nvalid = "This is invalid"
b88l = True
sp%$# = 1
message
Message
```

Variables

Begin with a letter, then letters, numbers and underscores

- No special characters

Case sensitive

Variables in Python

Primitive	Compound
Strings	Lists
Integers	Tuples
Booleans	Sets
Floats	

```
message = "This is a variable"
message
type(message)
message = 42
message
type(message)
```

Variables & Types

Create variable by assigning a value

Type is inferred

- Dynamic typing

Python vs. Scala

Python

```
total = 42
```

Scala

```
val total = 42
```

```
message = "Use double quotes
message = 'Use single quotes'
message = "But do not mix'
message = "Unless it's for combining quotes"
message = 'It\'s ok to escape as well"
message = "Learn also the escape chars \nThey are useful"
message[0]
```

Strings

Use double quotes or single quotes

Do not mix

Escape characters

```
True
False
one_var = "One"
"One" == one_var
"one" == one_var
"Oneis" == one_var
```

Booleans & Comparison

True & False

Truthy and Falsy

```
10 * 100 == 100 * 10
10 - 100 == 100 - 10
a_list = []
len(a_list)
len(a_list) == False
True == 1
True == 0
True == -1
```

Booleans & Comparison

True & False

Truthy and Falsy

```
message = "a string"
message
print message
print "This is %s" % message
message2 = "another string"
print "One %s, two %s" % (message, message2)
print "%d" % 3.141519
print "%f" % 3.141519
```

Output

Use print

- Special statement, not a function

Format output

```
result = "Pi-ish " + 'plus ten is...'

pi_ish_plus_ten = 3.14159 + 10

print result + pi_ish_plus_ten

print result + str(pi_ish_plus_ten)
```

Operations & Typecast Conversions

String and mathematical operations

Concatenate different types

- Type casting

```
10 + 100 + 1000
(10 + 100) + 1000
10 + (100 + 1000)
10 * 1000 * 1000
100 * 10 * 1000
1000 * 100 * 10
```

Associate & Commutative

And talking about operations...

Associate: how you group

Commutative: swap

```
10 - 1000 - 1000
(10 - 100) - 1000
10 - (100 - 1000)
```

Associate & Commutative

And talking about operations...

Associate: how you group

Commutative: swap

```
numbers = [1, 2, 3]
type(list)
numbers
other_list = ['xavier', 1, True]
other_list[0]
other_list[0] = 'Xavier Morera'
other_list
```

Lists

Mutable data structure consisting of an ordered sequence of elements

- Group related items
- Each element is called an item

Perform operations on lists and on individual items

Lists

```
numbers + 4
numbers + [4]
numbers + other_list
numbers = [1, 2, 3]
numbers.append([4, 5])
numbers
numbers = [1, 2, 3]
numbers.extend([4, 5])
numbers
numbers[2:4]
numbers[:2]
```

Mutable vs. Immutable

Immutable: object cannot be changed after it is created

Mutable: can be modified after it is created

Mutable vs. Immutable

Mutable	Immutable
byte array	int
list	float
set	long
dict	complex
	str
	tuple
	Boolean
	array

```
number_five = 5
hex(id(number_five))
number_five += 1
hex(id(number_five))
number_list = [1, 2]
hex(id(number_list))
number_list.append(3)
hex(id(number_list))
```

Mutable vs. Immutable

Let's confirm

Add one to an int and check the identity

Add an element to a list and check the identity

```
qa_people = {'xavier': 1}
qa_people = {'xavier': 1, 'xavier':2}
qa_people['xavier']
qa_people['xavier'] = 10
qa_people['irene'] = 100
qa_people.items()
```

Dictionaries

Unordered set of key:value pairs, defined with {}

Keys unique, no constraints for values

Access by key, and perform operations

```
one_tuple = ('xavier', 1)

one_tuple[0]

bigger_tuple = (1, 'xavier', 'morera')
```

Tuples

Sequence of elements, immutable, and defined ()

Commonly used to represent a record

Access individual elements using []

```
{1, 2, 2, 3, 3}
```

Sets

Unordered collection of objects, inside {}

Similar to list

Difference: do not contain duplicate items

```
def tell_running(count, user):
    print user + "is running a Python program"
    return count + 1

tell_running(2, 'Xavier')

tell_running
```

Functions

Define a function

Input and output

```
tell_running
x = tell_running
x(3, 'Irene)
def execute_it(my_function, count, user):
    my_function(count, user)
execute_it(tell_running, 3, 'Xavier')
```

Functions

Assign a function

Pass a function as parameter

```
sys
```

```
import sys
```

```
sys
```

```
sys.copyright
```

Import

Many built in functions

Others you need to import

```
def tell_running(count, user):
    print user + "is running a Python program"
    return count + 1
```

Indentation & Code Blocks

Code blocks represented with the same indent level

4 spaces recommended in PEP 8, don't use tabs

Indent levels

Lists

```
def tell_running(count, user):
    print user + "is running a Python program"
  return count + 1

def tell_running(count, user):
    if count == 1:
        print user + "is running a Python program"
    return count + 1
```

Flow Control

Statements executed in order

Change flow with if, for and while

```
def tell_running(count, user):
    if count == 1:
        print user + " is in the True part of the if"
    elif count == 2:
        print user + ' is in the elif'
    else:
        print user + " is in the False part of the if"
```

If

Checks for a condition

Executes the if block on true, or multiple conditions with elif

Then executes the else if all conditions evaluate to false

```
names = ['Xavier', 'Irene', 'Juli', 'Luci']
for n in names:
    print n
```

For

Statement to loop over a sequence

Define a variable to hold each item's value during iteration

Execute all statements in the code block

```
i = 0
executing = True
while executing:
    i = i + 1
    print i
    if i == 5:
        executing = False
```

While

Execute a block

While condition is True

```
map(add_one, numbers_list)

filter(is_even, numbers_list)

reduce(add_items, numbers_list)
```

Map, Filter & Reduce

Functions that facilitate a functional programming approach

- Map: apply a function to all items on a list
- Filter: create new list for items that meet certain criteria
- Reduce: perform a computation on a list

```
def add_one(this_item):
    return this_item + 1
numbers_list = [1, 2, 3, 4, 5]
map(add_one, numbers_list)

def is_even(this_item):
    return this_item % 2 == 0
numbers_list
filter(is_even, numbers_list)

def add_items(first, second):
    return first + second
reduce(add_items, numbers_list)
```

PySpark: Apache Spark with Python

Access with `pyspark` (`pyspark2`)

Python shell

- Able to use Spark

Use Python's functionality

```
python
globals()
spark
exit()
```

```
pyspark2
globals
spark
sc
exit()
```

PySpark Shell

Check out the Python shell

Now take a look at PySpark

Takeaway

Python is powerful, easy to use & learn

Adopted widely

Many libraries available

- Big Data
- Data Science

Takeaway

Two ways of running code

Terminal

```
# python test.py
```

REPL

- One statement at a time
- Great for prototyping and testing

Takeaway

Python's syntax

Quick tip on *"is this Python or Scala?"*

Types: primitive and compound

Functions

Flow control

Functional Programming

- Map, Filter & Reduce

```
[hdfs@dn01 stackexchange]$ python
Python 2.7.5 (default, Aug  4 2017, 00:39:18)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-16)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

```
[hdfs@dn01 stackexchange]$ pyspark2
Python 2.7.5 (default, Aug  4 2017, 00:39:18)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-16)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.2.0.cloudera1
      /_/

Using Python version 2.7.5 (default, Aug  4 2017 00:39:18)
SparkSession available as 'spark'.
>>>
```