CC - AI

Programación para la Computación Científica - IA

Python for Programmers

Universidad Sergio Arboleda
Prof. John Corredor

A little Python history

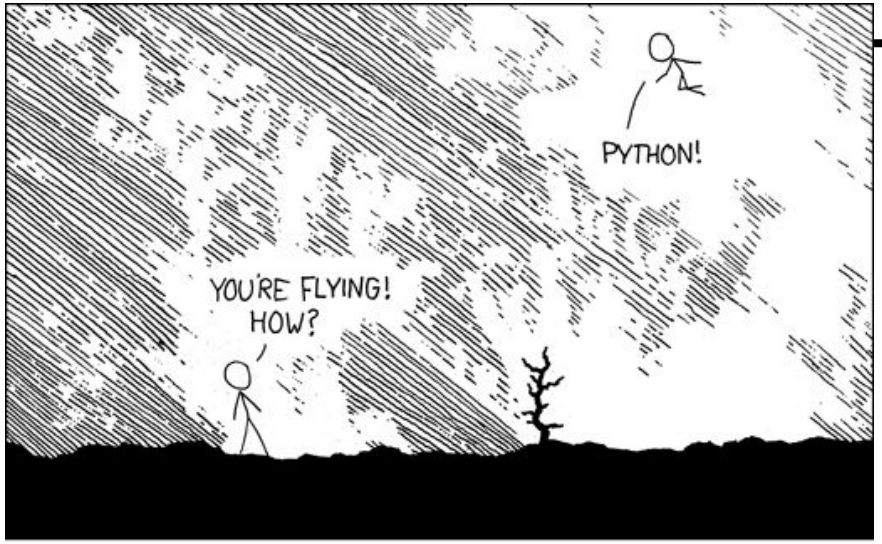
- Created by Guido van Rossum in late 80s
  - “Benevolent Dictator for Life”
  - Now works at Google
- Name is based off Monty Python

Why Python?

- Simple to get started, easy for beginners, powerful enough for professionals

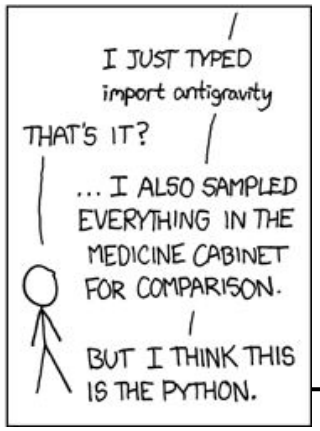
- Code is clean, modular, and elegant, making it easy to read and edit

- Whitespace enforcement


PYTHON!
YOU'RE FLYING! HOW?


I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST print "Hello, world!"


I DUNNO... DYNAMIC TYPING? WHITESPACE?
COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE!
BUT HOW ARE YOU FLYING?


I JUST TYPED import antigravity
THAT'S IT?
... I ALSO SAMPLED EVERYTHING IN THE MEDICINE CABINET FOR COMPARISON.
BUT I THINK THIS IS THE PYTHON.

Who is using Python?

- Google
- Reddit
- Twitter
- Pinterest/Instragram
- DreamHost
- YouTube
- BitTorrent
- DropBox
- ...and countless more!

Libraries: CherryPy and Jinja2

Why did we choose these?

- CherryPy:
  - Lightweight and easy to use microframework
  - Has direct interface with WSGI
  - It's fun to learn new things and we know you've all used django or flask ;)

- Jinja2
  - Commonly used python templating language
  - Easy to grok syntax
  - Integrates with many python libraries

So what is IPython, anyway?

It's a gussied up version of the standard Python interpreter.

Python is an **interpreted language**. This means that the interpreter executes the program source code directly, statement by statement.

In comparison, compiled languages (like C) must be explicitly translated into a lower-level machine language executable.

Brief Intro to Python Data Types

- Integers
- Strings
- Lists
- Tuples
- Dictionaries

Integers and basic math

```
In [6]: 2 + 2

Out[6]: 4

In [7]: 3 * 3

Out[7]: 9

In [8]: 2**3

Out[8]: 8

In [9]: 4/2

Out[9]: 2
```

Strings

```
In [1]: my_name = "Sandy Strong"

In [2]: print my_name

Sandy Strong
```

Now type, "`my_name.`" and hit the tab key:

```
In [3]: my_name.

my_name.capitalize  my_name.center      my_name.count
my_name.decode

my_name.encode  <snip>
```

Useful string methods

- strip
  - strips leading/trailing whitespace
- upper
  - makes all characters upper case
- lower
  - makes all characters lower case
- split
  - splits string into a list, whitespace delimited
- find
  - search for a string within your string
- startswith
  - test for what your string starts with

String formatting

You can pass variables into strings to format them in a specific way, for example:

```
In [14]: age = 28

In [15]: name = 'Sandy'

In [16]: print "Hello, my name is %s." % name

Hello, my name is Sandy.

In [17]: print "Hello, my name is %s, and I'm %s
years old." % (name, age)

Hello, my name is Sandy, and I'm 28 years old.
```

Lists

```
[2]: items = ['bacon', 3.14, ['bread',
'milk'] ]

In [3]: print items

['bacon', 3.14, ['bread', 'milk']]
```

You can put lists (and other Python data types) inside of lists.

- Useful list methods

- insert
  - provide index position and item to be inserted into the list
- append
  - append an item to the end of your list
- pop
  - provide index position to "pop" an item out of your list

Tuples

```
In [12]: colors = ('red', 'blue', 'green')


In [13]: print colors

('red', 'blue', 'green')
```

Tuples are **immutable**. Once they're created, they cannot be changed. Because they are immutable, they are hashable.

What does "hashble" mean?

An object is **hashable** if it has a hash value which never changes during its lifetime.

If we run a **hashing algorithm** on a tuple, it will return to us the hash value for that object. If we ran that same algorithm again on the same tuple, it would be identical-- because a tuple is immutable (it's values cannot change after it is defined).

Useful tuple methods

- index
  - provide an item in your tuple, and it returns the index position for that item

Dictionaries

```
In [1]: favorite_sports = {'John':
'Football', 'Sally': 'Soccer'}

In [2]: print favorite_sports

{'John': 'Football', 'Sally': 'Soccer'}
```

The first parameter is the "key" and the second parameter is the "value". A dictionary can have as many key/value pairs as you wish, and keys are **immutable**.

Useful dictionary methods

- get
  - retrieves the value of the given key or returns `None` if the key doesn't exist in the dictionary
- values
  - all values in your dictionary
- keys
  - all keys in your dictionary
- items
  - list of 2-element tuples that correspond to the key/value pairs in your dictionary
- update
  - add a new key/value pair to your dictionary

Wait, what is `None`?

It's a Python built-in constant that is frequently used to represent the **absence** of a value.

`None` is Python's version of NULL.

```
In [1]: None == None

Out[1]: True
```

Incrementing and Decrementing Values

```
# increment

counter += 1



# decrement

counter -= 1
```

Introduction to Python Loops, Code Blocks, and Boolean Logic

- for
- while

BUT before we can jump into loops....

... we have to talk about whitespace awareness

Python is a "whitespace aware" programming language.

This simply means that, within a **code block**, Python expects **uniform indentation**, the standard is 4 spaces.

What defines a code block? Let's try out some loops and find out...

For loops: "for each item in x, do y"

```
In [1]: items = ['socks', 'shoes', 'shirt', 'pants']

In [2]: for x in items:
   ...:     print x
   ...:
socks
shoes
shirt
pants
```

The **green highlighted code** represents a code block. See how the line "`print x`" is indented **in** 4 spaces from the line "`for x in items:`"? IPython does this for you, when coding outside of IPython, you must make your own indentations.

What happens if I don't indent?

```
In [1]: items = ['socks', 'shoes', 'shirt', 'pants']

In [2]: for x in items:

   ...: print x

   ...:
```

IndentationError: expected an indented block

I highly recommend this article explaining whitespace further:

http://www.secnetix.de/olli/Python/block_indentation.hawk

Enough boolean logic for this tutorial:

```
In [3]: True == True
Out[3]: True
In [4]: False == True
Out[4]: False
In [5]: False == False
Out[5]: True
In [6]: not False == True
Out[6]: True
In [7]: not True == False
Out[7]: True
```

While loops: "while x is true, do y"

```
In [4]: counter = 3
In [5]: while counter >= 0:
   ...:     print counter
   ...:     # decrement the counter
   ...:     counter -= 1
   ...:
3
2
1
0
```

This while-loop will continue to execute until the condition "counter >= 0" no longer evaluates to **`True`**.

Watch out for infinite loops!

In programming, an **infinite loop** is a loop of code which your program enters, but never returns from.

While-loops can be tricky, because they continue running while a condition remains True. The slightest logic error can lead to an infinite while-loop.

For-loops are less prone to infinity because they operate on iterable objects (lists, strings, dictionaries), which are finite by nature.

Basic Control Flow Tools

These are the keywords you need to know:

- `if`
- `else`
- `elif`

In Python we use the above keywords to control flows within our program.

Depending on which condition is **met** (evaluates to `True`) within a block of control flow code, we may want our application to behave differently.

Comparison operators

Equal to: ==

Less than: <

Greater than: >

Less than or equal to: <=

Greater than or equal to: >=

Let's try it out...

if and else

```
In [70]: age = 20
In [71]: if age <= 20:
   ....:     print "You're old enough!"
   ....:
You're old enough!

In [72]: if age > 20:
   ....:     print "You're too old!"
   ....: else:
   ....:     print "You're the right age."
   ....:
You're the right age.
```

— Using `elif`

The `elif` keyword allows you to expand your control flow block, and perform additional comparisons within an `if-else` statement.

```
In [74]: if age < 20:

   ....:     print "You're too young."

   ....: elif age >= 20:

   ....:     print "You're the right age!"

   ....:

You're the right age!
```

List comprehensions: *magical!*

```
In [75]: user_pets = {'Sandy': 'Dog',
'Katharine': 'Snake', 'Bob': 'Mouse', 'Sally':
'Fish'}

In [76]: pet_list = [ v for k,v in
user_pets.iteritems()]

In [77]: print pet_list

Out[77]: ['Snake', 'Mouse', 'Fish', 'Dog']
```

Python programmers pride themselves on writing tight, legible, efficient code. List comprehensions are a great example of that.

Using control flow and conditionals in list comprehensions

```
In [77]: user_pets = {'Sandy': 'Dog',
'Katharine': 'Snake', 'Bob': 'Mouse', 'Sally':
'Fish'}

In [78]: pet_list = [ v for k,v in
user_pets.iteritems() if k != 'Bob']

In [79]: print pet_list

['Snake', 'Fish', 'Dog']
```

We will only add the pet to `pet_list` if the user is not Bob.

_Writing Python functions

```
In [6]: def hello_world(user='John'):
   ...:     """ This is a docstring """
   ...:     message = "Hello, %s!" % user
   ...:     return message
   ...:

In [7]: my_message = hello_world(user='Sandy')

In [8]: print my_message

Hello, Sandy!
```

`*args` and `**kwargs`

In addition to single or comma-separated arguments, you can also pass lists or dictionaries of parameters into Python functions

```
def function(*args):

    # Expects something like:

    # ['he', 'she', 'it']

def function(**kwargs):

    # Expects something like:

    #{'me': 'you', 'we': they'}
```

. A word about `import` statements .

Python has an extensive standard library, filled with very useful modules that will prevent you from having to write boilerplate code.

```
# import the random library

In [60]: import random

# Print a random intiger between 1 and 5:

In [61]: print random.randint(1,5)

4
```

Put `import` statements at the top of your file.

Intro to Python classes

```
# import the imaginary 'cars' module
import cars
class MyCar:
    ''' This class defines a car object '''
    # member variables go here
    make = "Honda"
    model = "Civic"
    # class methods go here
    def car_color(self, clr="black"):
        self.color = cars.paint(clr)
        return True

    def tire_type(self, type="standard"):
        self.tires = cars.tire(type)
        return True
```

Wait-- what exactly is a class in Python?

In object-oriented programming, a class is a construct that is used to create instances of itself, called **class objects.**

A class defines constituent members which enable its instances to have **state and behavior. Data field** members (member variables) enable a class instance to maintain state.

Other kinds of members, especially **methods**, enable the behavior of class instances.

http://en.wikipedia.org/wiki/Class_(computer_programming)

Ok, so how do I use the `MyCar` class?

```
>>> import MyCar
>>> car = MyCar()
>>> print car.make
Honda
>>> print car.model
Civic
>>> car.car_color()
>>> print car.color
black
>>> car.tire_type(type="sport")
>>> print car.tires
sport
```

Be aware of the `__init__` method

There's a special method, named `__init__` , that is implicitly defined when you create a new class. By default, `__init__` is empty and does nothing special.

In some cases, you will want to override this method to have it do something in particular, you can do that by simply defining it at the top of your class:

```
class MyCar:

    def __init__():

        # do stuff here!
```

Methods vs. Functions

In Python, what we call a "method" is actually a **member function** of a class.

A standalone function that belongs to no class, is simply referred to as a "function".

References

- Kernighan, Brian W., and Dennis M. Ritchie. The C Programming Language. Vol. 2. Englewood Cliffs: prentice-Hall, 1988.
- Silberschatz, Abraham, Peter B. Galvin, and Greg Gagne. Operating System Concepts. Vol. 8. Wiley, 2013.
- https://planningtank.com/computer-applications/data-processing-cycle
- https://www.talend.com/resources/what-is-data-processing/
- https://peda.net/kenya/ass/subjects2/computer-studies/form-3/data-processing/dpc2
- https://www.academia.edu/38210518/What_is_Data_Processing
- https://www.studocu.com/en/document/polytechnic-university-of-the-philippines/information-and-communication-technology/lecture-notes/data-processing-lectures-in-data-processing/3167716/view
- http://download.nos.org/srsec330/330L2.pdf