Intro to Python

Python Workshop w/Prof Loren

python

Python? ...Like a snake? hissss...?

- Python is an interpreted, object-oriented, high-level programming language with dynamic semantics.
- Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance.
- It’s free!
- https://www.python.org/
- Currently there are two versions of Python: Python2 and Python3

Let’s dive right in!

Follow along in your terminal (if on Mac or Linux) or go to Powershell (if on Windows)!

```
➜  ~  python
Python 2.7.10 (default, Jun 30 2015, 15:30:23)
[GCC 4.8.4] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> print 'hello world'
hello world
>>>
```

python

Or let’s dive right in, online style.

https://www.repl.it/languages/python

https://coderpad.io/

Python’s shell aka The Interpreter

- ❑ Write functions
- ❑ Import libraries
- ❑ Explore Python
- ❑ CODE, CODE, CODE

I dare you to:

import this

```
>>> import this
The Zen of Python, by Tim Peters

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.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
>>>
```

python

Variables & Data Types

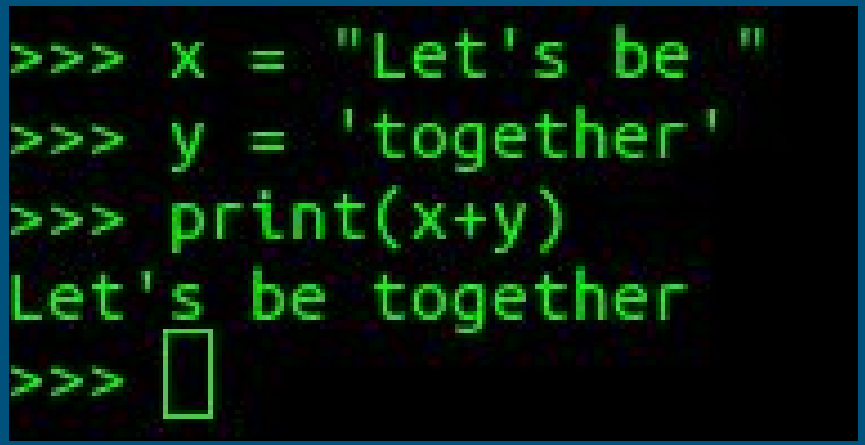
- **Variables** are reserved memory locations to store values. These variables can have different data types (or object types). This is done by the '=' (equal sign).

  You as the programmer assign values to variables (ALL THE POWER!)

- Data Types
  - String
  - Int
  - Float
  - Boolean

String? Like yarn? Like the cheese? Que-so?

- Strings are combinations of alphanumeric characters and special symbols defined between single (‘), double (“) quotes or triple (”’) for multiple lines
- You can do math with strings aka **concatenation**



python

```
ing = 'i am a string'
oby_string = "No, I'm a string"
cial_chars = "@#$%!'''
 "Chill."

nt string
string
nt snobby_string
 a string
nt special_chars

nt x


conformist = '''I

e! '''

nt nonconformist
```

Note:

- You can have quotes in a string just make sure they are not the same quotes surrounding the string itself (i.e. special_chars)
- When writing multiple lines when you <Enter> for a new line Python adds the ... at the beginning of each line, you just need to type away

python

Int? Float?

- **Int** meaning **integer** are whole numbers (i.e. 1,8, 12345678909876543 21)
- **Float** or **floating point number** are numbers that include decimals (i.e. 3.14, 1.0, 1234567890.0987654321)

python

```
>>> myint = 4
>>> mydec = 4.0
>>>
>>> type(myint)
<type 'int'>
>>> type(mydec)
<type 'float'>
>>> myint == mydec
True
>>>
>>> 4/5
0
>>> 4.0/5
0.8
>>> 4/5.0
0.8
>>> 4.0/5.0
0.8
>>>
```

Note:

- ➔ int & floats can still be the same values (i.e. 4 is the same as 4.0)
- ➔ However, they differ in division
  - ◆ just ints: you get the first part before the decimal
  - ◆ add a float: and you get what you will get the ‘accurate answer’ (if you want decimals for an answer anyways)

python

Boolean? ...Is that supposed to scare me? o.O

- Booleans are True/False values.
- Super powerful, they allow you to represent different possibilities in code.

```
>>> t = True
>>> f = False
>>>
>>> t
True
>>> f
False
>>>
```

raw_input

- Built-in function
- Gets whatever the user wrote (after they click <enter>) and reads it in as a **string.** **important, make a note of it**
- Allows you to put in a note for the user to know what they should input
- Notice how the variable with the raw_input function becomes what the user typed in

```
>>> x = raw_input("Tell me your name: ")
Tell me your name: Loren
>>> print(x)
Loren
>>>
```

type()

- Lets you know what type of an object an item is. This is important so we know what we can and can't do with an object
- Very useful when debugging

```
>>> x = raw_input("Tell me your name: ")
Tell me your name: Loren
>>> print(x)
Loren
>>> type(x)
<type 'str'>
>>>
```

OUR FIRST PROGRAM, AWW!

- Create a program that greets you by name and tells you a little something.
- Need:
  - Text editor to write code in (notepad, notepad++, sublime, atom, gedit, etc)
  - Python installed on your computer
  - How to navigate to a directory (cd)

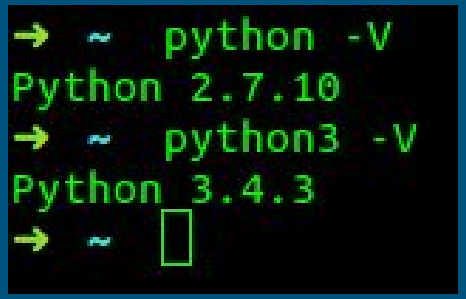
LET'S DO THIS!

Do you have Python?

- Mac/Linux users should have python already installed to check go to your command line (anywhere) and type:

  python -V or **python3 -V** (for those on who have python3 installed)

- Windows users download python and go to your system variables and append **C:/Python27** to your PATH variable (this allows you to call Python from the command line/Powershell.



Command line Basics:

- **cd** (change directory)
- **ls** (list stuff -- lists your directory contents)
- **pwd** (print working directory -- tells you where you are on your computer)

- Create a folder called Python_Workshop wherever you want (we will write our scripts here). To make it easy let's put it in our Desktop

Python files (.py)

- Create a file called **hello_me.py** (in the folder Python_Workshop)
  - .py is how Python knows what files have python code and can be executed

- Program:
  - Asks for my name
  - Tells me:

    Hello <name>, I hope you are enjoying python so far!

hello_me.py

```
1  #Instruction:  Hello <name>, I hope you are enjoying python so far!
2
3  user_name = raw_input("Hi there! Tell me your name: ")
4  #print user_name
5
6  greeting = "Hello " + user_name + " I hope you are enjoying python so far!"
7  print greeting
```

Notice the **comments** (the greyed out material)

- These are helpful when debugging and making notes for yourself
- They are lines in code that are hidden from Python and thus won't run

python

Running a Python program

- Go to the directory in which you have your .py file
- Type: **python hello_me.py**
  - Remember our file is called hello_me.py, if it was called x.py we'd run **python x.py**

```
➜  Python_Workshop  ls
hello_me.py
➜  Python_Workshop  python hello_me.py
Hi there! Tell me your name: Loren
Hello Loren I hope you are enjoying python so far!
➜  Python_Workshop
```

YOU JUST RAN YOUR FIRST PYTHON PROGRAM! AWESOME!

Arithmetic Operators

- Addition
- Subtraction
- Multiplication
- Division
- Exponential
- Modulo (remainder)
  - 20 / 3 --> 3*6 = 18 --> goes in 3 times remainder 2

```
>>> a = 10
>>> b = 20
>>> c = 3
>>>
>>> a + b
30
>>> b - a
10
>>> b/a
2
>>> b/c
6
>>> a**c
1000
>>> b*c
60
>>> b%c
2
>>> b%a
0
>>>
```

python

Comparison Operators: True or False

- **==** (equal)
- **!=** or **<>**(not equal)
- > (greater than)
- < (less than)
- >= (greater than or equal to)
- <= (less than or equal to)

```
>>> a = 1
>>> b = 2
>>> a == b
False
>>> a != b
True
>>> a <> b
True
>>> a > b
False
>>> a >= b
False
>>> a < b
True
>>> a <= b
True
>>>
```

python

Conditionals/Control Flow

- if/elif/else
- Evaluates to True for something to happen
- Needs to be written in order of logic (computers know nothing, you gotta spoon feed them logic)
- Indent 4 spaces

```
>>> x = 10
>>> if x > 25:
...     print 'x > 25'
... elif x > 15:
...     print 'x > 15'
... else:
...     print 'x < 15'
...
x < 15
>>>
```

Loops: While & For

- While loops: repeat a set of instructions while a condition is True.
  - Need condition to sometimes evaluate to False otherwise you get an infinite loop
- For loops: don’t require a condition, repeat instructions for some finite number of items.
  - Used for sequence/collection data like strings, arrays, etc
  - Used for iterating through (going through) objects

```
>>> count = 0
>>> condition = True
>>> while condition:
...         print True
...         count = count + 1
...         print count
...         condition = False
...
True
1
>>>
```

while loop ^

```
>>> name = 'Charles'
>>> for i in name:
...         print i
...
C
h
a
r
l
e
s
>>>
```

for loop >

Data Structures: Lists & Dictionaries

- Lists and Dictionaries:
  - Collection types
  - Hold a variety of object types (int, string, float, lists, dictionaries)
  - Mutable (can be changed)
  - Iterable (step through it)
- Lists are ordered (they stay in the order you put them in)
- Dictionaries are not ordered
  - Set of key (unique), value pairs. Access values through the key.
  - Like a phone numbers, name to numbers.

```
>>> mylist = []
>>> mylist.append(5)
>>> mylist
[5]
>>> mylist.append(4)
>>> mylist.append(3)
>>> mylist.append(2)
>>> mylist.append(1)
>>> mylist
[5, 4, 3, 2, 1]
>>> mylist[0]
5
>>> mylist[1]
4
>>> mylist[1] = 9
>>> mylist
[5, 9, 3, 2, 1]
>>> for idx in mylist:
...     print idx
...
5
9
3
2
1
```

< list

dictionary >

```
>>> phonebook = {}
>>> phonebook['Loren'] = '555-5555'
>>> phonebook['Charles'] = '555-5555'
>>> phonebook
{'Charles': '555-5555', 'Loren': '555-5555'}
>>> phonebook['Charles']
'555-5555'
>>> for name, number in phonebook.items():
...     print 'Name: '+name+ ' Phone: '+number
...
Name: Charles Phone: 555-5555
Name: Loren Phone: 555-5555
>>> for keys in phonebook:
...     print keys
...
Charles
Loren
>>> for keys in phonebook.keys():
...     print keys
...
Charles
Loren
>>> for values in phonebook.values():
...     print values
...
555-5555
555-5555
>>>
```

Functions

- Can have 0,1 or more parameters
- You define it however you want
- Instead of printing, you **return** (this stores that value to the function) to get something back from the function
- start out with:

def <function name> ():

- remember to indent 4 spaces!

```
>>> def say_hello():
...     return 'HELLO THERE'
...
>>> def greet_you(name):
...     return 'Hello ' + name
...
>>> say_hello()
'HELLO THERE'
>>> greet_you('Charles')
'Hello Charles'
>>>
```

python

Let’s create a program, young padawons

- Create a friendly **Guess the Number** game
  - Asks you for your name and uses it in the game
  - Tells you that you need to go higher or lower when you guess wrong
  - Max of 5 guesses per game and then tells you that you lost
  - Tells you that you won when you won

Thanks for learning Python with me!

Hope you enjoyed a taste of being a parseltongue!
Hiss!

Any questions?

Feel free to reach out: lm.velasquez12@gmail.com
or tweet me at @rebeldroid12
- Prof Loren