DOC SOC presents

Intermediate Python

What are our goals?:

- Learn **Python syntax** and **features**
- Understand why Python is so **powerful** and **popular** for so many different uses

Why are DOC SOC doing this?:

Because we're nice people, who want to provide others with the same opportunity to learn to code well in the same way we were

Who am I?:

- Jack
- 3rd year JMC
- DoCSoc Academic Events Director
- Worked as Software Engineer / Data Science Intern before, going to JPMorgan Tech this summer

What concepts should you know already?

- Data types and structures
- Control flow (If statements / conditionals)
- Iteration (Loops)
- Functions

Don’t worry if you’re rusty, we’ll do a recap!

What will you learn?

- Python syntax and features
- Object-Oriented Programming and good practices
- Scientific Computation Libraries such as Pandas, Numpy
- Applications of Python:
    - Statistics
    - Sciences
    - Machine Learning / Data Science
    - Web Servers, tools, etc
- Beyond the first/second week, you can vote on what's next

How will you learn?

- Lecture (Theory) Content
- Demos, Live Coding
- Use both Interpreter and write Python programs
- Exercises to do
- Menti Quizzes
- Questions are encouraged + Google yourself!

What will we need?

- A laptop with
  - Python 3.x (preferably 3.6 or 3.7)
  - A text editor
- Some dedication and effort!

Some **FAQ** before we begin

- I’ve done ___ before, but not Python?
- I know concept X but not sure about Y?
- Can I use an **IDE**?
- What the hell is going on in the demo?

And lastly……

A little bit about yourselves :)

Let’s begin!

We're going to begin with a recap of all the basics / as a crash course for those of you coming from other programming languages

Basic Data Types

What are some basic data types?

Basic Data Types

- Integer
- Float/Double
- Char
- String
- Bool
- Null

Basic Data Types

```
x = 5

y = 8

samplebool = True

message = "This is a string!"

z = 5.5
```

Basic Data Types

```
int x = 0;
x = x + 5;
int y;
y = age 10;
string message;
message = “A message”;
```

In other languages, you need to **declare the type** of the variable first. This is not the case for Python, as it is **dynamically typed**.

Dynamic Typing AKA Duck Typing

The type of variables are not checked until they are used in the program, at which either it works because it's the correct type, or all hell breaks loose because it's not

Basic Data Types

```
x = 5
y = 2
z = 3.0
x + y
x + 2.2
x / y
10 / 2
```

What are the types of these (if they work at all)?

Basic Data Types

```
x = 5
y = 2
z = 3.0
x + y
x + 2.2
x / y
10 / 2
```

What are the types of these (if they work at all)?

Type Casting

(Attempting to) Force a variable into another type

Basic Data Types

```
True and False
True or False
False + False
False - False
```

What are the types of these (if they work at all)?

Basic Data Types

```
True and False
True or False
False + False
False - False
```

What are the types of these (if they work at all)?

What happens when you try casting between ints, floats, and bools?

Basic Data Types

```
"message1" + "message2"
"message1" - "message2"
"message1" + ‘message2’
"message1" + 5
"message1" + True
"message1" * 3
"c"
‘c’
```

What are the types of these (if they work at all)?

Basic Data Types

```
"message1" + "message2"
"message1" - "message2"
"message1" + ‘message2’
"message1" + 5
"message1" + True
"message1" * 3
"c"
‘c’
```

What are the types of these (if they work at all)?

Play around with casting these

Basic Data Types

- Number (Integer or Float)
- String
- ~~Char~~
- Bool
- Null (None)

Python Data Structures

- Lists
- Dictionary
- Set

Lists

- Similar to arrays in other languages
- No fixed size, dynamically grows
- 0-indexed
- Item can be different types
- Can be nested, and “sliced”

Lists

```
l = [1,2,3,4]
l.append(5)
l.append("Six")
l.append(True)
l[2]
l[0:4]
```

Generating Lists

```
list_1 = list(range(5))
list_1 = [i for i in range(5)]
list_2 = list(range(5, 10))
list_2 = [i for i in range(5,10)]

summed_list = [x + y for x, y in zip(list_1, list_2)]
```

Generating Lists

```
list_1 = list(range(5))
list_1 = [i for i in range(5)]
list_2 = list(range(5, 10))
list_2 = [i for i in range(5,10)]

summed_list = [x + y for x, y in zip(list_1, list_2)]
```

Check out the actual documentation as well

Dictionaries

- Similar to (Hash)Maps in other languages
- Maps Keys to Values
- Not type-aware
- You can have dictionaries within dictionaries (within dictionaries.....)
- Similar to JSON

Dictionaries

```
released = {
        "iphone" : 2007,
        "iphone 3G" : 2008,
        "iphone 3GS" : 2009,
        "iphone 4" : 2010,
        "iphone 4S" : 2011,
        "iphone 5" : 2012
    }
released["iphone"]
```

Check out the actual documentation as well

Sets

- Pretty much the same as other languages
- Similar to lists. but
    - Duplicates are removed
    - No order
- Not type aware

Sets

```
myset = {1,2,3,4,5}
myset.add("Six")
```

Check out the actual documentation as well

Warning: Copying/Modifying structures/objects

"Objects" are referred to by reference, so always check what you are doing

Warning Example

```
l1 = [1,2,3,4,5]
l2 = l1
l2.append(6)
```

Control Flow

- If/else statements
- Loops

If statements

```
if True:
    print("Yay!")
```

If statements

```python
if True:
    print("Yay!")
elif False:
    print("Booooohooo")
```

If statements

```python
if 1:
    print("Yay!")
elif 0:
    print("Booohooo")
else:
    print("What's going on?")
```

The condition doesn’t have to be a bool itself!

While Loops

```
while True:
    print("This will go on forever!")
```

While a certain condition is true, do something

While Loops

```
i = 0
while i < 10:
    print(i)
    i += 1
```

While a certain condition is true, do something

While Loops

```
print("Marco....")

user_input = input()

while user_input != "Polo":
    print("Wrong Answer! Try Again!")
    print("Marco....")
    user_input = input()

print("Yay!")
```

While Loops

```
print("Marco....")

while input() != "Polo":
    print("Wrong Answer! Try Again!")
    print("Marco....")

print("Yay!")
```

For loops

In other languages:

```
for (int i = 0; i < array.length; i++)
{
  System.out.println(array[i]);
}
```

For loops in Python:

“For an element x in <something>, do <this>”

For loops

```python
for x in [1,2,3,4,5]:
    print(x)

for name in ["John", "Paul", "George", "Ringo"]:
    print(name + " is in the Beatles.")

for i in range(0,10):
    print(i)
```

For loops

```
for x in [1,2,3,4,5]:
    print(x)

for name in ["John", "Paul", "George", "Ringo"]:
    print(name + " is in the Beatles.")

for i in range(0,10):
    print(i)
```

Note**: a for loop doesn't have to take in a list, it can take in any **iterator

Quick Exercise:

Using either a For or While loop, print the first 10 multiples of 5

Functions

- Abstract code away
- Takes input **(maybe)**
- Does something **(maybe)**
- Returns output **(maybe)**

Functions example

```
beatles = ["John", "Paul", "George", "Ringo"]

def greet(name):
    print("Hello " + name + ", how are you doing?")

for beatle in beatles:
    greet(beatle)
```

Functions

- Variables created inside are forgotten once the function is finished executing
- Python allows for "**default arguments**"
- Does not have to consistently return (or not return) the same type
- Can call itself **(recursion)**

More function examples

```python
def greet(name=None):
    if name is None:
        name = input()
    print("Hello " + name + ", how are you doing today?")

def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        return fib(n - 1) + fib(n - 2)
```

Quick Exercise:

Write a function that given a positive integer n, returns the sum of all the numbers from 1 to n

Questions so far?

Break time!

Objects and Classes

What objects/classes do we already know?

Objects/Classes

- Encapsulates data, and functions centered around them (methods)
- Reduces code duplication
- With good design, can be reused, extended etc.
- Allows a “program” to be split into smaller components:
  - Easier to think about and write
  - Work can be split up easily

Example 1

- You want a class to represent a Person
    - What data do you want to have about a Person?
    - What methods should a Person be capable of?

Example 1

- You want a class to represent a Person
    - What data do you want to have about a Person?
        - Firstname, Lastname, Age……etc.
    - What methods should a Person be capable of?
        - Greet, Eat, Grow…..etc.

Example 1

By writing Person class, you can now instantiate as many Person objects as you want

Person Class

```
class Person:

    def __init__(self, name):
        self.name = name
```

Person Class

```
class Person:

    def __init__(self, name):
        self.name = name

    def greet(self):
        print("Hello! My name is", self.name)
```

Person Class

```
john = Person("John")
john.greet()
```

Person Class

```
class Person:
```

Constructor

```
    def __init__(self, name):
        self.name = name
```

Field

```
    def greet(self):
```

Method

```
        print("Hello! My name is", self.name)
```

Class-level Properties

```
class Person:

    race = "Human"

    def __init__(self, name):
        self.name = name
```

Example 1

What else can we add to the Person class?

Default Object Methods

All Objects have default methods for certain tasks, for example, print(a) will call a.__str__(). See documentation for more details.

Person class

- We created a single class to encapsulate a Person, so we can create multiple people.
- Each Person object
  - can be created simply and easily
  - has the same functionality
- You should now **begin** to see why Object-Oriented Programming is useful

Example 2

- You want a simple, command line rock-paper-scissors game
- How would you do this?

Example 2

- You want a simple, command line rock-paper-scissors game
- How would you do this?
  - **Option 1: Create a RockPaperScissors class to encapsulate the game**

Example 2

- You want a simple, command line rock-paper-scissors game
- How would you do this?
    - Option 1: Create a RockPaperScissors class to encapsulate the game
    - If you were to host a board games night, what would you need?

Example 2

- You want a simple, command line rock-paper-scissors game
- How would you do this?
    - Option 2: Create a RockPaperScissors game class, as well as a Player class.
    - This allows us to separate (and possibly later reuse) the logic

Example 2

- You want a simple, command line rock-paper-scissors game
- How would you do this?
  - Option 2: Create a RockPaperScissors game class, as well as a Player class.
  - This allows us to separate (and possibly later reuse) the logic

How should do the two classes interact with each other now?

Interfaces (AKA an **API**)

A set of rules which define how a component should interact with another

Interfaces

- A very general programming concept - not Python specific
  - Other languages have features to **enforce** an interface
  - You will also hear about **Web APIs**
- An interface isn't good or bad by itself - it depends on the **context and use cases**
- Abstracts away the usage from the implementation
- **"Design"** of a program/application

Our game interface

RockPaperScissors class:

- moves() method to give a list of possible moves (Rock, Paper, Scissors)
- play() method to play the game, and prints out the winner

Our game interface

RockPaperScissors class:

- moves() method to give a list of possible moves (Rock, Paper, Scissors)
- play() method to play the game, and prints out the winner

HumanPlayer class:

- pick_action() method to select a move to play

Our game interface

RockPaperScissors class:

- moves() method to give a list of possible moves (Rock, Paper, Scissors)
- play() method to play the game, and prints out the winner

HumanPlayer class:

- pick_action() method to select a move to play

Once this has been agreed, we can now get started!

What if we want to add an AI Player class?

What design decisions have we made?

- Think about type signatures of methods
- How we create each object and use them
- Would we change our design if our use case was different?

What design decisions have we made?

- Think about type signatures of methods
- How we create each object and use them
- Would we change our design if our use case was different?
- **What bad decisions could we have made?**

What if we want a Tic-Tac-Toe game now?

What could we reuse?

What would we need to add/change?

That's it for this week!

- docsoc.co.uk/education for all resources
- Next week(?), we will look at
  - a little more into classes (inheritance)
  - exceptions
  - importing libraries and project structure
  - numpy, and pandas, graph plotting, etc

Also, if you're interested....

DOC SOC

Learn To Build Your Own Personal Website!

Huxley 308 | Tuesday 12th February | 6:30pm - 7:30pm