Introduction to OOP

February 25, 2017

0.1 Introduction to Object Oriented Programming with Python

This prepares you for a 2 day challenge. The goals:

1. Be able to write Python code;
2. Understand the ideas behind object oriented programming.

0.1.1 Basic Python

1. Variables;
2. Flow Control;
3. Functions;
4. Data structure.

0.1.2 Basic Object Oriented Programming

1. Classes;
2. Attributes;
3. Methods;
4. Inheritance.

0.1.3 The Challenge

1. End of the day Thursday: Initial Feedback;
2. End of the day Friday: results.

0.2 Overview of Python

What is Python?

Python is a programming language. There are various other programming languages:

Java C C++ Ruby VBA and many more. A programming language allows you to write a program which is a sequence of instructions that specifies how to perform a computation.

When writing a program you need two things:

Something to save the code (a text editor for example) Something to run the code We will be using a combination of these 2 things called notebooks.

0.3 Installing Python

There are various distributions of Python, we will use Anaconda which comes packaged with a variety of other useful tools (including the notebooks I mentioned above).

To install it on your personal machine follow these steps:

Go to this webpage: https://www.continuum.io/downloads.

Identify and download the version of **Python 3** for your operating system (Windows, Mac OSX, Linux). Run the installer. We will use a Jupyter notebook which runs in your browser. To open a local server find the Continuum navigator and click on Jupyter. You do not need to be connected to the internet to use this.

0.4 Interacting with Python

Once you have installed Anaconda, you will now have Python on your machine. You can interact with Python in multiple ways:

0.4.1 The Python shell

- On Windows open a "Command Prompt":
- On Mac OS or Linux open a "Terminal":

This is a simple utility that allows you to give commands to your computer. In there type:

```
python
```

This should then look something like:

```
Python 3.5.2 |Anaconda 4.2.0 (64-bit)| (default, Jul  2 2016, 17:53:06)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-1)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

The >>> is a prompt for you to type a command. Let us carry out a straightforward addition:

```
>>> 2 + 2
```

and press ENTER:

```
>>> 2 + 2
4
```

This is not a very efficient and practical way of using Python. We will instead learn to use a Jupyter notebook.

0.4.2 Jupyter notebooks

If you are still in the Python shell (with a >>> waiting for you to give a command) then type:

```
>>> exit()
```

This will leave the prompt and you will now be back at the Command Prompt or the Terminal window. Here simply type:

```
jupyter notebook
```

After a little moment this will open a Jupyter notebook page in a browser. Note that this is all just running on your computer and you do not need to be connected to the internet to use a Jupyter notebook.

Click on `New` and create a notebook: we can start to write code now.

0.4.3 Basic Python

Write `2 + 2` in a cell and press `shift + Enter`:

```
In [2]: 2 + 2

Out[2]: 4
```

0.5 Variables

0.5.1 Character variables:

```
In [3]: string = "Hello world"
        string

Out[3]: 'Hello world'
```

0.5.2 Numeric variables:

```
In [4]: num_1 = 2
        num_2 = 3.5
        num_1 + num_2

Out[4]: 5.5
```

0.5.3 String manipulation

```
In [7]: #We define a variable called String
        # (note that # allows me to comment my code)
        string = "My name is Vince"
        #Let's get the 5th letter of String
        # (Note that Python starts counting at 0):
        string[4]

Out[7]: 'a'
```

```
In [8]: string[1:4]

Out[8]: 'y n'

In [11]: index_of_v = string.index("V")
         index_of_v

Out[11]: 11

In [12]: string[index_of_v:]
         string[:index_of_v]

Out[12]: 'My name is '
```

0.5.4 Numeric manipulation

```
In [13]: num = 3

         # The following two lines are equivalent
         num = num + 1
         num += 1
         num

Out[13]: 5

In [14]: num -= 2
         num *= 3
         num **= 2
         num

Out[14]: 81
```

0.6 Flow control

- In Python indentation is important!
- In all languages indentation is good practice, in Python it is a requirement.

0.6.1 If statements

```
In [23]: n = 11
         if n <= 5:
             value = 1
         elif n % 2 == 0:
             value = 2
         else:
             value = 3
         value

Out[23]: 3
```

0.6.2 While loops

```
In [25]: count = 0
         total = 0
         while count < 10:
             count += 1
             total += count
         total

Out[25]: 55
```

0.6.3 For loops

```
In [26]: for i in [1, 2, 3, 4]:
             print(i)

1
2
3
4
```

```
In [30]: for subject in ["Queueing Theory", "Game Theory",
                         "Inventory Theory", "Reliability Theory",
                         "Project Management", "Decision Analysis"]:
             if "Theory" in subject:
                 print(subject)

Queueing Theory
Game Theory
Inventory Theory
Reliability Theory
```

0.7 Functions

```
In [31]: #To create a function we use the 'def' statement:
         def hi():
             """
             This function simply prints a short statement.

             This is a shorter way of writing documentation,
             it is good practice to always include a
             description of what a function does.
             """
             print("Hello everybody!")

         hi()
```

```
Hello everybody!
```

```
In [34]: def fibonacci(n):
             """
             This returns the nth Fibonacci number.
             """
             if n == 0:
                 return 0
             if n == 1:
                 return 1
             return fibonacci(n - 1) + fibonacci(n - 2)
```

```
In [35]: fibonacci(5)

Out[35]: 5
```

0.8 Data structures: Lists

```
In [38]: my_list = list(range(6))
         my_list[0]

Out[38]: 0
```

```
In [39]: my_list.append(100)
         my_list

Out[39]: [0, 1, 2, 3, 4, 5, 100]
```

1 Object Oriented Programming

This is similar to cellular structure:
We can create "things" with:

- attributes: things those "things" have;
- methods: things those "things" can do.

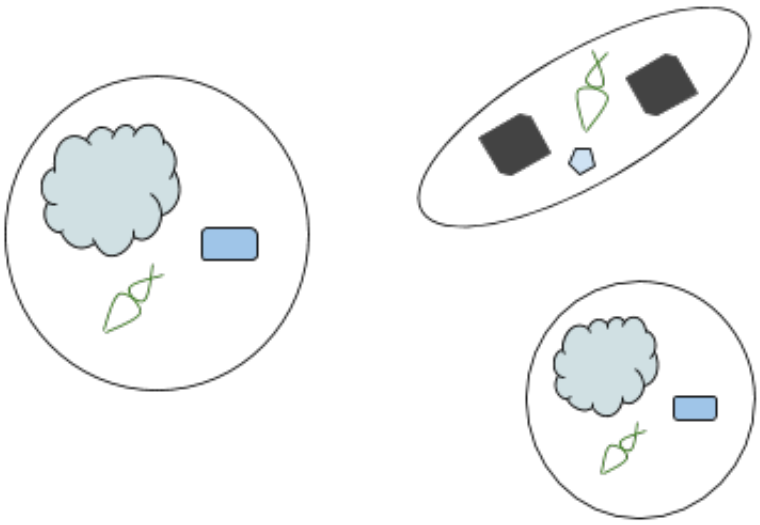
1.1 Defining a class

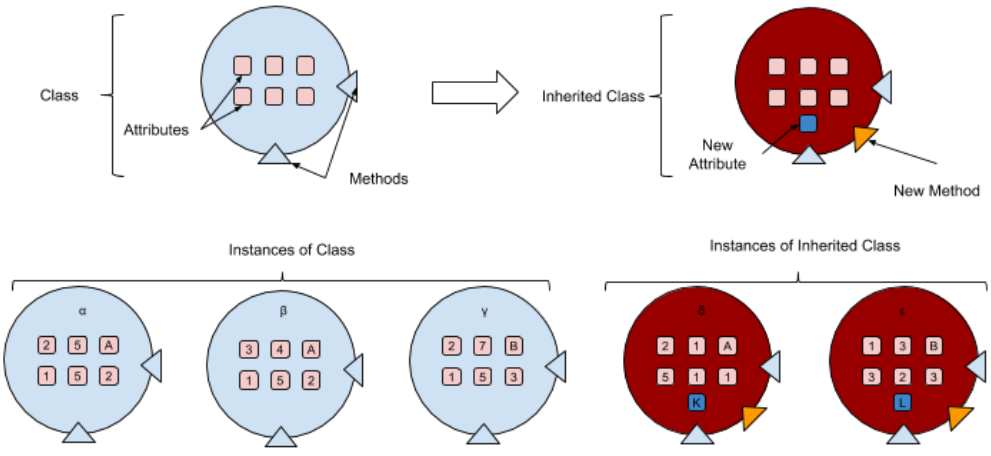
```
In [2]: class Student():
            """We can create a simple empty class.

            This is a set of rules that says what a student is.
            """
```

```
In [3]: vince = Student()  # Creating an instance
        vince

Out[3]: <__main__.Student at 0x7fef0874db70>
```




Class
Attributes
Methods
Inherited Class
New
Attribute
New Method
Instances of Class
Instances of Inherited Class
α
β
γ
δ
ε
2 5 A
1 5 2
3 4 A
1 5 2
2 7 B
1 5 3
2 1 A
5 1 1
K
1 3 B
3 2 3
L

```
In [4]: zoe = Student()  # Creating a different instance
        zoe

Out[4]: <__main__.Student at 0x7fef0874d8d0>
```

1.2 Attributes

```
In [8]: class Student():
            courses = ["Biology", "Mathematics", "English"]
            age = 5
            gender = "Male"
        #Let us now create Vince again:
        vince = Student()
```

Accessing these attributes:

```
In [6]: vince.courses

Out[6]: ['Biology', 'Mathematics', 'English']

In [7]: vince.age

Out[7]: 5

In [9]: vince.gender

Out[9]: 'Male'
```

We can manipulate these attributes just like **any other** python variable:

```
In [10]: vince.courses.append("Photography")
         vince.courses

Out[10]: ['Biology', 'Mathematics', 'English', 'Photography']

In [11]: vince.age = 28
         vince.age

Out[11]: 28

In [12]: vince.gender = "M"
         vince.gender

Out[12]: 'M'
```

1.3 Methods

```
In [15]: class Student():
             courses = ["Biology", "Mathematics", "English"]
             age = 5
             sex = "Male"

             def have_a_birthday(self):
                 """This method increments the age of our instance."""
                 self.age += 1
```

```
In [16]: vince = Student()
         vince.age
```

```
Out[16]: 5
```

```
In [17]: vince.have_a_birthday()
         vince.age
```

```
Out[17]: 6
```

1.4 The `__init__` method

```
In [18]: class Student():
             def __init__(self, courses, age, sex):
                 """
                 What the class should do when it
                 is used to create an instance
                 """
                 self.courses = courses
                 self.age = age
                 self.sex = sex

             def have_a_birthday(self):
                 self.age += 1
```

```
In [19]: vince = Student(["Biology","Math"],28,"Male")
         vince.courses, vince.age, vince.sex
```

```
Out[19]: (['Biology', 'Math'], 28, 'Male')
```

1.5 Inheritance

We can use a class to create new classes:

```
In [21]: class Math_Student(Student):
             """
             A Math student: behaves exactly like a Student
             but also has a favourite class attribute.
             """
             favourite_class = "Mathematics"
```

```
In [23]: becky = Math_Student(["Mathematics", "Biology"], 29, "Female")
         becky.courses, becky.age, becky.sex, becky.favourite_class

Out[23]: (['Mathematics', 'Biology'], 29, 'Female', 'Mathematics')

In [25]: #This class has the methods of the parent class:
         becky.have_a_birthday()
         becky.age

Out[25]: 31
```

1.6 Summary

- Classes
- Attributes
- Methods
- Inheritance

1.7 Advantages

- Simplicity
- Modularity
- Modifiability
- Extensibility
- Re-usability

2 Libraries

There are a number of built in libraries that extend what Python can do.

```
In [26]: import random
         random.seed(0)
         random.random()

Out[26]: 0.8444218515250481

In [27]: import math
         math.cos(math.pi)

Out[27]: -1.0
```

There are also a number of external libraries. This is one of the huge strengths of Python. Some of these come with Anaconda (the distribution of Python I recommend):

```
In [28]: import sympy  # symbolic mathematics
         x = sympy.symbols('x')
         sympy.diff(x ** 2, x)

Out[28]: 2*x
```

```
In [29]: import numpy  # Fast numeric computations
         A = numpy.matrix([[0, 2], [1, 2]])
         B = numpy.matrix([[3, 1], [1, -2]])
         A * B

Out[29]: matrix([[ 2, -4],
                 [ 5, -3]])
```

There are also a number of libraries outside of anaconda that are also very powerful. Some examples include:

- Ciw: modeling of queues;
- Axelrod: game theory.
- tqdm: adding progress bars to your code :)
- The list is very large...

To install these libraries from the internet you can open a command prompt (Windows) or a terminal (Mac OSX) and type:

```
pip install <library-name>
```

3 Further resources

There are a number of wonderful resources for learning Python. Here are some that I have made:

- My first year computing for Mathematics course: http://vknight.org/cfm/
- A short set of notebooks that used to teach Mathematics how to use Python: https://github.com/drvinceknight/Python-Mathematics-Handbook