Lecture 01: Introduction to Python

February 6, 2023

1 Introduction to Python

Content adopted from Pierian Data

2 What is Python and why use it

- **high-level**,
- **interpreted**,
- **general-purpose** programming language that is used for a wide range of applications.
- It is easy to learn, powerful, and
- has a large community of users.

3 Why is it called Python?

When he began implementing Python, Guido van Rossum was also reading the published scripts from "Monty Python's Flying Circus", a BBC comedy series from the 1970s. Van Rossum thought he needed a name that was short, unique, and slightly mysterious, so he decided to call the language Python.

–General Python FAQ

4 First Things First

As with any programming course, here is the `Hello World!` in Python.

```
[1]: print ("Hello World!")
```

```
Hello World!
```

5 Variable

A variable is a named storage location used to hold a value. The value of a variable can be changed and it can be used in expressions and operations

6 Variable Assignment

- Rules for variable names
- names can not start with a number
- names can not contain spaces, use _ intead
- names can not contain any of these symbols: `'",<>/?|\!@#%^&*~-+`
- accoding to Style Guide for Python Code (PEP8), it's considered best practice that names are lowercase with underscores
- avoid using Python built-in keywords like `list` and `str`
- avoid using the single characters `l` (lowercase letter el), `O` (uppercase letter oh) and `I` (uppercase letter eye) as they can be confused with 1 and 0

7 Dynamic Typing

Python uses *dynamic typing*, meaning you can reassign variables to different data types. This makes Python very flexible in assigning data types; it differs from other languages that are statically typed.

[2]:
```
my_cat = 2
my_cat
```

[2]:
```
2
```

[3]:
```
my_cat = ['Basbousa', 'Lucy']
my_cat
```

[3]:
```
['Basbousa', 'Lucy']
```

8 Pros and Cons of Dynamic Typing

- Pros of Dynamic Typing
  - very easy to work with
  - faster development time
- Cons of Dynamic Typing
  - may result in unexpected bugs!

9 Assigning Variables

Variable assignment follows `name = object`, where a single equals sign `=` is an assignment operator

[4]:
```
a = 5
a
```

[4]:
```
5
```

10 Reassigning Variables

Python lets you reassign variables with a reference to the same object.

```
[5]: a = a + 10
     a
```

```
[5]: 15
```

There's actually a shortcut for this. Python lets you add, subtract, multiply and divide numbers with reassignment using `+=`, `-=`, `*=`, and `/=`.

```
[6]: a += 10
     a
```

```
[6]: 25
```

```
[7]: a *= 2
     a
```

```
[7]: 50
```

11 Determining variable type with `type()`

You can check what type of object is assigned to a variable using Python's built-in type() function. Common data types include:

```
[8]: type(a)
```

```
[8]: int
```

12 Numbers

Basically there are two types of numbers: - 2 is interger `int` - 2.0 is floating point `float`

Example	Number Type
1,2,-5,1000	Integers
1.2,-0.5,2e2,3E2	Floating point

```
[9]: type(2)
```

```
[9]: int
```

```
[10]: type(2.0)
```

```
[10]: float
```

Basic Arithmetic 1/2

```
[11]: 2+1 # Addition
```

```
[11]: 3
```

```
[12]: 2-1 # Subtraction
```

```
[12]: 1
```

```
[13]: 2*2 # Multiplication
```

```
[13]: 4
```

```
[14]: 3/2 # Division
```

```
[14]: 1.5
```

Basic Arithmetic 2/2

```
[15]: 3//2 # Floor division (It returns the result of division rounded down to the␣
       ↪nearest integer)
```

```
[15]: 1
```

```
[16]: 2**3 # Powers
```

```
[16]: 8
```

Question: how to calculate the sequare root of 16?

13 Order of Operations

```
[17]: 2 + 10 * 10 + 3
```

```
[17]: 105
```

```
[18]: (2+10) * (10+3)
```

```
[18]: 156
```

14 Strings

Strings in Python are **text**, such as names, stored as a sequence or a list of characters. For example, Python understands the string `'AUC'` to be a sequence of letters in a specific order. This means we will be able to use indexing to grab particular letters (like the first letter `A`, or the last letter `C`).

15 Creating a String

To create a string in Python you need to use either single quotes `'` or double quotes `"`.

```
[19]: 'Hello'
```

```
[19]: 'Hello'
```

```
[20]: 'Hello World!'
```

```
[20]: 'Hello World!'
```

```
[21]: "This is also a string"
```

```
[21]: 'This is also a string'
```

```
[22]: 'I'm using single quotes, but this will create an error'
```

```
  Cell In[22], line 1
    'I'm using single quotes, but this will create an error'
       ^
SyntaxError: invalid syntax
```

```
[25]: 'Now I\'m ready to use the single quotes inside a string!' # Using escape␣
      ↪character
```

```
[25]: "Now I'm ready to use the single quotes inside a string!"
```

```
[26]: "Now I'm ready to use the single quotes inside a string!" # Using double quotes
```

```
[26]: "Now I'm ready to use the single quotes inside a string!"
```

16 Printing a String

Using Jupyter notebook with just a string in a cell will automatically output strings, but the correct way to display strings in your output is by using a `print` function.

[27]:
```
'Hello World'
```

[27]:
```
'Hello World'
```

[28]:
```
'Hello World 1'
'Hello World 2'
```

[28]:
```
'Hello World 2'
```

[29]:
```
print('Hello World 1')
print('Hello World 2')
```

```
Hello World 1
Hello World 2
```

[32]:
```
print('Hello World 1\nHello World 2') # using \n for new line
```

```
Hello World 1
Hello World 2
```

[ ]:
```
```