Little Intro

- Raza Ul Mustafa, PhD - Computer Engineering
- Research areas: NLP, Video, Quality of Experience, Quality of Service, Machine & Deep Learning
- Publications - 20 + → Languages used in my papers: Python :)
- Main Courses of Interests: Programming
  - Python, Web-programming, C/C++, Linux
- Worked & Working on Research Projects: 3+

Books

- The primary text for this course is a freely available online textbook called Think Python: How to Think like a Computer Scientist.
- We will be using the interactive version of the text that is available through Runestone Academy at https://runestone.academy.

Outline of course - Main topics

1. Variables & Operations
2. Strings & Numbers
3. Functions
4. Conditions / If/Elif/Else
5. Iterations – Very Important
6. Lists, Dictionaries, Tuples – Important in Python
7. Recursion, Sorting
8. OOP – Very important in all languages

Chapter 1

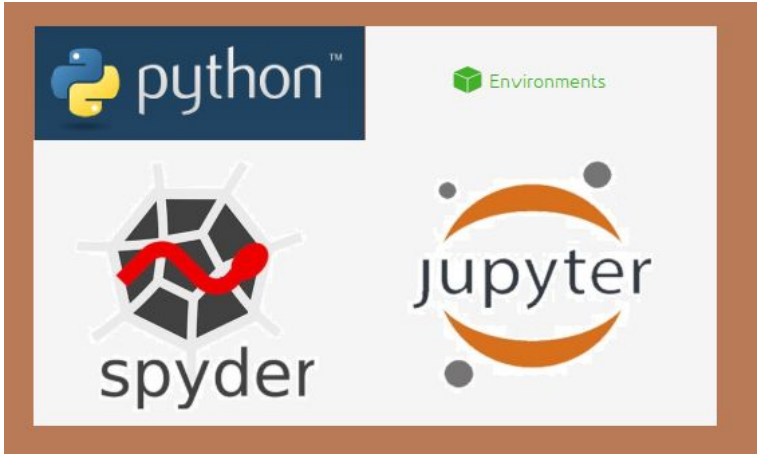
Software to Install

- Anaconda & Spyder

Windows Installation - https://docs.anaconda.com/free/anaconda/install/windows/

Mac Installation - https://docs.anaconda.com/free/anaconda/install/mac-os/

Linux - https://docs.anaconda.com/free/anaconda/install/linux/



First program

```
print("Hello, World!")
```

The way of program

- The single most important skill for a computer scientist is **problem solving**
- **A program is a sequence of instructions that specifies how to perform a computation.**

Algorithms

- If problem solving is a central part of computer science, then the solutions that you create through the problem solving process are also important.
- In computer science, we refer to these solutions as algorithms.
- **An algorithm is a step by step list of instructions that if followed exactly will solve the problem under consideration.**

Our goal in computer science is to take a problem and develop an algorithm that can serve as a general solution

The python programming languages

Python is an example of a high-level language; other high-level languages you might have heard of are **C++, PHP, and Java.**

High Level Languages: They are very easy to understand, i.e., resembles natural language or mathematical notation and is designed to reflect the requirements of a problem

Low Level Languages: They are also called machine-level languages. 0101

Advantages of high level languages

- First, it is much easier to program in a high-level language.
- Programs written in a high-level language take less time to write, they are shorter and easier to read, and they are more likely to be correct.
- High-level languages are portable, meaning that they can run on different kinds of computers with few or no modifications.

Then who process the codes?

Two kinds of programs process high-level languages into low-level languages: interpreters and compilers.

- Interpreters: An interpreter reads a high-level program and executes it, meaning that it does what the program says.
- Compiler: A compiler reads the program and translates it completely before the program starts running.
  - In this case, the high-level program is called the source code, and the translated program is called the object code or the executable.

Python Interpreter (1/2)

There are two ways to use the Python interpreter: shell mode and program mode.

In **shell mode**, you type Python expressions into the Python shell, and the interpreter immediately shows the result. The example below shows the Python shell at work.

```
$ python3
Python 3.2 (r32:88445, Mar 25 2011, 19:28:28)
[GCC 4.5.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> 2 + 3
5
>>>
```

Python Interpreter (2/2)

Program mode: Alternatively, you can write an entire program by placing lines of Python instructions in a file and then use the interpreter to execute the contents of the file as a whole. Such a file is often referred to as source code.

```
print("My first program adds two numbers, 2 and 3:")
print(2 + 3)
```

```
My first program adds two numbers, 2 and 3:
5
```

Saving and extension

By convention, files that contain Python programs have names that end with **.py**.

Following this convention will help your operating system and other programs identify a file as containing python code.

What is debugging

Programming is a complex process. Since it is done by human beings, errors may often occur.

Programming errors are called bugs and the process of tracking them down and correcting them is called debugging.

- **syntax errors,**
- **runtime errors,**
- **semantic errors.**

Syntax errors

Python can only execute a program if the program is syntactically correct; otherwise, the process fails and returns an error message.

Syntax refers to the structure of a program and the rules about that structure

Example – Next slide

Syntax errors example

```
#Missing Colon after a Block:

# Incorrect
if x == 5
    print("x is equal to 5")

# Correct
if x == 5:
    print("x is equal to 5")
```

```
  File "C:\Users\User\AppData\Local\Temp\ipykernel_2684\76303431.py", line 4
    if x == 5
             ^
SyntaxError: invalid syntax
```

Runtime errors

- The second type of error is a runtime error, so called because the error does not appear until you run the program.
- These errors are also called exceptions because they usually indicate that something exceptional (and bad) has happened.

Runtime errors example

In [2]:

```
# Division by Zero:
# Incorrect
result = 10 / 0

# Corrected
result = 10 / 2
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_2684\2919943611.py in <module>
      1 # Division by Zero:
      2 # Incorrect
----> 3 result = 10 / 0
      4
      5 # Corrected

ZeroDivisionError: division by zero
```

Semantic errors

- The third type of error is the semantic error.
- If there is a semantic error in your program, it will run successfully in the sense that the computer will not generate any error messages.
- However, your program will not do the right thing. It will do something else. Specifically, it will do what you told it to do.

Semantic errors example

In [6]:

```
# Incorrect Variable Reassignment:

# Incorrect
x = 5
x = x + 1
x = "hello"

# The above code changes the type of 'x' from an integer to a string

# Corrected
x = 5
x = x + 1
```

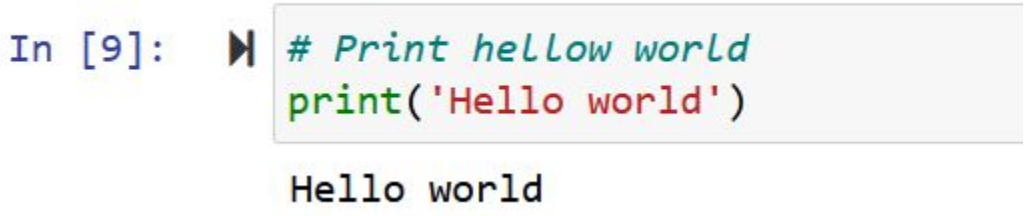
Experimental debugging

One of the most important skills you will acquire is debugging. Although it can be frustrating, debugging is one of the most intellectually rich, challenging, and interesting parts of programming.

Advice: Visualize

Comments in python

A comment in a computer program is text that is intended only for the human reader - it is completely ignored by the interpreter. In Python, the # token starts a comment. The rest of the line is ignored. Here is a new version of Hello, World!.



Chapter python notebook:

All material available on Canvas