# Data Manipulation in Python Part 1

**Ashish Rajendra Sai**

*Ph.D. Candidate - University of Limerick*

August 11, 2020


## Who am I?

***Ashish Rajendra Sai***

- Final year Ph.D. student at the University of Limerick under the supervision of Dr. Jim Buckley
- Doctoral Researcher in Lero: The Irish Software Research Centre
- Currently a Visiting Researcher at the University of California, Berkeley
- Long time interest in teaching programming to non-cs professionals and students:
    - Programming can be super fun and useful once we jump past some hoops

---

## How today might work

***Zoom Meeting:***

- I will present in approximately 15 - minute slots, over an hour.
- I would like to take all questions via the zoom 'chat'
    - What is a function? Why does it matter? ....
    - The importance of questions in this session.
- I will answer them in real-time, at the end of the 15 minutes or at the end of the session entirely (if I think they will be covered by something later)

## Welcome to today's class!

### *Today's Agenda*

1. Learn the basics of computer programming
2. What is Python and why should we care?
3. What do you need to write Python?
4. Basic building blocks of programming languages
5. Basics of reading CSV files in Python
6. Some best-practice for your code in Python

---

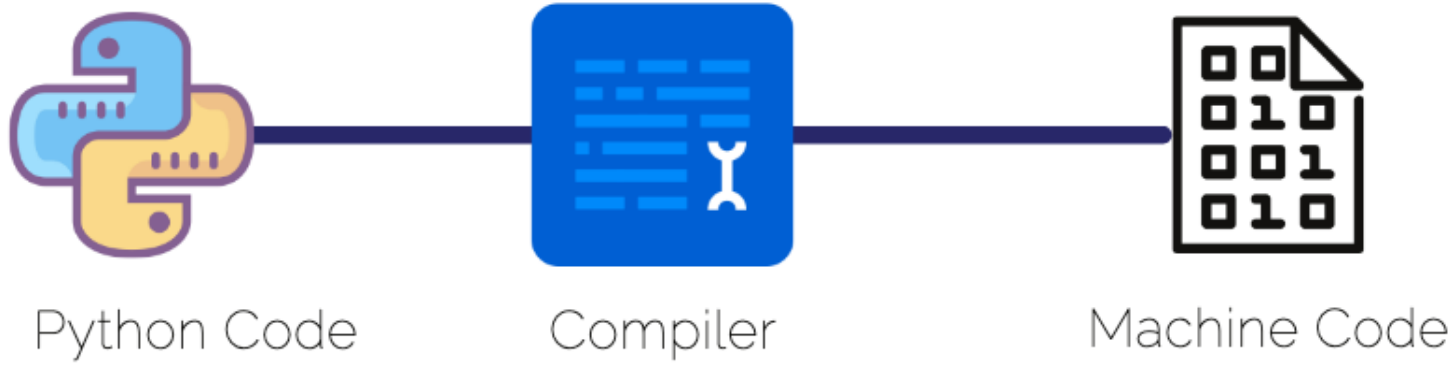
## Why learn to program?

- Traditional programs such as Excel are designed to perform some specific task (though they are very powerful tools at this).
    - Example: find people who have bought 10 masks in the last 6 days that live within 10 km of each other.
- Automation: If you have ever spent hours renaming files, Python can help you automate tedious tasks.
- Handling large data: no need to worry about Excel hanging/lagging.
- Predictive analysis: If you get comfortable with Python you can tab into the world of Machine Learning (We will touch the surface of ML in Python in the next lecture)

## Basics of Computer Programming

- Your computer only understands electronic pulses (On or Off) dictated by physical circuits in your machine
    - Boolean algebra plays a significant role in logical operations of computers (UCC was the academic home of George Boole)
- We moved from creating circuits to solve a problem (sum 8+9) to more general-purpose programable circuits (add a and b)
- These circuits use something called Machine Code that is specific to the hardware (not very intuitive)
- Nowadays we write programs in languages close to English, in CS lingo we call these High-Level Programming Languages.

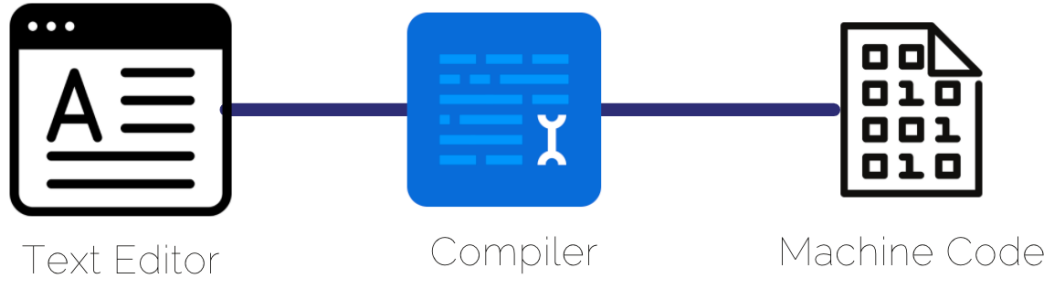
## High-Level Programming Language to Machine Code

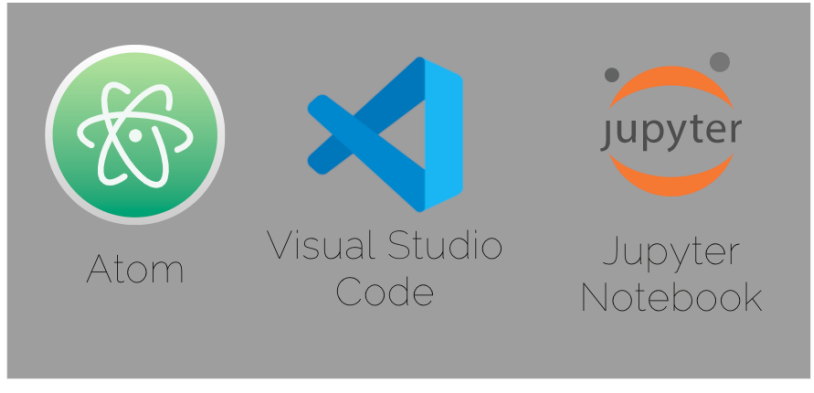


## What is Python and why should you care?

- Python is a High-Level programming language: Easy to read and write
- Widely used in both academia and industry
- General-purpose so you can use it to manipulate your data or create websites
- Good for machine learning or data analytics application due to the community
- Easy to set up and get started with

## What do you need to be able to code in Python?



- Compiler: https://www.python.org/downloads/ (https://www.python.org/downloads/)
- You can often run the Python machine binary using the text-editor (IDE) or use commands.

---

## What do you need to be able to code in Python?

- To make our life easy we are going to use free online hosted integrated development environments for Python.
- No need to download or set up anything: Be careful with your company policies, you do not want sensitive data on these third party computers.
- List of free online IDEs to get you started;
    - Google Colaboratory (Colab) Link: https://colab.research.google.com/ (https://colab.research.google.com/)
    - Microsoft Azure Notebooks Link: https://notebooks.azure.com/ (https://notebooks.azure.com/) (This is what we will use today)
    - CoCalc Link: https://cocalc.com/ (https://cocalc.com/) (Good but slow :( )
    - Kaggle Kernels Link: https://www.kaggle.com/ (https://www.kaggle.com/) (Mostly for data science nerds)
    - You can always use vanilla Jupyter Notebook on your own PC Link: https://jupyter.org/ (https://jupyter.org/)

## Basic Building Blocks of Programming Languages

- Let's start with the basics: Programming languages can often be used to perform basic mathematical operations

In [2]:
```
8+9
```

Out[2]:
```
17
```

- We can also make it more general-purpose: let's calculate the total marks of a student and the final percentage

In [4]:
```
# Student got 45/100 in Science, 50/100 in Maths, 52/100 in Biology and 80/100 in
 CS
45+50+52+80
```

Out[4]: 227

In [6]:
```
#  Here we calculate the percentage
(227/400)*100
```

Out[6]: 56.75

- We can solve this issue and make our calculation smarter

In [97]:
```
# Student got 45/100 in Science, 50/100 in Maths, 52/100 in Biology and 80/100 in
 CS
Science = 45
Maths = 50
Biology = 52
CS = 80
Total = Science + Maths + Biology + CS

print(Total)
```

```
227
```

In [18]:
```
Percentage = (Total/400)*100

print(Percentage)
```

```
56.75
```

- This is an important aspect of programming languages that make them powerful tools.
- We just created Variables and used them to make our calculation a bit versatile.

- There are different types of variables in programming languages
- So far we only created Integers and Floats (Decimal Point Values)
- The type of value that a program can understand is called a **Data Type**
    - Fun fact: What operations you perform on your data (variable) depends on your data type
- Some basic data types used in many programming languages:
    - Integers: 1,2,3,4 ....
    - Floating Point Numbers (Float): Decimal Point Number such as 1.2, 48.9999912, etc
    - Characters (Char): Alphabetic/Numeric characters such as A,a,B,b etc
    - String: Sequence of alphanumeric chars combined such as "Ashish"
    - Complex Number etc (Beyond the scope of this workshop but may be useful if you have a special type of data to deal with)

In [25]:
```
# Now we will see an example of these different data types

Name = "Luke"
Age = 24
Percentage = 84
Result = 'P'

# print(type(Name))
```

- That was all fun but what about calculating the percentage of Luke based on his subject results
    - Do we have to create one variable for each subject?
    - Certainly, this isn't simplifying our workflow
- This is where we can use a more complex data type called **List**:
    - Collection object types (a single variable that can store multiple data points)

In [30]:
```
ResultsBySubjects = [45,50,52,80]

# print(type(ResultsBySubjects))


# print(ResultsBySubjects[0])
# print(ResultsBySubjects[1])
# print(ResultsBySubjects[2])
# print(ResultsBySubjects[3])
```

- Only one issue with List we do not know which result refers to which subject (Not very intuitive for our use case)
- Worry not Python has us covered with another data type called **Dict**:
    - You can use a key-value to give some meaning to the data you are trying to store/manipulate

In [45]:
```
# 45/100 in Sciance, 50/100 in Maths, 52/100 in Biology and 80/100 in CS
ResultsBySubjects = {'Sciance': 45, 'Maths': 50, 'Biology': 52, 'CS': 80 }

# print(type(ResultsBySubjects))

# print(ResultsBySubjects['CS'])
```

In [38]:
```
# This is what we end up with when using Dict with our student example:

Name = "Luke"
Age = 24
ResultsBySubjects = {'Science': 45, 'Maths': 50, 'Biology': 52, 'CS': 80 }
Result = 'P'
```

- Now that we have results by subjects, we can calculate the percentage.
- Since ResultsBySubjects is not a single value we cannot perform addition (if you remember, we talked about how the types of operation that you can perform depend on the data type)
- How can we use the **Dict** type ResulsBySubjects to find the sum and ultimately the percentage
- Logically, we read each of the entries and sum them up.
    - In programming, we use a **loop** to perform this type of operations

In [44]:
```
ResultsBySubjects = {'Sciance': 45, 'Maths': 50, 'Biology': 52, 'CS': 80 }

for subject in ResultsBySubjects:
    print(subject)
    # print(ResultsBySubjects[subject])
```

```
Sciance
Maths
Biology
CS
```

- Now let's use the for loop to calculate the sum

In [48]:
```
ResultsBySubjects = {'Sciance': 45, 'Maths': 50, 'Biology': 52, 'CS': 80 }

# we need to store the sum somewhere so I have created a variable called sum and s
et its value to 0
sum = 0
for subject in ResultsBySubjects:
    print(subject)
    sum = sum + ResultsBySubjects[subject]
    print(sum)

# percentage = (sum/400)*100
# print(percentage)
```

```
Sciance
45
Maths
95
Biology
147
CS
227
56.75
```

- What if we wanted to add grades for another subject?

In [50]:
```
ResultsBySubjects = {'Sciance': 45, 'Maths': 50, 'Biology': 52, 'CS': 80, 'Histor
y': 65 }

# we need to store the sum somewhere so I have created a variable called sum and s
et its value to 0
sum = 0
for subject in ResultsBySubjects:
    print(subject)
    sum = sum + ResultsBySubjects[subject]
    print(sum)

percentage = (sum/400)*100
print(percentage)
```

```
Sciance
45
Maths
95
Biology
147
CS
227
History
292
73.0
```

- Can you identify any issue with our solution?

- We need to automate the logic of calculating **total**.
- Python has some really powerful tools called **functions**
- *A function is a bunch of code that only runs when it is called. You can often pass values to functions*
- We have pre-defined = and user-defined functions (more on this soon).
- We can use the pre-defined function called len (short for length) to find out how many subjects we have.

In [53]:
```
ResultsBySubjects = {'Sciance': 45, 'Maths': 50, 'Biology': 52, 'CS': 80, 'Histor
y': 65 }

print(len(ResultsBySubjects))
```

```
5
```

- Now we can just multiply it by 100 and get our total marks

In [56]:
```
ResultsBySubjects = {'Sciance': 45, 'Maths': 50, 'Biology': 52, 'CS': 80, 'Histor
y': 65 }

# we need to store the sum somewhere so I have created a variable called sum and s
et its value to 0
sum = 0
for subject in ResultsBySubjects:
    # print(subject)
    sum = sum + ResultsBySubjects[subject]
    # print(sum)

percentage = (sum/(len(ResultsBySubjects)*100))*100
# print(percentage)
```

## What have we learned so far?

- What a variable is and how to declare and use it?
- Complex data types such as a List and Dict
- How to read these data types by using a loop?
- How to use predefined functions?

## Basics of reading CSV files in Python

- A sample CSV file for a student
- We need to open this file using Python
- After opening the file we need to read it
- Now we can calculate and show the percentage results

- Reading files in a Python program is a very common task so rather than re-inventing the wheel again we will use the code written by others.
- A lot of functions that perform some related tasks are often bundled together and served as a *Library*
    - Example: All the functions used to draw graphs can be combined in a single library for graphs
- To read our CSV file, we will use one such Library called **Pandas**
- **Pandas** is a very popular data manipulation and analysis library.

- In order to use a library, we must first import it our program

In [59]:
```
import pandas

# Now we can use pandas to read the csv file

studentResults = pandas.read_csv('studentResults.csv')

print(studentResults)
```

```
   Name  Age  Science  Maths  Biology  CS  History
0  Luke   24       45     10       53  84       89
```

In [60]:
```
print(type(studentResults))
```

```
<class 'pandas.core.frame.DataFrame'>
```

- This is a new data type that we have never seen before. Nothing to be scared of here. Most of the new data types that you encounter will have all the logical operations we can think of.

- Now let us read the Name Age and subjectResults.
- To do this, we will use a predefined function of Pandas DataFrame

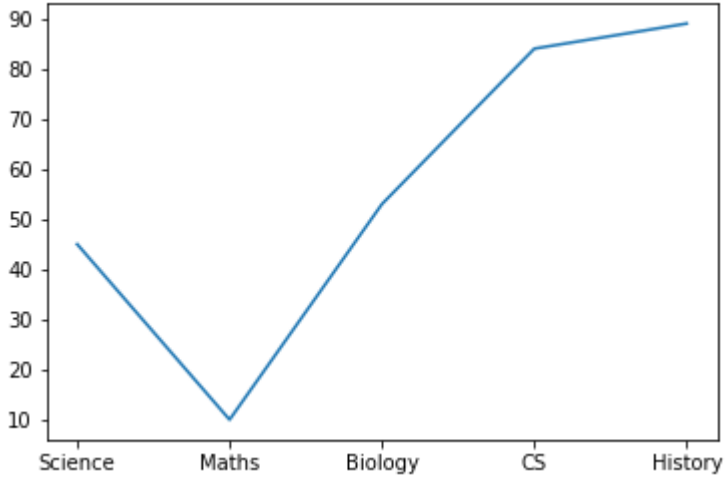
In [90]:

```
import pandas

studentResults = pandas.read_csv('studentResults.csv')

Luke = studentResults.iloc[0]

name = Luke[0]
age = Luke[1]
subjects = Luke[2:7]
# print(Luke[2:7])

sum = 0

for subject in subjects:
    sum = sum + subject

percentage = (sum/(len(subjects)*100))*100
# print(percentage)
```

- Before we wrap up today's session, let's have a look at graphs in Python
- We will use an amazing library known as Matplotlib

In [95]:
```
import pandas
import matplotlib.pyplot as plt

studentResults = pandas.read_csv('studentResults.csv')

Luke = studentResults.iloc[0]

name = Luke[0]
age = Luke[1]
subjects = Luke[2:7]

plt.plot(subjects)
```

Out[95]:
```
[<matplotlib.lines.Line2D at 0x7fafdd7ca3c8>]
```



## Some best-practice for your code in Python

- Write a lot of useful comments on your code.
- Use version control systems for your code (have a look at Github).
- Test your code frequently while implementing a logic.
- import this (Zen of Python)

In [96]:
```
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!
```

# Data Manipulation in Python Part 1

**Ashish Rajendra Sai**

***Ph.D. Candidate - University of Limerick***

Any outstanding questions?