Python - up and Running

https://github.com/BilalMaz/DevOps-Architect-BootCamp

About me

Hi there, my name is Bilal and I will Welcome you to DevOps boot camp! I am thrilled to have you join us for this exciting journey of learning and discovery.

In this boot camp, we will be exploring the principles and practices of DevOps, which is a set of methodologies and tools that aims to bridge the gap between software development and operations. DevOps is an increasingly important area in the field of software engineering, as it helps organizations to streamline their processes, improve their agility, and deliver better value to their customers.

By the end of this boot camp, you will have gained a comprehensive understanding of DevOps and its key concepts, as well as practical skills in areas such as infrastructure automation, continuous integration and delivery, monitoring and logging, and more. You will be equipped with the knowledge and tools to apply DevOps principles in your own work and contribute to the success of your organization.

I am always looking to connect with other professionals in the field, share ideas and insights, and stay up to date on the latest trends and developments. I welcome the opportunity to connect with you and explore ways in which we can collaborate and support each other.

Please find my Linkedin profile

https://www.linkedin.com/in/bilalmazhar-cyber-security-consultant/

What is Python Language ?

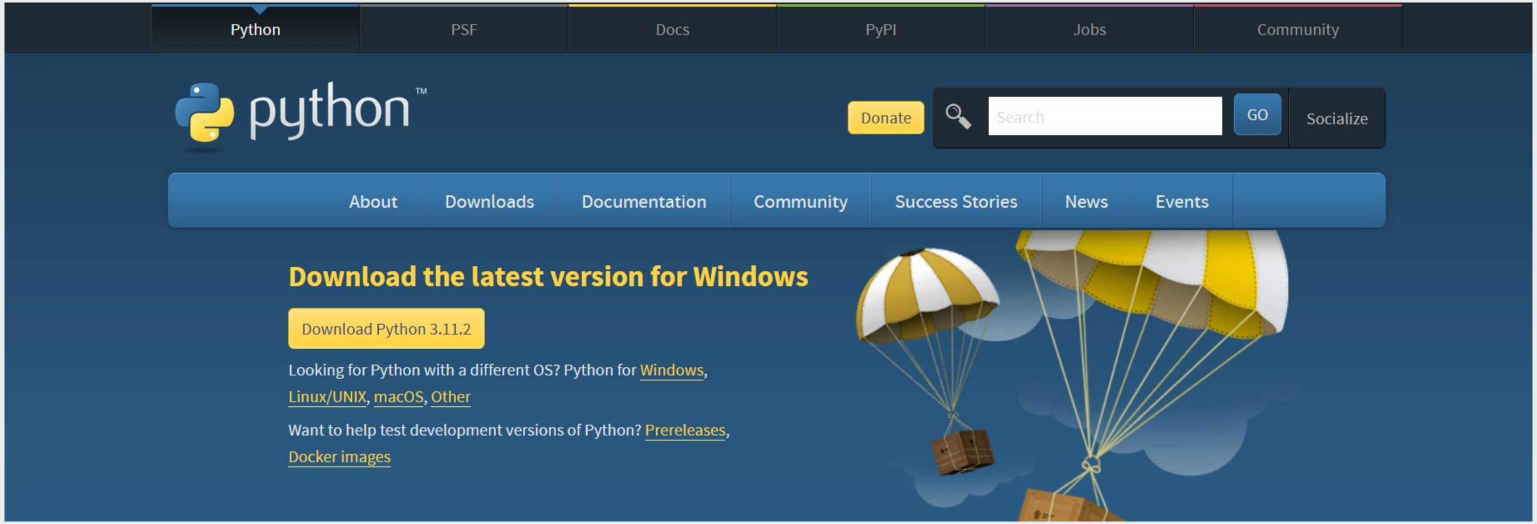
Python is a **high-level programming** language that is widely used for a variety of purposes, including web development, data analysis, artificial intelligence, machine learning, scientific computing, and more. It was created in the late 1980s by Guido van Rossum and has since become one of the most popular programming languages in the world.

Python is Interpreted − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
Python is Interactive − You can actually sit at a Python prompt and interact with the interpreter directly to write your programs
Python is Object-Oriented − Python supports Object-Oriented style or technique of programming that encapsulates code within objects.

Python is an excellent language for beginners who are just starting to learn programming. It provides a straightforward syntax that is easy to read and write, making it easier for novice programmers to understand and maintain their code. Despite its simplicity, Python is capable of developing a wide range of applications, from basic text processing and web browsers to complex games and scientific simulations

Python Installation

https://www.python.org/downloads/



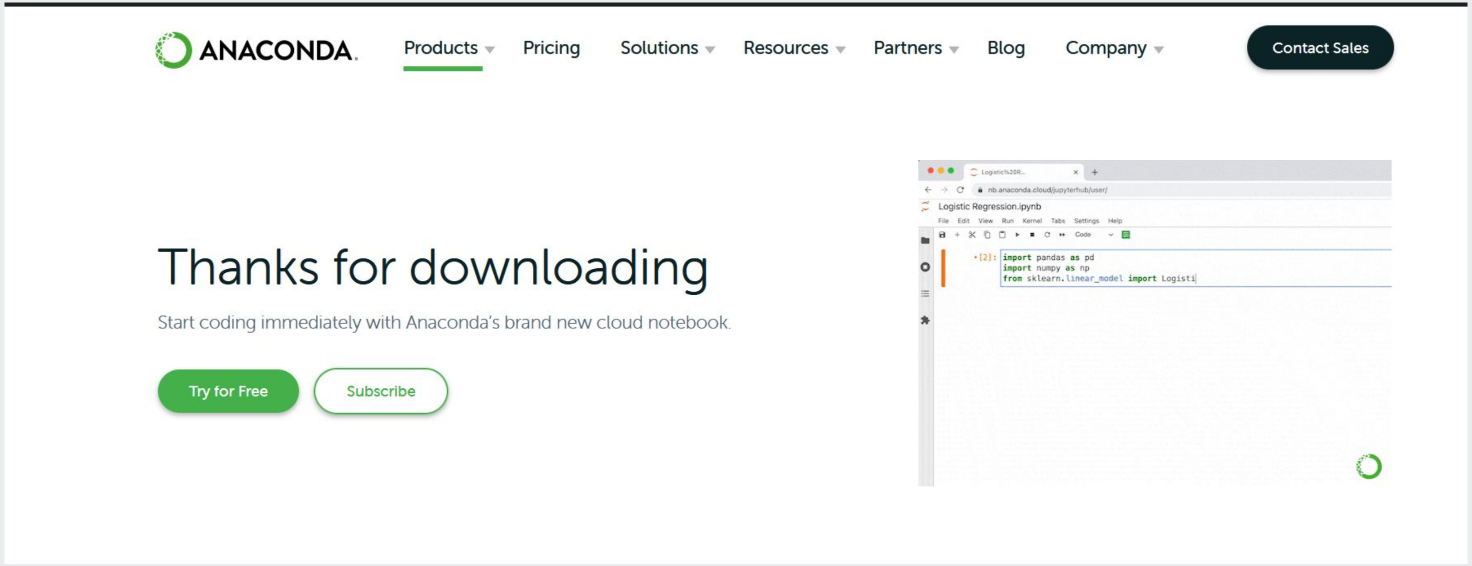
Download the latest version for Windows

Looking for Python with a different OS? Python for Windows, Linux/UNIX, macOS, Other

Want to help test development versions of Python? Prereleases, Docker images

Python - Environment Setup

https://www.anaconda.com/products/distribution/start-coding-immediately



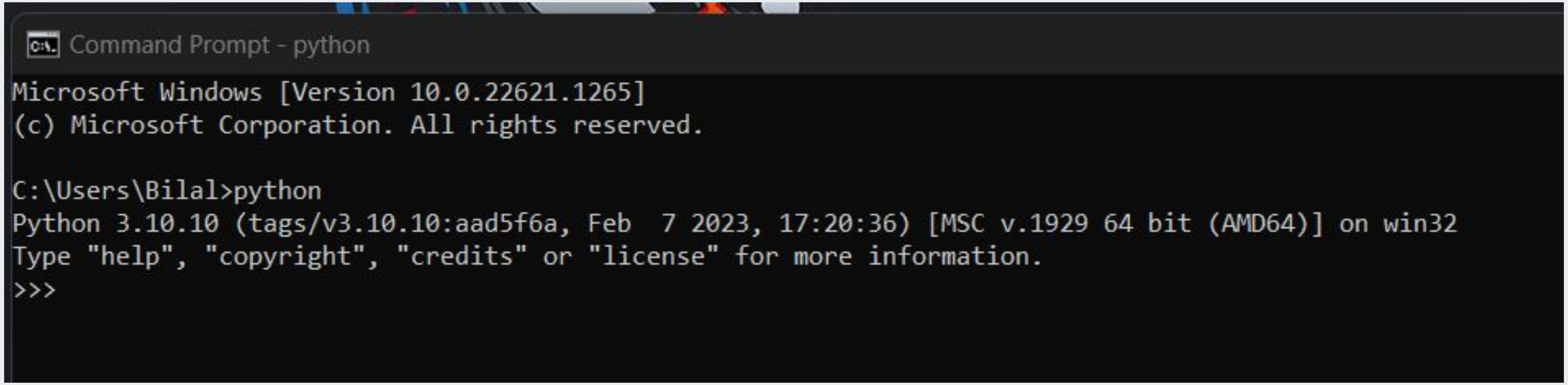
Thanks for downloading

Start coding immediately with Anaconda's brand new cloud notebook.

Try for Free Subscribe

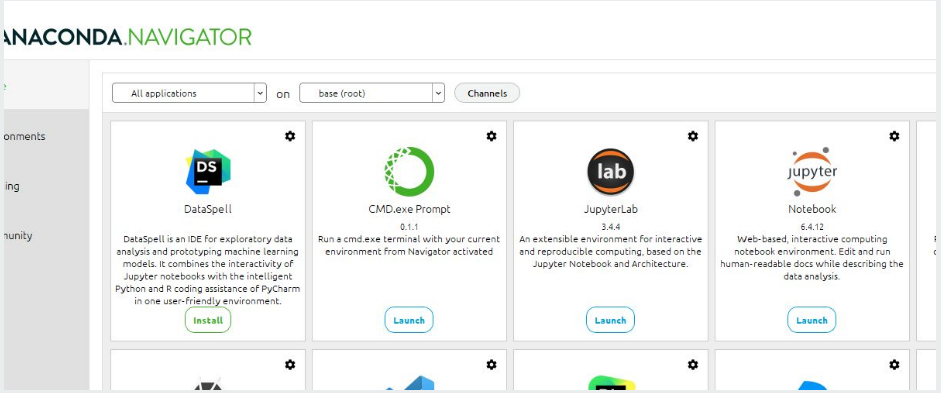
```
[2]: import pandas as pd
     import numpy as np
     from sklearn.linear_model import Logisti
```

Python version

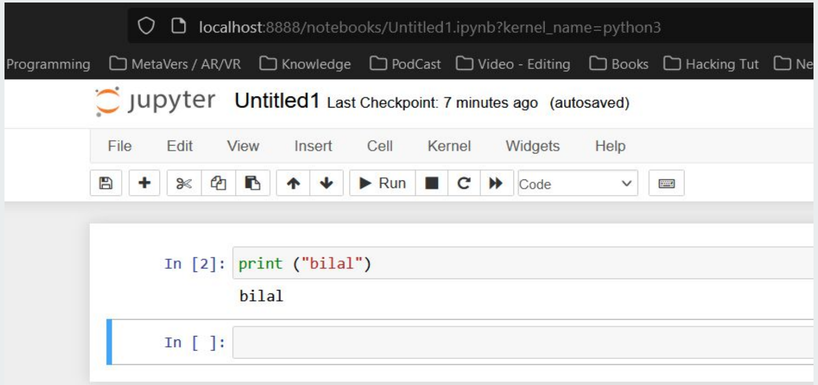


```
Microsoft Windows [Version 10.0.22621.1265]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Bilal>python
Python 3.10.10 (tags/v3.10.10:aad5f6a, Feb  7 2023, 17:20:36) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Write your first Program



ANACONDA.NAVIGATOR
All applications
on
base (root)
Channels
onments
ing
nunity
DataSpell
DataSpell is an IDE for exploratory data analysis and prototyping machine learning models. It combines the interactivity of Jupyter notebooks with the intelligent Python and R coding assistance of PyCharm in one user-friendly environment.
Install
CMD.exe Prompt
0.1.1
Run a cmd.exe terminal with your current environment from Navigator activated
Launch
JupyterLab
3.4.4
An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.
Launch
Notebook
6.4.12
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.
Launch



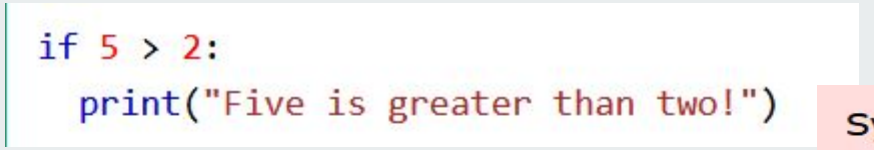
```
In [2]: print ("bilal")
        bilal

In [ ]:
```

Python Indentation

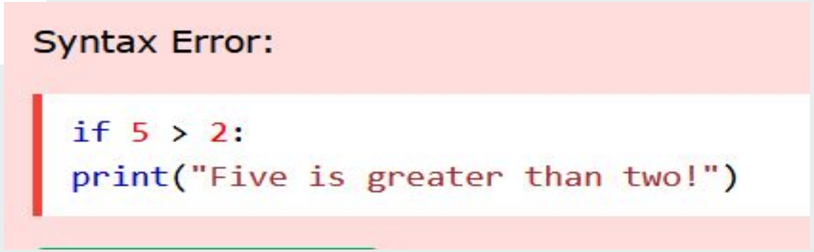
Indentation refers to the spaces at the beginning of a code line.

Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important.

Python uses indentation to indicate a block of code.



```python
if 5 > 2:
  print("Five is greater than two!")
```



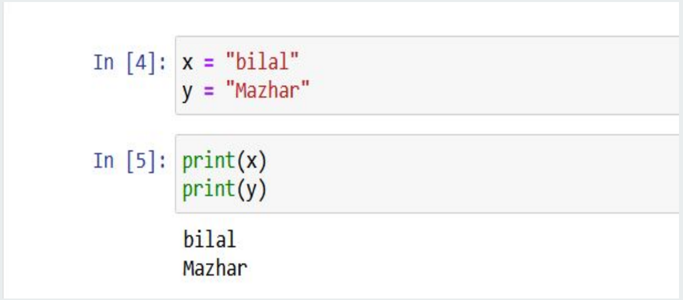
Syntax Error:

```python
if 5 > 2:
print("Five is greater than two!")
```

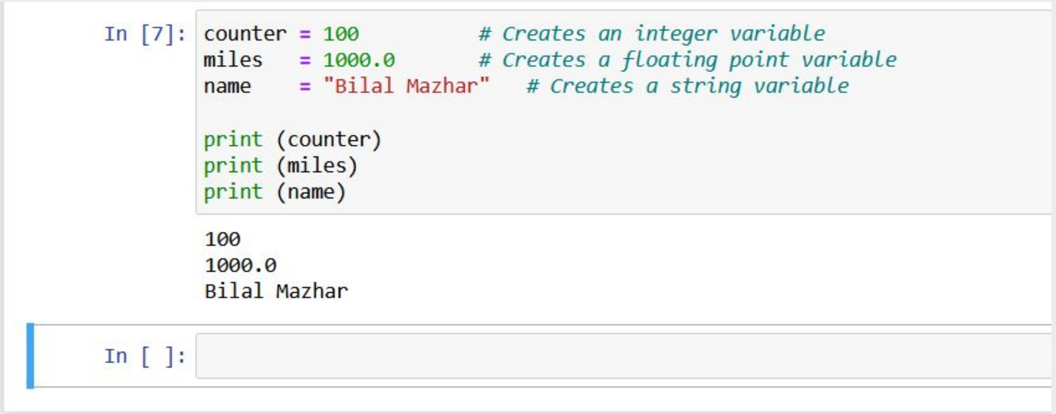
Python Variables

In Python, variables are created when you assign a value to it

Python has no command for declaring a variable.



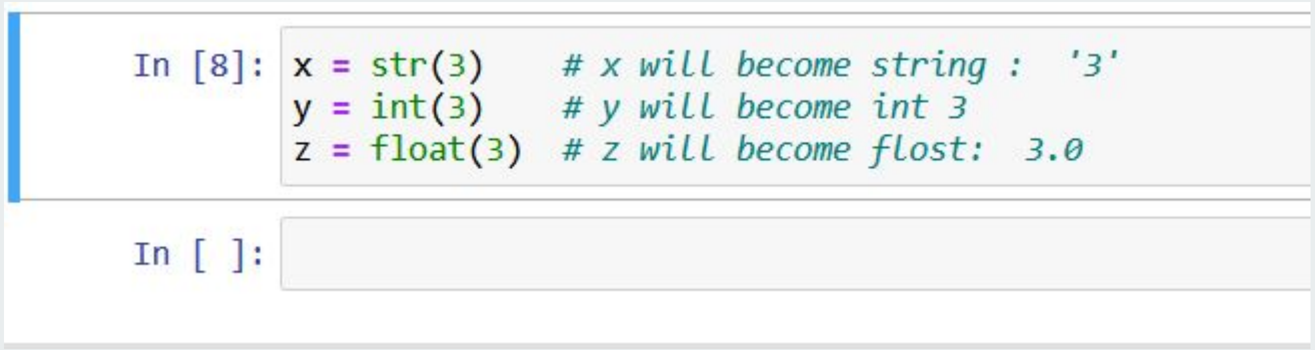
```
In [4]: x = "bilal"
        y = "Mazhar"

In [5]: print(x)
        print(y)

        bilal
        Mazhar
```



```
In [7]: counter = 100          # Creates an integer variable
        miles   = 1000.0       # Creates a floating point variable
        name    = "Bilal Mazhar"    # Creates a string variable

        print (counter)
        print (miles)
        print (name)

        100
        1000.0
        Bilal Mazhar

In [ ]:
```

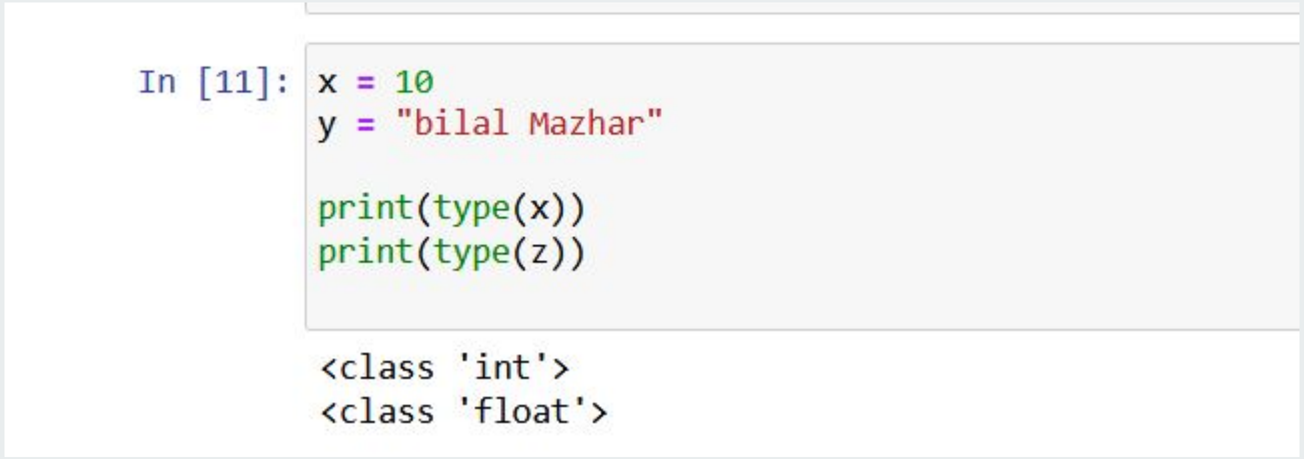
Python Variables : Casting

If you want to specify the data type of a variable, this can be done with casting.



```
In [8]: x = str(3)    # x will become string :  '3'
        y = int(3)    # y will become int 3
        z = float(3)  # z will become flost:  3.0

In [ ]:
```

Get the Type of variable

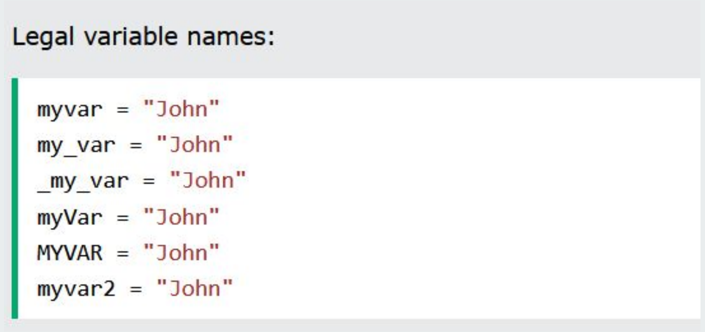


```
In [11]: x = 10
         y = "bilal Mazhar"

         print(type(x))
         print(type(z))

         <class 'int'>
         <class 'float'>
```

Variable Names

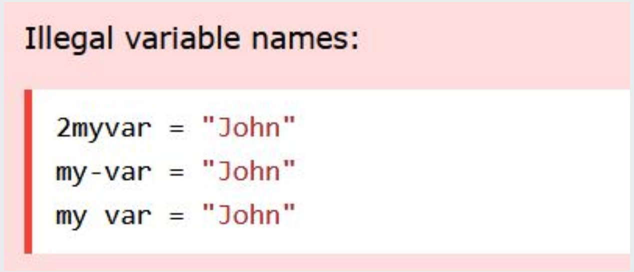
A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume). Rules for Python variables:

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alphanumeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)



Legal variable names:

```
myvar = "John"
my_var = "John"
_my_var = "John"
myVar = "John"
MYVAR = "John"
myvar2 = "John"
```



Illegal variable names:

```
2myvar = "John"
my-var = "John"
my var = "John"
```

Python Data Types

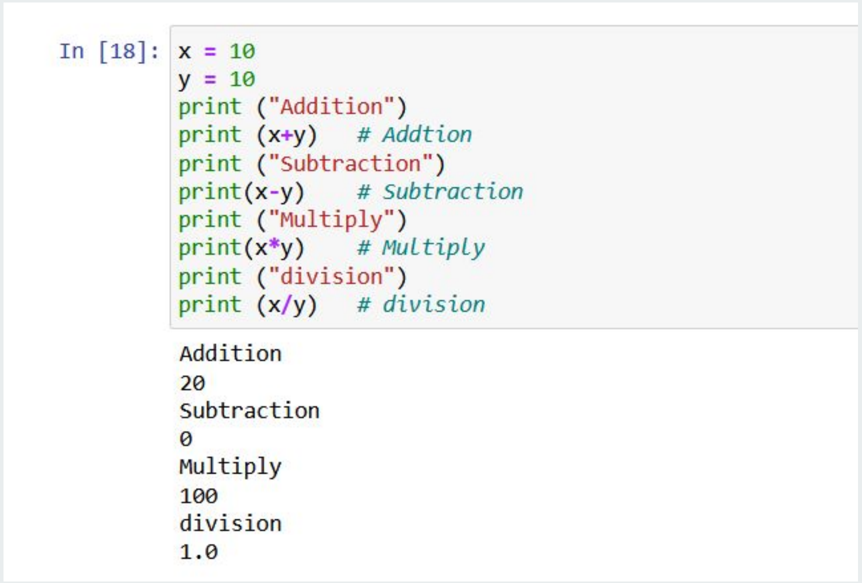
Variables can store data of different types, and different types can do different things.

Data Type	Syntax
Text Type	str
Numeric Types	int, float, complex
Sequence Types	list, tuple, range
Boolean Type	Bool

Data Type	Syntax
x = "Bilal Mazhar"	str
x = 20	int
x = 20.5	float
x = ["Bilal", "DevOps", "Mazhar"]	list
x = ("Bootcamp", "2023", "github”)	tuple
x = range(6)	range
x = {"name" : "Bilal", "age" : 36}	dist
x = {"apple", "banana", "cherry"}	set
x = memoryview(bytes(5))	Memory view

Python Operators

Operators are used to perform operations on variables and values.

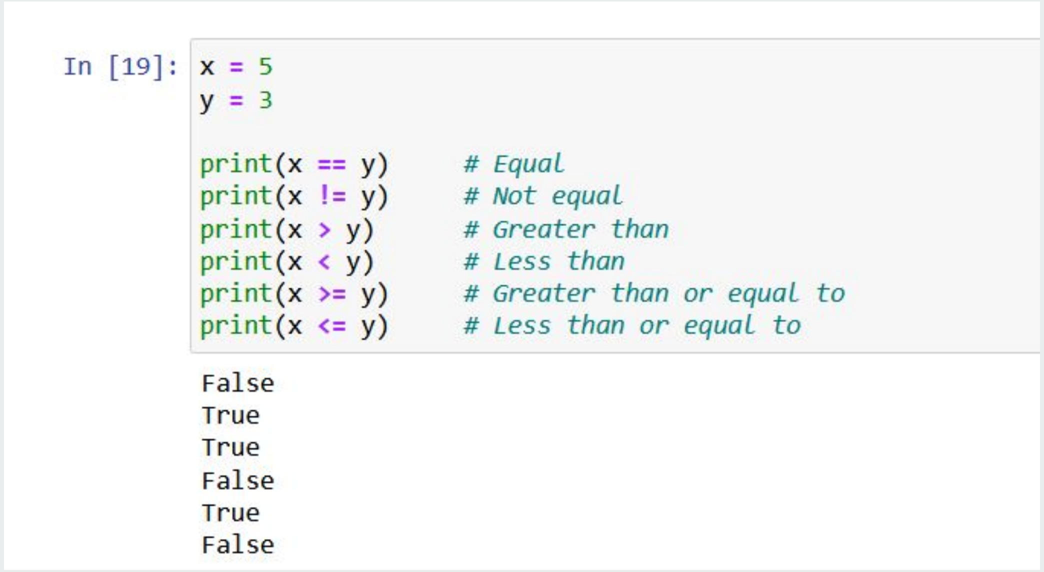


```
In [18]: x = 10
         y = 10
         print ("Addition")
         print (x+y)   # Addtion
         print ("Subtraction")
         print(x-y)    # Subtraction
         print ("Multiply")
         print(x*y)    # Multiply
         print ("division")
         print (x/y)   # division
```

```
Addition
20
Subtraction
0
Multiply
100
division
1.0
```

Python Comparison Operators

Comparison operators are used to compare two values:



```
In [19]: x = 5
         y = 3

         print(x == y)     # Equal
         print(x != y)     # Not equal
         print(x > y)      # Greater than
         print(x < y)      # Less than
         print(x >= y)     # Greater than or equal to
         print(x <= y)     # Less than or equal to

         False
         True
         True
         False
         True
         False
```

Python Collections (Arrays)

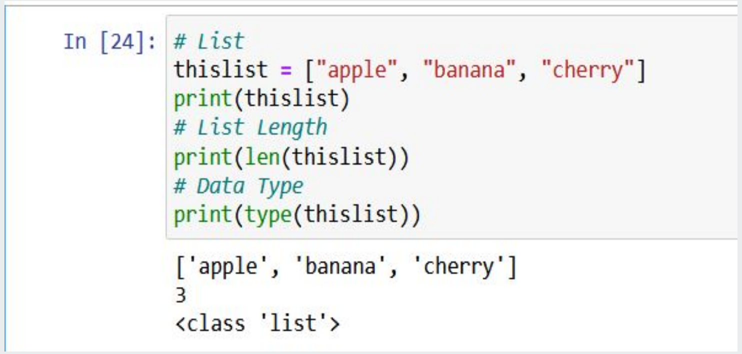
There are four collection data types in the Python programming language:

- **List** is a collection which is ordered and changeable. Allows duplicate members.
- **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
- **Set** is a collection which is unordered, unchangeable*, and unindexed. No duplicate members.
- **Dictionary** is a collection which is ordered** and changeable. No duplicate members.

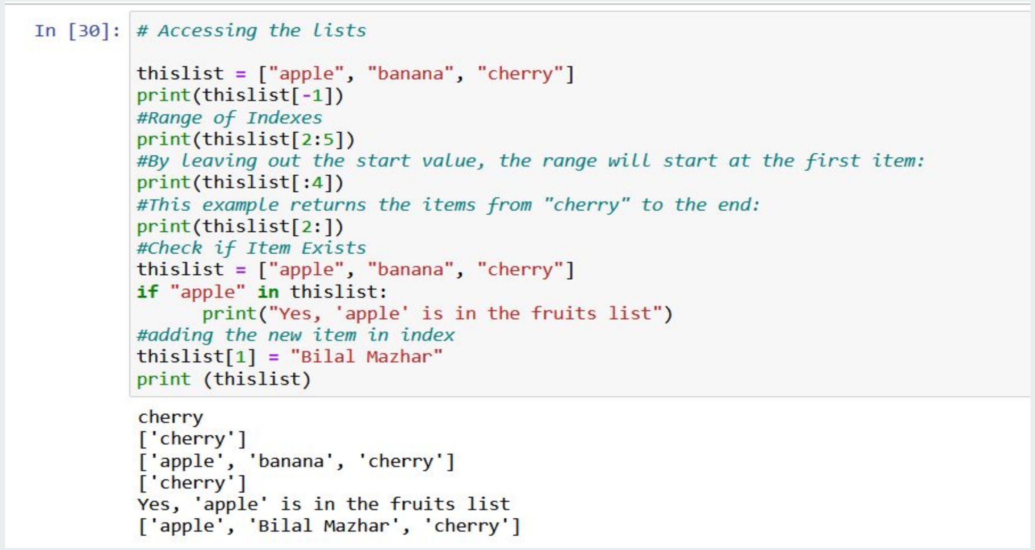
List

Lists are used to store multiple items in a single variable.

Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage.



```
In [24]: # List
         thislist = ["apple", "banana", "cherry"]
         print(thislist)
         # List Length
         print(len(thislist))
         # Data Type
         print(type(thislist))

         ['apple', 'banana', 'cherry']
         3
         <class 'list'>
```

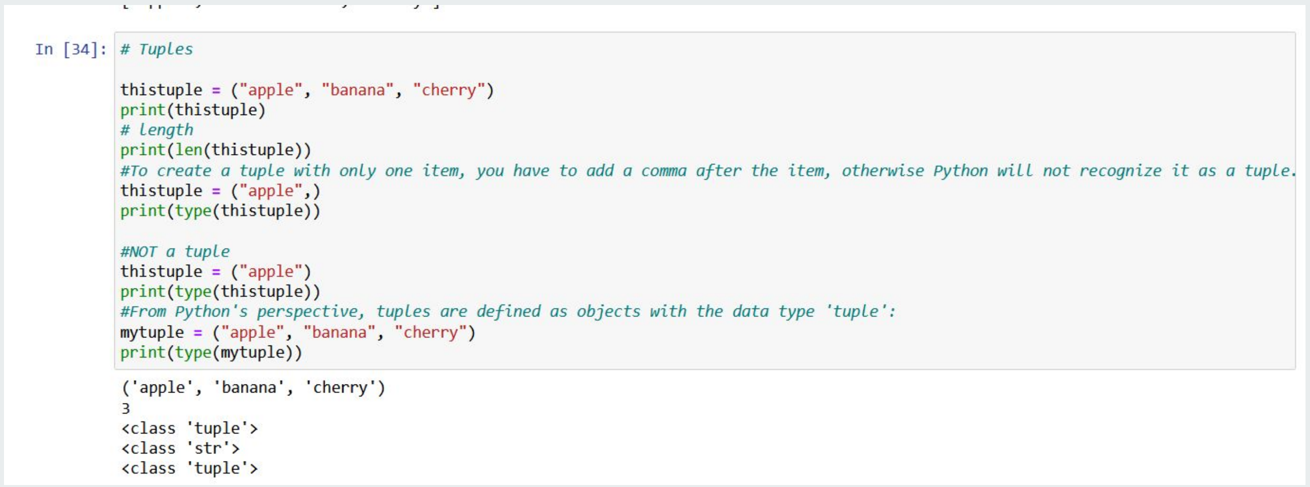


```
In [30]: # Accessing the lists

         thislist = ["apple", "banana", "cherry"]
         print(thislist[-1])
         #Range of Indexes
         print(thislist[2:5])
         #By leaving out the start value, the range will start at the first item:
         print(thislist[:4])
         #This example returns the items from "cherry" to the end:
         print(thislist[2:])
         #Check if Item Exists
         thislist = ["apple", "banana", "cherry"]
         if "apple" in thislist:
               print("Yes, 'apple' is in the fruits list")
         #adding the new item in index
         thislist[1] = "Bilal Mazhar"
         print (thislist)

         cherry
         ['cherry']
         ['apple', 'banana', 'cherry']
         ['cherry']
         Yes, 'apple' is in the fruits list
         ['apple', 'Bilal Mazhar', 'cherry']
```

Tuple

Tuples are used to store multiple items in a single variable.

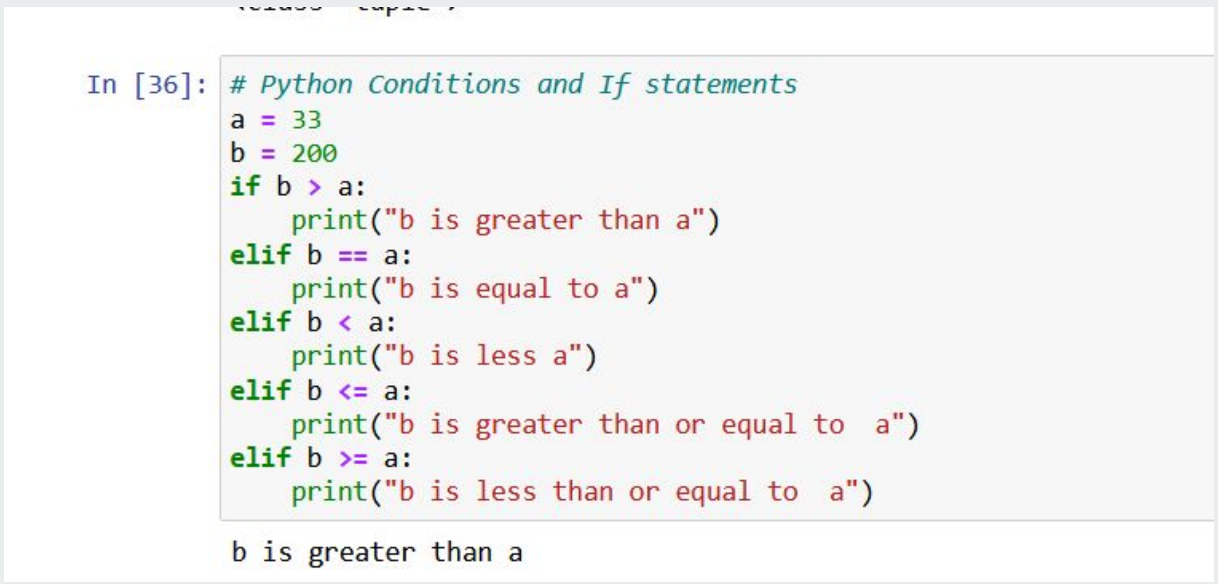
Tuple is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Set, and Dictionary, all with different qualities and usage. A tuple is a collection which is ordered and unchangeable.



```
In [34]: # Tuples

         thistuple = ("apple", "banana", "cherry")
         print(thistuple)
         # length
         print(len(thistuple))
         #To create a tuple with only one item, you have to add a comma after the item, otherwise Python will not recognize it as a tuple.
         thistuple = ("apple",)
         print(type(thistuple))

         #NOT a tuple
         thistuple = ("apple")
         print(type(thistuple))
         #From Python's perspective, tuples are defined as objects with the data type 'tuple':
         mytuple = ("apple", "banana", "cherry")
         print(type(mytuple))

         ('apple', 'banana', 'cherry')
         3
         <class 'tuple'>
         <class 'str'>
         <class 'tuple'>
```

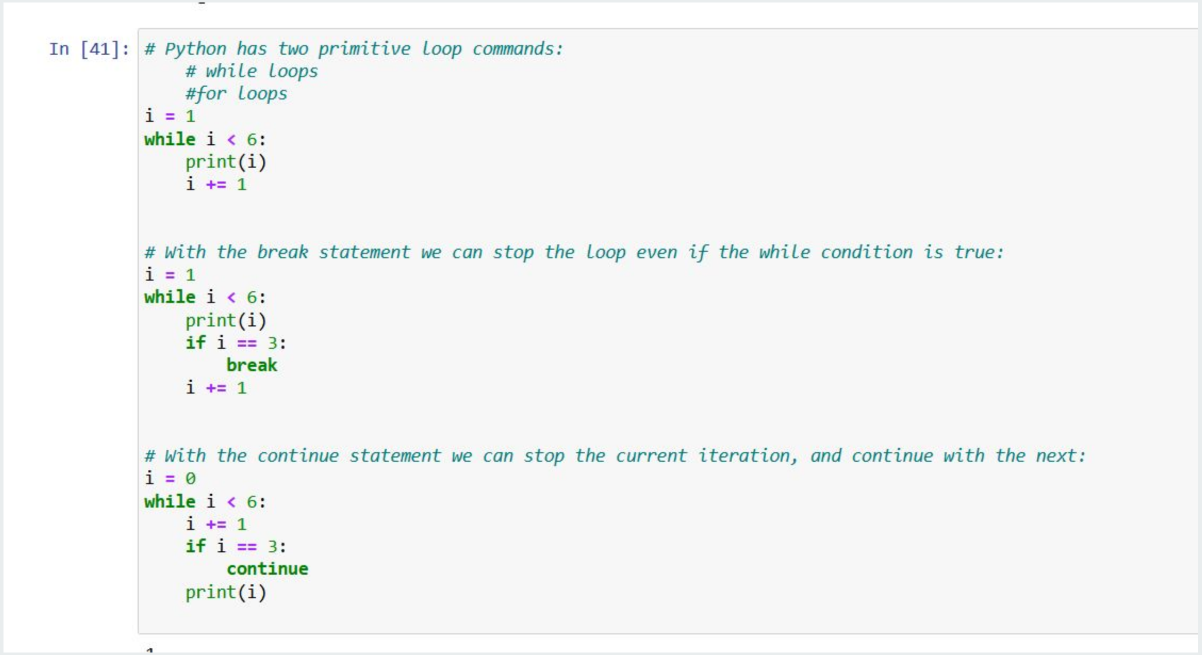
Python Conditions and If statements

Python supports the usual logical conditions from mathematics:

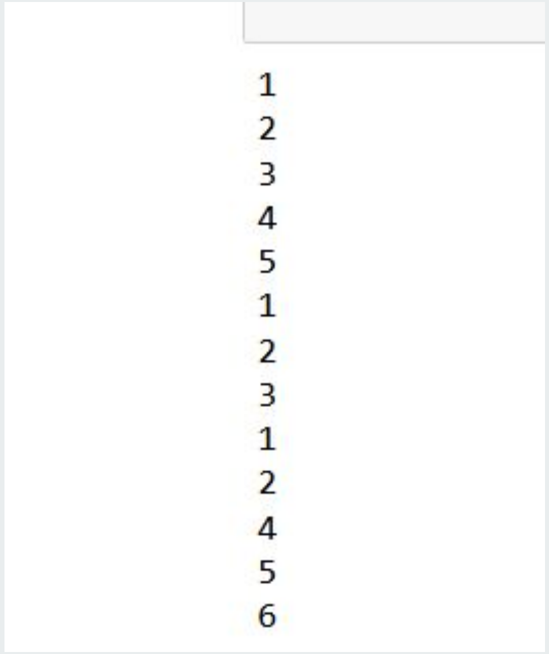


```
In [36]: # Python Conditions and If statements
         a = 33
         b = 200
         if b > a:
             print("b is greater than a")
         elif b == a:
             print("b is equal to a")
         elif b < a:
             print("b is less a")
         elif b <= a:
             print("b is greater than or equal to  a")
         elif b >= a:
             print("b is less than or equal to  a")

         b is greater than a
```

while Loop



```
In [41]: # Python has two primitive loop commands:
             # while loops
             #for loops
         i = 1
         while i < 6:
             print(i)
             i += 1


         # With the break statement we can stop the loop even if the while condition is true:
         i = 1
         while i < 6:
             print(i)
             if i == 3:
                 break
             i += 1


         # With the continue statement we can stop the current iteration, and continue with the next:
         i = 0
         while i < 6:
             i += 1
             if i == 3:
                 continue
             print(i)
```



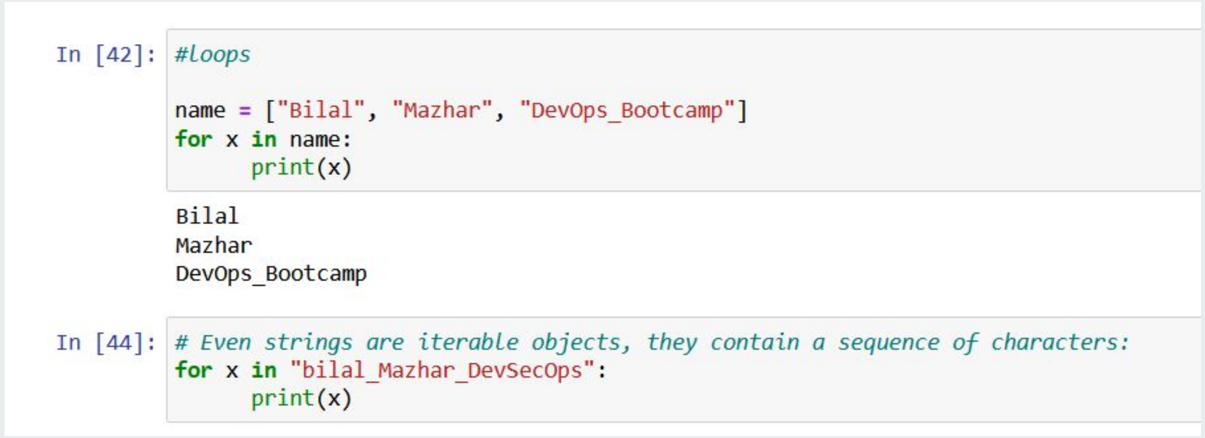
```
1
2
3
4
5
1
2
3
1
2
4
5
6
```

Loops

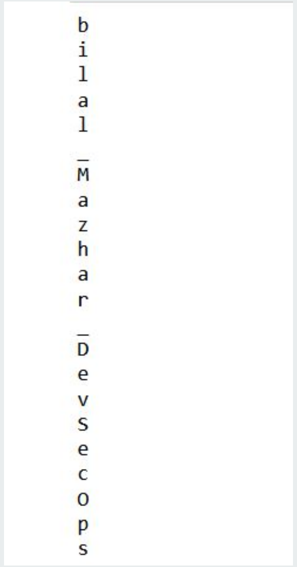
A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-oriented programming languages.

With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.



```
In [42]: #loops

         name = ["Bilal", "Mazhar", "DevOps_Bootcamp"]
         for x in name:
               print(x)

         Bilal
         Mazhar
         DevOps_Bootcamp

In [44]: # Even strings are iterable objects, they contain a sequence of characters:
         for x in "bilal_Mazhar_DevSecOps":
               print(x)
```

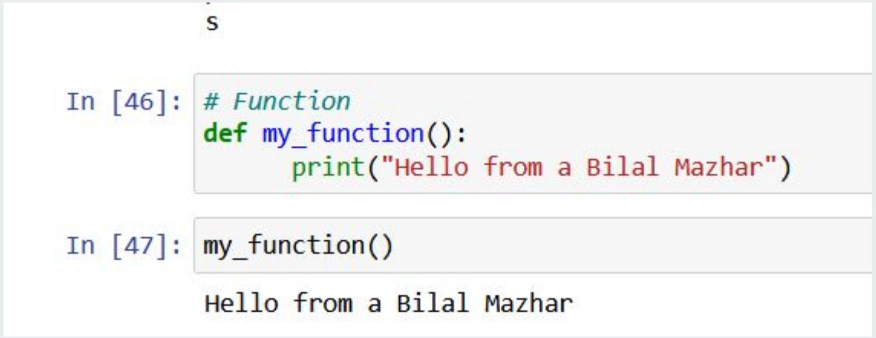


```
b
i
l
a
l
_
M
a
z
h
a
r
_
D
e
v
S
e
c
O
p
s
```

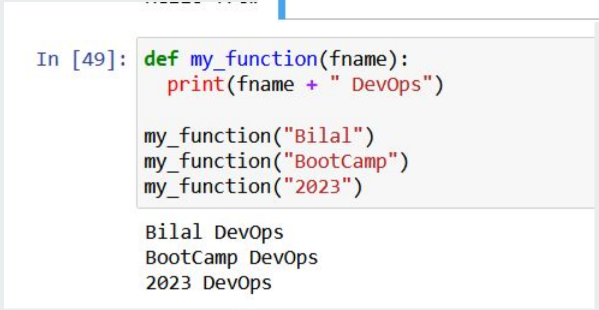
Python Functions

A function is a block of code which only runs when it is called.

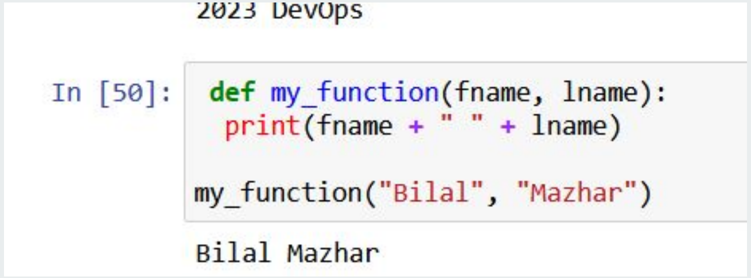
You can pass data, known as parameters, into a function.



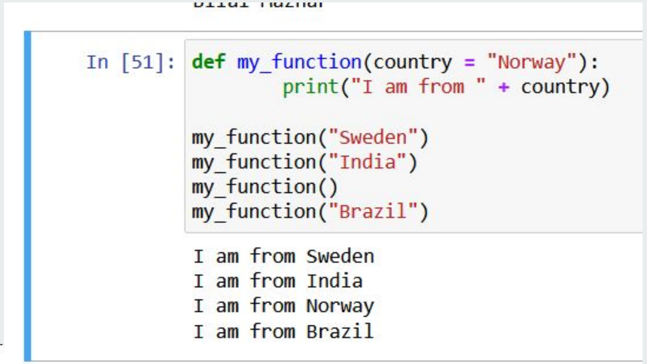
```
s

In [46]: # Function
         def my_function():
                 print("Hello from a Bilal Mazhar")

In [47]: my_function()

         Hello from a Bilal Mazhar
```



```
In [49]: def my_function(fname):
           print(fname + " DevOps")

         my_function("Bilal")
         my_function("BootCamp")
         my_function("2023")

         Bilal DevOps
         BootCamp DevOps
         2023 DevOps
```



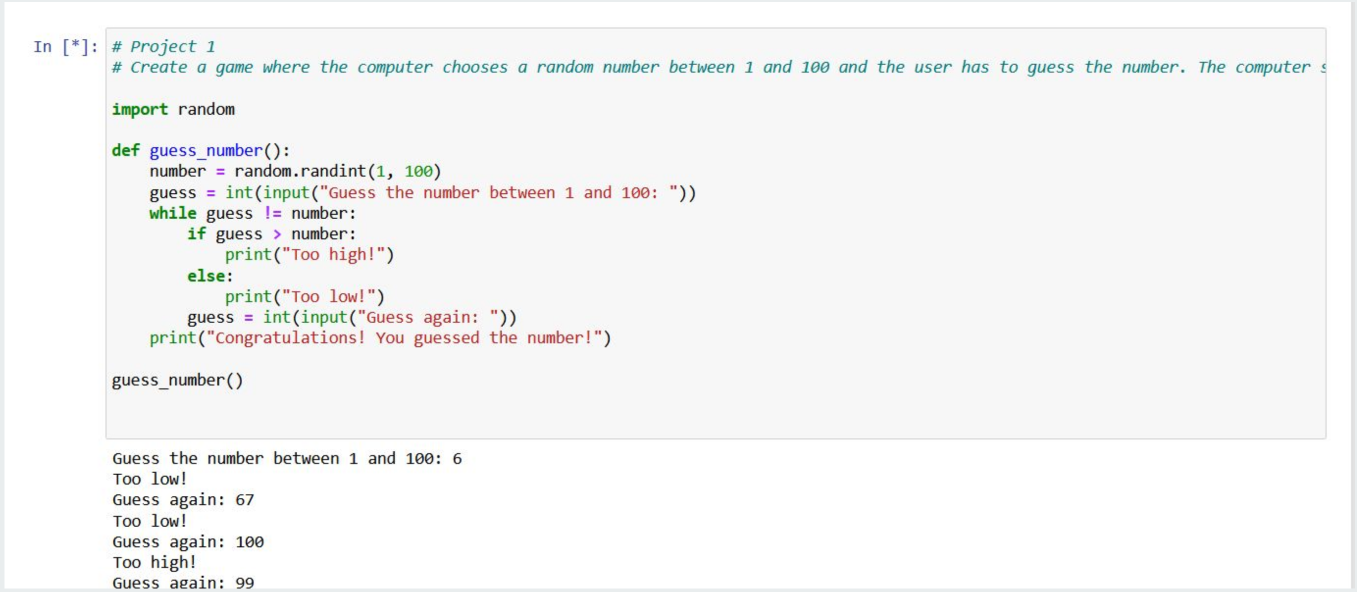
```
         2023 DevOps

In [50]:  def my_function(fname, lname):
           print(fname + " " + lname)

         my_function("Bilal", "Mazhar")

         Bilal Mazhar
```



```
         Bilal Mazhar

In [51]: def my_function(country = "Norway"):
                 print("I am from " + country)

         my_function("Sweden")
         my_function("India")
         my_function()
         my_function("Brazil")

         I am from Sweden
         I am from India
         I am from Norway
         I am from Brazil
```

Project: 1

Create a game where the computer chooses a random number between 1 and 100 and the user has to guess the number, The computer should provide feedback if the user's guess is too high or too low until the user guesses the correct number.

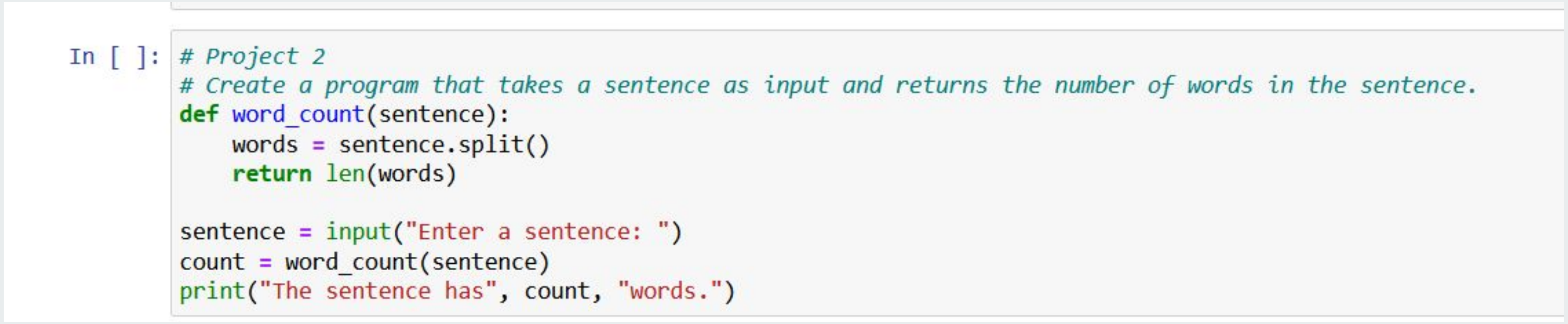


```
In [*]: # Project 1
        # Create a game where the computer chooses a random number between 1 and 100 and the user has to guess the number. The computer s

        import random

        def guess_number():
            number = random.randint(1, 100)
            guess = int(input("Guess the number between 1 and 100: "))
            while guess != number:
                if guess > number:
                    print("Too high!")
                else:
                    print("Too low!")
                guess = int(input("Guess again: "))
            print("Congratulations! You guessed the number!")

        guess_number()
```

```
Guess the number between 1 and 100: 6
Too low!
Guess again: 67
Too low!
Guess again: 100
Too high!
Guess again: 99
```

Project: 2

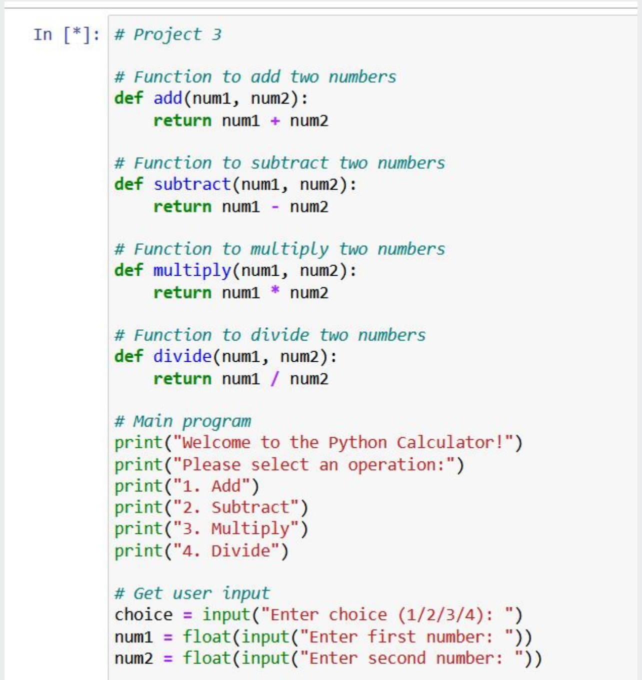
Create a program that takes a sentence as input and returns the number of words in the sentence.



```
In [ ]: # Project 2
        # Create a program that takes a sentence as input and returns the number of words in the sentence.
        def word_count(sentence):
            words = sentence.split()
            return len(words)

        sentence = input("Enter a sentence: ")
        count = word_count(sentence)
        print("The sentence has", count, "words.")
```

Project: 3

Python calculator code that performs basic arithmetic operations such as addition, subtraction, multiplication, and division:



```
In [*]: # Project 3

        # Function to add two numbers
        def add(num1, num2):
            return num1 + num2

        # Function to subtract two numbers
        def subtract(num1, num2):
            return num1 - num2

        # Function to multiply two numbers
        def multiply(num1, num2):
            return num1 * num2

        # Function to divide two numbers
        def divide(num1, num2):
            return num1 / num2

        # Main program
        print("Welcome to the Python Calculator!")
        print("Please select an operation:")
        print("1. Add")
        print("2. Subtract")
        print("3. Multiply")
        print("4. Divide")

        # Get user input
        choice = input("Enter choice (1/2/3/4): ")
        num1 = float(input("Enter first number: "))
        num2 = float(input("Enter second number: "))
```