1. Python Syntax, Comments, Variable Names

This is how we define the syntax in Python which is very much simple and easy to use unlike the other programming languages. The "#" is used to comment something in Python. We can directly declare and define variables without specifying the datatypes as python automatically detects datatypes.

In [1]:
```
# This is a comment
x = 5  # Variable assignment
y = 10

# Example code
sum = x + y
print("Sum:", sum)
```

```
Sum: 15
```

2. Python Data Types

These are the basic datatype which are used commonly in python.

1. Integer: Integer is used for the numbers and this number can be limitless.

In [10]:
```
#Integer Variable
a=100
print("The variable type of value", a, " is ", type(a))
```

```
The variable type of value 100  is  <class 'int'>
The variable type of value 10.2345  is  <class 'float'>
The variable type of value (100+3j)  is  <class 'complex'>
string in a double quote
string in a single quote
<class 'bool'>
```

Float data type is for the floating numbers which are precision numbers and it goes upto 15 decimal places

In [16]:
```
#Float Variable
b=10.2345
print("The variable type of value", b, " is ", type(b))
```

```
The variable type of value 10.2345  is  <class 'float'>
```

String : The string is a sequence of characters. Strings can be represented by single or double-quotes. We can use it to write alphbetic words

Another worth mentioning thing is that we can check the data type of any variable in python by using the "type(variable name)" as i have shown in the code above.

In [19]:
```
#String Variables
a = "string in a double quote"
b= 'string in a single quote'
print(a)
print(b)
```

```
string in a double quote
string in a single quote
```

The complex data type in python consists of two values, the first one is the real part of the complex number, and the second one is the imaginary part of the complex number.

In [18]:
```
#Complex Variable
c=100+3j
print("The variable type of value", c, " is ", type(c))
```

```
The variable type of value (100+3j)  is  <class 'complex'>
```

Boolean type: Bool: Boolean type is one of Python's built-in data types. It's used to represent the truth value of an expression. We can check the type of True and False with the built-in type(). Boolean type has only two possible values 'True' or 'False'

In [17]:
```
#Bool
a = False
print(type(a))
```

```
<class 'bool'>
```

3. Python Lists, Tuples, Sets, Dictionaries

These are also the data types of python and these are called Sequence Data Types List: A List can consist of multiple data types at a time. It is declared as square brackets([]) and commas(,).

In [35]:
```
#Python list having both integers and strings
c= ["hey","you",1,2,3,"go"]
print("List: ", c)

list1 = [1, 2, 3]
list2 = [4, 5, 6]

# Concatenation of two lists
concatenated_list = list1 + list2
print(concatenated_list)

# Length of a list
length = len(list1)
print("lenght: ",length)

# Sorting a list
numbers = [5, 2, 8, 1, 9]
numbers.sort()
print("sorted List: ", numbers)
```

```
List:  ['hey', 'you', 1, 2, 3, 'go']
[1, 2, 3, 4, 5, 6]
lenght:  3
sorted List:  [1, 2, 5, 8, 9]
```

Tuple: The tuple is a data type which is a sequence of data similar to a list. But in tuples the data is write-protected. And we Cannot make changes in it. Data in a tuple is written using parenthesis and commas.

In [21]:
```
#Tuple having multiple type of data.
b=("hello", 1,2,3,"go")
print("Tuple:", b)
```

```
Tuple: ('hello', 1, 2, 3, 'go')
```

Sets: Sets are used to store multiple items in a single variable. It is a collection that is unordered, and these dont have indexes. We can remove and add items in it but we cannot change its items.

In [33]:
```
#Sets
set_var = {"apple", "banana", "cherry"}
print("Set: ", set_var)

set1 = {1, 2, 3, 4, 5}
set2 = {4, 5, 6, 7, 8}

# Union of two sets
union_set = set1.union(set2)
print("union: ",union_set)

# Intersection of two sets
intersection_set = set1.intersection(set2)
print("intersection: ",intersection_set)
```

```
Set:  {'cherry', 'banana', 'apple'}
union:  {1, 2, 3, 4, 5, 6, 7, 8}
intersection:  {4, 5}
```

Dictionaries: A dictionary is a collection which is changeable and do not allow duplicates. Dictionaries are written with curly brackets, and have keys and values. We can also add the lists inside a dictionary.

In [23]:
```
#Dictionary
dict_var = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print("Dictionary: ", dict_var)

#Dictionary containing of a List
dict_var = {
  "brand": "Ford",
  "electric": False,
```

```
  "year": 1964,
  "colors": ["red", "white", "blue"]
}

#Accessing the values in dictionary using keys
dict_var = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(dict_var["brand"])
```

```
Dictionary:  {'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
Ford
```

4. Python Conditions and Loops

Python conditions allows to perform different actions based on the truth value of a condition. The syntax involves an "if" statement followed by a condition, and an "else" statement. If the condition is true, the code inside the "if" block executes; otherwise, the code inside the "else" block (if present) executes.

In [37]:
```
# If statement
x = 10
if x > 5:
    print("x is greater than 5")
elif x == 5:
    print("x is equal to 5")
else:
    print("x is less than 5")


#Example 2
x = 10
y = 5
if x > 5 and y < 10:
    print("Both conditions are true")

if x > 10 or y < 5:
    print("At least one condition is true")

if not x == y:
    print("x is not equal to y")
```

```
x is greater than 5
Both conditions are true
x is not equal to y
```

Loops in Python allow you to repeatedly execute a block of code until a certain condition is met. "For" loop iterates over a sequence and executes the block of code for each item in the sequence. "While" loop continues executing the block of code as long as a specified condition remains true. It is suitable when the number of iterations is unknown or based on a specific condition.

In [38]:
```
# For loop
fruits = ['apple', 'banana', 'cherry']
for fruit in fruits:
    print(fruit)

#Example 2
for num in range(1, 6):
    print(num)
```

```
apple
banana
cherry
1
2
3
4
5
```

In [39]:
```
# While loop
count = 0
while count < 5:
    print(count)
    count += 1
```

```
0
1
2
3
4
```

5.Python Functions, Lambda

A function is a block of reusable code that performs a specific task. It allows you to organize your code into logical units and make it more modular and manageable. Functions help promote code reuse, readability, and maintainability.

In [45]:
```
def greet(name):
    print("Hello " + name)

greet("Uzair")

#function2
def add_numbers(a, b):
    sum = a + b
    return sum

result = add_numbers(13, 5)
print("sum: ",result)
```

```
Hello Uzair
sum:  18
```

Lambda functions, also known as anonymous functions, are compact functions defined in a single line. They are useful for creating small, one-time use functions without a formal

function definition.

In [46]:
```
double = lambda x: x * 2

print(double(5))
```

```
10
```

6.Python Classes/Objects:

Classes are blueprints for creating objects. They define attributes (data) and methods (functions) that belong to the objects. Objects are instances of classes and can have unique data and behavior.

In [48]:
```
class Car:
    def __init__(self, brand):
        self.brand = brand

    def drive(self):
        print("I Drive the " + self.brand)

my_car = Car("Toyota")
my_car.drive()
```

```
I Drive the Toyota
```

In [54]:
```
class Car:
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year

    def start_engine(self):
        print("The engine is running.")

my_car = Car("Toyota", "Camry", 2022)
print(my_car.make)
print(my_car.model)
print(my_car.year)
my_car.start_engine()
```

```
Toyota
Camry
2022
The engine is running.
```

7.Python Inheritance Python Iterators

Inheritance is a fundamental concept in object-oriented programming that allows you to create new classes (derived classes) based on existing classes (base classes). The derived classes inherit attributes and methods from the base classes, enabling code reuse and the creation of class hierarchies. In Python, inheritance is implemented using the syntax class

DerivedClass(BaseClass):, where DerivedClass is the new class being created, and BaseClass is the existing class from which it inherits.

In [60]:
```
#Inheritance

class Animal:
    def __init__(self, name):
        self.name = name

    def sound(self):
        pass

class Dog(Animal):
    def sound(self):
        return "Woof!"

class Cat(Animal):
    def sound(self):
        return "Meow!"

dog = Dog("Sandy")
cat = Cat("Whiskers")

print(dog.name)
print(dog.sound())
print(cat.name)
print(cat.sound())
```

```
Sandy
Woof!
Whiskers
Meow!
```

In Python, an iterator is an object that implements the iterator protocol, which consists of the **iter**() and **next**() methods. Iterators are used to iterate over a collection of elements or to generate a sequence of values on-the-fly.

The **iter**() method returns the iterator object itself. It is called when an iterator is initialized or reset. The **next**() method returns the next element in the sequence. It is called repeatedly until it raises the StopIteration exception, indicating that there are no more elements to be returned

In [59]:
```
class MyIterator:
    def __init__(self, start, end):
        self.current = start
        self.end = end

    def __iter__(self):
        return self

    def __next__(self):
        if self.current > self.end:
            raise StopIteration
```

```
        else:
            self.current += 1
            return self.current - 1

my_iterator = MyIterator(1, 5)

for num in my_iterator:
    print(num)
```

```
1
2
3
4
5
```

8. Python Modules:

Modules are files containing Python code that can be imported and used in other programs. They help organize code, promote reusability, and provide a way to access pre-defined functions, classes, and variables. Modules allow you to break down your code into logical units and make it more modular and manageable. You can import modules into your Python programs and use their defined functions, classes, and variables.

There are two types of modules in Python:

Built-in Modules: These modules come pre-installed with the Python interpreter and provide a wide range of functionality.

User-defined Modules: These modules are created by users to encapsulate their own code or functionality. We can create a module by saving your Python code in a separate file with a .py extension. This file becomes a module that can be imported and used in other Python scripts.

In [61]:
```
#This is how we import a library/Module in python
import math

print(math.sqrt(25))
```

```
5.0
```

In [63]:
```
from math import sqrt, pi

# Using the imported functions directly
print(pi)
```

```
3.141592653589793
```

9. Python Try Except:

The try and except statements are used for error handling and exception handling. They allow you to catch and handle specific exceptions that may occur during the execution of

your code, preventing your program from crashing or displaying error messages to the user.

1. The code within the try block is executed. If an exception occurs within this block, the execution is immediately transferred to the corresponding except block.
2. The except block specifies the type of exception it can handle. You can catch specific exceptions (e.g., ValueError, TypeError) or use a more general Exception to catch any exception.
3. If an exception occurs that matches the specified type in the except block, the code within that block is executed to handle the exception. You can include error handling logic, display error messages, or perform any necessary actions.
4. If no exception occurs within the try block, the except block is skipped, and the program continues with the code following the except block.

In [68]:
```
try:
    # User input
    number = int(input("Enter a number: "))

    # Divide by zero to cause an exception
    result = 10 / number

    # Output the result
    print("Result:", result)

except DivisionError:
    print("Error: Division by zero is not allowed.")

except ValueError:
    print("Error: Invalid input. Please enter a valid number.")

except Exception as e:
    print("An error occurred:", str(e))
```

```
Enter a number: 6
Result: 1.6666666666666667
```

10. Python User Input and String Formatting:

User input can be obtained using the input() function, and string formatting allows us to create formatted strings with dynamic content.

User Input: The input() function is used to prompt the user for input. It displays a prompt message to the user, waits for input, and returns the entered value as a string

In [70]:
```
name = input("Enter your name: ")
print("Hello", name)
```

```
Enter your name: Uzair
Hello Uzair
```

String formatting allows us to create dynamic strings by inserting values into placeholders within the string. Python provides multiple ways to format strings, including:

Using the % operator: This is an older formatting method that uses the % operator as a placeholder for values. Using the str.format() method: This method provides more flexibility. Using f-strings (formatted string literals): This is new and concise way to format strings.

In [74]:
```
# Using the % operator
name = "Uzair"
age = 25
print("My name is %s and I am %d years old." % (name, age))

# Using the str.format() method
name = "Uzair"
age = 25
print("My name is {} and I am {} years old.".format(name, age))

# Using f-strings
name = "Uzair"
age = 25
print(f"My name is {name} and I am {age} years old.")
```

```
My name is Uzair and I am 25 years old.
My name is Uzair and I am 25 years old.
My name is Uzair and I am 25 years old.
```