# DS 542 assignment 02 - Mir Ahmed

## 1 .What will be the output of the following code?

In [1]:
```
print(type(1/2))
```

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

## 2. Define the two components of an object with the help of an example.

An object consists of :

Attributes: It reflects the properties of an object. \ Behavior: It is represented by the methods of an object. It also reflects the response of an object to other objects.

In [2]:
```
from datetime import date
class Person:
    "This is a person class"
    name = input("Enter your Name:")
    age = int(input("Enter your age:"))
    birth_year = date.today().year - age
    def greet(self):
        print(f"Hello {self.name}!")
        print(f"You were born in {self.birth_year}!")

# create a new object of Person class called "mir"
student = Person()
```

```
Enter your Name:Mir
Enter your age:32
```

In [3]:
```
#Attribute of the object
student.age
```

Out[3]:
```
32
```

In [4]:
```
# Behavior/ function of the object - Calling object's method
student.greet()
```

```
Hello Mir!
You were born in 1989!
```

## 3. Explain the difference between '/' and '//' operators with the help of examples.

'/' is used for float division. so the result will be a float. \ '//' is used for integer division. so the result will be an integer (rounded down).

In [5]:
```
# Scenario 1
var1 = 5
var2 = 2
# Integer division
print(var1//var2)
# Float division
print(var1/var2)
```

```
2
2.5
```

In [6]:
```
# Scenario 2
var1 = 4
var2 = 2
# Integer division
print(var1//var2)
# Float division
print(var1/var2)
```

```
2
2.0
```

In [7]:
```
# Scenario 4
var1 = 8
var2 = 3
# Integer division
print(var1//var2)
# Float division
print(var1/var2)
```

```
2
2.6666666666666665
```

## 4. Explain operator precedence rules in Python with the help of examples.

Python Operators Precedence Rule – PEMDAS \ P – Parentheses.\ E – Exponentiation.\ M – Multiplication.\ D – Division.\ A – Addition.\ S – Subtraction.

let's say we have the following problem\ (10+14) */3 5 + 2* **3 -6\ So python would first perform (10+14).\ Then it would perform 2** 3.\ *Then it would perform (10+14)*5.\ Then it would perform (10+14)*5/3. This will be a float.\ Then it would perform (10+14)*5/3 + 2* **3.\ Then it would perform (10+14)*5/3 + 2** 3 -6.\ And give us the output of 42.0.

In [8]:
```
(10+14)/3*5+2**3-6
```

Out[8]:
```
42.0
```

## 5. When do you use double quotes while performing string operations in Python? Explain with examples.

The choice between both the types (single quotes and double quotes) depends on the programmer's choice. Generally, double quotes are used for string representation and single quotes are used for regular expressions, dict keys or SQL. Hence both single quote and double quotes depict string in python but it's sometimes our need to use one type over the other.

Primarily, when we have single quote(s) in our string then we must use double quotes to wrap them.\ And when we have double quote(s) in our string then we must use single quotes to wrap them.

In [9]:
```
# Using double quotes
print("It's python")
```

```
It's python
```

In [10]:
```
# Using single quotes
print('"WithQuotes"')
```

```
"WithQuotes"
```

If we do not follow this then we would get invalid syntax error.

## 6. Explain the immutability of strings in Python with the help of examples.

In Python, strings are made immutable so that programmers cannot alter the contents of the object (even by mistake). This avoids unnecessary bugs.\ \ Some other immutable objects are integer, float, tuple, and bool.

In [11]:
```
## The following code will give you a type error
# name_1 = "Mir"
# name_1[0] = 'S'


# ---------------------------------------------------------------------------
# TypeError                                 Traceback (most recent call last)
# <ipython-input-11-82e3b2454340> in <module>
#       1 name_1 = "Mir"
# ----> 2 name_1[0] = 'S'

# TypeError: 'str' object does not support item assignment
```

In [12]:
```
# One possible solution is to create a new string object with necessary modifications:
name_1 = "Mir"
name_2 = "S"+ name_1[1:]
print("name_1 = ", name_1, "and name_2 = ", name_2)
```

```
name_1 =  Mir and name_2 =  Sir
```

In [13]:
```
# To identify that they are different strings, check with the id() function
print("id of name_1 = ", id(name_1))
print("id of name_2 = ", id(name_2))
```

```
id of name_1 =  2810055562288
id of name_2 =  2810055562800
```

In [14]:
```
# To understand more about the concept of string immutability, consider the following code:
name_1 = "Mir"
name_2 = "Mir"
print("id of name_1 = ", id(name_1))
print("id of name_2 = ", id(name_2))
```

```
id of name_1 =  2810055562288
id of name_2 =  2810055562288
```

When the above lines of code are executed, you will find that the id's of both name_1 and name_2 objects, which refer to the string "Mir", are the same.

In [15]:
```
# To dig deeper, execute the following statements:
name_1 = "Mir"
print("id of name_1 = ", id(name_1))

name_1 = "Sir"
print("id of name_1 afer initialing with new value = ", id(name_1))
```

```
id of name_1 =  2810055562288
id of name_1 afer initialing with new value =  2810055542384
```

As can be seen in the above example, when a string reference is reinitialized with a new value, it is creating a new object rather than overwriting the previous value.

## 7. What tools can you use to find a number's square root, as well as its square?

In [16]:
```
# we can use '**' operator to calculate exponentials as well as roots
# calculating square root
print(4**.5) # or print(4**(1/2))
# calculating square
print(4**2)
```

```
2.0
16
```

## 8. How can you truncate and round a floating-point number?

In [17]:
```
# by integer converter function which will round down the value of the varibale
var = 5.6
print(f" old value = {var}")
print(f" type of old value = {type(var)}")
print("----------------------------------")
var = int(var)
print(f" new value = {var}")
print(f" type of new value = {type(var)}")
```

```
 old value = 5.6
 type of old value = <class 'float'>
----------------------------------
 new value = 5
 type of new value = <class 'int'>
```

In [18]:
```
# by round function which will either round up or down based on the value of the varibale
# scenario 1
var = 5.5
print(f" old value = {var}")
print(f" type of old value = {type(var)}")
print("----------------------------------")
var = round(var)
print(f" new value = {var}")
print(f" type of new value = {type(var)}")
```

```
 old value = 5.5
 type of old value = <class 'float'>
----------------------------------
 new value = 6
 type of new value = <class 'int'>
```

In [19]:
```
# scenario 2
var = 5.4
print(f" old value = {var}")
print(f" type of old value = {type(var)}")
print("----------------------------------")
var = round(var)
print(f" new value = {var}")
print(f" type of new value = {type(var)}")
```

```
 old value = 5.4
 type of old value = <class 'float'>
----------------------------------
 new value = 5
 type of new value = <class 'int'>
```

## 9. How can you convert an integer to a floating-point number?

In [20]:
```
var = 5
print(f" old value = {var}")
print(f" type of old value = {type(var)}")
print("----------------------------------")
var = float(var)
print(f" new value = {var}")
print(f" type of new value = {type(var)}")
```

```
 old value = 5
 type of old value = <class 'int'>
----------------------------------
 new value = 5.0
 type of new value = <class 'float'>
```

## 10. Consider the following three statements. Do they change the value printed for A? Explain.

A = "spam" \ B= A\ B = "shrubbery"

In [21]:
```
A = "spam"
B= A
B = "shrubbery"
```

In [22]:
```
print(A)
```

```
spam
```

Python interpreter exceutes the block of codes line by line.\ In the first line, we assigned a new variable 'A'.\ In the second line, we are assigned a new variable 'B' with the value of 'A'.\ In the third line, we changed the value of 'B'.\ Code execution ends here.\ This is why the value of 'A' does not change as there are no other commands past the third line to manipulate the value of 'A'.