Python Object and Data Structures

1- Integers (int): Whole numbers without decimal points.

2- Floating-Point Numbers (float):* Numbers with decimal points.

3- Strings (str): Ordered sequences of characters.

4- Lists (list): Ordered, mutable collections of objects.

5- Tuples (tuple): Ordered, immutable collections of objects.

6- Dictionaries (dict): Unordered collections of key-value pairs.

7- Sets (set): Unordered collections of unique elements.

8- Booleans (bool): Represents True or False values.

9- NoneType (None): Represents the absence of a value or a null value.

These are the fundamental data structures in Python, and they play a crucial role in storing and manipulating data within your programs. Objects are instances of these data structures, and they can be manipulated using various methods and operations.

Mutable:*

Lists (list)

Dictionaries (dict)

Sets (set)

Immutable:

Integers (int)

Floating-Point Numbers (float)

Strings (str)

Tuples (tuple)

Booleans (bool)

NoneType (None)

Mutable data structures can be modified after they are created, while immutable data structures cannot be changed once they are created.

1- Python and Numbers

There are two main number types in Python:

• Integers which are whole numbers

• Numbers with decimals are known as floating point numbers.

In [1]:
```
# Assigning value to variables!!
Allah_names = 99
```

In [3]:
```
# Dynamic typing in Python
Uzair_creator = 1
```

In [5]:
```
Uzair_creator = ['cology' , 'pharma']
```

In [7]:
```
Float_num = 87.76
Float_num
```

Out[7]: 87.76

2- Python and Strings

Strings are sequences of characters, written either in single, or double quotes.

In [9]:
```
'Pharmacy is good'
"pharma is very good"
```

Out[9]:
```
'pharma is very good'
```

2.1- Strings Slicing and Indexing

INDEXING String are written in an order sequence, which means we can index them and slice them to grab a certain sub-section. Indexing uses [] after the string (or variable assigned the string). Indexing allows us to grab a single character from a string

Slicing in Python Slice allows us to grab a subsection of multiple characters, just like a slice of bread or slices of bread. The syntax is [start:stop:step] start is a numerical index for the slice stop is the index we will go up to (but not include) step is the size of the “jump” we make

2.2- String interpolation?

String interpolation is a process substituting values of variables into placeholders in a string.

In [10]:
```
name = "Uzair"
age = 23

# Using f-string for string interpolation
message = f"My name is {name} and I am {age} years old."

print(message)
```

```
My name is Uzair and I am 23 years old.
```

3- Lists and dictionaries

Lists in Python

• In Python Lists are ordered sequences that can hold a variety of object types.

• We can use [] brackets and commas to separate objects in the list for example [1,2,3,4,5,6,7,8]

• Lists can also be nested that we covered in Python Alpha.

Dictionaries in Python

A dictionary is a collection which is unordered, changeable and indexed. In Python dictionaries are written with curly brackets, and they have keys and values.

• Dictionaries use a key-value pairing instead {‘key1’:’value1, ’key2’:’value2’}

• The key value pair allows users to grab objects without needing to know an index location

4- Tuples, sets and Booleans in Python

Tuples in Python • Tuples are similar to lists. However lists are mutable and tuples are not i.e. immutable

• List can be changed tuples cannot be changed.

• Tuples are sequences just like lists.

• Elements inside tuple cannot be reassigned

• We use parenthesis ()

• The empty tuple is written as two parentheses containing nothing

• Like string index's, tuple index's start at 0, and they can be sliced, concatenated, and so on.

Sets in Python

• Sets are unordered and unindexed collection of unique elements.

• Meaning there can only be one representative of the same object

Booleans in Python

• Boolean values define two constant objects which can be either True or False

• This logic is key to control flow and other use cases.

5- Input and Output in Python

you can use the input() function to take user input and the print() function to display output

In [12]:
```
# example
def greet_user():
    name = input("Enter your name: ")
    print(f"Hello, {name}! Welcome to the Python world.")

# Call the function
greet_user()
```

```
Enter your name: Uzair Shafique
Hello, Uzair Shafique! Welcome to the Python world.
```

6- Python Boolean Operators

Boolean operators in Python are used to combine and manipulate boolean values (True or False). The main boolean operators are and, or, and not

and

Returns True if both operands are True, otherwise returns False.

or

Returns True if at least one of the operands is True, otherwise returns False.

not Operator

Returns the opposite of the operand's boolean value. python

In [13]:
```
# example and Operator

x = True
y = False
result = x and y
print(result)  # Output: False
```

```
False
```

In [14]:
```
# example or operator
a = True
b = False
result = a or b
print(result)  # Output: True
```

```
True
```

In [15]:
```
# example not operator
value = False
result = not value
print(result)  # Output: True
```

```
True
```

7- if, elif and else*

The if..else statement evaluates test expression and will execute body of if only when test condition is True . If the condition is False , body of else is executed. Indentation is used to separate the blocks.

The key statements are

if elif else

In [16]:
```
age = int(input("Enter your age: "))

if age < 18:
    print("You are a minor.")
elif age >= 18 and age < 65:
    print("You are an adult.")
else:
    print("You are a senior citizen.")
```

```
Enter your age: 50
You are an adult.
```

8- Loops in python

(i) For 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-orientated programming languages.

(ii) While loop

A while loop statement in Python programming language repeatedly executes a target statement as long as a given condition is true.

In [17]:
```
# Example using a for loop
fruits = ["apple", "banana", "cherry", "date"]

print("Using a for loop:")
for fruit in fruits:
    print(fruit)
```

```
Using a for loop:
apple
banana
cherry
date
```

In [18]:
```python
# Example using a while loop
count = 1

print("Using a while loop:")
while count <= 5:
    print(f"Count: {count}")
    count += 1
```

```
Using a while loop:
Count: 1
Count: 2
Count: 3
Count: 4
Count: 5
```

In [ ]: