PROJECT 08: “PYTHON”

SUBMITTED BY:

Manaswi M. Patil

INDEX

1. INTRODUCTON

Python is an interpreted language, This makes development faster and more interactive. It supports object-oriented programming, is cross-platform compatible, and remains an open-source language. Python's flexibility and ease of use make it a popular choice for both beginners and experienced programmer. Python is simple and easy to understand. It is Interpreted and platform-independent which makes debugging very easy. Python is an open-source programming language. Python provides very big library support. Some of the popular libraries include NumPy, Tensor flow , Selenium, OpenCV, etc. It is possible to integrate other programming languages within python

WHY USED PYTHON?

1. **Data Visualization:** Python helps make cool charts and graphs, showing data in a way that's easy to understand.
2. **Data Analytics:** It's like a detective for data, helping people find hidden patterns and trends in big sets of information.
3. **AI and Machine Learning:** Python makes computers learn and act smart by studying data, almost like how we learn from experience.
4. **Web Applications:** Python is like a builder for the web. It helps create websites and online tools that we use every day.
5. **Database Handling:** Think of Python as a librarian for data. It helps organize and manage large amounts of information.
6. **Business and Accounting:** Python is like a super-smart calculator. It helps businesses with complicated math and analysis, making smart decisions.

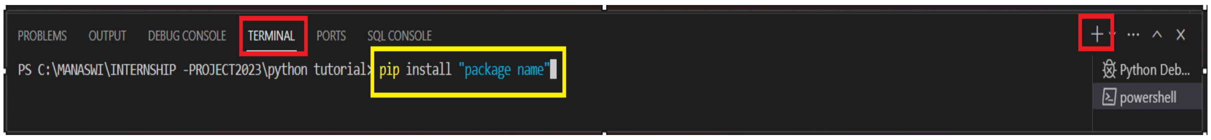
INSTALLATION STEPS ARE:

Open chrome browser→ and install latest version of python→ click on following link https://www.python.org/downloads/ .

After installation check the version which you has install→ 'python –version' .

How to install packages in python?

➢ To install packages in Python, we use the pip command.



```
PS C:\MANASWI\INTERNSHIP -PROJECT2023\python tutorial> pip install "package name"
```

- pip show command to display information about the installed package. ‘comment’ is used to explain the functioning of a block of code. It starts with a ‘#’.

What are the comments?

A comment in code is like a note for the computer. It's a message just for the human who wrote the code, explaining things or telling the computer to skip a particular part when running the program. It's like leaving a helpful hint for yourself or others who might read the code.

Single line comments:

To write a comment just add a ‘#’ at the start of the line.

Multi-Line Comments:

To write multi-line comments you can use ‘#’ at each line or you can use the multiline string.

2. PYTHON VARIABLES

Variables are used to store and manage data.

1. **Variable Naming Rules:** Variable names can contain letters (a-z, A-Z), numbers (0-9), and underscores (_).They cannot start with a number. Python is case-sensitive, so **myVar** and **myvar** are different variables.
2. **Variable Assignment:** You can assign a value to a variable using the = operator. The data type of the variable is determined dynamically based on the assigned value.
3. Data Types: Python has several built-in data types, including integers, floats, strings, booleans, lists, tuples, dictionaries, etc.You don't need to declare the data type explicitly; Python infers it based on the assigned value.
4. Variable Reassignment: You can change the value of a variable by assigning a new value to it.
5. Multiple Assignments: You can assign multiple variables in a single line.
6. Constants: Although Python doesn't have constants in the strict sense, it's a convention to use uppercase names for variables that are not intended to be changed.
7. Print Variables: You can print the value of a variable using the print function.
8. Variable Scope: The scope of a variable determines where in the code it can be accessed. Variables can be local (inside a function) or global (accessible throughout the program).

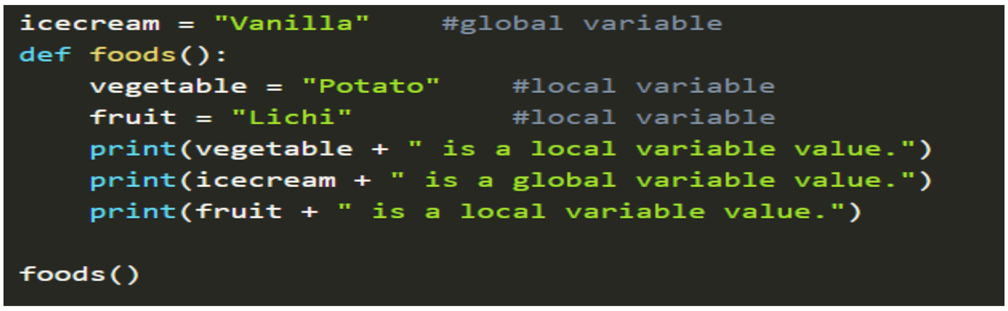
Rules for variables:

- Names cannot starts with a number.
- There can be no spaces between names use '_' underscore instead.
- Cant use any symbol like :,"", // ,etc.
- Avoid words which has special meaning in python eg. Str, list. And so on.

Scope of variable:

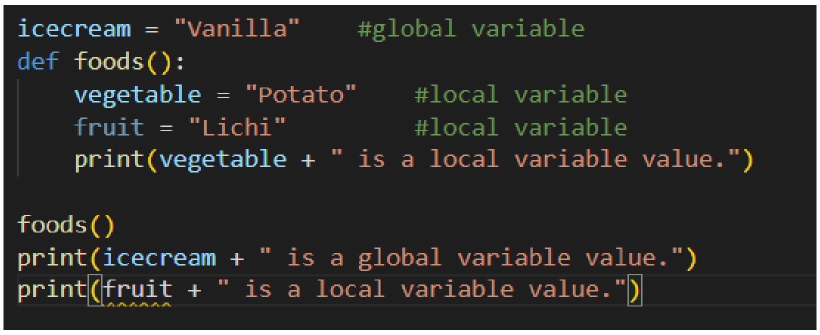
Variable scope refers to the region of a program where a particular variable can be accessed or modified.

Local Scope: A variable defined inside a function has a local scope. It is accessible only within that function and cannot be accessed outside of it.



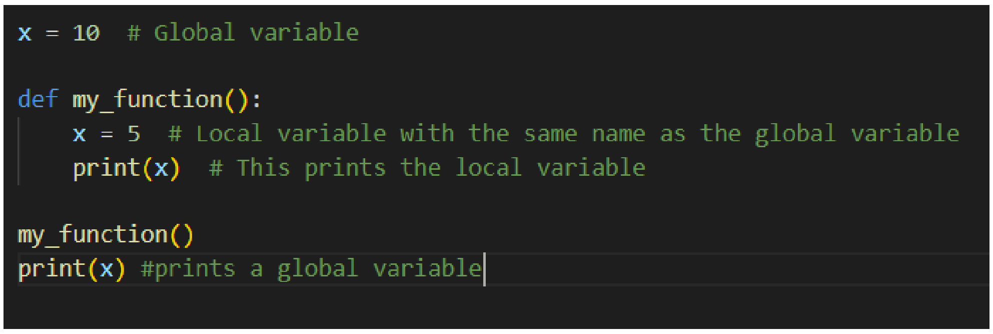
```python
icecream = "Vanilla"     #global variable
def foods():
    vegetable = "Potato"     #local variable
    fruit = "Lichi"          #local variable
    print(vegetable + " is a local variable value.")
    print(icecream + " is a global variable value.")
    print(fruit + " is a local variable value.")

foods()
```

Global Scope: A variable defined outside any function or block has a global scope. It can be accessed from anywhere in the program, including inside functions.



```python
icecream = "Vanilla"    #global variable
def foods():
    vegetable = "Potato"    #local variable
    fruit = "Lichi"         #local variable
    print(vegetable + " is a local variable value.")

foods()
print(icecream + " is a global variable value.")
print(fruit + " is a local variable value.")
```

Local vs. Global: When a variable is defined both globally and locally with the same name, the local variable takes precedence within its scope.



```python
x = 10  # Global variable

def my_function():
    x = 5  # Local variable with the same name as the global variable
    print(x)  # This prints the local variable

my_function()
print(x) #prints a global variable
```

3. PYTHON DATATYPES AND OPERATORS

Python supports several built-in data types, and each type is used to represent different kinds of information.

1. **Numeric Types:**

- **int:** Integer type represents whole numbers without any decimal point. Eg. x = 5
- **float:** Float type represents numbers with a decimal point. Eg. y = 3.14
- **complex:** Complex type represents complex numbers with real and imaginary parts.

Eg. z = 2 + 3j

2. **Text Type:**

- **str:** String type represents sequences of characters enclosed in single or double quotes.

Eg. text = "Hello, Python!"

3. **Boolean Type:**

- **bool:** Boolean type represents either True or False.eg. is_true = True is_false = False

4. **Sequence Types:**

- **list:** List is an ordered, mutable collection of elements.eg. my_list = [1, 2, 3, "four"]
- **tuple:** Tuple is an ordered, immutable collection of elements. Eg. my_tuple = (1, 2, 3, "four")
- **range:** Range represents an immutable sequence of numbers. Eg. my_range = range(5)

5. **Set Types:**

- **set:** Set is an unordered, mutable collection of unique elements.eg. my_set = {1, 2, 3, 3, 4}
- **frozenset:** Frozenset is an immutable set.eg. my_frozenset = frozenset([1, 2, 3, 3, 4])

6. **Mapping Type:**

- **dict:** Dictionary is an unordered collection of key-value pairs.

Eg. my_dict = {"name": "John", "age": 25, "city": "New York"}

7. **None Type:**

- **NoneType:** The None type represents the absence of a value or a null value.

Eg. my_variable = None

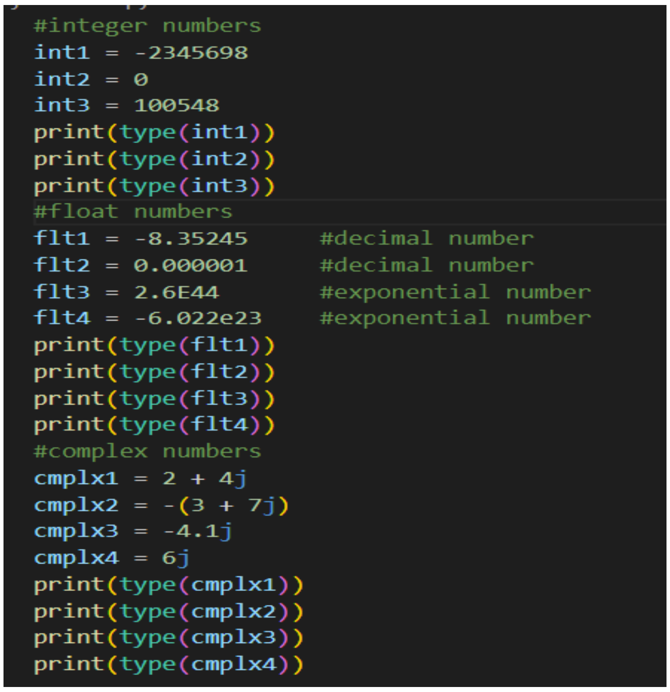
Python numbers

Python, numbers are represented by various numeric data types. The primary numeric types include integers (**int**), floating-point numbers (**float**), and complex numbers (**complex**).

Integers (int): Integers represent whole numbers without any decimal point.

Floating-point Numbers (float): Floating-point numbers represent real numbers and include a decimal point.

Complex Numbers (complex): Complex numbers have a real and an imaginary part and are represented as **a + bj**, where **a** is the real part, **b** is the imaginary part, and **j** is the imaginary unit (equal to the square root of -1).



```python
#integer numbers
int1 = -2345698
int2 = 0
int3 = 100548
print(type(int1))
print(type(int2))
print(type(int3))
#float numbers
flt1 = -8.35245     #decimal number
flt2 = 0.000001     #decimal number
flt3 = 2.6E44       #exponential number
flt4 = -6.022e23    #exponential number
print(type(flt1))
print(type(flt2))
print(type(flt3))
print(type(flt4))
#complex numbers
cmplx1 = 2 + 4j
cmplx2 = -(3 + 7j)
cmplx3 = -4.1j
cmplx4 = 6j
print(type(cmplx1))
print(type(cmplx2))
print(type(cmplx3))
print(type(cmplx4))
```

Data conversion :

Data conversion refers to the process of converting one data type into another.

- To convert from integer to float, we use float() function.
- To convert from integer to complex, we use the complex() function.
- To convert from float to integer, we use int() function.
- To convert from float to complex, we use the complex() function.

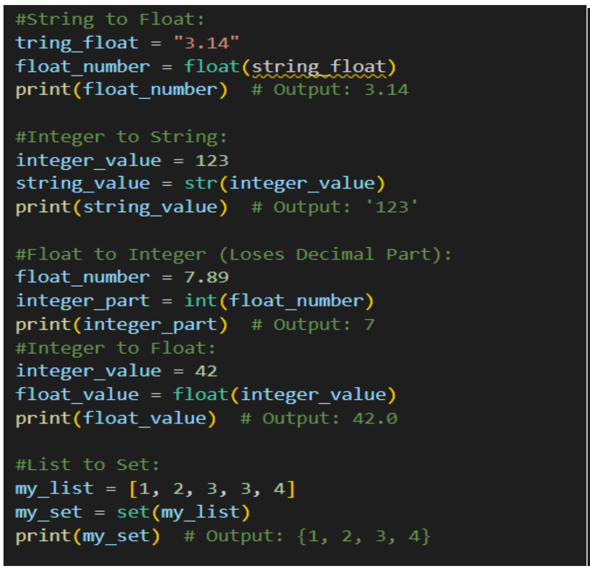
Complex numbers in Python cannot be directly converted to integers or floats because complex numbers have two components: a real part and an imaginary part.

Type casting:

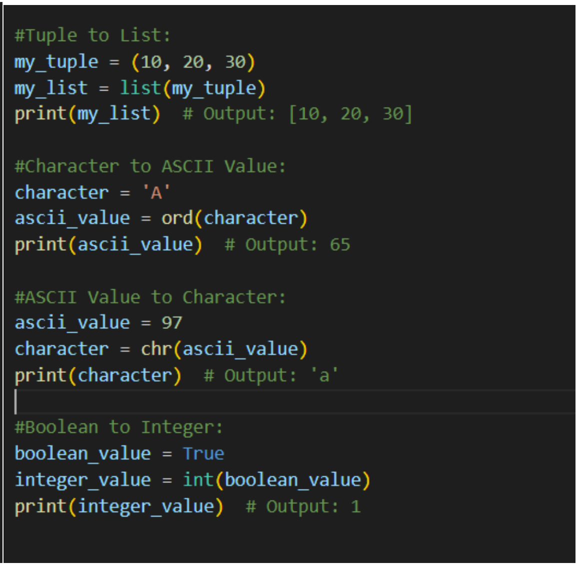
Type casting in Python refers to the process of converting a variable or value from one data type to another.

- String to Integer
- String to Float
- Integer to String
- Float to Integer (Loses Decimal Part)
- Integer to Float

- List to Set
- Tuple to List
- Character to ASCII Value
- ASCII Value to Character
- Boolean to Integer



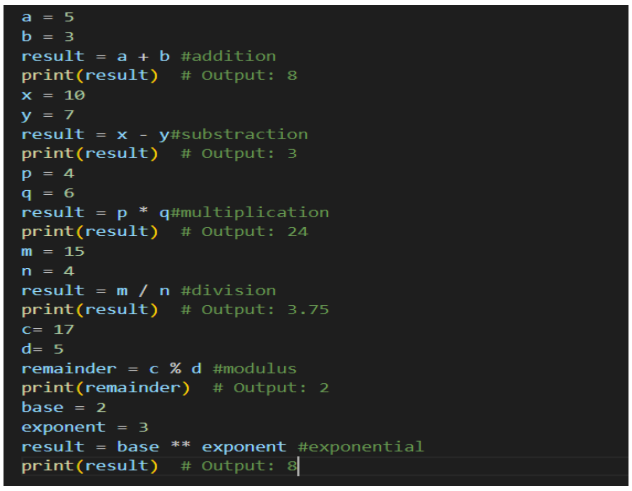
```
#String to Float:
tring_float = "3.14"
float_number = float(string_float)
print(float_number)  # Output: 3.14

#Integer to String:
integer_value = 123
string_value = str(integer_value)
print(string_value)  # Output: '123'

#Float to Integer (Loses Decimal Part):
float_number = 7.89
integer_part = int(float_number)
print(integer_part)  # Output: 7
#Integer to Float:
integer_value = 42
float_value = float(integer_value)
print(float_value)  # Output: 42.0

#List to Set:
my_list = [1, 2, 3, 3, 4]
my_set = set(my_list)
print(my_set)  # Output: {1, 2, 3, 4}
```



```
#Tuple to List:
my_tuple = (10, 20, 30)
my_list = list(my_tuple)
print(my_list)  # Output: [10, 20, 30]

#Character to ASCII Value:
character = 'A'
ascii_value = ord(character)
print(ascii_value)  # Output: 65

#ASCII Value to Character:
ascii_value = 97
character = chr(ascii_value)
print(character)  # Output: 'a'

#Boolean to Integer:
boolean_value = True
integer_value = int(boolean_value)
print(integer_value)  # Output: 1
```

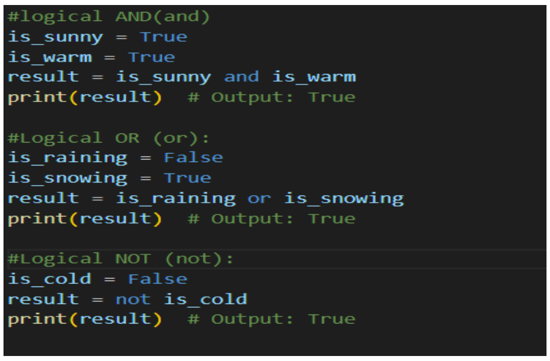
Python operators:

- **Arithmetic Operators:**
  - + (Addition): Adds two numbers together.
  - - (Subtraction): Subtracts the right operand from the left operand.
  - * (Multiplication): Multiplies two numbers.
  - / (Division): Divides the left operand by the right operand.
  - % (Modulus): Returns the remainder of the division of the left operand by the right operand.
  - ** (Exponent): Raises the left operand to the power of the right operand.



```
a = 5
b = 3
result = a + b #addition
print(result)  # Output: 8
x = 10
y = 7
result = x - y#substraction
print(result)  # Output: 3
p = 4
q = 6
result = p * q#multiplication
print(result)  # Output: 24
m = 15
n = 4
result = m / n #division
print(result)  # Output: 3.75
c= 17
d= 5
remainder = c % d #modulus
print(remainder)  # Output: 2
base = 2
exponent = 3
result = base ** exponent #exponential
print(result)  # Output: 8
```

- **Comparison Operators:**
  - == (Equal): Checks if the values of two operands are equal.
  - != (Not Equal): Checks if the values of two operands are not equal.
  - < (Less Than): Checks if the left operand is less than the right operand.
  - (Greater Than): Checks if the left operand is greater than the right operand.
  - <= (Less Than or Equal To): Checks if the left operand is less than or equal to the right operand.
  - >= (Greater Than or Equal To): Checks if the left operand is greater than or equal to the right operand.
- **Logical Operators:**
  - and (Logical AND): Returns True if both operands are true.
  - or (Logical OR): Returns True if at least one of the operands is true.
  - not (Logical NOT): Returns True if the operand is false, and vice versa.

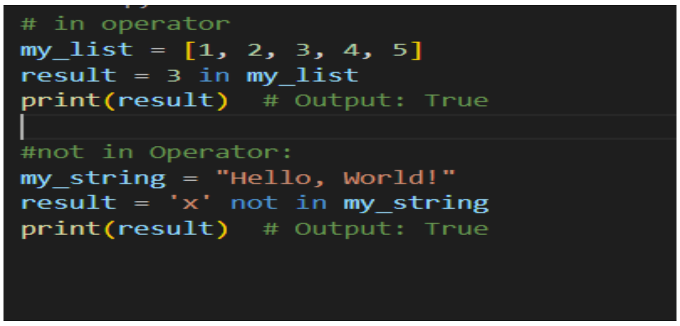


```python
#logical AND(and)
is_sunny = True
is_warm = True
result = is_sunny and is_warm
print(result)  # Output: True

#Logical OR (or):
is_raining = False
is_snowing = True
result = is_raining or is_snowing
print(result)  # Output: True

#Logical NOT (not):
is_cold = False
result = not is_cold
print(result)  # Output: True
```

- **Assignment Operators:**
  - = (Assignment): Assigns the value on the right to the variable on the left.
  - += (Add and Assign): Adds the right operand to the left operand and assigns the result to the left operand.
  - -= (Subtract and Assign): Subtracts the right operand from the left operand and assigns the result to the left operand.
  - *= (Multiply and Assign): Multiplies the left operand by the right operand and assigns the result to the left operand.
  - /= (Divide and Assign): Divides the left operand by the right operand and assigns the result to the left operand.
- **Membership Operators:**

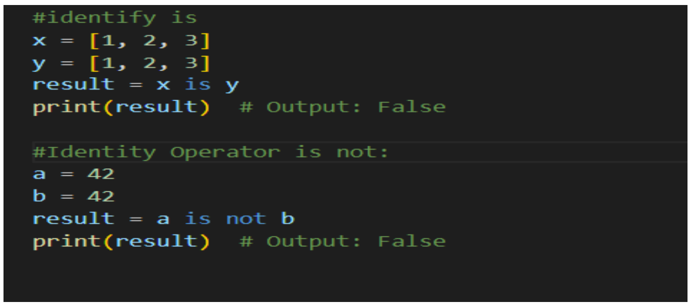
Membership operators in Python are used to test whether a value is a member of a sequence, such as a string, list, tuple, or set.

- in (Membership): Returns True if a specified value is present in a sequence (e.g., a string, list, tuple).
- not in (Not Membership): Returns True if a specified value is not present in a sequence.



```
# in operator
my_list = [1, 2, 3, 4, 5]
result = 3 in my_list
print(result)  # Output: True

#not in Operator:
my_string = "Hello, World!"
result = 'x' not in my_string
print(result)  # Output: True
```

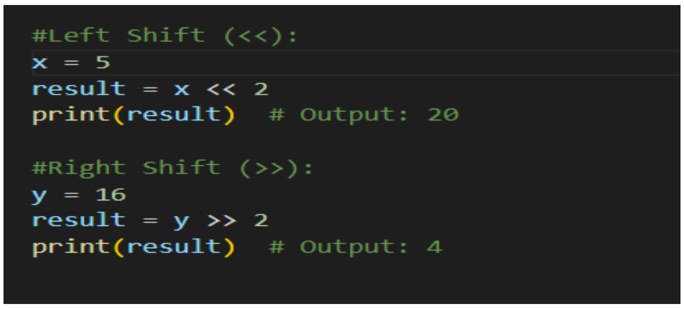
- **Identity Operators:**
  - is (Identity): Returns True if both operands are the same object.
  - is not (Not Identity): Returns True if both operands are not the same object.



```
#identify is
x = [1, 2, 3]
y = [1, 2, 3]
result = x is y
print(result)  # Output: False

#Identity Operator is not:
a = 42
b = 42
result = a is not b
print(result)  # Output: False
```

- **Left and right shift:**

1. Left Shift (<<):
   - The left shift operator (<<) shifts the bits of a binary representation to the left by a specified number of positions, effectively multiplying the integer by 2 raised to the power of the shift count.
2. Right Shift (>>):
   - The right shift operator (>>) shifts the bits of a binary representation to the right by a specified number of positions, effectively dividing the integer by 2 raised to the power of the shift count.



```
#Left Shift (<<):
x = 5
result = x << 2
print(result)  # Output: 20

#Right Shift (>>):
y = 16
result = y >> 2
print(result)  # Output: 4
```

Python Booleans:

bool() function evaluates values and returns True or False.

Boolean Values:

- There are only two boolean values in Python: True and False. These values are case-sensitive, so writing True and False with an uppercase initial letter is important.

Boolean Operations:

Python supports various boolean operations, including logical AND (and), logical OR (or), and logical NOT (not).

Comparison Operators:

Comparison operators return boolean values based on the comparison of two values.

Boolean Conversion:

Values of other types can be implicitly or explicitly converted to booleans. In general, empty or zero values are considered False, and non-empty or non-zero values are considered True.

Truthy and Falsy:

In Python, values that evaluate to False in a boolean context are considered "falsy," while values that evaluate to True are considered "truthy." Falsy values include False, None, 0, empty containers (e.g., empty strings, lists, dictionaries), and instances of custom classes that define a __bool__ or __len__ method that returns False or 0, respectively.

Sets and dictionaries :

Are both data structures in Python .

A set is an unordered collection of unique elements. It is defined using curly braces **{}** or the **set()** constructor. Sets do not allow duplicate elements.

A dictionary is an unordered collection of key-value pairs, where each key must be unique. Dictionaries are defined using curly braces **{}** or the **dict()** constructor.

Tuples:

A tuple in Python is an ordered, immutable collection of elements. It is similar to a list, but once a tuple is created, its elements cannot be modified or changed. Tuples are defined using parentheses () or the tuple() constructor.

4. PYTHON STRINGS

Strings in Python are sequences of characters and are one of the fundamental data types in the language. Strings can be created using single quotes ('), double quotes ("), or triple quotes (''' or """).

Sometimes you want to include a note, or a set of instructions, or just might want to explain a piece of code. Then used ‘"""’ or ‘""’.

Operation on Strings:

Length of a String:

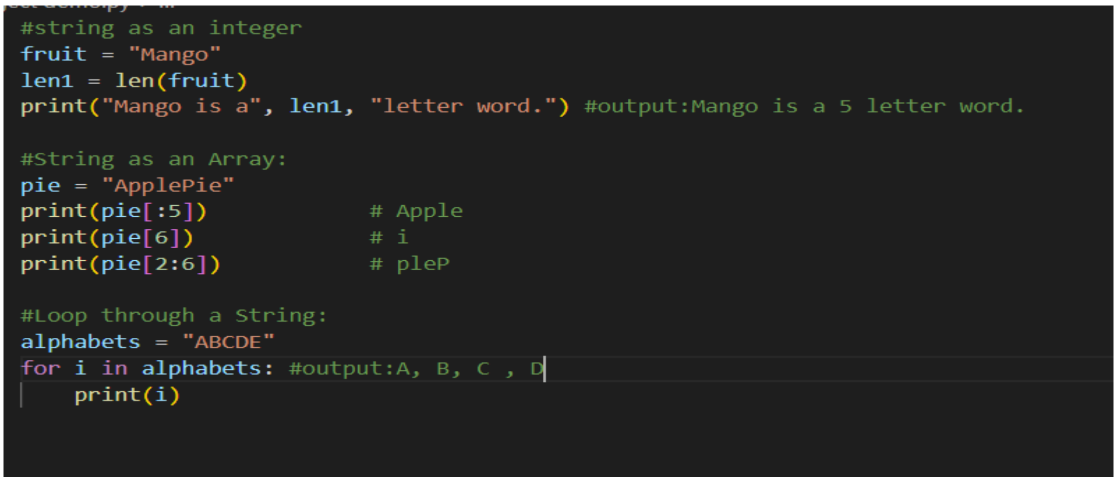
We can find the length of a string using len() function.

String as an Array:

A string is essentially a sequence of characters also called an array. Thus we can access the elements of this array.

Loop through a String:

Strings are arrays and arrays are iterable.



```
#string as an integer
fruit = "Mango"
len1 = len(fruit)
print("Mango is a", len1, "letter word.") #output:Mango is a 5 letter word.

#String as an Array:
pie = "ApplePie"
print(pie[:5])            # Apple
print(pie[6])             # i
print(pie[2:6])           # pleP

#Loop through a String:
alphabets = "ABCDE"
for i in alphabets: #output:A, B, C , D
    print(i)
```

String Methods:

upper() : The upper() method converts a string to upper case.

lower() : The lower() method converts a string to upper case.

strip() : The strip() method removes any white spaces before and after the string.

rstrip() : the rstrip() removes any trailing characters.

replace() : the replace() method replaces a string with another string.

split() : The split() method splits the give string at the specified instance and returns the separated strings as list items.

capitalize() : The capitalize() method turns only the first character of the string to uppercase and the rest other characters of the string are turned to lowercase. The string has no effect if the first character is already uppercase.

center() : The center() method aligns the string to the center as per the parameters given by the user.

count() : The count() method returns the number of times the given value has occurred within the given string.

endswith() : The endswith() method checks if the string ends with a given value. If yes then return True, else return False.

find() : The find() method searches for the first occurrence of the given value and returns the index where it is present. If given value is absent from the string then return -1.

index() : The index() method searches for the first occurrence of the given value and returns the index where it is present. If given value is absent from the string then raise an exception.

isalnum() : The isalnum() method returns True only if the entire string only consists of A-Z, a-z, 0-9. If any other characters or punctuations are present, then it returns False.

isalpha() : The isalnum() method returns True only if the entire string only consists of A-Z, a-z. If any other characters or punctuations or numbers(0-9) are present, then it returns False.

islower() : The islower() method returns True if all the characters in the string are lower case, else it returns False.

isprintable() : The isprintable() method returns True if all the values within the given string are printable, if not, then return False.

isspace() : The isspace() method returns True only and only if the string contains white spaces, else returns False.

istitle() : The istitile() returns True only if the first letter of each word of the string is capitalized, else it returns False.

isupper() : The isupper() method returns True if all the characters in the string are upper case, else it returns False.

replace() : The replace() method can be used to replace a part of the original string with another string.

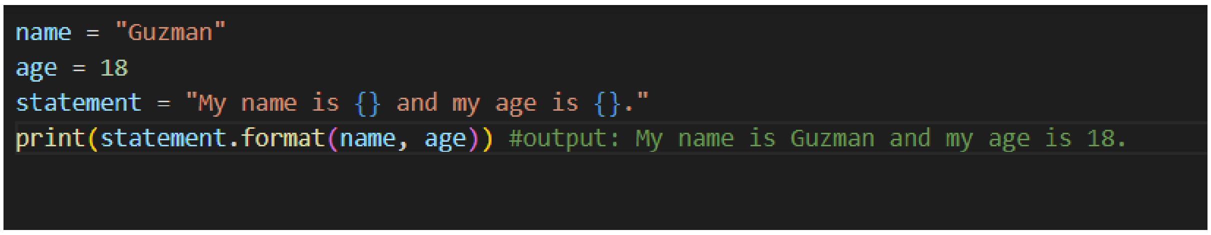
startswith() : The endswith() method checks if the string starts with a given value. If yes then return True, else return False.

swapcase() : The swapcase() method changes the character casing of the string. Upper case are converted to lower case and lower case to upper case**.**

title() : The title() method capitalizes each letter of the word within the string.

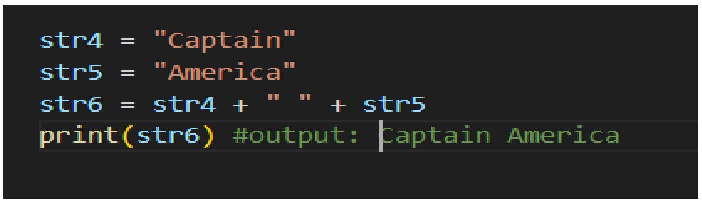
Format Strings:

The format() method allows you to insert values into a string at specified positions. The format() methods places the arguments within the string wherever the placeholders are specified.



```
name = "Guzman"
age = 18
statement = "My name is {} and my age is {}."
print(statement.format(name, age)) #output: My name is Guzman and my age is 18.
```

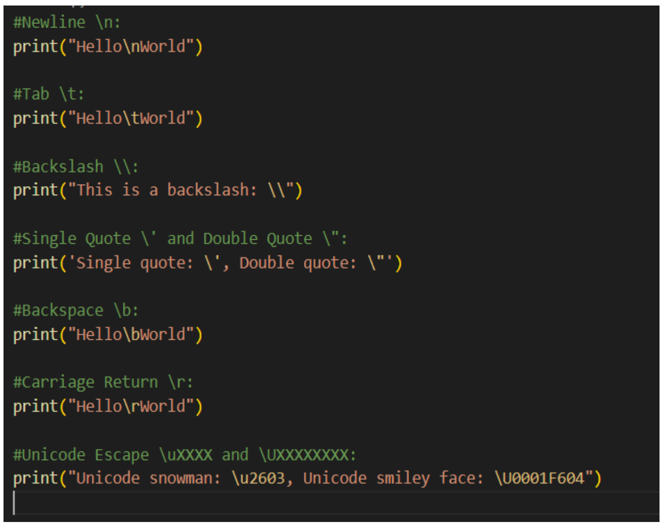
To Join two separate strings



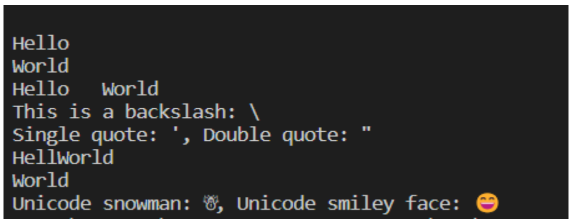
```
str4 = "Captain"
str5 = "America"
str6 = str4 + " " + str5
print(str6) #output: Captain America
```

Escape Characters:

1. **Newline \n:**
2. **Tab \t:**
3. **Backslash \\:**
4. **Single Quote \' and Double Quote \":**
5. **Backspace \b:**
6. **Carriage Return \r:**
7. **Unicode Escape \uXXXX and \UXXXXXXXX:**



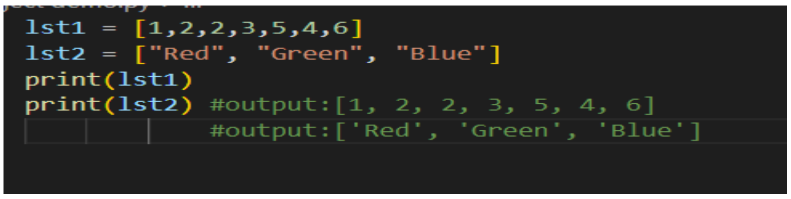
```
#Newline \n:
print("Hello\nWorld")

#Tab \t:
print("Hello\tWorld")

#Backslash \\:
print("This is a backslash: \\")

#Single Quote \' and Double Quote \":
print('Single quote: \', Double quote: \"')

#Backspace \b:
print("Hello\bWorld")

#Carriage Return \r:
print("Hello\rWorld")

#Unicode Escape \uXXXX and \UXXXXXXXX:
print("Unicode snowman: \u2603, Unicode smiley face: \U0001F604")
```



```
Hello
World
Hello	World
This is a backslash: \
Single quote: ', Double quote: "
HellWorld
World
Unicode snowman: ☃, Unicode smiley face: 😄
```

5. PYTHON LISTS

- Lists are ordered collection of data items.
- They store multiple items in a single variable.
- List items are separated by commas and enclosed within square brackets [].
- Lists are changeable meaning we can alter them after creation.
- List are mutable.

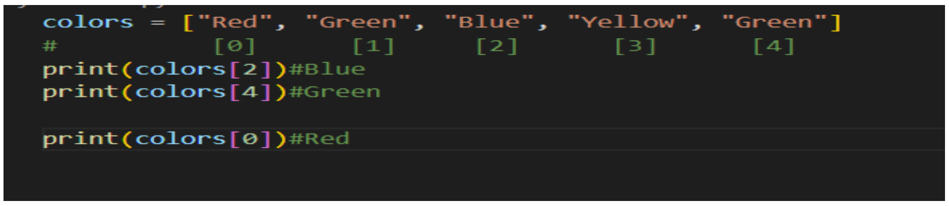


```
lst1 = [1,2,2,3,5,4,6]
lst2 = ["Red", "Green", "Blue"]
print(lst1)
print(lst2) #output:[1, 2, 2, 3, 5, 4, 6]
            #output:['Red', 'Green', 'Blue']
```

List Indexes

Positive Indexing:

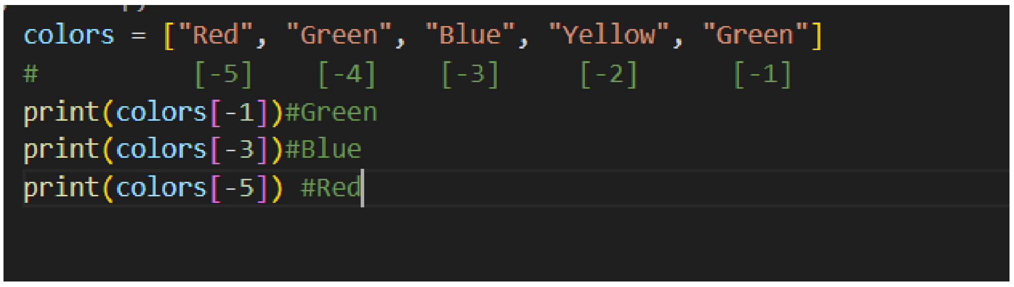
As we have seen that list items have index, as such we can access items using these indexes.



```
colors = ["Red", "Green", "Blue", "Yellow", "Green"]
#          [0]      [1]      [2]     [3]      [4]
print(colors[2])#Blue
print(colors[4])#Green

print(colors[0])#Red
```

Negative Indexing:

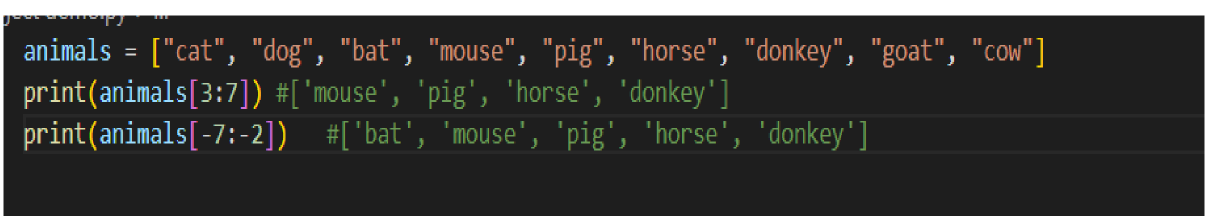
Similar to positive indexing, negative indexing is also used to access items, but from the end of the list. The last item has index [-1], second last item has index [-2], third last item has index [-3] and so on.



```
colors = ["Red", "Green", "Blue", "Yellow", "Green"]
#          [-5]    [-4]    [-3]     [-2]      [-1]
print(colors[-1])#Green
print(colors[-3])#Blue
print(colors[-5]) #Red
```

Range of Index:

You can print a range of list items by specifying where do you want to start, where do you want to end and if you want to skip elements in between the range.



```
animals = ["cat", "dog", "bat", "mouse", "pig", "horse", "donkey", "goat", "cow"]
print(animals[3:7]) #['mouse', 'pig', 'horse', 'donkey']
print(animals[-7:-2])   #['bat', 'mouse', 'pig', 'horse', 'donkey']
```

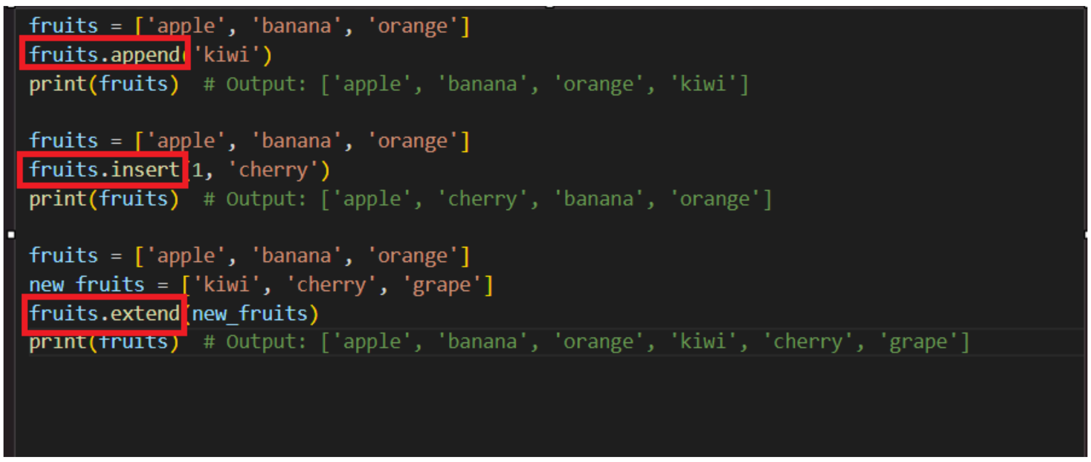
Add List Items:

There are three methods to add items to list: append(), insert() and extend().

append(): This method appends items to the end of the existing list.

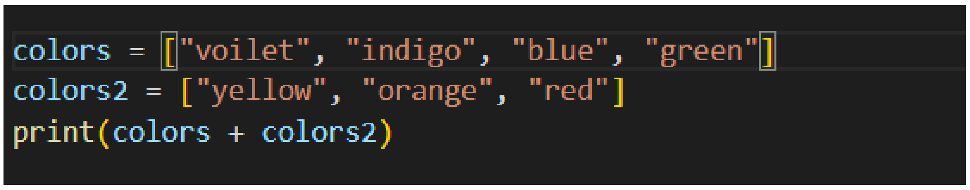
insert(): This method inserts an item at the given index. User has to specify index and the item to be inserted within the insert() method.

extend(): This method adds an entire list or any other collection datatype (set, tuple, dictionary) to the existing list.



```
fruits = ['apple', 'banana', 'orange']
fruits.append('kiwi')
print(fruits)  # Output: ['apple', 'banana', 'orange', 'kiwi']

fruits = ['apple', 'banana', 'orange']
fruits.insert(1, 'cherry')
print(fruits)  # Output: ['apple', 'cherry', 'banana', 'orange']

fruits = ['apple', 'banana', 'orange']
new_fruits = ['kiwi', 'cherry', 'grape']
fruits.extend(new_fruits)
print(fruits)  # Output: ['apple', 'banana', 'orange', 'kiwi', 'cherry', 'grape']
```

Concatenate two lists: You can simply concatenate two list to join two lists.



```
colors = ["voilet", "indigo", "blue", "green"]
colors2 = ["yellow", "orange", "red"]
print(colors + colors2)
```

Remove List Items:

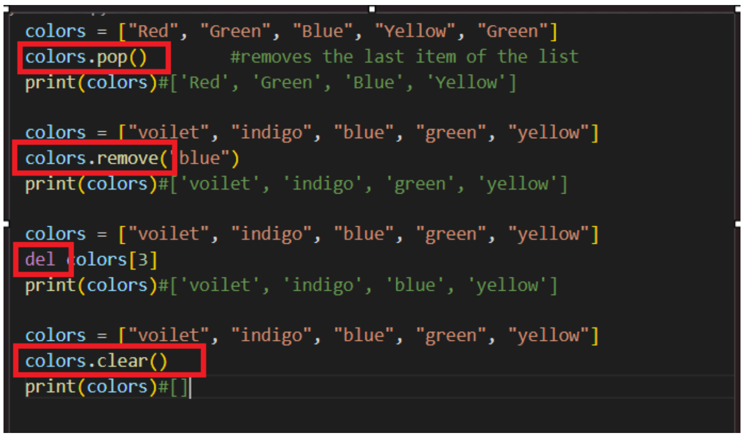
There are various methods to remove items from the list: pop(), remove(), del(), clear().

pop(): This method removes the last item of the list if no index is provided. If an index is provided, then it removes item at that specified index.

remove(): This method removes specific item from the list.

del:del is not a method, rather it is a keyword which deletes item at specific from the list, or deletes the list entirely.

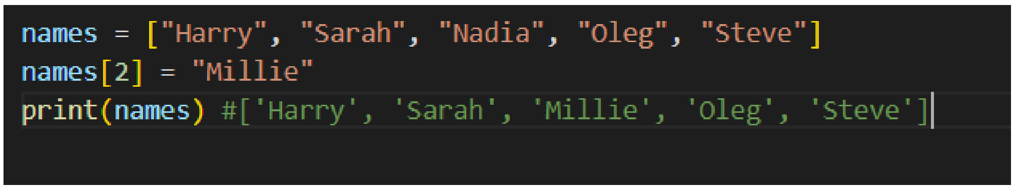
clear(): This method clears all items in the list and prints an empty list.



```python
colors = ["Red", "Green", "Blue", "Yellow", "Green"]
colors.pop()        #removes the last item of the list
print(colors)#['Red', 'Green', 'Blue', 'Yellow']

colors = ["voilet", "indigo", "blue", "green", "yellow"]
colors.remove("blue")
print(colors)#['voilet', 'indigo', 'green', 'yellow']

colors = ["voilet", "indigo", "blue", "green", "yellow"]
del colors[3]
print(colors)#['voilet', 'indigo', 'blue', 'yellow']

colors = ["voilet", "indigo", "blue", "green", "yellow"]
colors.clear()
print(colors)#[]
```

Change List Items:

Changing items from list is easier, you just have to specify the index where you want to replace the item with existing item.



```python
names = ["Harry", "Sarah", "Nadia", "Oleg", "Steve"]
names[2] = "Millie"
print(names) #['Harry', 'Sarah', 'Millie', 'Oleg', 'Steve']
```

List Methods:

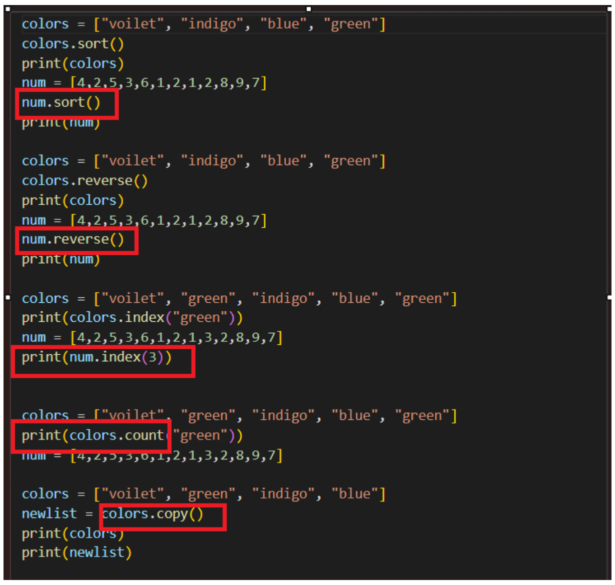
sort(): This method sorts the list in ascending order.

reverse(): This method reverses the order of the list.

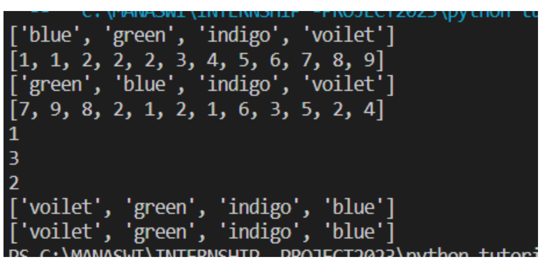
index(): This method returns the index of the first occurrence of the list item.

count(): Returns the count of the number of items with the given value.

copy(): Returns copy of the list. This can be done to perform operations on the list without modifying the original list.



```python
colors = ["voilet", "indigo", "blue", "green"]
colors.sort()
print(colors)
num = [4,2,5,3,6,1,2,1,2,8,9,7]
num.sort()
print(num)

colors = ["voilet", "indigo", "blue", "green"]
colors.reverse()
print(colors)
num = [4,2,5,3,6,1,2,1,2,8,9,7]
num.reverse()
print(num)

colors = ["voilet", "green", "indigo", "blue", "green"]
print(colors.index("green"))
num = [4,2,5,3,6,1,2,1,3,2,8,9,7]
print(num.index(3))


colors = ["voilet", "green", "indigo", "blue", "green"]
print(colors.count("green"))
num = [4,2,5,3,6,1,2,1,3,2,8,9,7]

colors = ["voilet", "green", "indigo", "blue"]
newlist = colors.copy()
print(colors)
print(newlist)
```

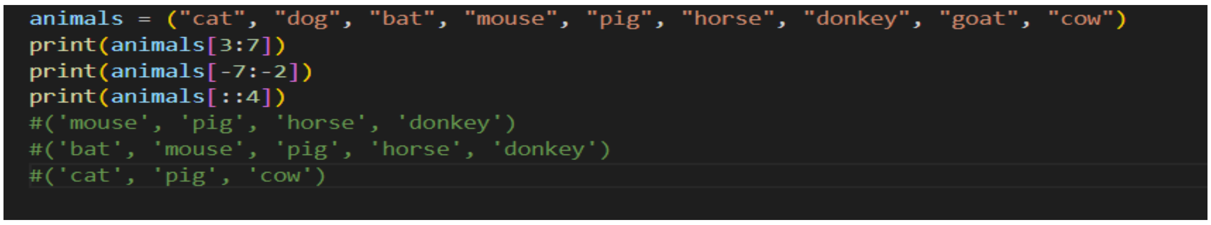


```
   C:\MANASWI\INTERNSHIP -PROJECT2023\python tu
['blue', 'green', 'indigo', 'voilet']
[1, 1, 2, 2, 2, 3, 4, 5, 6, 7, 8, 9]
['green', 'blue', 'indigo', 'voilet']
[7, 9, 8, 2, 1, 2, 1, 6, 3, 5, 2, 4]
1
3
2
['voilet', 'green', 'indigo', 'blue']
['voilet', 'green', 'indigo', 'blue']
PS C:\MANASWI\INTERNSHIP -PROJECT2023\python tutori
```

6. PYTHON TUPLES

Tuples are immutable. Tuple items are separated by commas and enclosed within round brackets (). Tuples are unchangeable means we cannot reassigned them after creation**.**

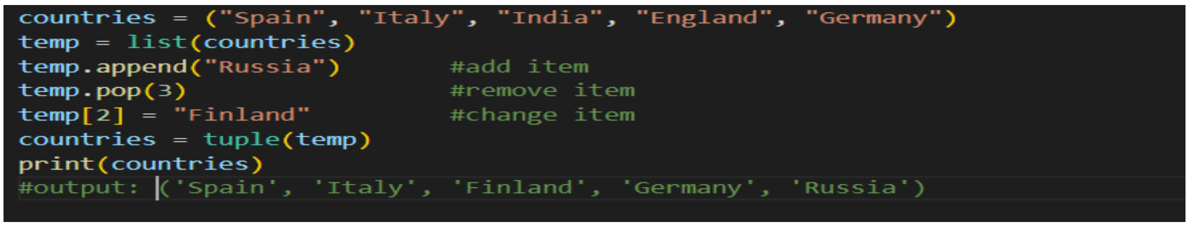
Tuple Indexes: Each item/element in a tuple has its own unique index. This index can be used to access any particular item from the tuple. The first item has index [0], second item has index [1], third item has index [2] and so on.Positive indexing and negative indexing and range of index.



```
animals = ("cat", "dog", "bat", "mouse", "pig", "horse", "donkey", "goat", "cow")
print(animals[3:7])
print(animals[-7:-2])
print(animals[::4])
#('mouse', 'pig', 'horse', 'donkey')
#('bat', 'mouse', 'pig', 'horse', 'donkey')
#('cat', 'pig', 'cow')
```

Manipulating Tuples:

Tuples are immutable, hence if you want to add, remove or change tuple items, then first you must convert the tuple to a list. Then perform operation on that list and convert it back to tuple.

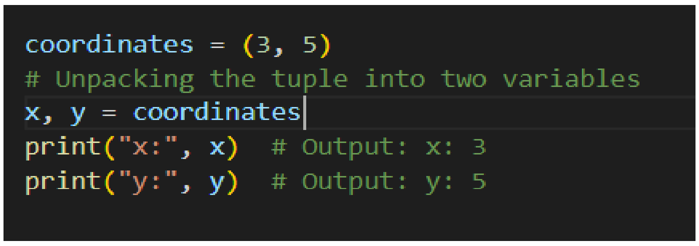


```
countries = ("Spain", "Italy", "India", "England", "Germany")
temp = list(countries)
temp.append("Russia")       #add item
temp.pop(3)                 #remove item
temp[2] = "Finland"         #change item
countries = tuple(temp)
print(countries)
#output: ('Spain', 'Italy', 'Finland', 'Germany', 'Russia')
```

Convert the tuple to a list, manipulate items of the list using list methods, then convert list back to a tuple.

Unpack Tuples:

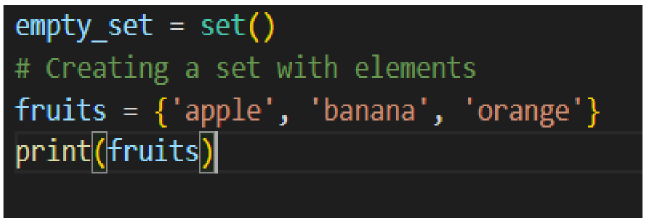
Unpacking is the process of assigning the tuple items as values to variables.



```
coordinates = (3, 5)
# Unpacking the tuple into two variables
x, y = coordinates
print("x:", x)  # Output: x: 3
print("y:", y)  # Output: y: 5
```

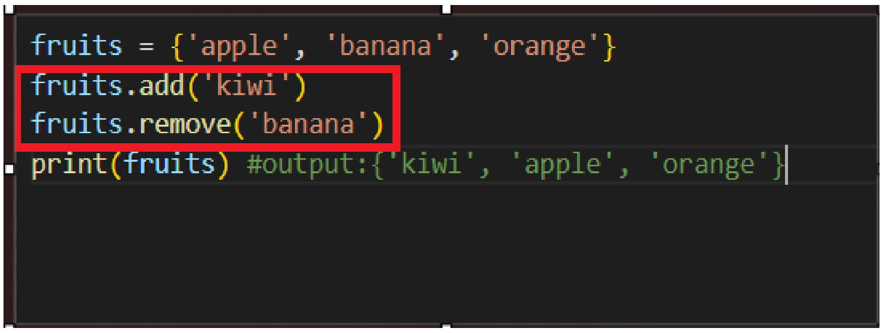
Python Sets:

Set is an unordered collection of unique elements. The primary purpose of using sets is to eliminate duplicate values and perform mathematical set operations, such as union, intersection, difference, and symmetric difference. Sets are mutable.



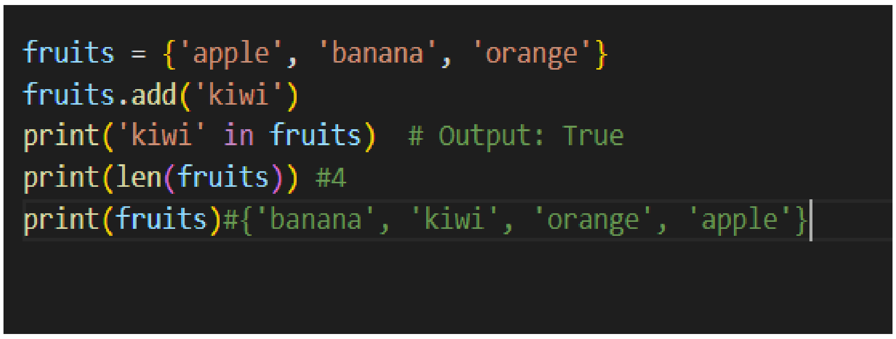
```python
empty_set = set()
# Creating a set with elements
fruits = {'apple', 'banana', 'orange'}
print(fruits)
```

Add/Remove Set Items:



```python
fruits = {'apple', 'banana', 'orange'}
fruits.add('kiwi')
fruits.remove('banana')
print(fruits) #output:{'kiwi', 'apple', 'orange'}
```

Membership and length:



```python
fruits = {'apple', 'banana', 'orange'}
fruits.add('kiwi')
print('kiwi' in fruits)  # Output: True
print(len(fruits)) #4
print(fruits)#{'banana', 'kiwi', 'orange', 'apple'}
```

Join Sets: union() and update(): The union() and update() methods prints all items that are present in the two sets. The union() method returns a new set whereas update() method adds item into the existing set from another set.

Intersection and intersection_update():

The intersection() and intersection_update() methods prints only items that are similar to both the sets. The intersection() method returns a new set whereas intersection_update() method updates into the existing set from another set.

Symmetric_difference and symmetric_difference_update():

The symmetric_difference() and symmetric_difference_update() methods prints only items that are not similar to both the sets. The symmetric_difference() method returns a new set whereas symmetric_difference_update() method updates into the existing set from another set.

Difference() and difference_update():

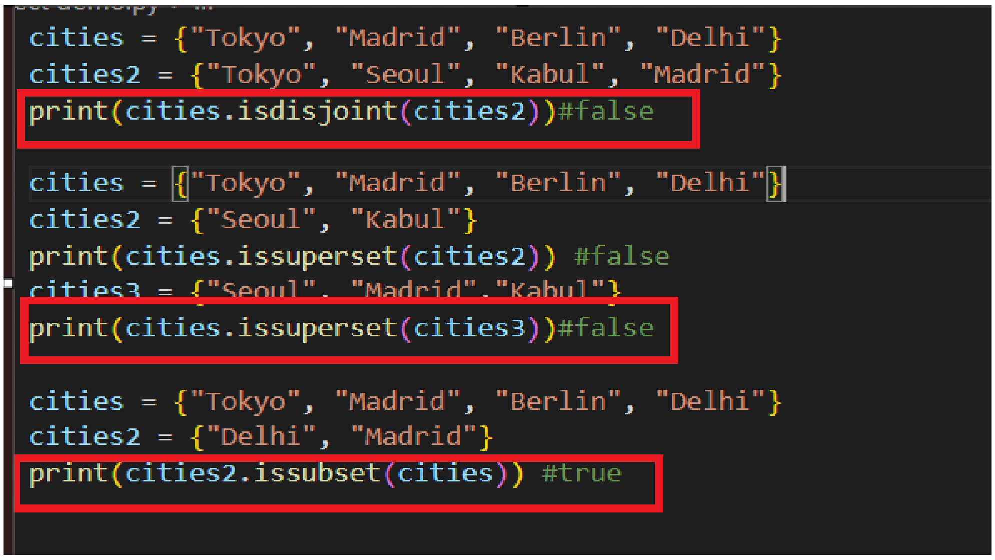
The difference() and difference_update() methods prints only items that are only present in the original set and not in both the sets. The difference() method returns a new set whereas difference_update() method updates into the existing set from another set.

Set Methods:

isdisjoint(): The isdisjoint() method checks if items of given set are present in another set. This method returns False if items are present, else it returns True.

issuperset(): The issuperset() method checks if all the items of a particular set are present in the original set. It returns True if all the items are present, else it returns False.

issubset(): The issubset() method checks if all the items of the original set are present in the particular set. It returns True if all the items are present, else it returns False.



```
cities = {"Tokyo", "Madrid", "Berlin", "Delhi"}
cities2 = {"Tokyo", "Seoul", "Kabul", "Madrid"}
print(cities.isdisjoint(cities2))#false

cities = {"Tokyo", "Madrid", "Berlin", "Delhi"}
cities2 = {"Seoul", "Kabul"}
print(cities.issuperset(cities2)) #false
cities3 = {"Seoul", "Madrid","Kabul"}
print(cities.issuperset(cities3))#false

cities = {"Tokyo", "Madrid", "Berlin", "Delhi"}
cities2 = {"Delhi", "Madrid"}
print(cities2.issubset(cities)) #true
```

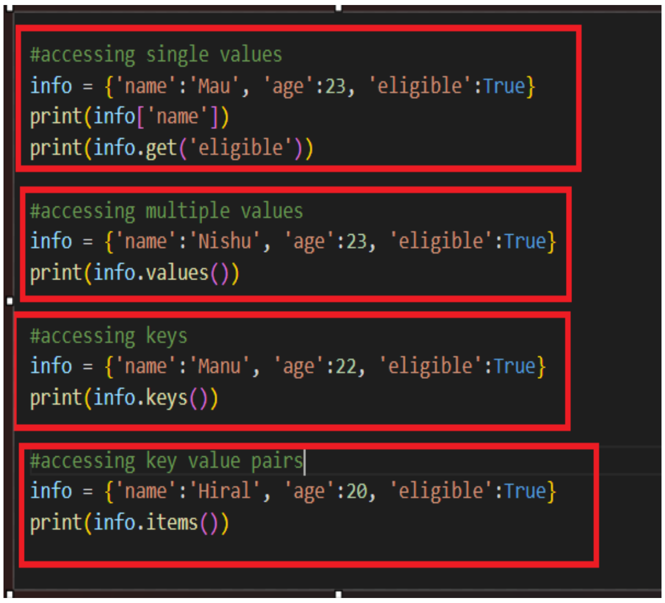
7. PYTHON DICTIONARIES

Python Dictionaries:

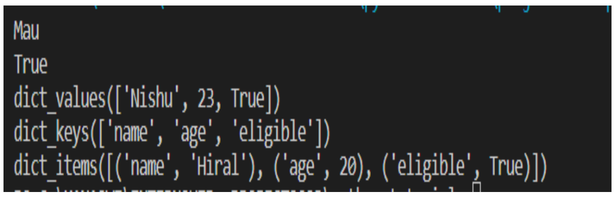
Dictionaries are collection of data items. They store multiple items in a single variable. Dictionaries items are key-value pairs that are separated by commas and enclosed within curly brackets {}. Dictionaries are mutable**.**

Access Items

- **Accessing single values:** Values in a dictionary can be accessed using keys. We can access dictionary values by mentioning keys either in square brackets or by using get method.
- **Accessing multiple values:** We can print all the values in the dictionary using values() method.
- **Accessing keys:** We can print all the keys in the dictionary using keys() method.
- **Accessing key-value pairs:** We can print all the key-value pairs in the dictionary using items() method.



```python
#accessing single values
info = {'name':'Mau', 'age':23, 'eligible':True}
print(info['name'])
print(info.get('eligible'))

#accessing multiple values
info = {'name':'Nishu', 'age':23, 'eligible':True}
print(info.values())

#accessing keys
info = {'name':'Manu', 'age':22, 'eligible':True}
print(info.keys())

#accessing key value pairs
info = {'name':'Hiral', 'age':20, 'eligible':True}
print(info.items())
```



```
Mau
True
dict_values(['Nishu', 23, True])
dict_keys(['name', 'age', 'eligible'])
dict_items([('name', 'Hiral'), ('age', 20), ('eligible', True)])
```

Add/Remove Items

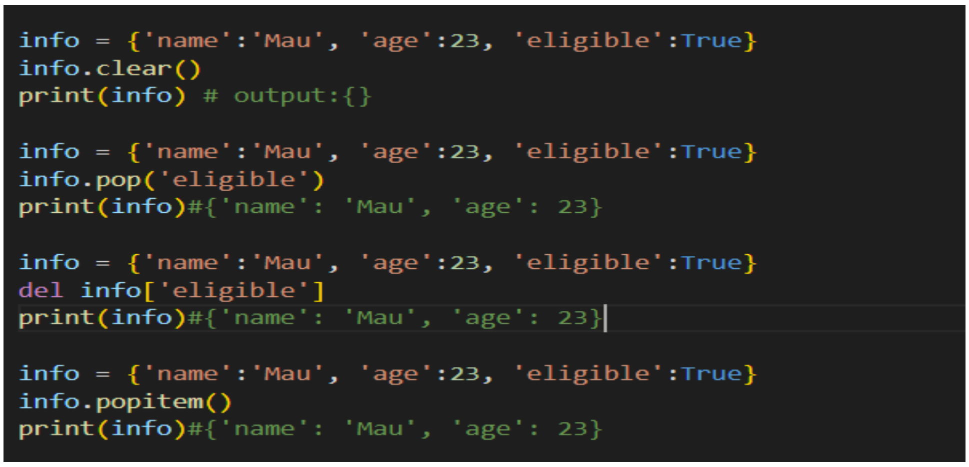
Create a new key and assign a value to it.

Use the **update()** method.

clear(): The clear() method removes all the items from the list.

pop(): The pop() method removes the key-value pair whose key is passed as a parameter.

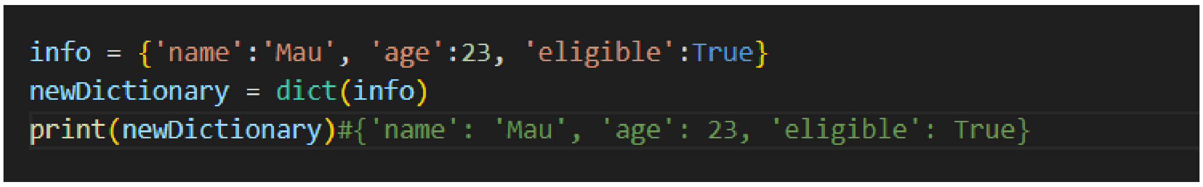
popitem(): The popitem() method removes the last key-value pair from the dictionary.



```
info = {'name':'Mau', 'age':23, 'eligible':True}
info.clear()
print(info) # output:{}

info = {'name':'Mau', 'age':23, 'eligible':True}
info.pop('eligible')
print(info)#{'name': 'Mau', 'age': 23}

info = {'name':'Mau', 'age':23, 'eligible':True}
del info['eligible']
print(info)#{'name': 'Mau', 'age': 23}

info = {'name':'Mau', 'age':23, 'eligible':True}
info.popitem()
print(info)#{'name': 'Mau', 'age': 23}
```

Copy Dictionaries:

We can use the copy() method to copy the contents of one dictionary into another dictionary.



```
info = {'name':'Mau', 'age':23, 'eligible':True}
newDictionary = dict(info)
print(newDictionary)#{'name': 'Mau', 'age': 23, 'eligible': True}
```

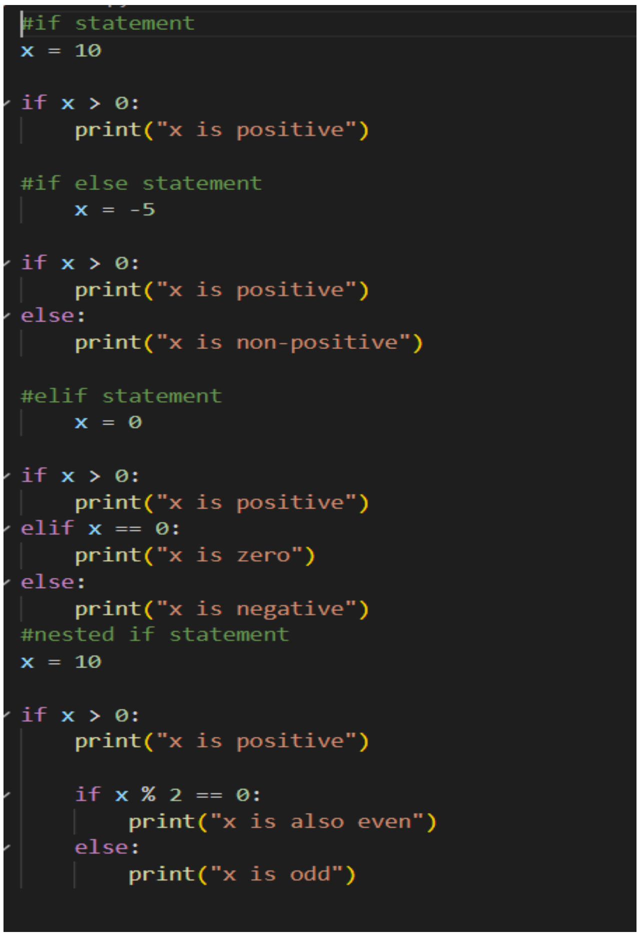
8. CONDITIONAL STATEMENTS

if Statement: The **if** statement is executed only if the condition **x > 0** is true.

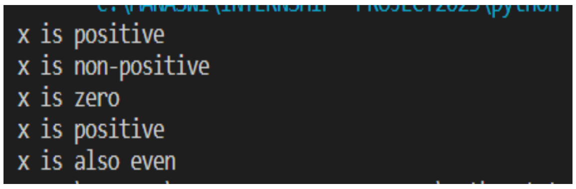
if-else Statement: if the condition x > 0 is true, the first block of code is executed; otherwise, the block under else is executed.

elif Statement: The if condition is checked first. If it is true, the corresponding block is executed. If not, the elif condition is checked. If none of the conditions is true, the block under else is executed.

Nested if Statement: The nested if statement is only checked if the outer if condition is true.



```python
#if statement
x = 10

if x > 0:
    print("x is positive")

#if else statement
    x = -5

if x > 0:
    print("x is positive")
else:
    print("x is non-positive")

#elif statement
    x = 0

if x > 0:
    print("x is positive")
elif x == 0:
    print("x is zero")
else:
    print("x is negative")
#nested if statement
x = 10

if x > 0:
    print("x is positive")

    if x % 2 == 0:
        print("x is also even")
    else:
        print("x is odd")
```



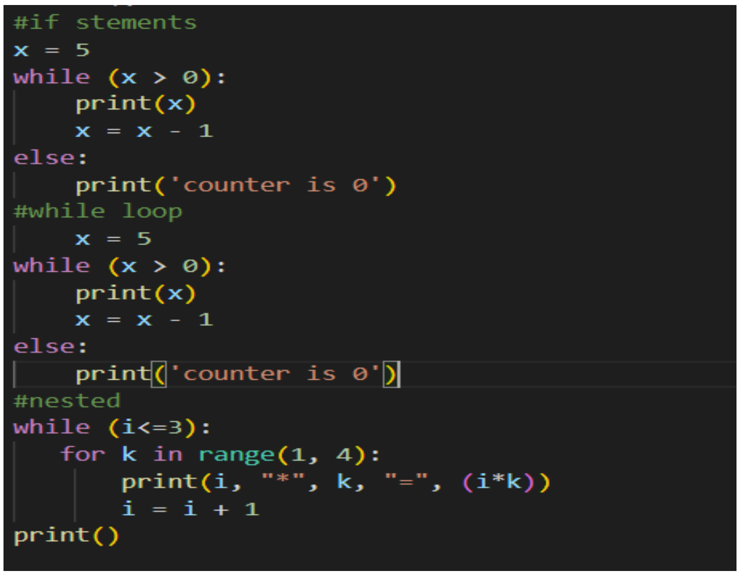
```
x is positive
x is non-positive
x is zero
x is positive
x is also even
```

9. PYTHON LOOPS

Python for loop: ‘for ‘loops can iterate over a sequence of iterable objects in python. Iterating over a sequence is nothing but iterating over strings, lists, tuples, sets and dictionaries.

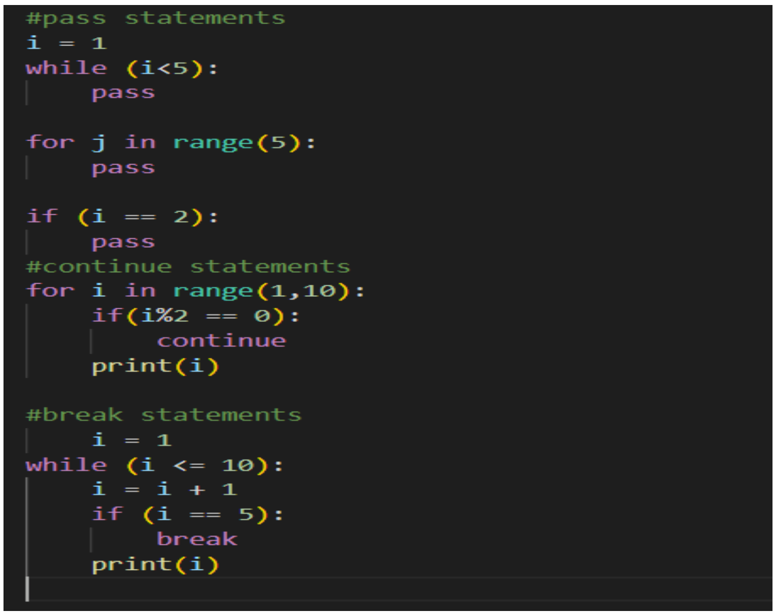
Python while Loop: while loops execute statements while the condition is True. As soon as the condition becomes False, the interpreter comes out of the while loop.

Nested Loops: use loops inside other loops, such types of loops are called as nested loops.



```
#if stements
x = 5
while (x > 0):
    print(x)
    x = x - 1
else:
    print('counter is 0')
#while loop
    x = 5
while (x > 0):
    print(x)
    x = x - 1
else:
    print('counter is 0')
#nested
while (i<=3):
   for k in range(1, 4):
       print(i, "*", k, "=", (i*k))
       i = i + 1
print()
```

Control Statements: There are three control statements.They are pass, continue and break.

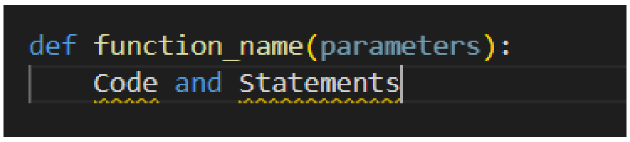


```
#pass statements
i = 1
while (i<5):
    pass

for j in range(5):
    pass

if (i == 2):
    pass
#continue statements
for i in range(1,10):
    if(i%2 == 0):
        continue
    print(i)

#break statements
    i = 1
while (i <= 10):
    i = i + 1
    if (i == 5):
        break
    print(i)
```

10. PYTHON FUNCTIONS

A function is a block of code.

- Create a function using the def keyword, followed by a function name, followed by a paranthesis (()) and a colon(:).
- Any parameters and arguments should be placed within the parentheses.
- Rules to naming function are similar to that of naming variables.
- Any statements and other code within the function should be indented.



```
def function_name(parameters):
    Code and Statements
```

1. built-in functions:

These functions are defined and pre-coded in python. Some examples of built-in functions are as follows:

min(), max(), len(), sum(), type(), range(), dict(), list(), tuple(), set(), print(), etc.

2. user-defined functions:

We can create functions to perform specific tasks as per our needs. Such functions are called user-defined functions.

return Statement

The return statement is used to return the value of the expression back to the main function**.**

Function arguments:

There are four types of arguments that we can provide in a function:

Default Arguments

Keyword Arguments

Required Arguments

Variable-length Arguments

Python Recursion

We can let the function call itself, such a process is known as calling a function recursively in python**.**

11. PYTHON MODULES

Python Modules

some popular python built-in modules: csv, datetime, json, math, random, sqlite3, statistics, tkinter, turtle, etc.

Creating and using module: Write code and save file with extension .py .

Using an alias: We can also use alias while importing module. This way we do not need to write the whole name of the module while calling it.

import from module: You can also import specific parts that you need from a module instead of importing the entire module.

dir() function: The dir() function lists all the function names (or variable names) in a module.

Python Packages

NumPy, SciPy, Pandas, Seaborn, sklearn, Matplotlib, etc.

NumPy is the fundamental package for scientific computing with Python. It provides support for large, multi-dimensional arrays and matrices, along with mathematical functions to operate on these arrays.

```
pip install numpy
```

SciPy builds on NumPy and provides additional modules for scientific computing. It includes modules for optimization, signal and image processing, statistical functions, linear algebra.

```
pip install scipy
```

12. PYTHON OOPS

Oops in python: Object-Oriented Programming (OOP) in Python provides a powerful and flexible way to structure code, promote code reuse, and improve code organization

Classes and Objects: OOP allows you to organize your code into classes and objects. Each class can encapsulate data and functionality related to a specific concept or entity.

Modularization: Classes provide a way to modularize code. Each class can be developed and tested independently, promoting a modular and scalable code structure.

Encapsulation

Data Hiding: Encapsulation allows you to hide the internal state of objects and expose a controlled interface.

Inheritance

Code Reuse: Inheritance enables you to create a new class (subclass) by inheriting attributes and methods from an existing class (superclass). This promotes code reuse and helps avoid redundancy.

Polymorphism: **Method Overriding:** Polymorphism allows you to override methods in subclasses, providing specific implementations. This makes it easy to extend and modify the behavior of existing classes.

Flexibility and Extensibility: **Code Extensibility:** OOP allows you to extend existing classes to create new ones. This makes it easy to add new features or modify behavior without modifying the existing codebase.

Real-world Modeling: Modeling Real-world Entities: OOP is well-suited for modeling real-world entities and their relationships.

Software Design Patterns

Design Patterns: OOP facilitates the use of design patterns, which are proven solutions to recurring design problems. Common design patterns, such as Singleton, Observer, and Factory, can be implemented more effectively with OOP principles.

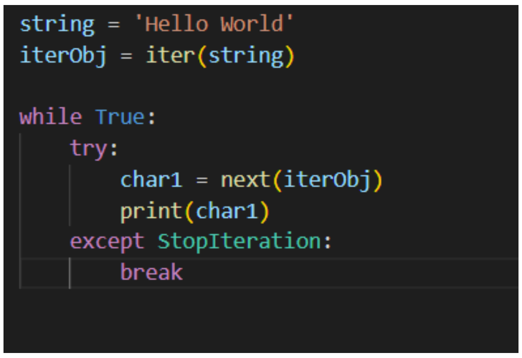
self method : The self parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class.

__init__ method: The **__init__** method is used to initialize the object's state and contains statements that are executed at the time of object creation.

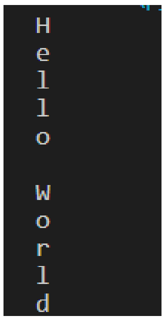
13. PYTHON ITERATORS

Iterators in python are used to iterate over iterable objects or container datatypes like lists, tuples, dictionaries, sets, etc.It consists of __iter__() and __next__() methods.

- __iter__() : to initialize an iterator, use the __iter__() method.
- __next__() : This method returns the next item of the sequence.



```python
string = 'Hello World'
iterObj = iter(string)

while True:
    try:
        char1 = next(iterObj)
        print(char1)
    except StopIteration:
        break
```



```
H
e
l
l
o

W
o
r
l
d
```

JSON

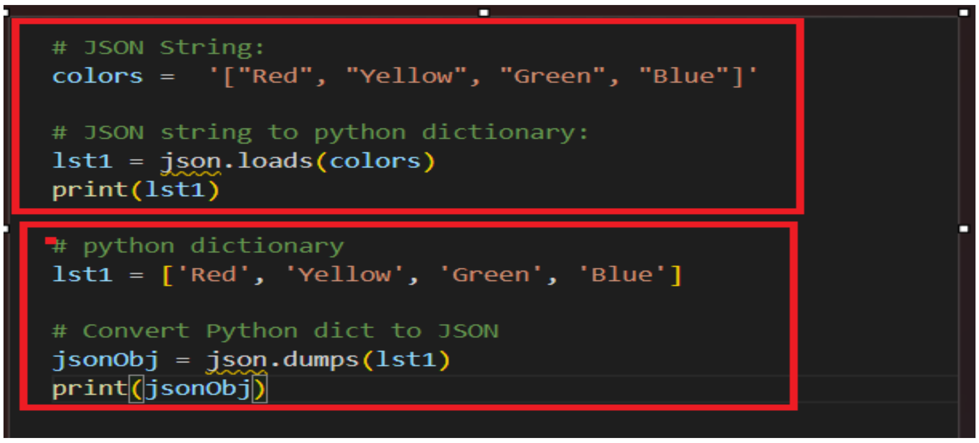
JSON stands for JavaScript Object Notation. It is a built-in package provided in python that is used to store and exchange data.

Why json in python ?

1. Data Interchange:
   - JSON is a lightweight and human-readable format that is widely used for data interchange between systems.
   - It is language-agnostic, meaning it can be easily shared between different programming languages.
2. Web APIs (Application Programming Interfaces):
   - Many web APIs use JSON as the standard format for sending and receiving data. Python applications interacting with web services often need to handle JSON data.
3. Configuration Files:
   - JSON is often used for configuration files due to its simplicity and readability. Python applications can read and write configuration settings in JSON format.
4. Data Serialization:
   - Python applications may need to serialize (convert to a format that can be stored or transmitted) complex data structures into a string representation. JSON is a convenient and widely supported serialization format.
5. Data Exchange with JavaScript:

- JSON is native to JavaScript, making it easy to exchange data between Python applications and web browsers, which predominantly use JavaScript.

6. Ease of Use:
   - The json module in Python provides convenient methods for encoding Python objects into JSON strings and decoding JSON strings into Python objects.
   - It offers functionalities like json.dumps() and json.loads() for serialization and deserialization.



```
# JSON String:
colors =  '["Red", "Yellow", "Green", "Blue"]'

# JSON string to python dictionary:
lst1 = json.loads(colors)
print(lst1)

# python dictionary
lst1 = ['Red', 'Yellow', 'Green', 'Blue']

# Convert Python dict to JSON
jsonObj = json.dumps(lst1)
print(jsonObj)
```

Python try...except: try….. except blocks are used in python to handle errors and exceptions. The code in try block runs when there is no error. If the try block catches the error, then the except block is executed.

Python PIP

pip stands for Package Installer for Python. It is used to install and manage software packages in python. 'Pip list' to view all installed packages in window.

Python inheritance

- Inheritance allows us to define a class that inherits all the methods and properties from another class.
- **Parent class** is the class being inherited from, also called base class.
- **Child class** is the class that inherits from another class, also called derived class.
- In Python, you can create a new class (subclass or derived class) by inheriting from an existing class (superclass or base class).

14. FILE HANDLING

Python File Handling

File handling in python lets you open, read, write, append, modify files.There are various modes in which we can open files.

- read (r): This mode opens the file for reading only and gives an error if the file does not exist. This is the default mode if no mode is passed as a parameter.
- write (w): This mode opens the file for writing only and creates a new file if the file does not exist.
- append (a): This mode opens the file for appending only and creates a new file if the file does not exist.
- create (x): This mode creates a file and gives an error if the file already exists.
- text (t): Used to handle text files.
- binary (b): used to handle binary files (images).

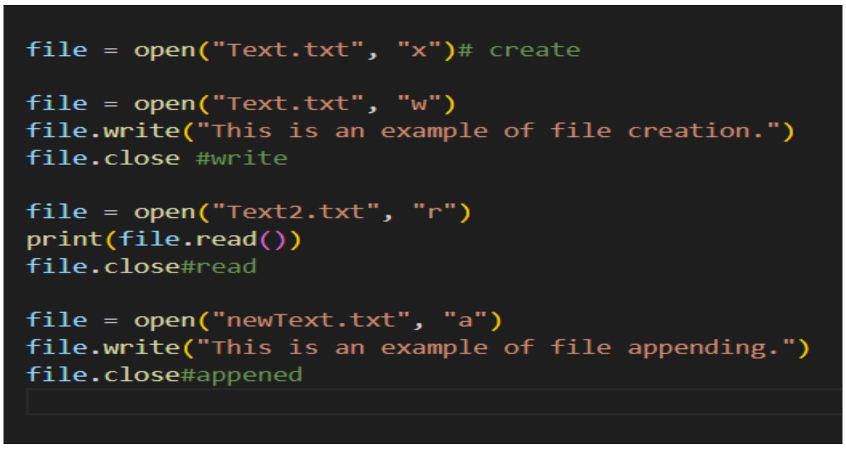
Read/Write Files

Create a File: Creating a file is primarily done using the create (x) mode.

Write onto a File: This method writes content onto a file.

Read a File: This method allows us to read the contents of the file.

Append a File: This method appends content into a file.



```
file = open("Text.txt", "x")# create

file = open("Text.txt", "w")
file.write("This is an example of file creation.")
file.close #write

file = open("Text2.txt", "r")
print(file.read())
file.close#read

file = open("newText.txt", "a")
file.write("This is an example of file appending.")
file.close#appened
```

15. PYTHON MYSQL

Python can be used in database applications. One of the most popular databases is MySQL.

- Install python library for mysql in python →run this command→pip install mysql-connector-python.
- Connecting to mysql→run this command→ python.exe -m pip install --upgrade pip

When working with MySQL in Python, you need specific details to establish a connection to the MySQL server.

- MySQL Server Hostname or IP Address: The address of the MySQL server you want to connect to. It can be an IP address or a domain name.
- Username and Password:A valid MySQL username and password with the necessary privileges to perform the required operations on the specified database.
- Database Name:The name of the MySQL database you want to interact with. Some operations might not require a specific database.
- Port (Optional):The port number on which the MySQL server is listening. The default port for MySQL is 3306.

Once you have these details, you can use them to create a connection to the MySQL server from your Python script or application.

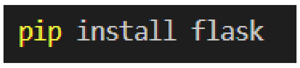
Python In Web Development:

Python is a versatile programming language that is widely used for web development. There are several frameworks and tools available that make web development in Python efficient and enjoyable.

Steps: Install Python→You can download the latest version from the official Python website.

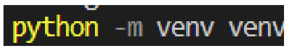
1. **Choose a Web Framework:** Python offers various web frameworks that simplify web development. Some popular ones include:
   - **Django:** A high-level web framework that follows the "batteries-included" philosophy, providing many built-in features for common tasks.
   - **Flask:** A lightweight and modular micro-framework that gives you more flexibility by allowing you to choose your components.

2. **Install a Web Framework:** Use a package manager like ‘pip to install’ the chosen framework. For example, to install Flask, you can use the following command in your terminal.



```
pip install flask
```

3. **Create a Virtual Environment:** It's good practice to create a virtual environment for your project to isolate dependencies. Use the following commands to create and activate a virtual environment.



```
python -m venv venv
```

4. **Create a Web Application:**
   - **Django:**
     - Create a new Django project: ‘django-admin start project project name’
     - Change into the project directory: ‘cd project name’
     - Run the development server: ‘python manage.py runserver’
   - **Flask:**
     - Create a new Flask application:
     - Run the Flask application: python yourfile.py
5. **Build Your Application:**
   - Define models, views, and templates in Django.
   - Define routes, views, and templates in Flask.
   - Connect to databases, if needed.
   - Implement business logic and application features.
6. **Template Engines:** Both Django and Flask support template engines to generate dynamic HTML. Django uses its own template engine.
7. **Static Files:** Manage static files (CSS, JavaScript, images) in your project. Both frameworks have mechanisms to handle static files.
8. **Deploy Your Application**: Choose a hosting solution to deploy your web application. Popular choices include Heroku, AWS, Digital Ocean, and others.
9. **Learn and Explore:** Continue learning and exploring advanced features of your chosen framework, as well as additional libraries and tools for specific needs.

CONCLUSION

Python is widely loved and used because it's easy to understand, flexible, and can be applied in many different areas. People appreciate its straightforward and clear syntax, making it a great choice for beginners and experts alike. Python's wide range of applications, from web development to data science, benefits from a vast library of tools and a supportive community. Whether you're just starting out or have been coding for a while, Python provides a friendly and effective platform for creating all sorts of programs.