# ­­Day-1

# Python Introduction

Python is a general-purpose, High-level programming language that is used for :

* Web development,
* Software development,
* Machine learning, and
* Data science
* Scripting etc

**Features/Advantages of Python:**

* 1. Simple and easy to learn
  2. Free and Open Source
  3. High Level Programming language
  4. Platform Independent
  5. Portable
  6. Dynamically Typed
  7. Both Procedure Oriented and Object Oriented
  8. Interpreted as well as compiler based
  9. Extensive Library(like numpy, panda, matplotlib etc…)

**Limitations of Python:**

* 1. **Performance and Speed:** Python is an interpreted language, which means that it is slower than compiled languages like C or Java. This can be a problem for certain types of applications that require high performance, such as real-time systems or heavy computation.(Slow at run time)
  2. **Dynamically Typing:** Python is a dynamically typed language, which means that the type of a variable is not checked at compile time. This can lead to errors at runtime.
  3. **Web Support:** Python does not have built-in support for web development. This means that programmers need to use third-party frameworks and libraries to develop web applications in Python
  4. **Mobile computing:** Python is not as well-suited for mobile computing as some other languages, such as Java or Kotlin.(High memory Consumption)
  5. **Memory Management:** Python is a dynamically typed language, which means that the data type of a variable can change at any time. This can lead to memory leaks and other memory usage problems.

**Application of Pythons:**

1. Healthcare

2. Web development

3. Data Science and Analytic

4. Scientific Computing

5. Automation

6. Scripting

7. Desktop GUI Application

8. Game Development

9. Networking etc

**Python can take almost all programming features from different languages:--**

1. Functional Programming Features from C

2. Object Oriented Programming Features from C++

3. Scripting Language Features from Perl and Shell Script

4. Modular Programming Features from Modula-3(Programming Language)

**Flavors of Python or types of python compilers:**

**1. CPython:** It is the standard flavor of Python. It can be used to work with C

lanugage Applications.

**2. Jython or JPython:** It is for Java Applications. It can run on JVM

**3. IronPython:** It is for C#.Net platform

**4. PyPy:** The main advantage of PyPy is performance will be improved because

JIT (just in time)compiler is available inside PVM.

**5. RPython:** For Ruby Platforms.

**6. AnacondaPython:** It is specially designed for handling large volume of data

processing.This makes it possible to run Python code in

web browsers.

**Day2**

**Python Versions:**

1. **Python 1.x:- Python 1.0 (January 1994)**
2. **Python 2.x:- Python 2.0 (October 2000)**
3. **Python 3.x:- Python 3.0 (December 2008)**

**Why you chosen python :-**

1. **Ease to Learn**
2. **Strong Demand in the Job Market**
3. **Versatility**
4. **Rich Libraries and Frameworks**
5. **Huge Community and Support**

**Complete working procedure :-----**

**Convert python code into byte-code:----**

python -m py\_compile first.py

python first.cpython-310.pyc

python -m dis first.py

**Day-3**

**Python Variable**

Variable/identifier

**x y**

object Value

1000

10

**10101010 10101010**

Memory Location

**Note:- In python, memory allocated to the object, not variable.**

**What is python variable?**

All the data which we create in the program will be saved in some memory location on the system. The data can be anything, an integer, a complex number, a set of mixed values, etc. A Python variable is a symbolic name that is a reference or pointer to an object. Once an object is assigned to a variable, you can refer to the object by that name.

A **Variable** is a name that is assigned to a memory location, which is used to contain the corresponding value in it.

**To summarize, a variable is a**

1. Name
2. Refers to a object
3. Hold some data

**Properties of variable:**

1. Data **Type**
2. **Scope**
3. **Value**
4. **Memory Location**

##### **Multiple values assign in multiple variables in a single line in Python.**

##### **Single values assign in multiple variables in a single line in Python.**

##### **Re-initialize the variables in Python**

**Note: Python doesn’t support const type variable. But if you want to create use capital letter for that.**

**Rules for creating variables/** **Identifiers in Python:**

* 1. A variable name must start with a letter or the underscore character.

( myvar, my\_var, \_my\_var, myVar, MYVAR, myvar2 )

* 1. A variable name cannot start with a number.

(2myvar,)

* 1. A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and \_ ).

( myvar, my\_var, \_my\_var, myVar, MYVAR, myvar2 )

* 1. Variable names are case-sensitive.

(age, Age and AGE are three different variables)

* 1. A variable name cannot be any of the python keywords.(if , else, for ,while, try, etc)

# how to fetch all python-keywords

import keyword

print("The list of keywords is : ")

print(keyword.kwlist)

**Recommendation:**

1. **Use descriptive variable names:** This means that your variable names should give you a good idea of what the variable stores. For example, the variable age is a good name for a variable that stores a person's age.
2. **Use snake\_case for variable names:** This means that variable names should be made up of lowercase letters, with underscores separating words. For example, the variable my\_variable is a good example of snake\_case.
3. **Avoid using abbreviations in variable names:** This can make your code more difficult to read and understand. For example, the variable num is not a good name for a variable that stores the number of student in a class room. A better name would be number\_of\_student.

# Assign Multiple Values in multiple variables in single line:-

## Many Values to Multiple Variables

### Example:-

x, y, z = "Neeraj", "Ravi", "Rahul"  
print(x)  
print(y)  
print(z)

## One Value to Multiple Variables in single line:

### Example:-

x = y = z = "Neeraj Kumar"  
print(x)  
print(y)  
print(z)

1. **Advance examples:-**

Example:-

city = ["Bhopal", "Indore", "Jabalpur"]  
x, y, z = city  
print(x)  
print(y)  
print(z)

**Python Comments:-**

1. single line comments:--- ( # ---------------) ctrl+/
2. Multi-line comments:---(‘‘‘ ------------

-----------’’’)

**Day-3**

**Data Type:**

Data-type is a property/attribute of variable that tell us, what kind of value stored in it.

|  |  |
| --- | --- |
| **Data Types in python** | |
| **Numeric** | int, float, complex |
| **String** | str |
| **Sequence** | list, tuple, range |
| **Mapping** | dict |
| **Set** | set, frozenset |
| **Boolean** | bool |
| **Binary** | bytes, bytearray |
| **None** | NoneType |

#----------------Numeric--------------

x=3

print(type(x))

x=3.5

print(type(x))

x=3+3j

print(type(x))

#----------------String--------------

x= 'Neeraj Kumar'

y= "Neeraj Kumar"

z='''My

        Name

            is

                Neeraj Kumar'''

print(type(x))

print(type(y))

print(type(z))

#----------------Sequesce data type -----------------

my\_list = [10,20,'Neeraj',10.8,True,2+3j]

print(type(my\_list))

my\_tuple = (10,20,'Neeraj',10.8,True,2+3j)

print(type(my\_tuple))

# --------------- mapping data type ----------------

my\_dict = {"name":"Neeraj","age":37,"Quali":"M.tech"}

print(type(my\_dict))

# -------------------set data\_type-------------------------

my\_set = {10,20,10,"Neeraj"}

print(my\_set)

print(type(my\_set))

my\_list = [10,20,30,40,50]

my\_frozenset = frozenset(my\_list)

# my\_frozenset[0]='New\_Value' # TypeError: 'frozenset' object does not support item assignment

print(my\_frozenset)

print(type(my\_frozenset))

my\_dict = {"name":"Neeraj","age":37,"Quali":"M.tech"}

my\_frozenset = frozenset(my\_dict.values())

print(my\_frozenset)

my\_frozenset = frozenset(my\_dict.items())

print(my\_frozenset)

my\_frozenset = frozenset(my\_dict.keys())

print(my\_frozenset)

print(type(my\_frozenset))

# my\_frozenset['name']="Neeraj Patel" # TypeError: 'frozenset' object does not support item assignment

print(my\_dict)

# -----------------------Boolean Data Type -------------------

x,y=True,False

print(type(x))

print(type(y))

# -----------------------Binary Data Type -------------------

# bytes()

x=(10,20,30,255)  # TypeError: 'str' object cannot be interpreted as an integer ,  ValueError: bytes must be in range(0, 256)

print(type(x))

y = bytes(x)

print(type(y))

# x[1]=15 # TypeError: 'bytes' object does not support item assignment

# bytearray()

x=(10,20,30,255)  # TypeError: 'str' object cannot be interpreted as an integer ,  ValueError: bytes must be in range(0, 256)

print(type(x))

x = bytearray(x)

print(type(x))

x[1]=15

print(x)

# -----------------------None Data Type -------------------

x=None

print(type(x))

**Day-5**

**Type of operetors:-**

1. **Arithmetic Operators:** The operators which are used to perform arithmetic operations like addition, subtraction, division etc. **(+, -, \*, /, %, \*\*, //)**

## 2. Comparison Operators: The operators which are used to check for some relation like greater than or less than etc.. between operands are called relational operators. ( <, >, <=, >=, ==, !=)

1. **Logical Operators:** The operators which do some logical operation on the operands and return True or False are called logical operators. The logical operations can be **‘and’, ‘or’, ‘not’** etc.

**4.** **Assignment Operators:** The operators which are used for assigning values to variables. **‘=’** is the assignment operator.

**5.** **Unary Operator:** The operator ‘-’ is called the Unary m­­inus operator. It is used to negate the number.

**6. Membership Operators:** The operators that are used to check whether a particular variable is part of another variable or not. **(‘in’ and ‘not in’ --** this is basically used to check object is present in this collection or not**)**

7. **Identity Operators:** The operators which are used to check for identity. **(‘is’ and ‘is not’--** this is basically used to check two objects are having same memory location or not**)**

## 8. Python Bitwise Operators: Bitwise operators are used to compare (binary) numbers:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Name** | **Description** |
| & | AND | Sets each bit to 1 if both bits are 1 |
| | | OR | Sets each bit to 1 if one of two bits is 1 |
| ^ | XOR | Sets each bit to 1 if only one of two bits is 1 |
| ~ | NOT | Inverts all the bits |
| << | Zero fill left shift | Shift left by pushing zeros in from the right and let the leftmost bits fall off |
| >> | Signed right shift | Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off |

**Day-6**

**----: Indexing in Python :---**

Indexing is a process to access any object/element of collection (string, list, tuple) which are present in a sequence.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | **Positive indexing** |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| P | Y | T | H | O | N | **Collections** |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| -6 | -5 | -4 | -3 | -2 | -1 | **Negative indexing** |

In Python, indexing starts from 0, which means the first element in a sequence is at position 0, the second element is at position 1, and so on. To access an element in a sequence, you can use square brackets [] with the index of the element you want to access.

In Python, indexing refers to the process of accessing a specific element in a sequence, such as a string or list, using its position or index number. Indexing in Python starts at 0, which means that the first element in a sequence has an index of 0, the second element has an index of 1, and so on.

For example, if we have a string "HELLO", we can access the first letter "H" using its index 0 by using the square bracket notation: string[0]

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Positive Index** | **0** | **1** | **2** | **3** | **4** |
|  | **H** | **E** | **L** | **L** | **O** |
| **Negative Index** | **-5** | **-4** | **-3** | **-2** | **-1** |

Python's built-in index() function is a useful tool for finding the index of a specific element in a sequence. This function takes an argument representing the value to search for and returns the index of the first occurrence of that value in the sequence.

If the value is not found in the sequence, the function raises a ValueError. For example, if we have a list [1, 2, 3, 4, 5], we can find the index of the value 3 by calling list.index(3), which will return the value 2 (since 3 is the third element in the list, and indexing starts at 0).

## Python Index Examples

The method index() returns the lowest index in the list where the element searched for appears. If any element which is not present is searched, it returns a **ValueError**.

Example:--

list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

element = 3

print(list.index(element))

O/P:-

2

Example:--( Throws a ValueError)

list = [4, 5, 6, 7, 8, 9, 10]

element = 3 # Not in the list

print(list.index(element))

O/P:-

Traceback (most recent call last):

  File "e:\DataSciencePythonBatch\index.py", line 7, in <module>

    print(list.index(element))

ValueError: 3 is not in list

### Example:--(Index of a string element)

list = [1, 'two', 3, 4, 5, 6, 7, 8, 9, 10]

element = 'two'

print(list.index(element))

O/P:-

1

### ****What does it mean to return the lowest index?****

list = [3, 1, 2, 3, 3, 4, 5, 6, 3, 7, 8, 9, 10]

element = 3

print(list.index(element))

O/P:-

0

##### **Find element with particular start and end point:--**

##### **Syntax:-list\_name.index(element, start, stop)**

##### **Example:- index() provides you an option to give it hints to where the value searched for might lie.**

list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

element = 7

print(list.index(element, 5, 8))

O/P:-

6

**Day-7**

**-----: Slicing in Python:-----**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | P | R | O | G | R | A | M | M | I | N | G |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | P | R | O |  | O | G | R | A |  | R | A | M | M |  | M | I | N | G |

Slicing is the extraction of a part of a string, list, or tuple. It enables users to access the specific range of elements by mentioning their indices.

**Syntax: Object [start : stop : step]**

**Object [start : stop]**

**Note: if step value not available that means 1 present by-default.**

* **start:** The start parameter in the slice function is used to set the starting position or index of the slicing. If not given that means 0 is the default starting index.
* **stop:** The stop parameter in the slice function is used to set the end position or index of the slicing[(n-1) for positive value and (n+1) for negative value. If not given that means it goes to end index position.
* **step:** The step parameter in the slice function is used to set the number of steps to jump. The default value of the step is 1(positive indexing).

**Rules for working:---**

**Step1**:-- Need to check step direction by default it’s goes to positive direction.

**Setp2**:- Need to check start-point and end-point direction.

**Step3**:-If both directions are matched, then working fine.

**Step4**:- Otherwise it gives empty subsequence.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| -13 | -12 | -11 | -10 | -9 | -8 | -7 | -6 | -5 | -4 | -3 | -2 | -1 |
| I |  | L | O | V | E |  | P | Y | T | H | O | N |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |

Ex:-1

var = "I love python"

print(var[::])

O/P:-

I love python

Ex:2

var = "I love python"

print(var[::-1])

O/P:-

nohtyp evol I

Ex:-3

var = "I love python"

print(var[-2:-5:])

O/P:-

Ex:-4

var = "I love python"

print(var[2:5:-1])

O/P:-

Ex:-5

var = "I love python"

print(var[::2])

O/P:-

Ilv yhn

Ex:-6

var = "I love python"

print(var[::-2])

O/P:-

nhy vlI

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| -18 | -17 | -16 | -15 | -14 | -13 | -12 | -11 | -10 | -9 | -8 | -7 | -6 | -5 | -4 | -3 | -2 | -1 |
| W | E | L | C | O | M | E |  | T | O |  | M | Y |  | B | L | O | G |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |

Ex:-7,8,9,10,11,12

var = "WELCOME TO MY BLOG"

print(var[3:18]) O/P:- COME TO MY BLOG

print(var[2:14:2]) O/P:- LOET Y

print(var[:7]) O/P:- WELCOME

print(var[8:-1:1]) O/P:- TO MY BLO

print(var[-6:-9:-3]) O/P:- Y

print(var[-9:-9:-1]) O/P:-

**Day-8**

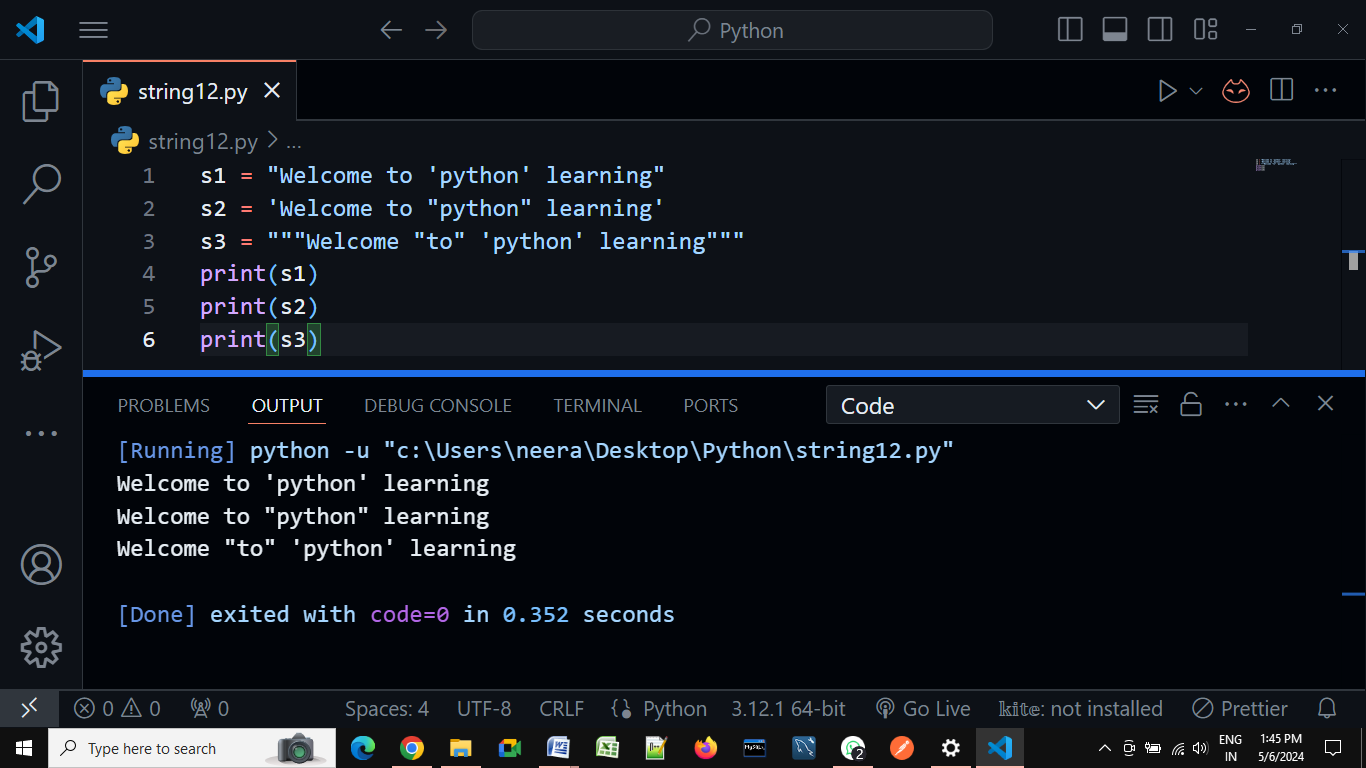
|  |  |
| --- | --- |
| **Type casting** | |
| **Function** | **Description** |
| **int(x)** | converts any data type x to an integer with the mentioned base |
| **float(x)** | converts x data type to a floating-point number |
| **complex(x)** | converts a real number to a complex number |
| **str(x)** | converts x data type to a string |
| **tuple(x)** | converts x data type to a tuple |
| **list(x)** | converts x data type to a list |
| **set(x)** | converts x data type to a set |
| **dict(x)** | creates a dictionary and x data type should be a sequence of (key,value) tuples |
| **ord(x)** | converts a character x into an integer |
| **hex(x)** | converts an integer x to a hexadecimal string |
| **oct(x)** | converts an integer x to an octal string |
| **chr(x)** | converts a number x to its corresponding ASCII character |

**Day-9**

**String**

A group of characters enclosed within single or double or triple quotes is called a string. In other word we can say the string is a sequential collection of characters.

1. **Duplicates are allowed.**
2. **Order is preserved.**
3. **Objects are immutable.**
4. **Indexing is allowed.**
5. **Slicing is allowed.**
6. **Iteration is allowed**.



**In-built python functions for string:-**

* 1. len() - To check how many objects/characters present in string.
  2. max() - To check which object/character may have maximum

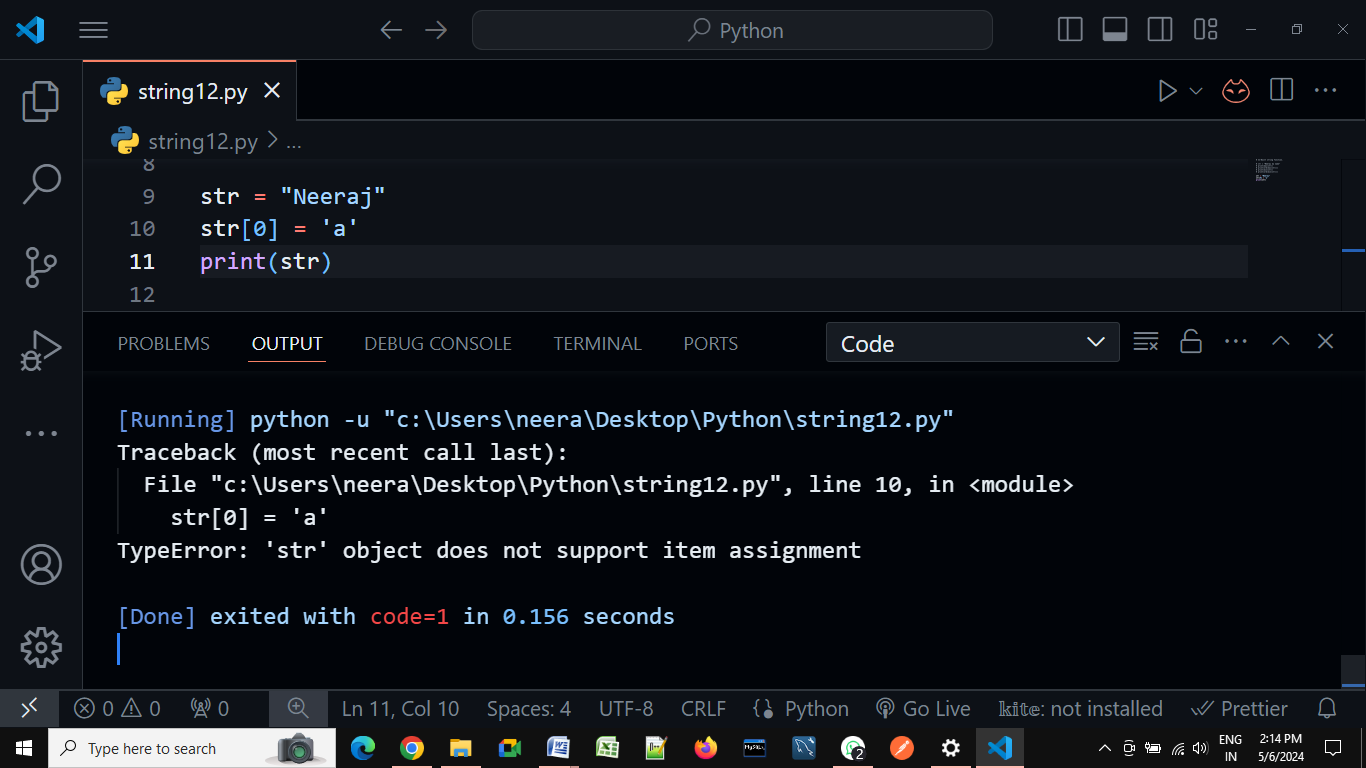
ASCII value.

* 1. min() - To check which object/character may have minimum

ASCII value.

* 1. type() - To check data-type
  2. str() - For type casting
  3. ord() - Character to ASCII value
  4. chr() - ASCII value to character

**Strings are immutable in Python**:-



**Mathematical operators on string objects in Python**

We can perform two mathematical operators on a string. Those operators are:

1. Addition (+) operator.
2. Multiplication (\*) operator.

##### **Addition operator on strings in Python:**

The + operator works like concatenation or joins the strings. While using the + operator on the string then compulsory both arguments should be string type, otherwise, we will get an error.

a = "Python"

b = "Programming"

print(a+b)

O/P:--

PythonProgramming

a = "Python"

b = "Programming"

print(a+" "+b)

O/P:--

Python Programming

##### **Multiplication operator on Strings in Python:**

This is used for string repetition. While using the \* operator on the string then the compulsory one argument should be a string and other arguments should be int type.

a = "Python"

b = 3

print(a\*b)

O/P:--

PythonPythonPython

**Pre-define methods:--**

1. **upper()** – This method converts all characters into upper case

str1 = 'python programming language'

print('converted to using title():', str1.upper())

str2 = 'JAVA proGramming laNGuage'

print('converted to using upper():', str2.upper())

str3 = 'WE ARE SOFTWARE DEVELOPER'

print('converted to using upper ():', str3.upper())

O/P:--

converted to using upper(): PYTHON PROGRAMMING LANGUAGE

converted to using upper (): JAVA PROGRAMMING LANGUAGE

converted to using upper (): WE ARE SOFTWARE DEVELOPER

1. **lower()** – This method converts all characters into lower case

str1 = 'python programming language'

print('converted to using lover():', str1.lower())

str2 = 'JAVA proGramming laNGuage'

print('converted to using lover ():', str2.lower())

str3 = 'WE ARE SOFTWARE DEVELOPER'

print('converted to using lover ():', str3.lower())

O/P:--

converted to using lover(): python programming language

converted to using lover (): java programming language

converted to using lover (): we are software developer

1. **swapcase()** – This method converts all lower-case characters to uppercase and all upper-case characters to lowercase

str1 = 'python programming language'

print('converted to using swapcase():', str1.swapcase())

str2 = 'JAVA proGramming laNGuage'

print('converted to using swapcase ():', str2.swapcase())

str3 = 'WE ARE SOFTWARE DEVELOPER'

print('converted to using swapcase ():', str3.swapcase())

O/P:--

converted to using title(): PYTHON PROGRAMMING LANGUAGE

converted to using title(): java PROgRAMMING LAngUAGE

converted to using title(): we are software developer

1. **title()**– This method converts all character to title case (The first character in every word will be in upper case and all remaining characters will be in lower case)

str1 = 'python programming language'

print('converted to using title():', str1.title())

str2 = 'JAVA proGramming laNGuage'.title()

print('converted to using title():', str2.title())

str3 = 'WE ARE SOFTWARE DEVELOPER'.title()

print('converted to using title():', str3.title())

O/P:--

converted to using title(): Python Programming Language

converted to using title(): Java Programming Language

converted to using title(): We Are Software Developer

1. **capitalize()** – Only the first character will be converted to upper case and all remaining characters can be converted to lowercase.

str1 = 'python programming language'

print('converted to using capitalize():', str1.capitalize())

str2 = 'JAVA proGramming laNGuage'

print('converted to using capitalize ():', str2.capitalize())

str3 = 'WE ARE SOFTWARE DEVELOPER'

print('converted to using capitalize ():', str3.capitalize())

O/P:--

converted to using capitalize(): Python programming language

converted to using capitalize (): Java programming language

converted to using capitalize (): We are software developer

1. **center():-**[Python](https://www.geeksforgeeks.org/python-programming-language/) String center() Method tries to keep the new [string](https://www.geeksforgeeks.org/python-string/) length equal to the given length value and fills the extra characters using the default character (space in this case).

str = "python programming language"

new\_str = str.center(40)

# here fillchar not provided so takes space by default.

print("After padding String is: ", new\_str)

O/P:--

python programming language .

str = "python programming language"

new\_str = str.center(40,'#')

# here fillchar not provided so takes space by default.

print("After padding String is: ", new\_str)

O/P:--

######python programming language#######

str = "python programming language"

new\_str = str.center(15,'#')

# here fillchar not provided so takes space by default.

print("After padding String is: ", new\_str)

O/P:--

python programming language

1. **count():--** **count()** function is an inbuilt function in Python programming language that returns the number of occurrences of a substring in the given string.

**Syntax:** string.Count(substring, start= …., end= ….)

**Parameters:**

The count() function has one compulsory and two optional parameters.

**Mandatory parameter:**

substring – string whose count is to be found.

**Optional Parameters:**

start (Optional) – starting index within the string where the search starts.

end (Optional) – ending index within the string where the search ends.

str = "python programming language"

count = str.count('o')

# here fillchar not provided so takes space by default.

print("count of given charactor is: ", count)

O/P:--

ount of given charactor is: 2

str = "python programming language"

count = str.count('o',5,9)

# here fillchar not provided so takes space by default.

print("count of given charactor is: ", count)

O/P:--

count of given charactor is: 0

1. **Join():---** The string join() method returns a string by joining all the elements of an iterable (list, string, tuple), separated by the given separator.

The join() method takes an iterable (objects capable of returning its members one at a time) as its parameter.Some of the example of iterables are: Native data types - List, Tuple, String, Dictionary and Set.

str = ['Python', 'is', 'a', 'programming', 'language']

# join elements of text with space

print(' '.join(str))

O/P:--

Python is a programming language

str = ['Python', 'is', 'a', 'programming', 'language']

# join elements of text with space

print('\_'.join(str))

O/P:-

Python\_is\_a\_programming\_language

# .join() with lists

numList = ['1', '2', '3', '4']

separator = ', '

print(separator.join(numList))

O/P:--

1, 2, 3, 4

# .join() with tuples

numTuple = ('1', '2', '3', '4')

print(separator.join(numTuple))

O/P:--

1, 2, 3, 4

s1 = 'abc'

s2 = '123'

# each element of s2 is separated by s1

# '1'+ 'abc'+ '2'+ 'abc'+ '3'

print('s1.join(s2):', s1.join(s2))

O/P:--

s1.join(s2): 1abc2abc3

# each element of s1 is separated by s2

# 'a'+ '123'+ 'b'+ '123'+ 'b'

print('s2.join(s1):', s2.join(s1))

O/P:--

s2.join(s1): a123b123c

# .join() with sets

test = {'2', '1', '3'}

s = ', '

print(s.join(test))

O/P:--

2, 3, 1

test = {'Python', 'Java', 'Ruby'}

s = '->->'

print(s.join(test))

O/P:--

Ruby->->Java->->Python

# .join() with dictionaries

test = {'mat': 1, 'that': 2}

s = '->'

# joins the keys only

print(s.join(test))

O/P:--

mat->that

1. **split():--** The split() method splits a string at the specified separator and returns a list of substrings.

str = "Python is a programming language"

print(str.split(" "))

str = "Python is a programming language"

print(str.split(",",2))

print(str.split(":",4))

print(str.split(" ",1))

print(str.split(" ",0))

O/P:--

['Python', 'is', 'a', 'programming', 'language']

['Python is a programming language']

['Python is a programming language']

['Python', 'is a programming language']

['Python is a programming language']

**Day-9**

**---: List :---**

Whenever we want to create a group of objects where we want below mention properties, thenwe are using list sequence.

1. Collection of Homogeneous and Heterogeneous objects.
2. Represented in square bracket with comma separated objects.
3. Duplicates are allowed.
4. Order is preserved.
5. Indexing are allowed.
6. Slicing are allowed.
7. Objects are mutable.
8. Iterations are allowed.

**1. Duplicates are allowed.**

List=['neeraj', 10,20,30,10,20]

print(List)

O/P:--

['neeraj', 10, 20, 30, 10, 20]

**2. Order is preserved:**

List=['neeraj', 10,20,30,10,20]

x=0

for i in List:

    print('List[{}] = '.format(x),i)

    x=x+1

O/P:--

List[0] = neeraj

List[1] = 10

List[2] = 20

List[3] = 30

List[4] = 10

List[5] = 20

**3. Objects are mutable.**

List=['neeraj', 10,20,30,10,20]

x=0

for i in List:

    print('List[{}] = '.format(x),i)

    x=x+1

List[0]="Arvind"

print(List)

O/P:--

List[0] = neeraj

List[1] = 10

List[2] = 20

List[3] = 30

List[4] = 10

List[5] = 20

['Arvind', 10, 20, 30, 10, 20]

**4. Indexing are allowed.**

List=['neeraj', 10,20,30,10,20]

print(List[0])

print(List[1])

print(List[2])

print(List[3])

print(List[4])

print(List[5])

O/P:--

neeraj

10

20

30

10

20

**5. Slicing are allowed:**

List=['neeraj', 10,20,30,10,20]

print(List[:5])

O/P:--

['neeraj', 10, 20, 30, 10]

List=['neeraj', 10,20,30,10,20]

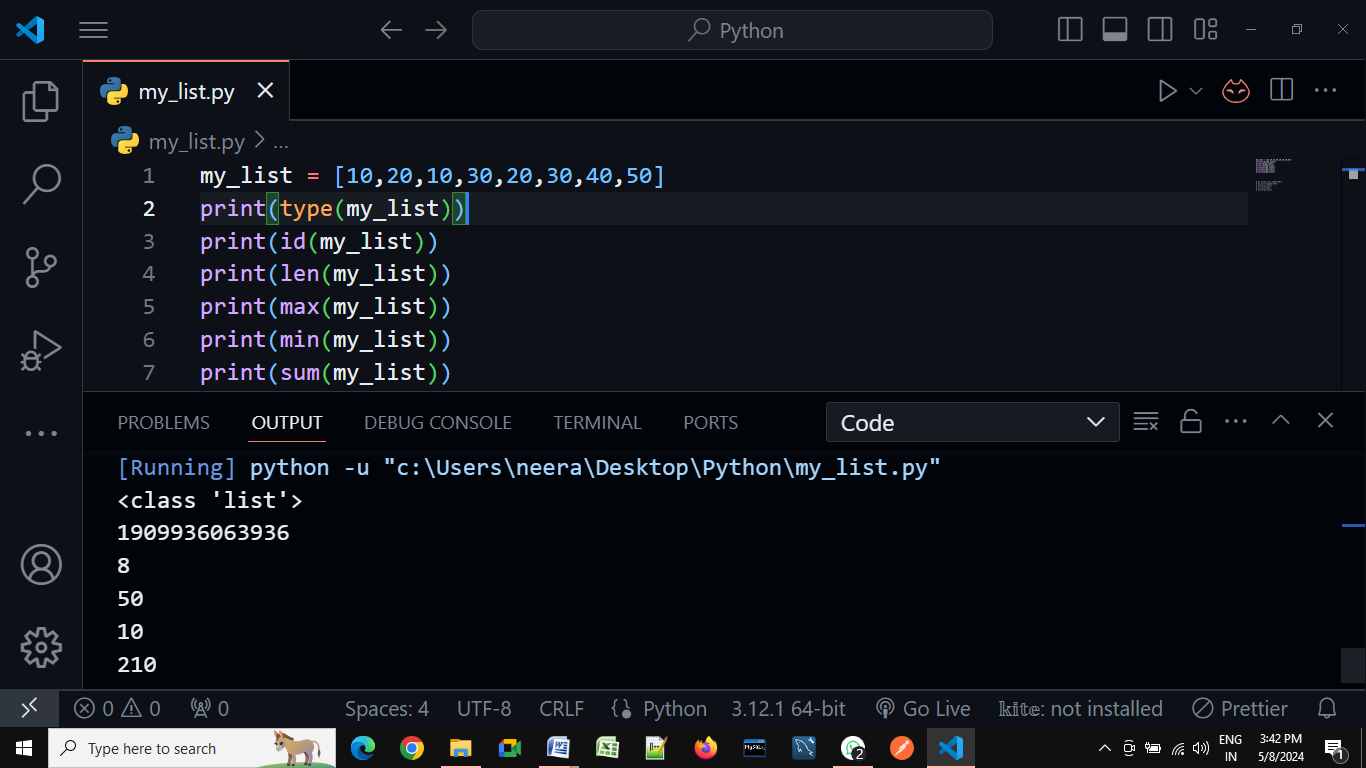
print(List[::-1])

O/P:--

[20, 10, 30, 20, 10, 'neeraj']

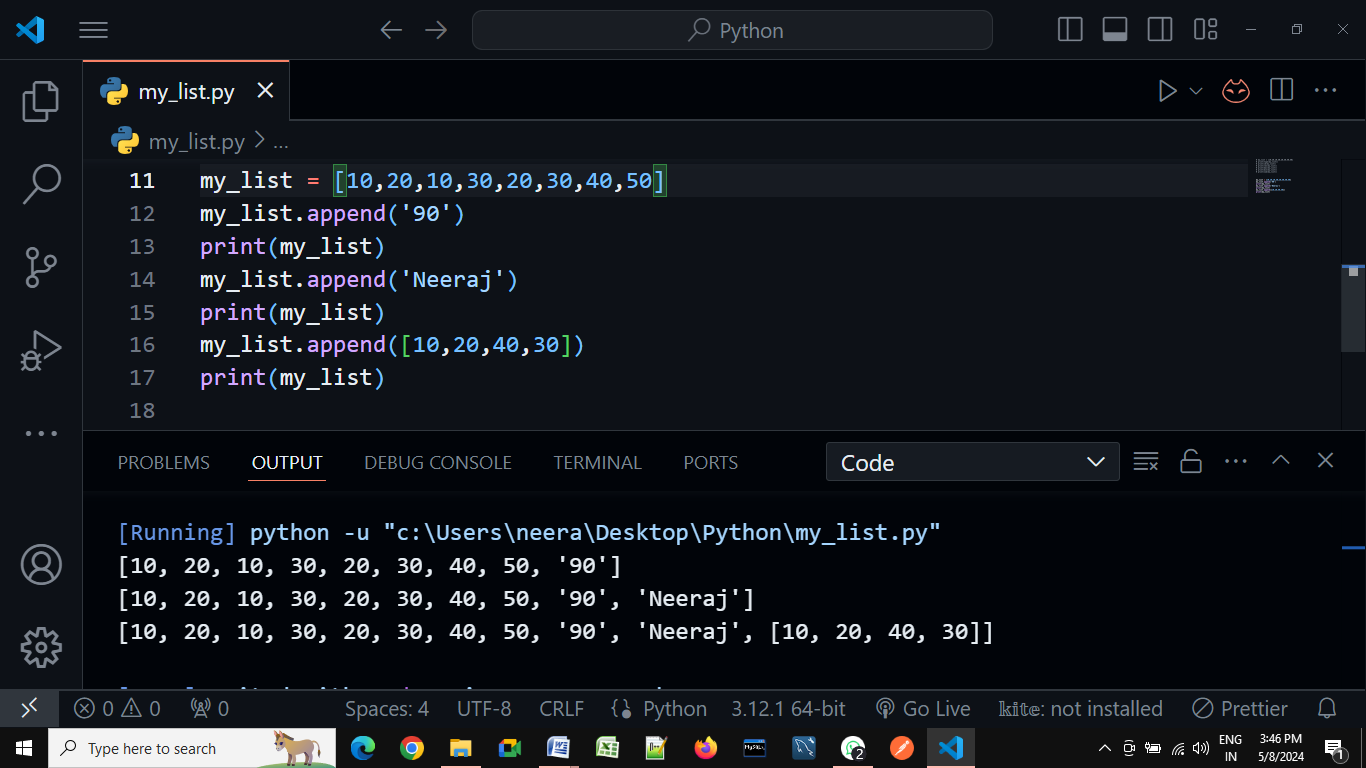
**Inbuilt functions in list:**

* 1. **len(list)**
  2. **max(list)**
  3. **min(list)**
  4. **sum(list)**
  5. **list(tuple)**
  6. **type(list)**

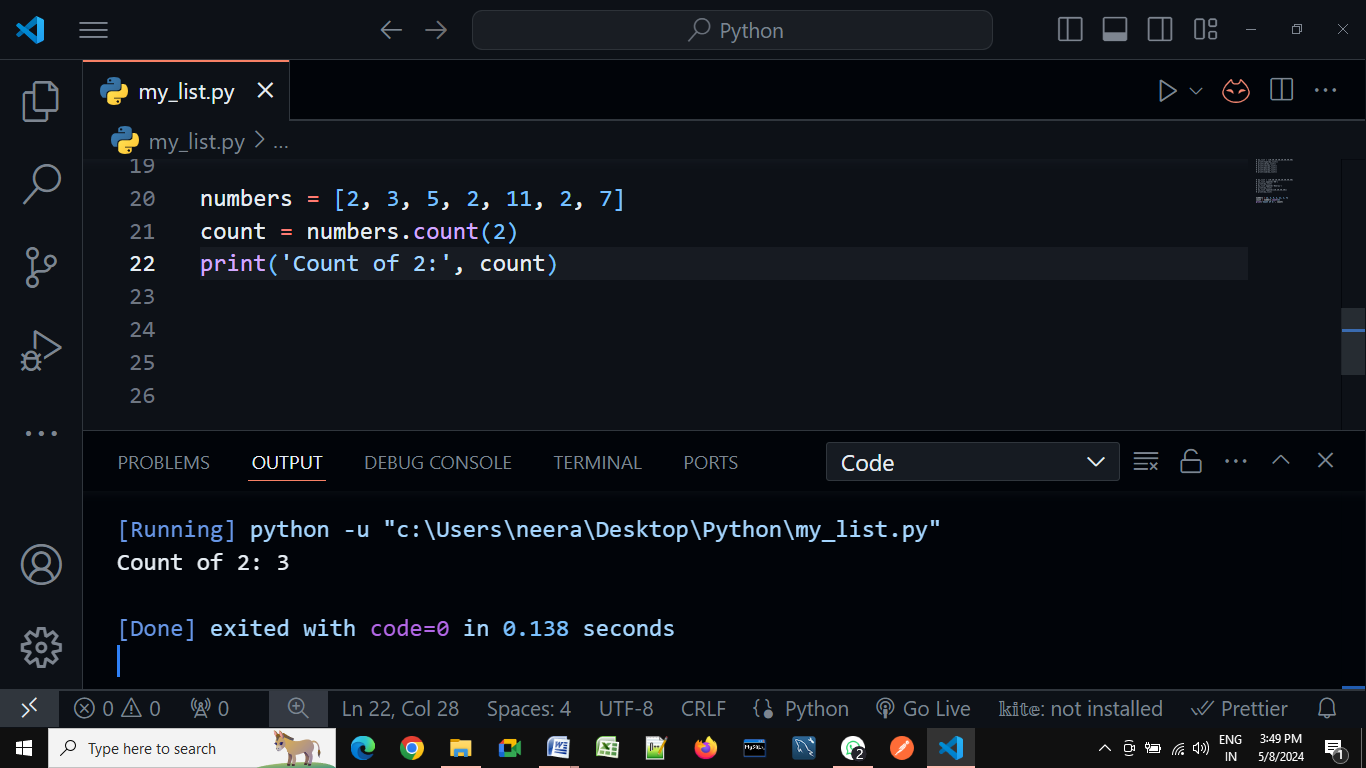
****

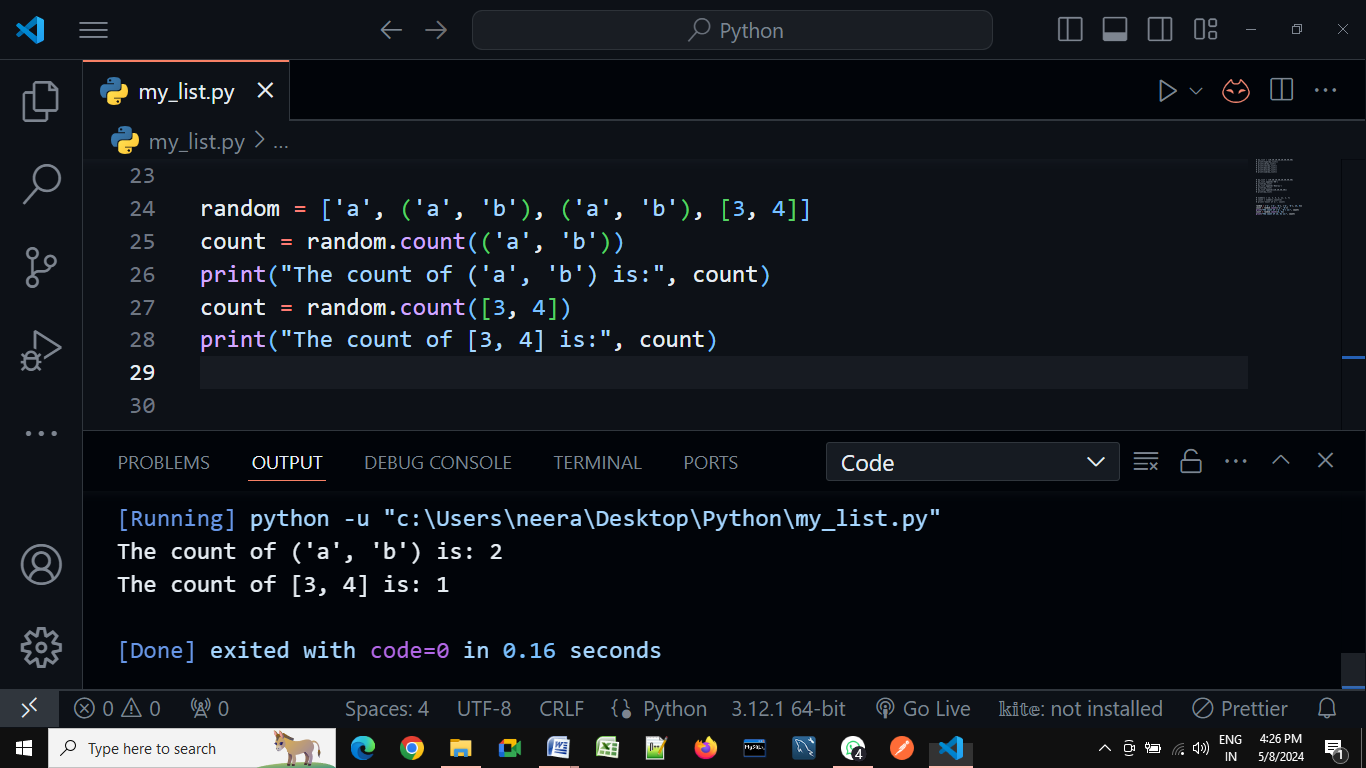
**In-Built Methos:--**

1. **list.append(obj/list/str)-** add object in last:-

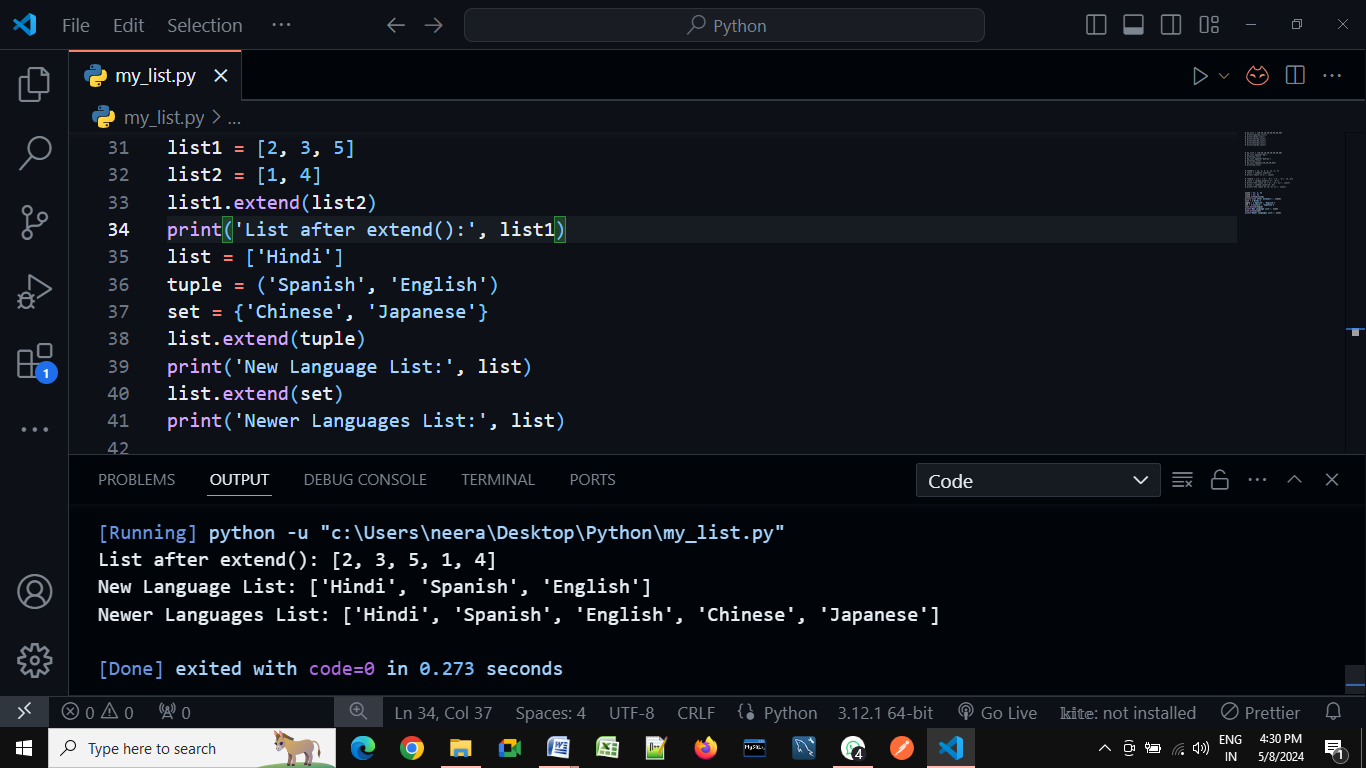


1. **list.count(obj)** – count how many times given-object are present in list





1. **list.extend(list1)** – add list1 in last of list.



1. **list.insert(index,obj)** – insert given object in given index.
2. **list.pop()** – delete last object from given list.
3. **list.remove(obj)** – Remove given object from given list.
4. **list.reverse() –**

Example:---

numbers = ['Neeraj',2, 3, 5, 7]

numbers.reverse()

print('Reversed List:', numbers)

O/P:--

Reversed List: [7, 5, 3, 2, 'Neeraj']

Example:----

numbers = ['Neeraj',2, 3, 5, 7]

print(numbers[::-1])

O/P:--

[7, 5, 3, 2, 'Neeraj']

1. **list.sort**(reverse=True/False) default-False

Example:---

numbers = [2, 3, 7, 5, 4]

numbers.sort()

print('Sort\_List:', numbers)

O/P:--

Sort\_List: [2, 3, 4, 5, 7]

Example:---

numbers = [2, 3, 7, 5, 4]

numbers.sort(reverse=True)

print('Sort\_List:', numbers)

O/P:--

Sort\_List: [7, 5, 4, 3, 2]

**Day-9**

**---: Tuple :---**

In Python, tuples are immutables. Meaning, you cannot change items of a tuple once it is assigned. There are only two tuple methods count() and index() that a tuple object can call.

1. **Represented in parenthesis () with comma separated objects.**
2. **Homogeneous and Heterogeneous both objects are allowed.**
3. **Duplicates are allowed.**
4. **Order is preserved.**
5. **Indexing is allowed.**
6. **Slicing is allowed.**
7. **Objects are immutable.**

Tuple occupies less memory as compare to list, that’s why tuple is more faster as compare to list**.**

**Example:--**

list = [10,20,30,40,50,60,70]

tuple = (10,20,30,40,50,60,70)

print(sys.getsizeof('Size of list = ',list))

print(sys.getsizeof('Size of tuple',tuple))

O/P- 64

62

**Built-in functions:-**

1. **Len(tuple) # tuple variable must be a iterable.**
2. **Max(tuple)**
3. **Min(tuple)**
4. **Sum(tuple)**
5. **Tuple(list)**
6. **Type(tuple)**

**Methods:--**

1. **Count(obj). (How many occurrences)**

# Creating tuples

Tuple = (0, 1, (2, 3), (2, 3), 1, [3, 2],'Neeraj', (0), (0,))

res = Tuple.count((2, 3))

print('Count of (2, 3) in Tuple is:', res)

res = Tuple.count(0)

print('Count of 0 in Tuple is:', res)

res = Tuple.count((0,))

print('Count of (0,) in Tuple is:', res)

O/P:--

Count of (2, 3) in Tuple is: 2

Count of 0 in Tuple is: 2

Count of (0,) in Tuple is: 1

1. **Index(obj,start,stop)(obj is compulsory argument but rest are optional)**

Tuple = (0, 1, 2, 3, 2, 3, 1, 3, 2)

# getting the index of 3

res = Tuple.index(3)

print(res)

O/P:--

3

Tuple = (0, 1, 2, 3, 2, 3, 1, 3, 2)

# getting the index of 3

print(Tuple.index(3,4))

O/P:--

5

Tuple = (0, 1, 2, 3, 2, 3, 1, 3, 2)

# getting the index of 3

print(Tuple.index(3,0,4))

o/p:--

3

**Day-10**

**---: Dictionary** :---

If we want to represent a group of objects as key-value pairs then we should go for dictionaries.

##### **Characteristics of Dictionary**

1. Dictionary will contain data in the form of key, value pairs.
2. One key-value pair can be represented as an single item/object.
3. Duplicate keys are not allowed.
4. Duplicate values can be allowed.
5. Heterogeneous objects are allowed for both keys and values.
6. Insertion order is not preserved.
7. Dictionary object having mutable nature.
8. Dictionary objects are dynamic.
9. Indexing and slicing concepts are not applicable

**syntax** for creating dictionaries with key, value pairs is: **d = { key1:value1, key2:value2, …., keyN:valueN }**

##### **Creating an Empty dictionary in Python:**

d = {}

print(d)

print(type(d))

O/P:--

{}

<class 'dict'>

**Adding the items in empty dictionary:--**

d = {}

d[1] = "Neeraj"

d[2] = "Rahul"

d[3] = "Ravi"

print(d)

O/P:--

{1: 'Neeraj', 2: 'Rahul', 3: 'Ravi'}

**Accessing dictionary values by using keys:--**

d={1: 'Neeraj', 2: 'Rahul', 3: 'Ravi'}

print(d[1])

print(d[2])

print(d[3])

O/P:--

Neeraj

Rahul

Ravi

**Note:---** While accessing, if the specified key is not available then we will get **KeyError**

d={1: 'Neeraj', 2: 'Rahul', 3: 'Ravi'}

print(d[1])

print(d[2])

print(d[3])

print(d[10])

O/P:--

Neeraj

Rahul

Ravi

Traceback (most recent call last):

File "E:\DataSciencePythonBatch\dict.py", line 16, in <module>

print(d[10])

KeyError: 10

##### **Updating dictionary elements:**

We can update the value for a particular key in a dictionary. The syntax is:

**d[key] = value**

**Case1:** While updating the key in the dictionary, if the key is not available then a new key will be added at the end of the dictionary with the specified value.

d={1: 'Neeraj', 2: 'Rahul', 3: 'Ravi'}

print("Old dict data",d)

d[10]="Arvind"

print("Nwe dict data",d)

O/P:--

Old dict data {1: 'Neeraj', 2: 'Rahul', 3: 'Ravi'}

Nwe dict data {1: 'Neeraj', 2: 'Rahul', 3: 'Ravi', 10: 'Arvind'}

**Case2:** If the key already exists in the dictionary, then the old value will be replaced with a new value.

d={1: 'Neeraj', 2: 'Rahul', 3: 'Ravi'}

print("Old dict data",d)

d[2]="Arvind"

print("New dict data",d)

O/P:--

Old dict data {1: 'Neeraj', 2: 'Rahul', 3: 'Ravi'}

New dict data {1: 'Neeraj', 2: 'Arvind', 3: 'Ravi'}

##### **Removing or deleting elements from the dictionary:**

1. By using the del keyword, we can remove the keys
2. By using clear() we can clear the objects in the dictionary

###### **By using the del keyword**

**Syntax: del d[key]**

d={1: 'Neeraj', 2: 'Rahul', 3: 'Ravi'}

del d[3]

print("New dict is",d)

O/P:--

New dict is {1: 'Neeraj', 2: 'Rahul'}

**By using clear() keyword**

d={1: 'Neeraj', 2: 'Rahul', 3: 'Ravi'}

d.clear()

print("New dict is",d)

O/P:--

New dict is {}

##### **Delete entire dictionary object:-** We can also use the del keyword to delete the total dictionary object. Before deleting we just have to note that once it is deleted then we cannot access the dictionary.

d={1: 'Neeraj', 2: 'Rahul', 3: 'Ravi'}

del d

print("New dict is",d) O/P:--

Traceback (most recent call last):

File "E:\DataSciencePythonBatch\dict.py", line 51, in <module>

print("New dict is",d)

NameError: name 'd' is not defined. Did you mean: 'id'?

##### **methods of dictionary in Python**

1. clear() method
2. get() method
3. pop() method
4. popitem() method
5. keys() method
6. items() method
7. values()
8. copy() method

##### **dict() function:**

This can be used to create an empty dictionary.

d=dict()

print(d)

print(type(d))

O/P:--

{}

<class 'dict'>

##### **len() function:** This function returns the number of items in the dictionary.

d={1: 'Neeraj', 2: 'Rahul', 3: 'Ravi'}

print(len(d))

O/P:--

3

##### **clear() method:** This method can remove all elements from the dictionary.

d={1: 'Neeraj', 2: 'Rahul', 3: 'Ravi'}

print(d.clear())O/P:--

O/P:--

None

##### **get() method:**

This method used to get the value associated with the key. This is another way to get the values of the dictionary based on the key. The biggest advantage it gives over the normal way of accessing a dictionary is, this doesn’t give any error if the key is not present. Let’s see through some examples:

**Case1:**If the key is available, then it returns the corresponding value otherwise returns None. It won’t raise any errors.

**Syntax: d.get(key)**

d={1: 'Neeraj', 2: 'Rahul', 3: 'Ravi'}

print(d.get(1))

print(d.get(2))

print(d.get(3))

O/P:--

Neeraj

Rahul

Ravi

**Case 2:** If the key is available, then returns the corresponding value otherwise returns the default value that we give.

**Syntax: d.get(key, defaultvalue)**

d={1: 'Neeraj', 2: 'Rahul', 3: 'Ravi'}

print(d.get(7,"Neeraj"))

print(d.get(6,"Neeraj"))

print(d.get(5,"Neeraj"))

O/P:--

Neeraj

Neeraj

Neeraj

##### **pop() method:** This method removes the entry associated with the specified key and returns the corresponding value. If the specified key is not available, then we will get KeyError.

**Syntax: d.pop(key)**

d={1: 'Neeraj', 2: 'Rahul', 3: 'Ravi'}

d.pop(3)

print(d)

O/P:

{1: 'Neeraj', 2: 'Rahul'}

d={1: 'Neeraj', 2: 'Rahul', 3: 'Ravi'}

print(d.pop(3)) O/P:-- Ravi

##### **popitem() method:** This method removes an arbitrary item(key-value) from the dictionary and returns it.

d={1: 'Neeraj', 2: 'Rahul', 3: 'Ravi',4:'Jai',5:'Santosh'}

print(d.popitem())

print(d)

O/P:--

(5, 'Santosh')

{1: 'Neeraj', 2: 'Rahul', 3: 'Ravi', 4: 'Jai'}

##### **keys() method:**This method returns all keys associated with the dictionary

d = {1: 'Ramesh', 2: 'Suresh', 3: 'Mahesh'}

print(d)

for k in d.keys():

   print(k)

O/P:--

1

2

3

##### **values() method:** This method returns all values associated with the dictionary

d = {1: 'Ramesh', 2: 'Suresh', 3: 'Mahesh'}

print(d)

for k in d.values():

   print(k)

O/P:--

Ramesh

Suresh

Mahesh

##### **items() method:** A key-value pair in a dictionary is called an item. items() method returns the list of tuples representing key-value pairs.

d = {1: 'Ramesh', 2: 'Suresh', 3: 'Mahesh'}

for k, v in d.items():

   print(k, "---", v)

O/P:--

1 --- Ramesh

2 --- Suresh

3 --- Mahesh

**Day-9**

**---: Set :---**

If we want to represent a group of unique elements then we can go for sets. Set cannot store duplicate elements.

1. **Duplicates are not allowed.**
2. **Order is not preserved.**
3. **Objects are mutable.**
4. **Indexing is not allowed.**
5. **Slicing is not allowed.**
6. **Represented in { } with comma separated objects.**
7. **Homogeneous and Heterogeneous both objects are allowed.**

# Creating a set

s = {10,20,30,40}

print(s)

print(type(s))

O/P:--

{40, 10, 20, 30}

<class 'set'>

# Creating a set with different elements

s = {10,'20','Rahul', 234.56, True}

print(s)

print(type(s))

O/P:--

{'20', True, 234.56, 10, 'Rahul'}

<class 'set'>

# Creating a set using range function

s=set(range(5))

print(s)

O/P:--

{0, 1, 2, 3, 4}

# Duplicates not allowed

s = {10, 20, 30, 40, 10, 10}

print(s)

print(type(s))

O/P:--

{40, 10, 20, 30}

<class 'set'>

# Creating an empty set

s=set()

print(s)

print(type(s))

O/P:--

set()

<class 'set'>

**# Methods in set:----**

1. **add(only\_one\_argument not iterable)**

s={10,20,30,50}

s.add(40)

print(s)

O/P:--

{40, 10, 50, 20, 30}

1. **update(iterable\_obj1,iterable\_obj2)**

s = {10,20,30}

l = [40,50,60,10]

s.update(l)

print(s)

O/P:--

{40, 10, 50, 20, 60, 30}

s = {10,20,30}

l = [40,50,60,10]

s.update(l, range(5))

print(s)

O/P:--

{0, 1, 2, 3, 4, 40, 10, 50, 20, 60, 30}

##### **Difference between add() and update() methods in set:**

1. We can use add() to add individual items to the set, whereas we can use update() method to add multiple items to the set.
2. The add() method can take one argument whereas the update() method can take any number of arguments but the only point is all of them should be iterable objects.
3. **copy() --Clone of set**

s={10,20,30}

s1=s.copy()

print(s1)

O/P:--

{10, 20, 30}

1. **pop()---** This method removes and returns some random element from the set.

s = {40,10,30,20}

print(s)

print(s.pop())

print(s)

O/P:--

{40, 10, 20, 30}

40

{10, 20, 30}

1. **remove(element) ---** This method removes specific elements from the set. If the specified element is not present in the set then we will get KeyError.

s={40,10,30,20}

s.remove(30)

print(s)

O/P:--

{40, 10, 20}

s={40,10,30,20}

s.remove(50)

print(s)

O/P:--

Traceback (most recent call last):

File "E:\DataSciencePythonBatch\sets.py", line 65, in <module>

s.remove(50)

KeyError: 50

1. **discard(element) ---** This method removes the specified element from the set. If the specified element is not present in the set, then we won’t get any error.

s={10,20,30}

s.discard(10)

print(s)

O/P:--

{20, 30}

s={10,20,30}

s.discard(40)

print(s)

O/P:--

{10, 20, 30}

1. **clear() --- removes all elements from the set.**

s={10,20,30}

print(s)

s.clear()

print(s)

O/P:--

{10, 20, 30}

set()

**MATHEMATICAL OPERATIONS ON SETS**

1. **union() ---** This method return all elements present in both sets.

x={10,20,30,40}

y={30,40,50,60}

print(x.union(y))

O/P:--

{40, 10, 50, 20, 60, 30}

1. **intersection() ---** This method returns common elements present in both x and y.

x = {10,20,30,40}

y = {30,40,50,60}

print(x.intersection(y))

print(x&y)

print(y.intersection(x))

print(y&x)

O/P:--

{40, 30}

{40, 30}

{40, 30}

{40, 30}

1. **difference() ---** This method returns the elements present in x but not in y

x = {"apple", "banana", "cherry"}

y = {"google", "microsoft", "apple"}

z = x.difference(y)

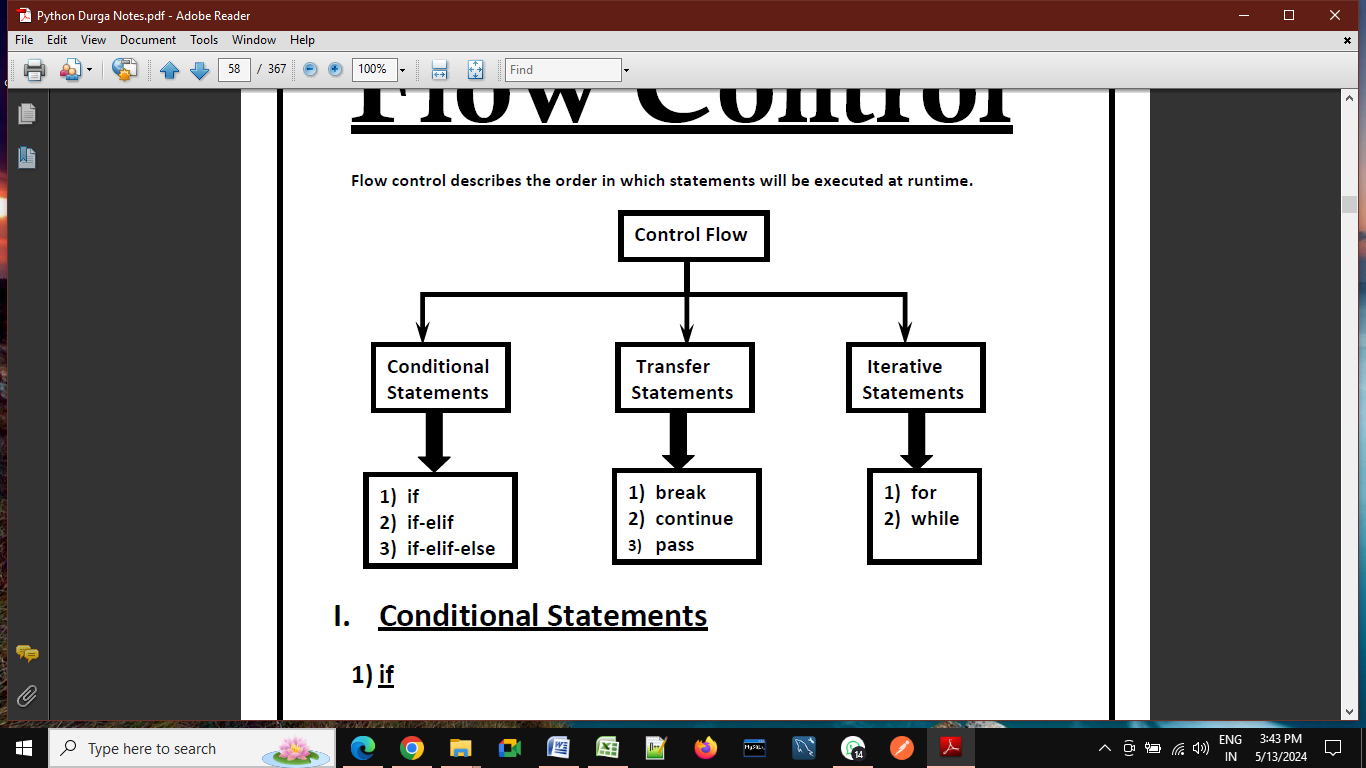
print(z)

O/P:--- {'banana', 'cherry'}

**Day-11**

**Control Flow Statements**

In programming languages, flow control means the order in which the statements or instructions, that we write, get executed. In order to understand a program, we should be aware of what statements are being executed and in which order. So, understanding the flow of the program is very important. There are, generally, three ways in which the statements will be executed. They are,



**Conditional:**Statements are executed based on the condition. As shown above in the flow graph, if the condition is true then one set of statements are executed, and if false then the other set. Conditional statements are used much in complex programs.

Conditional statements are also called decision-making statements. Let’s discuss some conditions making statements in detail. There are three types of conditional statements in python. They are as follows:

1. **if statement**
2. **if-else statement**
3. **nested-if (if-elif-elif-else)**

**if-statement:-**

syntax:-

if condition:

print("Block statement")

print("Out of block statement")

**Example:-**

num=int(input("Enter any no: "))

if num>=18:

    print("if block statment executed")

print("out of if block statements")

O/P:

Enter any no: 10

out of if block statements

PS E:\DataSciencePythonBatch> python control.py

Enter any no: 18

if block statment executed

out of if block statements

**Example:---**

# Example:--- Checking if a number is positive, negative, or zero.

num = float(input("Enter a number: "))

if num > 0:

    print("The number is positive.")

elif num < 0:

    print("The number is negative.")

else:

    print("The number is zero.")

**if-else condition:-**

**syntax:-**

if condition:

print("if block statement executed")

else:

print("else block statement executed ")

**Example:---**

num=int(input("Enter any no: "))

if num>=18:

    print("if block statement executed")

else:

    print("else block statement executed")

O/P:-

PS E:\DataSciencePythonBatch> python control.py

Enter any no: 18

if block statement executed

PS E:\DataSciencePythonBatch> python control.py

Enter any no: 15

else block statement executed

**Example:---**

# Example:-- Checking if a person is eligible to vote

age = int(input("Enter your age: "))

if age >= 18:

    print("You are eligible to vote.")

else:

    print("You are not eligible to vote.")

O/P:-

Enter your age: 35

You are eligible to vote.

**Example:---**

# Example:-- Checking if a year is a leap year

year = int(input("Enter a year: "))

if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0 and year %100==0):

    print("It's a leap year.")

else:

    print("It's not a leap year.")

O/P:-

Enter a year: 2000

It's a leap year.

**Nested-If else:--**

**Syntex:-**

if condition:

print("if block statement executed")

elif:

print("elif block statement executed ")

elif:

print("elif block statement executed ")

elif:

print("elif block statement executed ")

else:

print("else block statement executed ")

# Example:-- Check your gread based on your own score

score = int(input("Enter your score: "))

if score >= 90:

    print("You got an A.")

else:

    if score >= 80:

        print("You got a B.")

    else:

        if score >= 70:

            print("You got a C.")

        else:

            if score >= 60:

                print("You got a D.")

            else:

                print("You got an F.")

O/P:-

Enter your score: 90

You got an A.

Example:---

# example:-- Please choose value within range of o to 4.

print("Please enter the values from 0 to 4")

x=int(input("Enter a number: "))

if x==0:

       print("You entered:", x)

elif x==1:

       print("You entered:", x)

elif x==2:

       print("You entered:", x)

elif x==3:

       print("You entered:", x)

elif x==4:

       print("You entered:", x)

else:

       print("Beyond the range than specified")

O/P:-

Enter a number: 5

Beyond the range than specified

PS E:\DataSciencePythonBatch> python control.py

Please enter the values from 0 to 4

Enter a number: 4

You entered: 4

**Example:- Python Program to calculate the square root.**

# Example:-Python

num = float(input('Enter a number: '))

num\_sqrt = num \*\* 0.5

print('The square root of Num :', num\_sqrt)

O/P:-

Enter a number: 4

The square root of 4.000 is 2.000

PS E:\DataSciencePythonBatch> python control.py

Enter a number: 8

The square root of 8.000 is 2.828

**Example:-- Python Program to find the area of triangle.**

# Python Program to find the area of triangle

# s = (a+b+c)/2

# area = √(s(s-a)\*(s-b)\*(s-c))

a = float(input('Enter first side: '))

b = float(input('Enter second side: '))

c = float(input('Enter third side: '))

s = (a + b + c) / 2

area = (s\*(s-a)\*(s-b)\*(s-c)) \*\* 0.5

print('The area of the triangle is :', area)

O/P:---

Enter first side: 5

Enter second side: 6

Enter third side: 7

The area of the triangle is : 14.696938456699069

**Example:-- Python program to swap two variables.**

# Python program to swap two variables

x = input('Enter value of x: ')

y = input('Enter value of y: ')

# create a temporary variable and swap the values

temp = x

x = y

y = temp

print('The value of x after swapping: {}'.format(x))

print('The value of y after swapping: {}'.format(y))

O/P:---

Enter value of x: 5

Enter value of y: 8

The value of x after swapping: 8

The value of y after swapping: 5

# without using third variable

x = input('Enter value of x: ')

y = input('Enter value of y: ')

x, y = y, x

print('The value of x after swapping: {}'.format(x))

print('The value of y after swapping: {}'.format(y))

O/P:---

Enter value of x: 4

Enter value of y: 6

The value of x after swapping: 6

The value of y after swapping: 4

# By-using Addition and Subtraction.

x = int(input('Enter value of x: '))

y = int(input('Enter value of y: '))

x = x + y

y = x - y

x = x - y

print('The value of x after swapping: {}'.format(x))

print('The value of y after swapping: {}'.format(y))

O/P:---

Enter value of x: 4

Enter value of y: 6

The value of x after swapping: 6

The value of y after swapping: 4

# By-using Multiplication and division.

x = int(input('Enter value of x: '))

y = int(input('Enter value of y: '))

x = x \* y

y = x / y

x = x / y

print('The value of x after swapping: {}'.format(x))

print('The value of y after swapping: {}'.format(y))

O/P:---

Enter value of x: 2

Enter value of y: 5

The value of x after swapping: 5.0

The value of y after swapping: 2.0

# By-using x-or(^) operator.

x = int(input('Enter value of x: '))

y = int(input('Enter value of y: '))

x = x ^ y

y = x ^ y

x = x ^ y

print('The value of x after swapping: {}'.format(x))

print('The value of y after swapping: {}'.format(y))

O/P:--

Enter value of x: 10

Enter value of y: 20

The value of x after swapping: 20

The value of y after swapping: 10

**Day-12**

**Transfer Statements:**

* 1. Break
  2. continue
  3. pass

**Break statement:---**

We can use break statement inside loops to break loop execution based on some condition.

for i in range(10):

    if i==7:

        print("processing is enough..  plz break !!!!!!! ")

        break

    print(i)

O/P:---

0

1

2

3

4

5

6

processing is enough.. plz break !!!!!!!

list=[10,20,600,60,70]

for i in list:

    if i>500:

        print("no need to check next object of list")

        break

    print(i)

O/P:--

10

20

no need to check next object of list

**continue statement:--** We can use continue statement to skip current iteration and continue next iteration.

for i in range(10):

    if i%2==0:

        continue

    print(i)

O/P:--

1

3

5

7

9

list=[10,20,600,60,70]

for i in list:

    if i>500:

        print("no need to print this object")

        continue

    print(i)

O/P:--

10

20

no need to print this object

60

70

list=[10,20,600,60,70]

for i in list:

    if i>500:

        continue

    print(i)

O/P:--

10

20

60

70

**pass statement:--**

pass is a keyword in Python. In our programming syntactically if block is required which won't do anything then we can define that empty block with pass keyword.

1. It is an empty statement
2. It is null statement
3. It won't do anything

for i in range(100):

    if i%9==0:

        print(i)

    else:

        pass

O/P:--

0

9

18

27

36

45

54

63

72

81

90

99

**Day-13**

**----:LOOPING Statement in Python (Iterations):----**

If we want to execute a group of statements multiple times, then we should go for a looping kind of execution. There are two types of loops available in python.

They are:

1. **while loop**
2. **for loop**

**1. while loop:-** The while loop contains an expression/condition. As per the syntax colon (:) is mandatory otherwise it throws a syntax error. The condition gives the result as bool type, either True or False. The loop keeps on executing the statements until the condition becomes False. i.e. With the while loop we can execute a set of statements as long as a condition is true.

##### **Parts of while loop in Python:**

**Initialization:**

This is the first part of the while loop. Before entering the condition section, some initialization is required.

**Condition:**

Once the initializations are done, then it will go for condition checking which is the heart of the while loop. The condition is checked and if it returns True, then execution enters the loop for executing the statements inside.

After executing the statements, the execution goes to the increment/decrement section to increment the iterator. Mostly, the condition will be based on this iterator value, which keeps on changing for each iteration. This completes the first iteration. In the second iteration, if the condition is False, then the execution comes out of the loop else it will proceed as explained in the above point.

**Increment/Decrement section:**This is the section where the iterator is increased or decreased. Generally, we use arithmetic operators in the loop for this section.

**Example: Printing numbers from 1 to 5 by using while loop**

1. The program is to print the number from 1 to 5
2. Before starting the loop, we have made some assignments( x = 1). This is called the Initialization section.
3. After initialization, we started the while loop with a condition x<=5. This condition returns True until x is less than 5.
4. Inside the loop, we are printing the value of x.
5. After printing the x value, we are incrementing it using the operator x+=1. This is called the increment/decrement section.
6. For each iteration, the value of x will increase and when the x value reaches 6, then the condition x<=5 returns False. At this iteration, the execution comes out of the loop without executing the statements inside. Hence in the output ‘6’ is not printed.

x=1

while x<=5:

    print(x)

    x+=1

O/P:--

1

2

3

4

5

*# Printing numbers from 1 to 5 by using while loop.*

x=1

while x<=5:

    print(x)

    x+=1

*# Printing numbers from 1 to 5 by using while loop.*

x=1

while x<=5:

    if x<5:

        print(x,end=",")

    else:

        print(x,end="")

    x+=1

*# Printing even numbers from 10 to 20 by using while loop.*

x=10

while (x>=10) and (x<=20):

   print(x)

   x+=2

print("End")

*# print sun of given n netural no*

x=int(input("Enter any no : "))

sum=0

i=1

while i<=x:

    sum=sum+i

    if i<x:

        print(i,end="+")

    else:

        print(i,end="=")

    i=i+1

print(sum)

*# print n even numbers*

x= int(input("Enter how many even number you want :"))

n=1

while n<=x:

    print(2\*n)

    n=n+1

*# print n even numbers(1,2,3,4,5,6,--------)*

x= int(input("Enter how many even numbers you want :"))

n=1

while n<=x:

    if n<x:

        print(2\*n,end=",")

    else:

        print(2\*n,end="")

    n=n+1

*# Print sum of given even numbers.*

x= int(input("Enter how many even numbers sum you want :"))

n=1

sum=0

while n<=x:

    sum=sum+2\*n

    if n<x:

        print(2\*n,end="+")

    else:

        print(2\*n,end="=")

    n=n+1

print(sum)

*# Print n odd numbers*

x= int(input("Enter how many odd number you want :"))

n=1

while n<=x:

    if n<x:

        print((2\*n-1),end=",")

    else:

        print((2\*n-1),end="")

    n=n+1

*# Print sum of n odd numbers*

x= int(input("Enter how many odd number you want :"))

n=1

sum = 0

while n<=x:

    sum=sum+(2\*n-1)

    if n<x:

        print((2\*n-1),end="+")

    else:

        print((2\*n-1),end="=")

    n=n+1

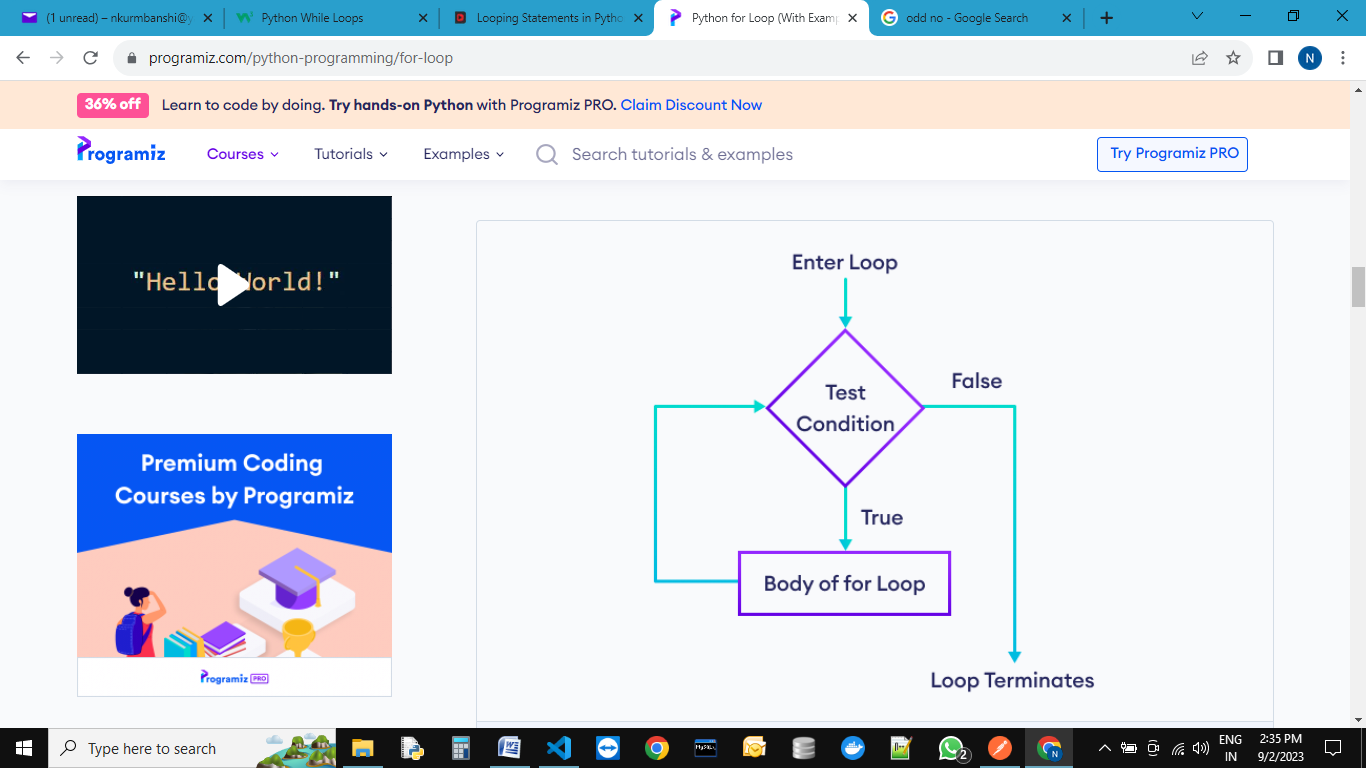
print(sum)

**Day-14**

##### **for loop in python:**

Basically, a for loop is used to iterate elements one by one from sequences like string, list, tuple, etc. This loop can be easily understood when compared to the while loop. While iterating elements from the sequence we can perform operations on every element.

The Python For Loop is used to repeat a block of statements until there are no items in the Object may be String, List, Tuple, or any other object.



1. Initialization: We initialize the variable(s) here. Example i=1.
2. Items in Sequence / Object: It will check the items in Objects. For example, individual letters in String word. If there are items in sequence (True), then it will execute the statements within it or inside. If no item is in sequence (False), it will exit.
3. After completing the current iteration, the controller will traverse to the next item.
4. Again it will check the new items in sequence. The statements inside it will be executed as long as the items are in sequence.

**Range() function :----**

The Python range function helps to generate a sequence of numbers. Or this Python range function helps to iterate items in iterables such as Lists, Tuples, Sets, Strings, etc.

The syntax of this Python function contains range start, stop, step. All the arguments such as start, stop, and step accepts positive or Negative integers.

Syntax: range(start, stop, step)

* **Start**(optional) – Starting position number. If you omit this, the Python range function will start from 0.
* **Stop** – A value before this number is the end value. For example, (1, 10) print values from 1 to 9.
* **Step**(optional) – Sequence of numbers generated. It determines the space or difference between each integer value. For example, (1, 10, 2) returns 1, 3, 5, 7, and 9. You can notice that the difference between each integer is 2.

Python's built-in range() function is mainly used when working with for loops – you can use it to loop through certain blocks of code a specified number of times. The range() function accepts three arguments – one is required, and two are optional. By default, the syntax for the range() function looks similar to the following:

**range(stop)---**The stop argument is **required**.

The range() function returns a sequence of numbers starting from 0, incrementing by 1, and ending at the value you specify as stop (non-inclusive). But what if you want to iterate through a range of two numbers you specify and don't want to start the counting from 0? You can pass a second **optional** start argument, start, to specify the starting number. The syntax to do so looks like this:

**range(start, stop)**

This syntax generates a sequence of numbers based on the start (inclusive) and stop (non-inclusive) values that increment by 1.

Lastly, if you don't want the default increment to be 1, you can specify a third **optional** argument, step. The syntax to do that looks like this:

**range(start, stop, step)**

x=range(11)

print(x)

print(list(x))

print(tuple(x))

print(set(x))

O/P:--

range(0, 11)

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

**Day-15**

**Practice Questions**

**1. Write a code to print below mention sequences:--(for loop with the help of range)**

1. 1,2,3,4,5,6,7,8,9,10

2. 1+2+3+4+5+6+7+8+9+10=(Sum of all no)

3. 2,4,6,8,10,12,14,16,18,20

4. 2+4+6+8+10+12+14+16+18+20=(Sum of all no)

5. 1,3,5,7,9,11,13,15,17,19

6. 1+3+5+7+9+11+13+15+17+19=(Sum of all no)

**2. Write a code to print below mention sequences:--(while loop)**

1. 1,2,3,4,5,6,7,8,9,10

2. 1+2+3+4+5+6+7+8+9+10=(Sum of all no)

3. 2,4,6,8,10,12,14,16,18,20

4. 2+4+6+8+10+12+14+16+18+20=(Sum of all no)

5. 1,3,5,7,9,11,13,15,17,19

6. 1+3+5+7+9+11+13+15+17+19=(Sum of all no)

1. Write a program to print below mention series- 1,3,6,10,15,21,28,36,45,55

n=int(input("Enter any Netural No:"))

i=1

sum=0

while(i<=n):

 sum=sum+i

 i=i+1

 print("",sum)

1. Write a program to check given no is Armstrong or not ( 153 = 1\*\*3+5\*\*3+3\*\*3 )

# 1st method-------------------------

n=int(input("Enter Anu Number"))

i=0

sum=0

count=0

m=n

p=m

while n>0:

    n=n//10

    count+=1

while m>0:

    r=m%10

    k=r\*\*count

    sum=sum+k

    m=m//10

if(sum==p):

    print("Number is armstrong")

else:

    print("Number is not armstrong")

# 2nd Method------------------

n=int(input("Enter Anu Number"))

i=0

sum=0

m=n

count = len(str(n))

while m>0:

    r=m%10

    # k=r\*\*count

    sum=sum+k(r\*\*count)

    m=m//10

if(sum==n):

    print("Number is armstrong")

else:

    print("Number is not armstrong")

1. Write a code to check given no is prime no or not:

n=int(input("Enter Any Nomber: "))

i=1

count=0

while i<=n:

        if n%i==0:

            count+=1

        i+=1

if count<2:

    print("Number is prime")

else:

    print("Not Prime Number")

6 . Write a code to print Fibonacci series up to given no.(0,1,1,2,3,5,8,13,21….)

n=int(input("Enter Any No."))

a,b,c,i=0,1,0,1

print(a)

print(b)

while i<=n:

 c=a+b

 a=b

 b=c

 i=i+1

 print(c)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Leep year-Concept** | | | | |
| **Year** | **Day** | **hour** | **minutes** | **seconds** |
| 1-Year | 365 | 5 | 48 | 47.5 |
| 2-Year | 365 | 5 | 48 | 47.5 |
| 3-Year | 365 | 5 | 48 | 47.5 |
| 4-Year | 365 | 5 | 48 | 47.5 |
|  | **Remain times** | 20 | 192 | 190 |
|  |  | 195/60=3h,15m | 190/6= 3min,10sec |
| Approx 1-day which is added in every 4years | **23** | **15** | **10** |
| Add some time |  |  |
| leep year | 366 |  |  |  |
|  |  |  |  |  |
|  | negative time in leep year(-) | extra added time | 44 | 50 |
|  |  |  |  |  |
| 100 year |  |  | 44\*25=1100 | 50\*25=1250 |
|  |  |  |  | 1250/60 |
|  |  |  | 1120/60 | 20 minutes, 50 sec |
|  |  |  | 18hours,40 minutes |  |
|  |  | 18 | 40 | 50 |
| 100-years | 1-day remove in feb month | Not a Leep year | | |
|  |  |  |  |  |
|  |  |  |  |  |
|  | **( approx 6hour behind in 100th year)** |  |  |  |
|  | **( approx 6hour behind in 100th year)** |  |  |  |
|  | **( approx 6hour behind in 100th year)** |  |  |  |
| **400 years** | **Now add 1-day in 400th year** | **That’s why it is leep year** | | |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  | 44\*400 | 50\*400 |
|  |  |  | 4400 | 5000 |
|  |  |  | 4480/60=74h,40m | 5000/60=80minutes,20sec |
|  |  | 74 | 40 | 20 |

7 . Write a code to check given year is leep year or not

year = int(input("Enter a year: "))

if (year % 4 ==0) and (year % 100 != 0):

    print("{0} is a leap year".format(year))

elif (year % 400 == 0) and (year % 100 == 0):

    # print("{0} is a leap year".format(year))

    print(f'{year} is a leep year by using f string')

else:

    print("{0} is not a leap year".format(year))

8 . Write a code to check given no is palindrome or not(madam=madam)

1st method:---(only for integer)

n = int(input("enter any num :"))

m = n

rev = 0

while m > 0:

    r = m % 10

    rev = (rev \* 10) + r

    m = m // 10

if n == rev:

  print('Palindrome')

else:

  print("Not Palindrome")

2nd mehud for string:

Write a code to print your marksheet :

A1=str(input("Enter Your School Name"))

B1=str(input("Enter Your Center Code"))

C1=str(input("Enter Your School Code"))

D1=str(input("Enter Your Class"))

E1=str(input("Enter Your D.O.B"))

R1=str(input("Enter Your Roll no"))

P1=str(input("Enter Your Name"))

F1=str(input("Enter Your FName"))

M1=str(input("Enter Your MName"))

H1=int(input("Enter Hindi Mark"))

En=int(input("Enter English Mark"))

S1=int(input("Enter Sanskrit Mark"))

M2=int(input("Enter Math Mark"))

S2=int(input("Enter Science Mark"))

S3=int(input("Enter So.Science Mark"))

print("\t\tBOARDOF SECONDARY EDUCATION, M.P Bhopal")

print("\tHIGH SCHOOL CERTIFICATE EXAMINATION (10+2)2002, M.P Bhopal",end=" ")

print("\nSchool:",A1,end="")

print("\n\nRoll No:",R1,"\t\t\t\t\tCenter:",B1,end="")

print("\nName:",P1,"\t\t\t\tSchool:",C1,end="")

print("\nFName:",F1,"\t\t\tClass:",D1,end="")

print("\nMName:",M1,"\t\t\tD.O.B:",E1,end="")

print("\n\n\tSubcode   Sub Name  Maximun  Minimum  Obtain")

print("\t   H0001     Hindi     100      33      :",H1)

print("\t   E0001     English   50       17      :",En)

print("\t   S0001     Sanskrit  50       17      :",S1)

print("\t   M0001     Math      100      33      :",M2)

print("\t   SC001     Science   100      33      :",S2)

print("\t   SOS01     Sosal Sc. 100      33      :",S3)

if(H1<0 or H1>100):

  print("Invailid Marks")

elif(En<0 or En>100):

  print("Invailid Marks")

elif(S1<0 or S1>100):

  print("Invailid Marks")

elif(M2<0 or M2>100):

  print("Invailid Marks")

elif(S2<0 or S2>100):

  print("Invailid Marks")

elif(S3<0 or S3>100):

  print("Invailid Marks")

else:

 total=H1+En+S1+M2+S2+S3

 Per=total/5

 print("Total Marks:",total)

 print("Total Percentage:",Per)

 if(Per>=60):

  print("Pass in first division")

 elif(Per>=45):

  print("Pass in Second division")

 elif(Per>=33):

  print("Pass in third division")

 else:

  print("Fail")



**Day-16**

**---: Pattern Questions :---**

# \*

# \*\*

# \*\*\*

# \*\*\*\*

# \*\*\*\*\*

n=int(input("Enter the number of rows: "))

for i in range(1,n+1):

    print("\*"\*i)

# \*

# \* \*

# \* \* \*

# \* \* \* \*

# \* \* \* \* \*

n=int(input("Enter the number of rows: "))

for i in range(1,n+1):

    print("\* "\*i)

#      \*

#     \*\*

#    \*\*\*

#   \*\*\*\*

#  \*\*\*\*\*

n=int(input("Enter the number of rows: "))

for i in range(1,n+1):

    print(" "\*(n-i),"\*"\*i)

#      \*

#     \*\*\*

#    \*\*\*\*\*

#   \*\*\*\*\*\*\*

#  \*\*\*\*\*\*\*\*\*

n=int(input("Enter the number of rows: "))

for i in range(1,n+1):

    print(" "\*(n-i),"\*"\*(2\*i-1))

#       \*

#      \* \*

#     \* \* \*

#    \* \* \* \*

#   \* \* \* \* \*

n=int(input("Enter the number of rows: "))

for i in range(1,n+1):

    print(" "\*(n-i)," \*"\*i)

# 1

# 1 2

# 1 2 3

# 1 2 3 4

# 1 2 3 4 5

n=int(input("Enter the number of rows:"))

for i in range(1,n+1):

    for j in range(1,i+1):

        print(j,end=" ")

    print()

#  \*\*\*\*\*

#   \*\*\*\*

#    \*\*\*

#     \*\*

#      \*

n=int(input("Enter the number of rows: "))

for i in range(n,0,-1):

    print(" "\*(n-i),"\*"\*i)

# \*\*\*\*\*

# \*\*\*\*

# \*\*\*

# \*\*

# \*

n=int(input("Enter the number of rows: "))

for i in range(n,0,-1):

    print("\*"\*i)

#   \* \* \* \* \*

#    \* \* \* \*

#     \* \* \*

#      \* \*

#       \*

n=int(input("Enter the number of rows: "))

for i in range(n,0,-1):

    print(" "\*(n-i)," \*"\*i)

#       \*

#      \* \*

#     \* \* \*

#    \* \* \* \*

#   \* \* \* \* \*

#   \* \* \* \* \*

#    \* \* \* \*

#     \* \* \*

#      \* \*

#       \*

n=int(input("Enter the number of rows: "))

for i in range(1,n+1):

    print(" "\*(n-i),"\* "\*i)

m=n-1

for i in range(m,0,-1):

    print(" "\*(m-i)," \*"\*i)

#      \*

#     \*\*

#    \*\*\*

#   \*\*\*\*

#  \*\*\*\*\*

# \*\*\*\*\*

# \*\*\*\*

# \*\*\*

# \*\*

# \*

n=int(input("Enter the number of rows: "))

for i in range(0,n+1):

    print(" "\*(n-i),"\*"\*i)

for i in range(n,0,-1):

    print("\*"\*i," "\*(n-i))

# \*

# \* \*

# \* \* \*

# \* \* \* \*

# \* \* \* \* \*

# \* \* \* \*

# \* \* \*

# \* \*

# \*

n=int(input("Enter the number of rows: "))

for i in range(0,n+1):

    print("\* "\*i)

m=n-1

for i in range(m,0,-1):

    print("\* "\*i)

#      \*

#     \*\*

#    \*\*\*

#   \*\*\*\*

#  \*\*\*\*\*

#  \*\*\*\*\*

#   \*\*\*\*

#    \*\*\*

#     \*\*

#      \*

n=int(input("Enter the number of rows: "))

for i in range(0,n+1):

    print(" "\*(n-i),"\*"\*i)

for i in range(n,0,-1):

    print(" "\*(n-i),"\*"\*i)

**­­**

**----:Functions:----**

If a group of statements is repeatedly required then it is not recommended to write these statements every time separately. We have to define these statements as a single unit and we can call that unit any number of times based on our requirement without rewriting.This unit is nothing but function.

x=10

y=20

print("Addition of x & y =",x+y)

print("Addition of x & y =",x-y)

print("Addition of x & y =",x\*y)

x=200

y=100

print("Addition of x & y =",x+y)

print("Addition of x & y =",x-y)

print("Addition of x & y =",x\*y)

x=10

y=5

print("Addition of x & y =",x+y)

print("Addition of x & y =",x-y)

print("Addition of x & y =",x\*y)

O/P:--

Addition of x & y = 30

Addition of x & y = -10

Addition of x & y = 200

Addition of x & y = 300

Addition of x & y = 100

Addition of x & y = 20000

Addition of x & y = 15

Addition of x & y = 5

Addition of x & y = 50

**Now,** repeated code can be bound into single unit that is called function.

The advantages of function:

1. **Maintaining the code is an easy way.**
2. **Code re-usability.**

Now,

def calculate(x, y):

    print("Addition of x & y =",x+y)

    print("Addition of x & y =",x-y)

    print("Addition of x & y =",x\*y)

calculate(10,20)

calculate(200,100)

calculate(10,5)

O/P:--

Addition of x & y = 30

Addition of x & y = -10

Addition of x & y = 200

Addition of x & y = 300

Addition of x & y = 100

Addition of x & y = 20000

Addition of x & y = 15

Addition of x & y = 5

Addition of x & y = 50

**Types of function:---**

1. **In-built function :--** The functions which are coming along with Python software automatically,are called built-in functions or pre defined functions.

Examples:--

1. print()
2. id()
3. type()
4. len()
5. eval()
6. sorted()
7. count() etc…….

**2. User define function:---** The functions which are defined by the developer as per the requirement are called user-defined functions.

**Syntax:---**

**def** fun\_name(**parameters….):**

‘‘‘ doc string….’’’

Statment1…….

Statment2…….

Statment3…….

**return** (anything)

# call function

fun\_name(**arguments....**)

|  |  |
| --- | --- |
| **Important terminology** | |
| def-keyword | mandatory |
| return-keyword | optional |
| arguments | optional |
| parameters | optional |
| fun**\_**name | mandatory |

1. **def keyword –** Every function in python should start with the keyword ‘def’. In other words, python can understand the code as part of a function if it contains the ‘def’ keyword only.
2. **Name of the function –** Every function should be given a name, which can later be used to call it.
3. **Parenthesis**– After the name ‘()’ parentheses are required
4. **Parameters**– The parameters, if any, should be included within the parenthesis.
5. **Colon symbol ‘:’**should be mandatorily placed immediately after closing the parentheses.
6. **Body –**All the code that does some operation should go into the body of the function. The body of the function should have an indentation of one level with respect to the line containing the ‘def’ keyword.
7. **Return statement** – Return statement should be in the body of the function. It’s not mandatory to have a return statement.If we are not writing return statement then default return value is **None**
8. **Arguments**:-- At the time of calling any function, in between the parentheses we passes arguments.

**Relation between parameters and arguments:--**

When we are creating a function, if we are using parameters in between parenthesis, then it is compulsory to at the time of calling this function, you need to pass correspond arguments.

Parameters are inputs to the function. If a function contains parameters, then at the time of calling, compulsory we should provide values as a arguments, otherwise we will get error.

def calculate(x, y):

    print("Addition of x & y =",x+y)

    print("Addition of x & y =",x-y)

    print("Addition of x & y =",x\*y)

calculate(10,20)

calculate(200,100)

calculate(10,5)

O/P:--

Addition of x & y = 30

Addition of x & y = -10

Addition of x & y = 200

Addition of x & y = 300

Addition of x & y = 100

Addition of x & y = 20000

Addition of x & y = 15

Addition of x & y = 5

Addition of x & y = 50

# Write a function to take number as input and print its square value

def square(x):

    print("The Square of",x,"is", x\*x)

square(4)

square(5)

O/P:--

The Square of 4 is 16

The Square of 5 is 25

# Write a function to check whether the given number is even or odd?

def even\_odd(num):

    if num%2==0:

        print(num,"is Even Number")

    else:

        print(num,"is Odd Number")

even\_odd(10)

even\_odd(15)

O/P:--

10 is Even Number

15 is Odd Number

# Write a function to find factorial of given number?

def fact(num):

    result=1

    while num>=1:

        result=result\*num

        num=num-1

    return result

i=int(input("Enter any no "))

print("The Factorial of",i,"is :",fact(i))

O/P:--

Enter any no 5

The Factorial of 5 is : 120

**Returning multiple values from a function:** In other languages like C, C++ and Java, function can return almost one value. But in Python, a function can return any number of values.

def add\_sub(a,b):

    add=a+b

    sub=a-b

    return add,sub

x,y=add\_sub(100,50)

print("The Addition is :",x)

print("The Subtraction is :",y)

O/P:--

The Addition is : 150

The Subtraction is : 50

Or

def add\_sub(a,b):

    add=a+b

    sub=a-b

    return add,sub

x,y=int(input("Enter first value:")),int(input("Enter second value: "))

print("The Addition is :",x)

print("The Subtraction is :",y)

O/P:--

The Addition is : 100

The Subtraction is : 50

def calc(a,b):

    add=a+b

    sub=a-b

    mul=a\*b

    div=a/b

    return add,sub,mul,div

x,y,z,p=calc(int(input("Enter first value:")),int(input("Enter second value: ")))

print("The Addition is",x)

print("The Substraction is",y)

print("The Multip is",z)

print("The Division is",p)

O/P:--

Enter first value:100

Enter second value: 10

The Addition is 110

The Substraction is 90

The Multip is 1000

The Division is 10.0

**Types of arguments:**

def f1(a,b):

------

------

f1(10,20)

There are 4 types are actual arguments are allowed in Python.

1. **positional arguments:**

def f1(a,b):

------

------

f1(10,20)

def square(x):

    print("The Square of",x,"is", x\*x)

square(4)

square(5)

O/P:-

The Square of 4 is 16

The Square of 5 is 25

1. **keyword arguments:**

def f1(a,b):

------

------

f1(a=10,b=20)

def square(x):

    print("The Square of",x,"is", x\*x)

square(x=4)

square(x=5)

O/P:--

The Square of 4 is 16

The Square of 5 is 25

1. **default arguments:**

def f1(a=0,b=0):

------

------

f1(10,20)

f1()

def square(x=0):

    print("The Square of",x,"is", x\*x)

square(x=4)

square()

O/P:--

The Square of 4 is 16

The Square of 0 is 0

1. **Variable length arguments:**

def f1(\*n):

------

------

f1(10)

f1(10,20)

f1(10,20,30)

def sum(\*n):

    total=0

    for i in n:

        total=total+i

    print("The Sum=",total)

sum()

sum(10)

sum(10,20)

sum(10,20,30,40)

O/P:--

The Sum= 0

The Sum= 10

The Sum= 30

The Sum= 100

1. **key word variable length arguments:**

def f1(\*\*n):

------

------

f1(n1=10, n2=20)

def display(\*\*kwargs):

    for k,v in kwargs.items():

        print(k,"=",v)

display(n1=10,n2=20,n3=30)

print("-----------")

display(rno=100, name="Neeraj", marks=70, subject="Java")

O/P:--

n1 = 10

n2 = 20

n3 = 30

-----------

rno = 100

name = Neeraj

marks = 70

subject = Java

##### **Types of Variables in Python**

The variables based on their scope can be classified into two types:

1. **Local variables**
2. **Global variables**

##### **Local Variables in Python:**

The variables which are declared inside of the function or inside any block are called local variables. Their scope is limited i.e we can access local variables within the function/block only. If we are trying to access local variables outside of the function, then we will get an error.

def a():

    x=10

    return "value of Local variable is:",x

def b():

    return "value of Local variable is:",x

p=a()

print(p)

y=b()

print(y)

O/P:-

('value of Local variable is:', 10)

Traceback (most recent call last):

File "E:\Python Core\_Advance\local.py", line 10, in <module>

y=b()

File "E:\Python Core\_Advance\local.py", line 6, in b

return "value of Local variable is:",x

NameError: name 'x' is not defined

We Can’t access local variable outside the function:

def a():

    x=10

    return "value of Local variable is:",x

def b():

    return "value of Local variable is:",x

p=a()

print(p)

print(x)

O/P:--

('value of Local variable is:', 10)

Traceback (most recent call last):

File "E:\Python Core\_Advance\local.py", line 12, in <module>

print(x)

NameError: name 'x' is not defined

##### **Global variables in Python:**

The variables which are declared outside of the function or outside of the block are called global variables. Global variables can be access throughout the program.

a=11

b=12

def m():

   print("a from function m(): ",a)

   print("b from function m(): ",b)

def n():

   print("a from function n(): ",a)

   print("b from function n(): ",b)

m()

n()

O/P:--

a from function m(): 11

b from function m(): 12

a from function n(): 11

b from function n(): 12

##### **Global keyword and globals() inbuilt methods:**

The keyword global can be used for the following 2 purposes:

1. To declare a global variable inside a function

def m1():

   global a

   a=2

   print("a value from m1() function: ", a)

def m2():

   print("a value from m2() function:", a)

m1()

m2()

O/P:--

a value from m1() function: 2

a value from m2() function: 2

1. To make global variables available to the function.

a=10

def a1():

   print("a value from m1() function: ", a)

def a2():

   print("a value from m2() function:", a)

a1()

a2()

O/P:---

a value from m1() function:  10

a value from m2() function: 10

**Note:-If global and Local variables having the same name in function, and within the function block we use a global keyword for same variable, then**

a=1

def a1():

   global a

   a=2

   print("a value from m1() function:", a)

def a2():

   print("a value from m2() function:", a)

a1()

a2()

print("Value of a = ",a)

O/P:-

a value from m1() function: 2

a value from m2() function: 2

Value of a =  2

##### **globals() built-in function in python:**

##### **If we are having local and global variable with having same name,but we want to use global variable instead of local variable then we are using** globals()method.

a=1

def m1():

   a=2

   print("a value from m1() function:", a)

   print("a value from m1() function:", globals()['a'])

m1()

O/P:--

a value from m1() function: 2

a value from m1() function: 1

Write a code for calculator:-----

def add(x,y):

   return x+y

def subtract(x,y):

   return x-y

def multiply(x,y):

   return x\*y

def divide(x,y):

   return x/y

while True:

    print ("Please select the operation.")

    print ("1. Add\n2. Subtract\n3. Multiply\n4. Divide\n5. Off\n")

    n = int(input("Please enter choice (1,2,3,4,5): "))

    p = int (input ("Please enter the first number: "))

    q = int (input ("Please enter the second number: "))

    print(type(n),type(p),type(q))

    if n == 1:

        print (p, " + ", p, " = ", add(p, q))

    elif n == 2:

        print (p, " - ", q, " = ", subtract(p, q))

    elif n == 3:

        print (p, " \* ", q, " = ", multiply(p, q))

    elif n == 4:

        print (p, " / ", q, " = ", divide(p, q))

    elif n==5 :

        break

    else:

        print ("This is an invalid input")

O/P:--

Please select the operation.

1. Add

2. Subtract

3. Multiply

4. Divide

5. Off

Please enter choice (1,2,3,4,5): 1

Please enter the first number: 5

Please enter the second number: 6

<class 'int'> <class 'int'> <class 'int'>

5 + 5 = 11

Please select the operation.

1. Add

2. Subtract

3. Multiply

4. Divide

5. Off

Please enter choice (1,2,3,4,5):