Course Agenda

- Introduction to Python Programming
- Programming Basics - variables, operators, decision making, iteration, sequences
- Data Structures in Python - list, tuple, set, dictionary
- Functions - Functiona Arguments, lambda functions, function objects, map - filter - reduce
- Arrays and NumPy functions
- Data Frames and data cleaning using pandas
- Data Visualisation using matplotlib and seaborn

Introduction to Python Programming

- Data Types in Python
- Variables in Python
- Data Type conversion
- Accept inputs from user
- Operators in Python

Python is –

- Open source
- Interpreter based
- Platform independent
- Current version – 3.12.x
- Download it from here - https://www.python.org/download/releases/3.0/

Python Data Types

 No description has been provided for this image

In [ ]:
```
10 # int
```

In [ ]:
```
15.5 # float
```

In [ ]:
```
"abcd" # str
```

In [ ]:
```
True # bool
```

In [ ]:
```
complex(2,3) # complex
```

Note - Everything in Python is an Object

Python Containers

Containers are any object that holds an arbitrary number of other objects. Generally, containers provide a way to access the contained objects and to iterate over them. Examples of built-in containers include tuples, lists, sets, dictionary.

 No description has been provided for this image

In [ ]:
```
[1, 2, 3, 4, 'abc'] # list
```

In [ ]:
```
(1, 2, 3, 4) # tuple
```

In [ ]:
```
{2, 's', 3, 5, 5} # set
```

In [ ]:
```
{1 : "Jane", 2:"George", 3:"Sam"} # dictionary
```

Variables in Python

- A Python variable points to a reserved memory location
- Data or objects are stored in these memory locations
- Variables can be declared by any name or even alphabets

Ex. Define variable name and assign value to the variable

In [ ]:
```
name = "Jane"
print("Welcome", name)
```

In [ ]:
```
print("Welcome", name, sep = "_")
```

formatted string

In [ ]:
```
a = 10
b = 20
c = a + b
print(f"Addition of {a} and {b} = {c}")
print(f"Addition of",a, "and" ,b, "=", c)
```

raw string

In [ ]:
```
path = r"C:\Users\Admin\Newfolder\newfile"
print(path)
```

Python feature - dynamically typed

In [ ]:
```
name
```

In [ ]:
```
name = 10
```

type() - returns class type of the argument(object) passed as parameter

In [ ]:
```
a = True
print(type(a))
```

In [ ]:
```
a = 10
print(type(a))
```

In [ ]:
```
a = 10.5
print(type(a))
```

In [ ]:
```
a = "abcd"
print(type(a))
```

In [ ]:
```
lst = [1, 2, 3, 4]
print(type(lst))
```

Ex. WAP to take name of user as input and print a welcome message

In [ ]:
```
name = input("Enter your name - ")
print("Welcome", name)
```

Ex. WAP to take two numbers as input from user and print their sum.

In [ ]:
```
num1 = input("Enter a number - ")
num2 = input("Enter a number - ")

print(num1 + num2)
```

In [ ]:
```
type(num1)
```

Note - input() always stores the value in str format

Data Type Conversion

`Implicit Conversion:` Conversion done by Python interpreter without programmer's intervention

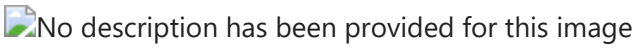
In [ ]:
```
a = 10  # int
b = 2.5 # float
a + b
```

`Explicit Conversion:` Conversion that is user-defined that forces an expression to be of specific data type

In [ ]:
```
a = "10"  # str
b = 5     # int

int(a) + b
```

In [ ]:
```
num1 = int(input("Enter a number - "))
num2 = int(input("Enter a number - "))
print(num1 + num2)
```


No description has been provided for this image

`int()` conversion

In [ ]:
```
x = 10.8 # float
int(x)
```

In [ ]:
```
x = "10" # int value in str format
int(x)
```

In [ ]:
```
x = True # bool
int(x)
```

In [ ]:
```
x = False # bool
int(x)
```

In [ ]:
```
x = "abcd" # str
int(x)
```

In [ ]:
```
x = "10.8" # float value in str format
int(x)
```

`float()` conversion

In [ ]:
```
x = True # bool
float(x)
```

In [ ]:
```
x = False # bool
float(x)
```

In [ ]:
```
x = 10 # int
float(x)
```

In [ ]:
```
x = "10.8" # float value in str format
float(x)
```

In [ ]:
```
x = "abcd" # str
float(x)
```

`str()` conversion

In [ ]:
```
x = 10
str(x)
```

In [ ]:
```
lst = [1,2,3,4,5]
str(lst)
```

`bool()` conversion

In [ ]:
```
x = 0
bool(x)
```

In [ ]:
```
x = 1
bool(x)
```

In [ ]:
```
x = 0.0
bool(x)
```

In [ ]:
```
x = 1.0
bool(x)
```

In [ ]:
```
x = 10
bool(x)
```

In [ ]:
```
x = "abc"
bool(x)
```

In [ ]:
```
x = "True"
bool(x)
```

In [ ]:
```
x = "False"
bool(x)
```

In [ ]:
```
x = None
bool(x)
```

Note - bool() returns True for any value except 0 or 0.0 or None

---

Operators in Python

Operators are special symbols in Python that carry out computations. The value that the operator operates on is called as operand.

 No description has been provided for this image

Arithmetic Operators

- are used to perform mathematical operations like addition, subtraction, multiplication and division.

In [ ]:
```
a = 10
```

```
b = 6
```

In [ ]:
```
print("Addition - ", a+b)
```

In [ ]:
```
print("Substraction - ", a-b)
```

In [ ]:
```
print("Multiplication - ", a*b)
```

In [ ]:
```
print("Division - ", a/b) # returns result in float format
```

In [ ]:
```
print("Floor Division - ", a//b) # returns integer part of the division
```

In [ ]:
```
print("Exponential - ", a**b) # returns the result of a to the power b
```

In [ ]:
```
print("Modulous - ", a%b) # returns remainder of the division
```

Ex. WAP to accept hours and rate per hour from user and compute gross pay.

In [ ]:
```
hrs = int(input("Enter no of hrs worked - "))
rate = int(input("Enter rate per hour - "))

gross_pay = hrs * rate
print(gross_pay)
```

Ex. WAP to calculate BMI of a person.

In [ ]:
```
weight = float(input("Enter your weight in kgs - "))
height = float(input("Enter your height in mtrs - "))
bmi = round(weight / (height ** 2), 2)
print(bmi)
```

Relational Operators

- It either returns True or False according to the condition.

In [ ]:
```
a = 10
b = 7
```

In [ ]:
```
print(a < b)
```

In [ ]:
```
print(a <= b)
```

In [ ]:
```
print(a > b)
```

In [ ]:
```
print(a >= b)
```

In [ ]:
```
print(a==b)
```

In [ ]:
```
print(a!=b)
```

Ex. WAP to take a number as input and write condition to check if the number is greater than 10.. (Output must be a bool value)

In [ ]:
```
num = int(input("Enter a number - "))
num > 10
```

Ex. WAP to take a number as input and write condition to check if the number is divisible by 5. (Output must be a bool value)

In [ ]:
```
num = int(input("Enter a number - "))
num % 5 == 0
```

Logical Operators

- perform Logical AND, Logical OR, and Logical NOT operations. It is used to combine conditional statements.

`and` - returns True if both the conditions are True else returns False

Ex. WAP to check if the number is divisible by 5 and greater than 10

In [ ]:
```
num = int(input("Enter a number - "))
num % 5 == 0 and num > 10
```

`or` - returns True if either of the conditions is True else. Returns false if both conditions are False

Ex. WAP to check if the number is either divisible by 5 or greater than 10 or both

In [ ]:
```
num = int(input("Enter a number - "))
num % 5 == 0 or num > 10
```

`not` - reverse the bool value

In [ ]:
```
not True
```

In [ ]:
```
not False
```

In [ ]:
```
not (10 - 5 * 2)
```

Assignment Operators

`=` - assigns the result of expression on RHS to variable on LHS

In [ ]:
```
var = 10 + 2 * 5
var
```

Assignment can also be clubbed with arithmetic operators.

+=, -=, *=, /=, //=, **=, %=

In [ ]:
```
a = 10
a = a + (5 * 2 - 3)

a += (5 * 2 - 3)
```

In [ ]:
```
a += 1
```

Objects

- elements in the Python environments
- generic objects
- Sequence objects - collection of elements - str, range()
    - Container sequences/Objects - list, tuple, dict, set

Properties of Sequences

- `Operations on Sequences`
    - Membership - `in` | `not in`
    - Iteration - `for-loop`
- `Functions on Sequences`
    - len() - gives the number of elements in the sequence
    - max() - gives the largest element in the sequence
    - min() - gives the smallest element in the sequence
    - sum() - applicable to numeric sequences, returns the sum of all elements in the sequence
    - math.prod() - applicable to numeric sequences, returns the product of all elements in the sequence
    - sorted() - sorts the elements in the sequence in ASC order and returns a list object

Membership Operators

Membership operators checks whether a value is a member of a sequence.

Sequence Object -

A `sequence` is defined as a collection of arbitrary number of elements. The sequence may be a list, a string, a tuple, or a dictionary

- in - The `in` operator is used to check if a value exists in any sequence object or not.

- not in - A `not in` works in an opposite way to an 'in' operator. A 'not in' evaluates to True if a value is not found in the specified sequence object. Else it returns a False.

In [ ]:
```
"a" in "india"
```

Ex. WAP to check if entered name is present in the mentioned list or not

In [ ]:
```
names = ["Jane", "George", "Sam"]
"Jane" in names
```

In [ ]:
```
names = ["Jane", "George", "Sam"]
"Jack" in names
```

In [ ]:
```
names = ["Jane", "George", "Sam"]
"J" in names
```

Ex. WAP to check if entered character is a vowel or not

In [ ]:
```
ch = input("Enter a single character - ")
ch in "aeiou"
```

Ex. WAP to calculate the hypoteneous of a right angled triangle when sides are given

In [ ]:
```
import math
base = 4
height = 3

hypt = math.sqrt((base ** 2) + (height ** 2))
hypt
```

List of all built-in functions

In [ ]:
```
print(dir(__builtins__))
```

List of all built-in modules

In [ ]:
```
import sys
print(sys.builtin_module_names)
```

List all function in a module

In [ ]:
```
print(dir(math))
```

Display documentation of a function

In [ ]:
```
help(math.sqrt)
```

In [ ]:
```
import math
math.sqrt()
```

In [ ]:
```
import math as m
m.sqrt()
```

In [ ]:
```
from math import sqrt
sqrt()
```

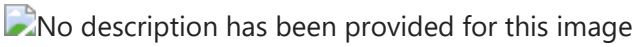
In [105…
```
from math import *
```

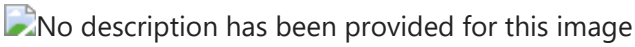
In [103…
```
sqrt = 10
```

In [106…
```
sqrt(16)
```

Out[106…
```
4.0
```

Decision Making





Ex1 - WAP to take a single character as input and check if it is a vowel or a consonant?

In [4]:
```
ch = input("Enter a character - ")
if ch.lower() in "aeiou" :
    print("Vowel")
else:
    print("Consonant")
```

```
Vowel
```

In [12]:
```
ch = input("Enter a character - ")
if len(ch) == 1 and ch.isalpha():
    if ch.lower() in "aeiou" :
        print("Vowel")
    else:
        print("Consonant")
else:
    print("Invalid")
```

```
Invalid
```

Ex. Print customised message for invalid inputs

In [16]:
```
ch = input("Enter a character - ")
if len(ch) != 1 :
    print("Length exceeds the limit")
elif not ch.isalpha():
    print("Input is invalid. Must be alphabet")
else:
    if ch.lower() in "aeiou" :
        print("Vowel")
    else:
        print("Consonant")
```

```
Consonant
```

Examples -

Ex. WAP to accept a single character from user and check if it is a vowel or not.

In [ ]:
```

```

Ex. WAP to take weight in kgs and height in mtrs from user. Calculate BMI and print if the the health status based on following chart

BMI Categories

- Underweight - BMI < 18.5
- Normal weight - BMI between 18.5-24.9
- Overweight - BMI between 25-29.9
- Obesity - BMI > 30

In [ ]:

Ex. WAP to accept hours and rate per hour from user and compute gross pay.

- for 40 hrs pay the standard rate
- if overtime then pay 1.5 times of rate for the additional hrs.

In [18]:
```
hrs = int(input("Enter no of hrs worked - "))
rate = int(input("Enter rate per hour - "))

if hrs <= 40 :
    gross_pay = hrs * rate
else:
    gross_pay = (40 * rate) + ((hrs - 40) * 1.5 * rate)
print(gross_pay)
```

```
4750.0
```

Ex. Toss a coin and guess the outcome

WAP to simulate coin toss and compare the outcome with guess made by user. The program must return "Invalid input" if user enters a wrong or invalid value as a guess.

1. Take a guess as input from user.
2. Validate the input as "Heads" or "Tails", case-sensitive
3. If valid
    - generate random outcomes as heads ot tails
    - Compare the guess with outcome and print Win or lost

In [31]:
```
import random as r
guess = input("Enter your guess as heads or tails - ").lower()
choice_lst = ["heads", "tails"]
if guess in choice_lst :
    outcome = r.choice(choice_lst)
    print("Outcome - ", outcome)
    if guess == outcome :
        print("Win")
    else:
        print("Lost")
else:
    print("Invalid")
```

```
Outcome -  tails
Win
```

random module in Python

- Generates a random value
- Importing random module
  import random as r
- Frequently used functions
  - **r.random()** – Generates a random float number between 0.0 to 1.0
  - **r.randint()** – Returns a random integer between the specified integers
  - **r.randrange()** – Returns a randomly selected element from the range created by the start, stop and step arguments
  - **r.choice()** – Returns a randomly selected element from a non-empty sequence
  - **r.shuffle()** – This functions randomly reorders the elements in a list

```
In [22]: import random as r # as is an operator - used to assign a variable to a moudule whi
```

```
In [23]: r.random() # generates a random value between 0 and 1
```

```
Out[23]: 0.541609545617024
```

```
In [46]: r.randint(-4,5) # generates a random integer between the mentioned start and end va
```

```
Out[46]: 3
```

```
In [25]: r.choice([1,2,3,4]) # generates a random value from the mentioned iterable or seque
```

```
Out[25]: 4
```

```
In [26]: r.choice(["abc", "pqr", "xyz"])
```

```
Out[26]: 'xyz'
```

```
In [27]: r.choice("abcde")
```

```
Out[27]: 'c'
```

```
In [28]: help(r.choice)
```

```
Help on method choice in module random:

choice(seq) method of random.Random instance
    Choose a random element from a non-empty sequence.
```

Ex. 7up and 7down

WAP to simulate the below mentioned scenario -

1. Player enters the game with initial amount as Rs. 1,000/-

2. Generate a random value between 1 to 14 and store it in variable "outcome"
3. if outcome = 7, player hits a jackpot and wins Rs. 1,00,00,000.
4. if outcome < 7, player looses amount by (outcome*100)
5. if outcome > 7, player earns amount by (outcome*100)
6. Print the final amount with the player.

In [47]:
```
import random as r
amount = 1000
outcome = r.randint(1, 14)
print("Your Score - ", outcome)

if outcome < 7 :
    amount -= (outcome * 100)
elif outcome > 7 :
    amount += (outcome * 100)
else:
    print("You have won the jackpot!!!!")
    amount += 1000000
print("Final Balance - ", amount)
```

```
Your Score -  3
Final Balance -  700
```

Loops

Loops are used to execute of a specific block of code in repetitively

while Loop -

```
- Event based or condition based loop
- requires a condition to terminate the loop
- number of iterations is not fixed
```

for loop -

```
- counter based and operates only on sequences
- will execute till the last element of the sequence
- number of iterations = number of elements in the sequence
```

while loop

- The 'while loop' in Python is used to iterate over a block of code as long as the test expression holds true
- Event based loop
- Event occurring inside the loop determines the number of iterations
- This loop is used when the number of times to iterate is not known to us beforehand

Ex. Modify the 7up 7down program based on following rules -

- Ask user his choice to play again as `yes/no`.
- First round starts with amount balance as Rs. 1000. However, further rounds will be played on the balance amount generated from previous round. Example - in round 1 user earned Rs. 800. So for his next round amount will be Rs. 1,800 which is balance generated in previous round.
- The game will terminate if user -
  - choice to play again is `no`
  - hits the `jackpot`
  - has insufficient funds to play the next round.

In [51]:
```
import random as r
amount = 1000
choice = "yes"

while choice == "yes" :
    outcome = r.randint(1, 14)
    print("Your Score - ", outcome)

    if outcome < 7 :
        amount -= (outcome * 100)
    elif outcome > 7 :
        amount += (outcome * 100)
    else:
        print("You have won the jackpot!!!!")
        amount += 1000000
        break

    # check for insufficient funds
    if amount <= 600 :
        print("Insufficient funds, Balance - ", amount)
        choice = input("Top-up with 1000 to continue or quit. yes/no? ")
        if choice == "yes" :
            amount += 1000
            continue
        break

    print(f"Your current balance is {amount}")
    choice = input("Do you wish to play again? yes/no? ").lower()

print("Final Balance - ", amount)
```

```
Your Score -  4
Insufficient funds, Balance -  600
Your Score -  13
Your current balance is 2900
Final Balance -  2900
```

break statement

- The 'break' statement ends the loop and resumes execution at the next statement
- The break statement can be used in both 'while' loop and 'for' loop
- It is always used with conditional statements

continue statement

- The 'continue' statement in Python ignores all the remaining statements in the iteration of the current loop and moves the control back to the beginning of the loop
- The continue statement can be used in both 'while' loop and 'for' loop
- It is always used with conditional statements

Ex. Write a code to validate user input for an integer input

In [ ]:

```
```

for loop

- The 'for loop' in Python is used to iterate over the items of a sequence object like list, tuple, string and other iterable objects
- The iteration continues until we reach the last item in the sequence object
- Counter driven loop
- This loop is used when the number of times to iterate is predefined

Ex. WAP to print square of numbers in the given list

In [53]:

```
numbers = [1, 2, 3, 4, 5]
for i in numbers :
    print(i, " - ", i **2)
```

```
1  -  1
2  -  4
3  -  9
4  -  16
5  -  25
```

Ex. WAP to print square of all even numbers in the given list

In [55]:

```
numbers = [1, 2, 3, 4, 5]
for i in numbers :
    if i % 2 == 0 :
        print(i, " - ", i **2)
```

```
2  -  4
4  -  16
```

Ex. WAP to accept a word from user and print vowels in the word, no duplicates

In [59]:
```
word = input("Enter a word - ")
for ch in "aeiou" :
    if ch in word :
        print(ch)
```

```
i
```

In [62]:
```
word = input("Enter a word - ")
for ch in set(word) :
    if ch in "aeiou" :
        print(ch)
```

```
i
```

Ex. Write a python code to print the product of all elements in the numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11, 43]?

In [63]:
```
numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11, 43]
product = 1
for i in numbers :
    product *= i
product
```

Out[63]: 54489600

In [68]:
```
import math as m
m.prod(numbers)
```

Out[68]: 54489600

In [66]:
```
sum(numbers)
```

Out[66]: 91

Ex. WAP to perform product of first 10 natural numbers

In [69]:
```
m.prod(range(1, 11))
```

Out[69]: 3628800

In [70]:
```
sum(range(1, 11))
```

Out[70]: 55

range(`[start], stop, [step]`)

- `start` - Optional. An integer number specifying at which position to start. Default is 0
- `stop` - Required. An integer number specifying at which position to end.
- `step` - Optional. An integer number specifying the incrementation. Default is 1

In [71]:
```
range(1, 11)  # 1 - 10
```

Out[71]:
```
range(1, 11)
```

In [72]:
```
list(range(1, 11))
```

Out[72]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

In [73]:
```
for i in range(1,11):
    print(i, end = ",")
```

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

In [74]:
```
for i in range(11):
    print(i, end = ",")
```

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

In [75]:
```
for i in range(1, 50, 3):
    print(i, end = ", ")
```

```
1, 4, 7, 10, 13, 16, 19, 22, 25, 28, 31, 34, 37, 40, 43, 46, 49,
```

In [76]:
```
for i in range(1, 50, 5):
    print(i, end = ", ")
```

```
1, 6, 11, 16, 21, 26, 31, 36, 41, 46,
```

In [77]:
```
for i in range(50, 0, -1):
    print(i, end = ", ")
```

```
50, 49, 48, 47, 46, 45, 44, 43, 42, 41, 40, 39, 38, 37, 36, 35, 34, 33, 32, 31, 30,
29, 28, 27, 26, 25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9,
8, 7, 6, 5, 4, 3, 2, 1,
```

Examples

Ex. WAP to accept an integer from user and print a table for given number.

3 x 1 = 3

.

.

3 x 10 = 30

In [78]:
```
num = int(input("Enter a number - "))
for i in range(1, 11) :
    print(f"{num} x {i} = {num*i}")
```

```
3 x 1 = 3
3 x 2 = 6
3 x 3 = 9
3 x 4 = 12
3 x 5 = 15
3 x 6 = 18
3 x 7 = 21
3 x 8 = 24
3 x 9 = 27
3 x 10 = 30
```

Ex. WAP to print prime number between 1 - 100

In [80]:
```
num = int(input("Enter a number - "))
flag = True
for i in range(2, num):
    if num % i == 0 :
        flag = False
        break
if flag :
    print("Prime Number")
else:
    print("Not a prime number")
```

```
Not a prime number
```

for .. else

- if a for loop is `terminated` abruptly using a `break` statement then else block is `NOT` executed
- if for loop is executed successfully till its last iteration, then else block is `executed`

In [82]:
```
for i in range(1, 10) :
    print(i)
    if i == 7 :
        print("Terminate")
        break
else:
    print("Else executed")
```

```
1
2
3
4
5
6
7
Terminate
```

In [86]:
```
for i in range(1, 6) :
    print(i)
    if i == 7 :
        print("Terminate")
        break
```

```
else:
    print("Else executed")
```

```
1
2
3
4
5
Else executed
```

In [94]:
```
# resultset = conn.execute("Select * from inventory")
resultset = [
    ("product1", "Coffee", 5, "30 days"),
    ("product2", "Tea", 30, "35 days")
]
```

Ex. Check for products less than 10 units

In [95]:
```
for tup in resultset :
    if tup[2] < 10 :
        break
else:
    print("No products less than 10 units")
```

Ex. Print prime numbers between 1-100 using for-else

In [97]:
```
num = int(input("Enter a number - "))
for i in range(2, num):
    if num % i == 0 :
        print("Not Prime")
        break
else :
    print("Prime")
```

```
Not Prime
```

In [98]:
```
for num in range(1, 101):
    for i in range(2, num):
        if num % i == 0 :
            break
    else :
        print(num, end = " ")
```

```
1 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
```

In [ ]:

In [ ]:

Strings in Python

Strings are -

- an ordered sequence of characters
- enclosed in a pair of single quotes or pair of double quotes
- immutable

Empty string

In [ ]:
```
string = ''

string = ""
```

Note - bool() of empty str is always

Defining a string

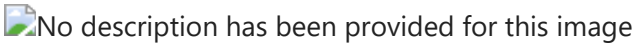
In [ ]:
```
string = "aero-plane"
```

Extract first element from the string

In [ ]:

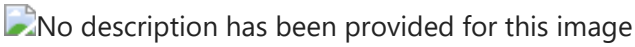
Extract 5th element from the string

In [ ]:



Ex. Extract last element from the string

In [ ]:



Ex. Extract first 3 characters from string

In [ ]:

Ex. Extract all characters from index position 3

In [ ]:

Ex. Extract last 4 characters from the string

In [ ]:

Reverse of string

In [ ]:

Ex. WAP to check if entered string is a palindrome or not.

`Palindrome` - the string reads same characters left to right or right to left

Ex - madam

In [ ]:

Ex. WAP to generate a new string by swapping first and last characers

ex - abcde - ebcda

In [ ]:

Operations on Strings

- Concatenation
- Repetition
- Membership
- Iteration

In [ ]:

`Concatenation` - merging two strings into a single object using the + operator.

In [ ]:

`Repetition` - The repetition operator * will make multiple copies of that particular object and combines them together.

In [ ]:

Strings are immutable and cannot be modified

In [ ]:

Built-in Functions

- **len()** - returns length of the string
- **min(), max()** - returns minimum and maximum element from the string
- **sorted()** - sorts the characters of the string and returns a list

In [ ]:
```
string = "Mississippi"

len(string)
```

In [ ]:
```
min(string)
```

In [ ]:
```
max(string)
```

In [ ]:
```
sorted(string) # returns a list object
```

Strings Methods

- **str.index(`obj`)** - returns index of the first occurence of the character
- **str.count(`obj`)** - returns count of number of occurences of the charcter
- **str.upper()** - returns string of uppercase characters
- **str.lower()** - returns string of lowercase characters
- **str.title()** - returns string of sentence case charaters
- **str.isupper()** - checks if all characters are uppercase
- **str.islower()** - checks if all characters are lowercase
- **str.isdigit()** - checks if all characters are digits
- **str.isalpha()** - checks if all characters are alphabets
- **str.isalnum()** - checks if all characters are either alphabets or digits
- **str.split(`delimiter`)** - splits the string on the mentioned delimiter and returns a list of obtained parts of the string
- **str.replace(`str` , `str`)** - replaces all the mentioned characters with the specified string and returns a new string
- **str.strip(`delimiter`)** - removes whitespace characters from start and end of the string (delimiter can also be specified)
- `delimiter` **.join(** `sequence`) - it is called on a string object which acts as a delimiter to join all string elements in the sequence passed as an argument to join()

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Examples

Ex. WAP to convert the given string -

string = "I am in Python class"

o/p - 'ssalc nohtyP ni ma I'

o/p - 'I Am In Python Class'

In [ ]:
```
string = "I am in Python class"
```

Ex. WAP to print foloowing pattern

In [ ]:
```
*
**
***
****
*****
```

Ex. WAP to replace all vowels in a word with and asterisk.

In [ ]:
```

```

Ex. WAP to accept numbers from user in comma seperated format. Extract the integers and perform their summation

In [ ]:
```

```

Ex. Write a Python program to get a string from a given string where all occurrences of its first char have been changed to '#', except the first char itself.

Sample String : 'restart'

Expected Result : 'resta#t'

In [ ]:
```

```