27/09/21

String and operations on string

str class

S_1 = str( ) → # creates an empty string object

S_2 = str("Hello")

Inbuilt function :-

a = "PYTHON"

→ len(a) → returns length of a string #6

→ min(a) → returns smallest char # 'H'

→ max(a) → return largest character # 'Y'

Indexing of a string :-

S →

I	N	D	I	A
S[0]	S[1]	S[2]	S[3]	S[4]

S_1 → = "PYTHON"
0 1 2 3 4 5

S[0] → P

S[5] → N

⇒ Negative indexes can also be used in a string as used in a list

S = "PYTHON"
-6 -5 -4 -3 -2 -1

S[-1] = 'N'

S[-2] = 'O'

$$S[-n] = S[len(S) - n]$$

S = IIT-BOMBAY
0 1 2 3 4 5 6 7 8 9

S[-3] = S[len(S) - 3]

= S(10-3] = S[7]

O/P ⇒ 'B'

String traversal :-

→ Program to traverse all elements of string using for loop.

```
S = "INDIA"
for ch in S :
    print (ch, end = " ")
```

O/P → INDIA

→ Program to traverse every second element in a list.

```
S = "ILOVEPYTHONPROGRAMMING"
for ch in range(0, len(s), 2):
    print(S[ch], end = " ")
```

O/P ⇒ I O E Y H I N R G A M N

Traversing using while loop :-

```
S = "INDIA"
index = 0
while  index < len(s)
print(s[index], end= " ")
  index = index+1
```

O/P → INDIA

29/09/21

String slicing → same as slicing in a list.

ex:- S = "IIT- BOMBAY"
(0 1 2 3 4 5 6 7 8 9)

S[4:10] ⇒ 'BOMBAY'

S[0. len(s) : 2] ⇒ 'IT'

S[::] ⇒ Print the entire string → 'IIT-BOMBAY'

S[::-1] ⇒ Print the string in reverse → 'YABMOB-TII'

S = 'IIT-MADRAS'
(0 1 2 3 4 5 6 7 8 9)
(-10 -9 -8 -7 -6 -5 -4 -3 -2 -1)

S[-1:0:-1] = 'SARDAM-IT'

S[:-1] ⇒ 'IIT- MADRA'

String operators :-

'+' Operators :- Used for concatenation of two strings

S1 = "IIT"
S2 = "DELHI"
S3 = S1+S2
print(S3) ⇒ "IITDELHI"

'*' Operator :-

S1 = "Hello"
S2 = 3*S1
Print(S2) ⇒ "Hello Hello Hello"

'in' and 'not in' operator :-

S1 = "INFORMATION TECHNOLOGY"

"TECHNOLOGY" in S1 ⇒ True
"Hello" not in S1 ⇒ True
"ENGINEERING" in S1 ⇒ False

Q Write a program to print all the letters from Word 1 that also appear in Word 2.

Program

```
Word_1 = "USA North America"
Word_2 = "USA south America"
print the letter that appear in Word 1 also appears in WORD 2.
for letter in Word_1:
    if letter in Word_2:
        print (letter, end = " ")
print ('outside')
```

O/P → USA orth America

I case :- not using [end = " "].

```
U
S
A
o
r
t
h
```

II case :- Use [end= " "]

USA orth America.

06/10/2001

Function in Python

Ex

```
def show():
    print("I am good")
```

Without Arg.

show() ⟶ "I am good"

Ex.2 Write a program to add the sum of digits from 1 to 25, 50 to 76 and 90 to 101 using different for loops.

```
sum = 0
for i in range(1, 25)
    sum = sum + i
    print("sum is", sum)
```

```
def sum(x, y)
    s = 0
    for i in range(x, y+1):
        s = s+i
    print("sum of integers is :", s)
    return s
```

k = sum(1,25) ⟶ 1+2+..... 25 → sum

l = sum(50,75) ⟶ 50+51+..... 75 → sum

m = sum(90,100)

o = k+l+m.

Q Write a program to find maximum of given two numbers.

→ Parameters in functions :-

```
def printMax(num1, num2)
    if (num1 > num2):
        print(num1, " is greater than ", num2)

    elif(num2 > num1):
        print(num2, " is greater than ", num1)

    else:
        print("Both are equal")

printMax(2,6)
```

Q write a program to find the factorial of a number

Factorial

→

```
def facto(num):
    fact = 1
    print("number entered", num)
    for i in range(1, num+1):
        Fact = fact * i
    print("factorial is", fact)

number = int(input("Enter num"))
facto(number)
```

funct define

Argument

function call

13/10/2021

Loops in python :-

While loop

```
count = 0
while (count < 3):
    count = count + 1
    print ("Hello").
```

O/P → Hello
Hello.

for loop

```
n = 4
for i in range (0, n).
    print (i).
```

Continue vs break

```
for val in "String"
    [if val == "i":
        break
    print (val)
    print ("the end").
```

→ continue O/P

```
S
t
r
n
g
The end
```

Output :-

```
S
t
V
The end
```

Continue :-

operation
<<
>>
||
/
**
|
&
|

Python data types :-

- List
- Tuple
- Sets
- dictionary

→ Dictionary are used to store values in key value.

```
The dict = { "brand" : "Ford"
             "Model" : "X1"
             "Year" : "1964" }

print (the dict['brand'])
```

⇒ print (len (the diet))

Python program :-

```
sum of n odd numbers:
name = int (input("sum of odd numbers till"))
sum = 0
for i in range (1, num+1)
    if (not (i%2) == 0):
        sum += i
    print ("sum", sum).
```

Tuple → are the immutable data types i.e. cannot change the value in Tuple.

declaration

```
ice cream = ("choco", "Vanilla", 'Mint')
print (ice-cream) - print the Tuple.
```

accessing :- print (ice-cream[2]) ⇓ 'mint'

print (ice-cream [-1]) ⇓ 'Mint'.

→ we can also slice the tuple.

Sets - are use to store multiple items in a single variable.

→ Unique elements.

they are unordered and unindered.

declaration

this set = {"apple", "banana", "cherry"}

Unordered :-

It means the set element appear every time in different order when used, can not be indexed.

Ordered d.T :→ list, Tuple

Unordered → sets, dictionary

→ other program :-

→ sum of n natural number

→ sum of squares of first 'n' natural numbers.

→ sum of cubes

→ prime number

→ Leap year

→ List comprehension

→ List functions

- reverse()
- extend()
- sort()
- append()

Condition for a leap year :-

→ divisible by 4 except for century years (ending with 00).

→ century year in a leap year if perfectly divisible by 400.

```
year = 2000
if (year % 4) == 0:
    if (year % 100) == 0:
        if (year % 400) == 0:
            print ("leap year")
        else:
            print ("not leap year")
    else:
        print (" leap year")
else: print ("not").
```

Positional & keyword arguments :-

```
def display (Name, age):
    print (" Name =" Name, "age = ", age)
```

Keyword args → When a programmar knows the args defined as 'keyword = value'

Syntax Name of function (pos_args, keyword 1 = value 1, keyword 2 = value 2)

```
Display (age= 25, Name = "John")
# called function using keyword args.
```

def Display (num1, num2)

→ Display (40, num2=10) ✓

→ Display (num2=10, 40) ✗

→ Display (40, num1=40) ✗ (No duplicate values can be assigned to parameters)

Parameters with default values :-

```
def greet (name, msg = "Hello")
    print ("Hello", name, msg)
```

greet ("Sachin") ⇒ Hello Sachin Hello

greet ("Sachin", "Hi") ⇒ Hello Sachin Hi

```
def greet (msg = "Hello", name) # Error
```

→ Non-default arg cannot be followed by default arg

```
def display (a, b=10, c=20)
    print (a,b,c)
```

display (15) → 15,10,20

(15, b=30) ⇒ 50, 30, 20

(c=50, b=30, a=25) ⇒ 25, 30, 80.

Exception handling :—

What are the exceptions ?

Python has many built in exception that are raised when any program encounters an errors.
When these excep occur the python interpreter stops the current process and passes it to the calling proce until it is handled.
If these are not handled the program will crash.
For example :- Let us consider a program where we have a function 'A' that calls function 'B' which intern calls function 'C' and if an exception occur in function 'C' but it is not handled in 'C'. The exception passes through 'B' and then to 'A'.

⇒ ### Catching exceptions in Python :—

In python, the exceptions can be handled using a try statement.
The critical operation which can raise an exception is placed inside the try clause.
The code that handles the exception is written except block

```
import sys
randomlist = ['a', 0, 2]
for entry in randomlist :
    try :
        print (" the entry is ", entry)
        r = 1/int (entry)
        break
```

```
except :
    print (sys.exc_info() [0], "occurred")
    print (next entry)
print (" the reciprocal of", entry, "is", r)
```

O/P →

The entry is a
<class 'Value Error'> occurred
next entry
the entry is 0
<class 'zero division Error'> occurred.
next entry.
the entry is 2
reciprocal = 0.5.

Catching specific exception in Python :—

```
try :
    # do something
    pass
except  Value Error :
    # handle value error.
    pass
except ( Type Error , Zero Division Error) :
    # handle multiple exception
    pass
except :
    # handle all other errors
    pass
```

try-finally

the statement written in finally clause are executed, no matter what the condition is.

ex:

```
try
    f = open(t.txt)
# perform file operation
finally:
    f.close()
```

file handling in Python:-

* python supports many file operations such as reading, writing, appending, to and from the file. Other operations are also supported.

syntax:

open(filename, mode)

mode = "r" ⟶ for reading
"w" ⟶ for writing
"a" ⟶ for appending
"r+" ⟶ for both reading & writing.

Ex:

```
file = open("note.txt", 'r')
for each in file:
    print(each)
```

```
file = open("file.txt","r")
print(file.read(5)
```

↓

the interpreter will lead first five characters of data stored in a file "file.txt".

write( )

This will write the given strings in a file x.txt.

```
file = open("x.txt",'w')
file.write ("Hello")
file.write ("Next string")
file.close( )
```

```
file = open('x.txt', 'a')
# open file in append mode
file.write (" add this line")
file.close( )
```

Using write( ) with with( ) function :-

```
with open ("h.txt", w) as f :
    f.write (" Hello world")
```

```
with open("f.txt", 'r') as f:
    data = file.readlines( )
for line in data :
    word = line.split
    print(word)
```

f.txt

```
Hello my Name  → line 1
This is my     → line 2
```

⇒ readlines( ) funct. will read each line in a file

⇒ split( ) will split each line into words.

Regular expression :—

```
import re
txt = "The rain in Spain"
x = re.search("^The.*spain$")
```

This will read search the string that starts with The and ends with spain.

Regular expression and their meaning :—

^ → starts with → "^hello"

$ → ends with → "plant $"

* → zero or more occurence → he.*o

+ → one or more occurence

| → either → "falls / stays"

? → zero or one character → "he.?o"

. → Any character → "hello"

[a-m] → A set of characters between a to m → "[a-m]"

[arn] → either a or r or n

[^arn] → any char except a, r and n

[0-5][0-9] → any two digit number
first digit → 0 to 5
second → 0 to 9.

[a-z A-Z] → any character (lower or upper case)

Local & global scope of variable in python :-

local scope :

This variable is local function and cannot be used outside the function

```
def my func ()
    x = 300
    print(x)

my func()
```

def

call

```
def myfunc():
    x = 300
    def myinner fun()
        print (x)
    my inner func()
my fun()
```

→ A local variable can be accessed from a func written function.

Global scope :

```
x = 30
def my func():
    print (x)
my func()
```

print (x) ⟶ global variable can be accessed outside func def.

```
def my func():
    global x
    x = 30

myfunc()
    print (x) ⟶ 30
```