Taking user input in Python

syntex : input()

input(prompt=None, /)

```
In [1]: a = input('enter your name : ')
```

```
enter your name : harsh
```

```
In [2]: b = input(prompt='enter your name : ')
```

```
enter your name : shah
```

```
In [ ]:
```

Program to add 2 numbers

```
In [5]: a = int(input('1st number : '))
        b = int(input('2nd number : '))
        print('result : ',a+b)
```

```
1st number : 12
2nd number : 12
result :  24
```

```
In [ ]:
```

Type conversion

```
In [7]: list('hello')
Out[7]: ['h', 'e', 'l', 'l', 'o']
```

```
In [8]: set('welcome')
Out[8]: {'c', 'e', 'l', 'm', 'o', 'w'}
```

```
In [14]: tuple('harsh')
Out[14]: ('h', 'a', 'r', 's', 'h')
```

```
In [10]: # type conversion does not chaneges the value permanently
         a = 4
         float(4)
Out[10]: 4.0
```

```
In [11]: a
Out[11]: 4
```

There are mainly 2 types of type conversion

- Implicit (python automatically does that)
- Explicit (we need to do manually)

```
In [ ]: # Implicit
```

```
In [12]: 3+4.5
Out[12]: 7.5
```

```
In [13]: 1+1+3j
Out[13]: (2+3j)
```

```
In [ ]:
```

```
In [15]: # explicit
```

```
In [16]: str(4)
Out[16]: '4'
```

```
In [17]: complex(4)
Out[17]: (4+0j)
```

```
In [18]: int('3434')
Out[18]: 3434
```

```
In [ ]:
```

Variables are containers for storing data values

Literals :

Literals in Python is defined as the raw data assigned to variables or constants while programming.

There are five types of literal in Python,

- String Literals
  - Single-line String (Using single/double quotes)
  - Multi-line String (Using triple quotes)
- Numeric Literals
  - Integer
  - Float
  - Complex
  - Long
- Boolean Literals
  - True- True represents the value 1.
  - False-False represents the value 0.
- Literal Collections
  - List Literals
  - Tuple Literals
  - Dictionary Literals
  - Set Literals
- Special Literals

- special literal known as None

“A constant can be defined as an entity that has a fixed value or an entity whose value does not vary.”

or.

A constant is a type of variable whose value cannot be changed.

In Python, constants do not exist, but you can indicate that a variable is a constant and must not be changed

In [ ]:

```

```

string literals : group of characters in single, double or triple quotes

1. single line literal
   - String literals that are enclosed within single quotes (‘’) are known as single-line strings.

In [33]:

```
single_quotes_string='Scaler Academy'
double_quotes_string="Hello World"
print(single_quotes_string)
print(double_quotes_string)
```

```
Scaler Academy
Hello World
```

2. Multi-line String
   - A collection of characters or a string that goes on for multiple lines is a multi-line string.

In [21]:

```
#string literals
#multi line literal
str="Welcome \
to \
Scaler \
Academy"
print(str)
```

```
Welcome to Scaler Academy
```

In [29]:

```
str = 'Hi \
i \
am \
harsh'
print(str)
```

```
Hi i am harsh
```

In [31]:
```
str = '''
Hi
my
name
is
harsh
'''
print(str)
```

```

Hi
my
name
is
harsh

```

In [33]:
```
# unicode literal

# creating variables to holds
# the letters in python word.
p = "\u2119"
y = "\u01b4"
t = "\u2602"
h = "\u210c"
o = "\u00f8"
n = "\u1f24"

print(p+y+t+h+o+n)
```

```
ℙƴ☂ℌøἤ
```

or.

In [37]:
```
unicode_literal = u"\u2119\u01b4\u2602\u210c\u00f8\u1f24"
unicode_literal
```

Out[37]:
```
'ℙƴ☂ℌøἤ'
```

In [ ]:
```
```

2. Numeric Literals
    - Numerical literals are those literals that contain digits only and are immutable/ cannot be changed once assigned

Integer

```
different types of integers are-

+ Decimal- It contains digits from 0 to 9. The base for decimal v
alues is 10.
+ Binary- It contains only two digits- 0 and 1. The base for bina
ry values is 2 and prefixed with “0b”.
+ Octal- It contains the digits from 0 to 7. The base for octal v
alues is 8. In Python, such values are prefixed with “0o”.
+ Hexadecimal- It contains digits from 0 to 9 and alphabets from
 A to F.
```

In [1]:
```
# integer literal

#positive whole numbers
x = 2586

#negative whole numbers
y = -9856

# binary literal
a = 0b10101

# decimal literal
b = 505

# octal literal
c = 0o350

# hexadecimal literal
d = 0x12b

print (x,y)
print(a, b, c, d)
```

```
2586 -9856
21 505 232 299
```

Float

- Unlike integers, these contain decimal points.

  Float literals are primarily of two types-
  - Fractional- Fractional literals contain both whole numbers and decimal points.
  - Exponential- Exponential literals in Python are represented in the powers of 10.

In [5]:
```
print(78.256)
```

```
78.256
78.256
```

In [22]:
```
print(1.5e2) # 10.5 to the power 2
print(1.5e-1) # 10.5 to the power -1
```

```
150.0
0.15
```

Complex

- Complex literals are represented by A+Bj.
- Over here, A is the real part. And the entire B part, along with j, is the imaginary or complex part.

In [23]:
```
# complex literal
a=7 + 8j
b=5j
print(a)
print(b)
```

```
(7+8j)
5j
```

In [25]:
```
x = 23+5j
print(x,type(x))

# extracting real & imaginary part
print(x.real,x.imag)
```

```
(23+5j) <class 'complex'>
23.0 5.0
```

In [ ]:
```

```

Long - Long literals were nothing but integers with unlimited length

3. Boolean

Boolean literals in Python are pretty straight-forward and have only two values-

- True- True represents the value 1.
- False-False represents the value 0.

In [14]:
```
#boolean literals
x = (1 == 1)
y = (7 == False)
print("x is", x)
print("y is", y)
```

```
x is True
y is False
```

ex. explicit type conversion

In [38]:
```
a = True + 1 # 1 + 1
b = False + 2 # 0 + 2
print(a)
print(b)
```

```
2
2
```

4. Special Literals in Python

- Python literals have one special literal known as None.
- Python will print None as output when we print the variable with no value assigned to it.

In [15]:
```
#special literals
val=None  # None is the absence od anything
print(val)

# None : use for variable declaration
```

```
None
```

5. Literal Collections

   If we wish to work with more than one value, then we can go for literal collections in Python.

   - List Literals
   - Tuple Literals
   - Dictionary Literals
   - Set Literals

List

- Lists are a collection of data declared using the square brackets([]), and commas separate the elements of the list (,).
- This data can be of different types. Another important thing to know about lists is that they are mutable.

In [17]:
```
# list literals
numbers = [10, 20, 30, 40, 50]
names = ['John', 'Jake', 'Jason', 25]
print(numbers)
print(names)
```

```
[10, 20, 30, 40, 50]
['John', 'Jake', 'Jason', 25]
```

Tuple

- Tuple are a collection of data declared using round brackets(), and commas separate the elements of the tuple (,).
- unlike lists, tuples are immutable.

In [18]:
```
# tuple literals
even_numbers = (2, 4, 6, 8)
vowels=('a','e','i','o','u')
print(even_numbers)
print(vowels)
```

```
(2, 4, 6, 8)
('a', 'e', 'i', 'o', 'u')
```

Dictionaries

- Dictionaries are used to store data values in key:value pairs.
- A dictionary is a collection which is ordered*, changeable and do not allow duplicates.

In [19]:
```
# dictionary literals
my_dict = {'a': 'apple', 'b': 'bat', 'c': 'car'}
print(my_dict)
```

```
{'a': 'apple', 'b': 'bat', 'c': 'car'}
```

Set Literals

- Set literals are a collection of unordered data that cannot be modified.

In [20]:
```
#set literals
vowels = {'a', 'e', 'i', 'o', 'u'}
print(vowels)
```

```
{'e', 'a', 'i', 'u', 'o'}
```

In [ ]:
```
```

Operators :

Operators are used to perform operations on variables and values.

- Arithmatic operator
- Comparison operator
- Logical operator
- Bitwise operator
- Assignment operator
- Identity operator
- Membership operator

Arithmatic operator

```
+
-
*
/ [division]
% [modulus] -> gives remainder after dividing a from b (a%b)
** [Exponential] -> gives the result of a to the power b
// [Floor division] -> Ignores the decimal point if present
```

In [9]:
```
a = 20
b = 10
print(f'{a}+{b} = {a+b}')
print(f'{a}-{b} = {a-b}')
print(f'{a}%{b} = {a%b}')
```

```
20+10 = 30
20-10 = 10
20%10 = 0
```

In [17]:
```
a = 3.5
b = 2
print(f'{a}/{b} = {a/b}')
print(f'{a}//{b} = {a//b}')
```

```
3.5/2 = 1.75
3.5//2 = 1.0
```

In [18]:
```
a = 3
b = 2
print(f'{a}**{b} = {a**b}')
```

```
3**2 = 9
```

In [ ]:
```
```

Comparison operator

- operators compare the value of the left operand and the right operand and return either True or False.

```
==
!=
<> -> this is similiar to !=
>
<
>=
<=
```

In [20]:
```
a = 10
b = 20
c = 10

print(f'{a}=={b} : {a==b}')
print(f'{a}=={c} : {a==c}')
```

```
10==20 : False
10==10 : True
```

In [21]:
```
print(f'{a}!={b} : {a!=b}')
print(f'{a}!={c} : {a!=c}')
```

```
10!=20 : True
10!=10 : False
```

In [25]:
```
print(f'{a}>{b} : {a>b}')
print(f'{a}>{c} : {a>c}')
```

```
10>20 : False
10>10 : False
```

In [26]:
```
print(f'{a}<{b} : {a<b}')
print(f'{a}<{c} : {a<c}')
```

```
10<20 : True
10<10 : False
```

In [27]:
```
print(f'{a}>={b} : {a>=b}')
print(f'{a}>={c} : {a>=c}')
```
```
10>=20 : False
10>=10 : True
```

In [28]:
```
print(f'{a}<={b} : {a<=b}')
print(f'{a}<={c} : {a<=c}')
```
```
10<=20 : True
10<=10 : True
```

In [ ]:

Assignment operator

- Assignment operators used for assigning values to a variable.
- The values to be assigned must be on the right side, and the variable must be on the left-hand side of the operator.

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

In [37]:
```
a = 10
b = 20
c = 10

a+=b
print(a)
```
```
30
```

In [38]:
```
a = 10
b = 20
c = 10

a-=b
print(a)
```
```
-10
```

In [39]:
```
a = 10
b = 20
c = 10

a/=b
print(a)
```
```
0.5
```

In [40]:
```
a = 10
b = 20
c = 10

a*=b
print(a)
```
```
200
```

In [ ]:

```

```

When n1 % n2

- If n1 is smaller than n2 then n1 is returned

In [45]:

```
a = 9
b = 3


a%=b # 9%3 gives remainder
print(a)
```

```
0
```

In [4]:

```
a = 4
b = 3

a%=b # 4%3 remainder 1
print(a)
```

```
1
```

In [6]:

```
a = 3
b = 9

a%=b
print(a)
```

```
3
```

In [ ]:

```

```

In [8]:

```
a = 10
b = 2

a**=b # 10**2=100
print(
```

```
100
```

In [13]:

```
a = 5
b = 2

a//=b
print(a)
```

```
2
```

In [16]:

```
a = 5.5
b = 2

a//=b
print(a)
```

```
2.0
```

```
In [ ]:
```

Logical operator

- And (T * T = T)
- Or (T * anything = T)
- Not (Reverse the result)

Prog. Find largest number among 3 nummbers

```
In [19]: n1 = int(input('Enter 1st number : '))
         n2 = int(input('Enter 2nd number : '))
         n3 = int(input('Enter 3rd number : '))

         if n1>n2 and n1>n3:
             print(f'{n1} is greatest among {n1,n2,n3}')
         elif n2>n1 and n2>n3:
             print(f'{n2} is greatest among {n1,n2,n3}')
         else:
             print(f'{n3} is greatest among {n1,n2,n3}')
```

```
Enter 1st number : 44
Enter 2nd number : 33
Enter 3rd number : 22
44 is greatest among (44, 33, 22)
```

```
In [ ]:
```

```
In [22]: x = 10
         print(x<5 and x<11)
```

```
False
```

```
In [23]: print(x<5 or x<11)
```

```
True
```

```
In [24]: print(not(x<5 or x<11))
```

```
False
```

```
In [ ]:
```

Membership operator

- These operators search for the value in a specified sequence and return True or False accordingly.
- If the value is found in the given sequence, it gives the output as True, otherwise False.
- The not in operator returns true if the value specified is not found in the given sequence.

Ex. in, not in, is

```
In [3]: print(5 in [1,2,3,4,5])

        # returns true if the value is found in the specified sequence

        True

In [4]: print(15 in [1,2,3,4,5])

        False

In [5]: print(5.0 in [1,2,3,4,5])

        True

In [ ]: 

In [6]: print(5 in {1,2,3,4,5})

        True

In [9]: print(5.0 in {1,2,3,4,5})

        True

In [10]: print(15 in {1,2,3,4,5})

        False

In [ ]: 

In [11]: print(5 not in {1,2,3,4,5})

        False

In [12]: print(5.0 not in {1,2,3,4,5})

        False

In [13]: print(15 not in {1,2,3,4,5})

        True

In [ ]: 
```

is

- is mainly used for checking whether the 2 values are pointing to the same memory locatio

Note :

- If the 2 values look similiar does not mean that they are pointing to the same memory location

In [32]:
```
a = 10
b = 20
print(a is b)
```
```
False
```

In [33]:
```
a = 10
b = 10
print(a is b)
```
```
True
```

In [34]:
```
a = [1,2,3]
b = [1,2,3]
print(a is b)
```
```
False
```

In [35]:
```
a = {1:2,2:4}
b = {1:2,2:4}
print(a is b)
```
```
False
```

In [36]:
```
a = {1,2,3}
b = {1,2,3}
print(a is b)
```
```
False
```

In [37]:
```
a = 'harsh'
b = 'harsh'
print(a is b)
```
```
True
```

In [38]:
```
a = 'hello-harsh'
b = 'hello-harsh'
print(a is b)
```
```
False
```

In [39]:
```
a = 'hello_harsh'
b = 'hello_harsh'
print(a is b)
```
```
True
```

In [ ]:
```

```

Bitwise operator

- These operators perform operations on binary numbers.
- So if the number given is not in binary, the number is converted to binary internally, and then an operation is performed.
- These operations are generally performed bit by bit.
- For this operator : mainly refer the truth table

AND operator (&)

```
In [17]: a = 4 # so 4 in binary is 0100,
         b = 3 # 3 in binary is 0011.
```

```
In [18]: 
         '''
         0100
         0011
         ----
         ffff 0
         '''

         print(a&b)
```

```
0
```

OR operator (|)

```
In [19]: '''
         0100
         0011
         ----
         0111 (4+2+1)
         '''

         #8421 binary to number
         print(a|b)
```

```
7
```

XOR operator(^)

```
In [20]: '''
         0100
         0011
         ----
         0111 (4+2+1)
         '''

         #8421 binary to number
         print(a^b)
```

```
7
```

NOT operator(~)

In [24]:
```
# invert all the bits

a = 4 # so 4 in binary is 0100,
b = 3 # 3 in binary is 0011.

'''
0100
----
1011 (8+2+1)
'''


#8421 binary to number
print(~a)
print(~b)
```

```
-5
-4
```

Negative number to binary conversion [Concept]

https://www.youtube.com/watch?v=MXUVr7dB7Uo (https://www.youtube.com/watch?v=MXUVr7dB7Uo)

Bitwise left shift:

- << Zero fill - Left shift
- Shifts the bits of the number to the left and fills 0 on voids right as a result. Similar effect as of multiplying the number with some power of two.

In [27]:
```
a = 5 # 0000 0101 (Binary)
print(a << 1) # 0000 1010 = 10
```

```
10
```

In [28]:
```
a = 5 # 0000 0101 (Binary)
print(a << 2) # 0001 0100 = 20
```

```
20
```

Bitwise right shift:

- Shifts the bits of the number to the right and fills 0 on voids left(fills 1 in the case of a negative number) as a result.
- Similar effect as of dividing the number with some power of two.

In [29]:
```
a = 10 # 0000 1010 (Binary)
print(a >> 1) # 0000 0101 = 5
```

```
5
```

In [31]:
```
a = -10 # 1111 0110 (Binary)
print(a >> 1) # 1111 1011 = -5
```

```
-5
```

In [ ]:

If-else Statement

In [46]:
```
# default email : python@gmail.com
# default pass : 1234

email = input('Enter the email : ')

if '@' in email:
    pwd = input('Enter the password : ')
    if email=='python@gmail.com' and int(pwd)==1234:
        print('Welcome User !!')
    if email=='python@gmail.com' and int(pwd)!=1234:
        print('Invalid pass ')
        pwd = input('Re-enter the password')
        if 1234==int(pwd):
            print('Welcome user !!')
        else:
            print('Still Incorrect pass')
else:
    print('Invalid Email !!')
```

```
Enter the email : python@gmail.com
Enter the password : 123
Invalid pass
Re-enter the password1234
Welcome user !!
```

In [ ]:

Indentation in Python

- Python does not use curly brances {} or semi-colon :
- Improves code-readibility & debugging process easy

Loops

- Repeatative task
- real life eg: flipkart search results
    - phone details [img + data] -> container
    - difference content
    - similarity -> format of display of information

we will create a single container and keep it in loop & fetch data from db and the container will be printed multiple times

Types of loops :

- While
- For

Printing tables n

In [52]:
```python
tno = int(input("enter the table no . : "))

i = 1
while i<11:
    print(f'{tno} x {i} = {tno*i}')
    i+=1
```

```
enter the table no . : 3
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
```

```
In [54]: eno = int(input("print table till . : "))

         c = 1
         while c<=eno:
             i = 1
             while i<11:
                 print(f'{c} x {i} = {c*i}')
                 i+=1
             c+=1
             print()
```

```
print table till . : 3
1 x 1 = 1
1 x 2 = 2
1 x 3 = 3
1 x 4 = 4
1 x 5 = 5
1 x 6 = 6
1 x 7 = 7
1 x 8 = 8
1 x 9 = 9
1 x 10 = 10

2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
2 x 4 = 8
2 x 5 = 10
2 x 6 = 12
2 x 7 = 14
2 x 8 = 16
2 x 9 = 18
2 x 10 = 20

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
```

```
In [ ]:
```

Guessing Game in Python

- gen 1-100 random no.
- ask user to enter guess number
  - give instruction guess higher or lower
- save no of attempts

In [64]:
```
import random
rno = random.randint(1,100) # boundary included

tries = 0
while True:
    guess = int(input('Guess the number : '))
    print('-------------------------------------------------------------'
    tries+=1
    if guess==rno:
        print(f'You guessed the number successfully in {tries} tries !!')
        print('you won $5000')
        print('-----------------------------------------------------------
        break
    elif guess>rno:
        print('Hint :: Lower your guess number ')
        print('-----------------------------------------------------------
    elif guess<rno:
        print('Hint :: Increse your guess number')
        print('-----------------------------------------------------------
```

```
Guess the number : 23
-------------------------------------------------------------
Hint :: Increse your guess number
-------------------------------------------------------------
Guess the number : 34
-------------------------------------------------------------
Hint :: Lower your guess number
-------------------------------------------------------------
Guess the number : 30
-------------------------------------------------------------
Hint :: Increse your guess number
-------------------------------------------------------------
Guess the number : 31
-------------------------------------------------------------
Hint :: Increse your guess number
-------------------------------------------------------------
Guess the number : 32
-------------------------------------------------------------
You guessed the number successfully in 5 tries !!
you won $5000
-------------------------------------------------------------
```

In [ ]:
```
```

Range Fucntion

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

```
0 1 2 3 4
```

In [3]:
```
for i in range(2,5):
    print(i,end=' ')
```

```
2 3 4
```

In [6]:
```
for i in range(1,10,2):
    print(i,end=' ')
```
```
1 3 5 7 9
```

In [ ]:

In [7]:
```
list(range(1,11))
```
Out[7]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

In [ ]:

String :

- Is a sequence of characters

Tuple, list, set, dic :

- sequence of words

For loop

- Only iterates over a range fucntion or the sequence

In [8]:
```
for i in 'happy harsh':
    print(i,end=' ')
```
```
h a p p y   h a r s h
```

In [9]:
```
for i in range(2,5):
    print(i,end=' ')
```
```
2 3 4
```

In [12]:
```
for i in (4,2,1):
    print(i,end=' ')
```
```
4 2 1
```

In [14]:
```
for i in [1,5,2]:
    print(i, end=' ')
```
```
1 5 2
```

In [ ]:

Use case of differerent Loop :

- For loop use : when we know how many times to loop through
- while loop use : unknown how many times loop through

Nested Loop :

- using loop inside another loop

```
In [22]: n = int(input('Enter the rows : '))
         for i in range(n+1):
             for j in range(i):
                 print('$',end = '')
             print()
```

```
Enter the rows : 5

$
$$
$$$
$$$$
$$$$$
```

```
In [23]: n = int(input('Enter the rows : '))
         for i in range(n+1):
             for j in range(i):
                 print('$ ',end = '')
             print()
```

```
Enter the rows : 5

$
$ $
$ $ $
$ $ $ $
$ $ $ $ $
```

```
In [ ]:
```

Break-continue and pass Statements

Break ex - Linear Search

- searching user from a db and braking the execution when the user is found

Continue ex - continue ke aage ka code - skip that iteration (skip ho jata hai)

- flipkart : current product stock mai hai toh display if not in stock the display code is skipped

pass -

- The pass statement is used as a placeholder for future code.
- When the pass statement is executed, nothing happens, but you avoid getting an error when empty code is not allowed. Empty code is not allowed in loops, function definitions, class definitions, or in if statements.

```
In [ ]:
```

Build-in Fucntions

- print()
- type()
- input()

- type conversion fucntions ex. int, str, float, list
- abs # absolute function
- pow # popwer fucntn
- min/max
- round # round(var,places)
- divmod
- bin/oct/hex
- id : returns the address
- ord : returns the ascii code of the char
- len()
- sum()
- help()

```
In [25]: abs(3)
Out[25]: 3

In [26]: abs(-3)
Out[26]: 3

In [27]: pow(2,3) # 2 to the power 3
Out[27]: 8

In [ ]: 

In [28]: min([1,2,3,46,6])
Out[28]: 1

In [29]: max([1,2,3,46,6])
Out[29]: 46

In [31]: max('harsh')
         # on the basis of ascii value
Out[31]: 's'

In [32]: min('harsh')
Out[32]: 'a'

In [ ]: 

In [34]: q = 22/7
         q
Out[34]: 3.142857142857143

In [35]: round(q,2)
Out[35]: 3.14
```

```
In [ ]:
```

Signature: divmod(x, y, /)

Docstring: Return the tuple (x//y, x%y).

```
In [36]: divmod(5,2)

Out[36]: (2, 1)

In [ ]:

In [37]: print(bin(4))
         print(oct(4))
         print(hex(4))

         0b100
         0o4
         0x4

In [ ]:

In [38]: a = 1
         id(a)

Out[38]: 9801248

In [ ]:

In [39]: ord('a')

Out[39]: 97

In [40]: ord('A')

Out[40]: 65

In [ ]:

In [42]: sum([1,2,34,4])

Out[42]: 41

In [45]:  sum({4,2,34,4})

Out[45]: 40

In [ ]:
```

In [48]:
```
help(len)
```

```
Help on built-in function len in module builtins:

len(obj, /)
    Return the number of items in a container.
```

In [ ]:

Modules

- Module is same as a code library
- Module is a set of fuctn's you want to include in your application

Examples of python modules

- Math
- random
- os
- time

In [50]:
```
help('modules') # list of modules present int the sys
```

```

Please wait a moment while I gather a list of all available modules...


/usr/lib/python3.8/pkgutil.py:107: VisibleDeprecationWarning: zmq.even
tloop.minitornado is deprecated in pyzmq 14.0 and will be removed.
    Install tornado itself to use zmq with the tornado IOLoop.

  yield from walk_packages(path, info.name+'.', onerror)

INFO:root:Generating grammar tables from /usr/lib/python3.8/lib2to3/Gr
ammar.txt
INFO:root:Generating grammar tables from /usr/lib/python3.8/lib2to3/Pa
tternGrammar.txt

/usr/lib/python3/dist-packages/UpdateManager/backend/__init__.py:11: P
yGIWarning: Gtk was imported without specifying a version first. Use g
i.require_version('Gtk', '3.0') before import to ensure that the right
version gets loaded.
  from gi.repository import GLib, Gtk, Snapd
```

In [ ]:

In [51]:
```
import math
math.pi
```

Out[51]:
```
3.141592653589793
```

In [52]:
```
math.e
```

Out[52]:
```
2.718281828459045
```

```
In [53]: math.factorial(5)

Out[53]: 120

In [54]: math.ceil(5.5)

Out[54]: 6

In [55]: math.floor(5.5)

Out[55]: 5

In [56]: math.sqrt(25)

Out[56]: 5.0

In [ ]:

In [57]: import random
         random.randint(1,20)

Out[57]: 4

In [58]: a = [1,2,3,4]
         random.shuffle(a)

In [59]: a

Out[59]: [2, 3, 1, 4]

In [ ]:

In [60]: import time
         time.time()


Out[60]: 1659114724.7055874

In [61]: time.ctime() # current time

Out[61]: 'Fri Jul 29 22:42:55 2022'

In [62]: print('harsh')
         time.sleep(2)
         print('ok')

         harsh
         ok

In [ ]:

In [63]: import os
         os.getcwd()

Out[63]: '/home/harsh'

In [ ]:
```

String in Python

- String is a sequence of characters
- In python, string's are a sequence of unicode characters

Creating a string

In [1]:
```
msg = 'hello rin'
print(msg)
```
```
hello rin
```

In [3]:
```
msg1 = "hi rin what's up"
print(msg1)
```
```
hi rin what's up
```

In [14]:
```
msg3 = '''hey buddy
my name is
harsh'''
print(msg3)
```
```
hey buddy
my name is
harsh
```

In [15]:
```
msg4 = '''hey buddy \
my name is \
harsh'''
print(msg4)
```
```
hey buddy my name is harsh
```

In [ ]:
```

```

type casting in string

In [17]:
```
a = 2323
conv = str(a)
print(conv,type(conv))
```
```
2323 <class 'str'>
```

In [ ]:
```

```

Accessing sub-string from a string

Types of indexing :

- Positive Indexing
- negative indexing

In [21]:
```
# Concept of indexing
a = 'life is beautifulx'
print(a)
```
```
life is beautifulx
```

In [22]:
```
a[0]
```
Out[22]:
```
'l'
```

In [23]:
```
a[-1]
```
Out[23]:
```
'x'
```

In [25]:
```
a[4]
```
Out[25]:
```
' '
```

In [ ]:
```

```

Slicing in string

In [26]:
```
a = 'life is beautifulx'
print(a[0:4])
```
```
life
```

In [28]:
```
print(a[5:])
```
```
is beautifulx
```

In [29]:
```
print(a[:-1])
```
```
life is beautiful
```

In [30]:
```
print(a[:5])
```
```
life
```

In [31]:
```
print(a[:])
```
```
life is beautifulx
```

In [35]:
```
print(a[2:9:3])
```
```
fib
```

In [ ]:
```

```

Note :

- When we are working woth a positive indexing we cannot take -ve (negative) steps

In [36]:
```
print(a[2:9:-3])
```

```
In [37]: msg = 'Hello World'
         print(msg[-5:-1:2])

         Wr
```

```
In [38]: # to reverse the string
         msg[::-1]

Out[38]: 'dlroW olleH'
```

```
In [49]: msg = 'Hello World'
         msg[-1:-5:-1]

Out[49]: 'dlro'
```

```
In [ ]:
```

Editing and Deleting String's

Note:

- String is a immutable data type
- Once assigned you cannot make chages to it
  - I can neither add new char to the str
  - Nor u can make chages to the exisiting char

```
In [1]: s = 'new string'
        print(s)

        new string
```

```
In [2]: s[0]

Out[2]: 'n'
```

```
In [3]: s[0]='x'

        ------------------------------------------------------------------------
        ----
        TypeError                                 Traceback (most recent call l
        ast)
        Input In [3], in <module>
        ----> 1 s[0]='x'

        TypeError: 'str' object does not support item assignment
```

```
In [5]:  # del of string

        del s
```

```
In [6]: s
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [6], in <module>
----> 1 s

NameError: name 's' is not defined
```

```
In [ ]:
```

Operations on String

- Arithmatic Operations
- Relational Operations
- Logical Operations
- Loops on string
- Membership Operations

Adding 2 string (string concatination)

```
In [7]: 'hello'+'harsh'
```

```
Out[7]: 'helloharsh'
```

```
In [8]: 'hello'+'harsh'+'bhai'
```

```
Out[8]: 'helloharshbhai'
```

String Multiplication

```
In [9]: '#'*5
```

```
Out[9]: '#####'
```

```
In [10]: 'hello'*3
```

```
Out[10]: 'hellohellohello'
```

Comparison of string's

```
In [11]: 'hello'=='hello'
```

```
Out[11]: True
```

```
In [12]: 'Hello'=='hello'
```

```
Out[12]: False
```

```
In [ ]:
```

In [13]:
```
'Hello'!='hello'
```
Out[13]:
```
True
```

In [ ]:

Lexiographycally comaparison

- Dicting based comparison
- word jo baad mai aayega wo bada hoga
- word jo pehla aayega wo chota hoga

....

- small letters baad mai aate hai
- capital letters pehla aata hai

In [14]:
```
'Mumbai'<'Ahmedabad'
```
Out[14]:
```
False
```

In [15]:
```
'apple'<'carrot'
```
Out[15]:
```
True
```

In [16]:
```
'zen'<'rat'
```
Out[16]:
```
False
```

In [ ]:

In [17]:
```
'kim'>'Kim'
```
Out[17]:
```
True
```

In [ ]:

Using logical operators on string

- Empty string : Python false
- Non Empty String : python true

In [19]:
```
'hello' and 'world'
# T and T = T
```
Out[19]:
```
'world'
```

In [ ]:

In [20]:
```
'' and 'world'
# F and T = F
```
Out[20]:
```
''
```

In [21]:
```
'' or 'wprlds'
# f or T = T
```
Out[21]:
```
'wprlds'
```

In [ ]:

In [23]:
```
'hello' or 'wordls'
# 1st value T hai toh we dont check further = !st value
```
Out[23]:
```
'hello'
```

In [28]:
```

'''
checking of the 2nd value gives us the surety in and
operator. Whether the result is True or false
'''

'hello' and 'wordls'
```
Out[28]:
```
'wordls'
```

In [ ]:

In [29]:
```
not ''
```
Out[29]:
```
True
```

In [30]:
```
not 'true'
```
Out[30]:
```
False
```

In [ ]:

Using Loop on string

In [33]:
```
msg = 'hello buddy'

for i in msg:
    print(i,end=' ')
```
```
h e l l o   b u d d y
```

In [34]:
```
msg = 'hello buddy'

for i in msg[::-1]:
    print(i,end=' ')
```
```
y d d u b   o l l e h
```

In [ ]:

reference : string slicing

https://www.youtube.com/watch?v=USw-dS6fHm4 (https://www.youtube.com/watch?v=USw-dS6fHm4)

In [35]:
```
msg = 'hello buddy'

for i in msg[1::-1]:
    print(i,end=' ')
```
```
e h
```

In [ ]:

In [38]:
```
msg = 'hello buddy'

for i in msg[2:7:2]:
    print(i,end=' ')
```
```
l o b
```

In [ ]:

Using membership operator with string in py

- in
- not in

In [39]:
```
msg = 'my name is harsh'

'x' in msg
```
Out[39]:
```
False
```

In [40]:
```
'h' in msg
```
Out[40]:
```
True
```

In [45]:
```
'  is' in msg
```
Out[45]:
```
False
```

In [47]:
```
'is' in msg
```
Out[47]:
```
True
```

In [ ]:

In [48]:
```
msg = 'my name is harsh'

'x' not in msg
```
Out[48]:
```
True
```

In [49]:
```
' is ' not in msg
```
Out[49]:
```
False
```

```
In [51]: '  is' not in msg

Out[51]: True

In [52]: 'harsh' not in msg

Out[52]: False

In [ ]: 
```

Common Functions

- len
- max
- min
- sorted

```
In [56]: msg = 'welcome'
         len(msg)

Out[56]: 7

In [55]: # on the basis of ascii value it will tell us
         # which char is the greatest
         max(msg)

Out[55]: 'w'

In [57]: min(msg)

Out[57]: 'c'

In [61]: '''
         on the basis of ascii value it will sort the list
         in ascending order as default
         - it return type is list

         But if you want it in descending order, we can
         use reverse=True

         '''
         print('msg : ',msg)
         sorted(msg)

         msg :  welcome

Out[61]: ['c', 'e', 'e', 'l', 'm', 'o', 'w']

In [ ]: 
```

Functions that can only be used with string data type

- Capitalize : 1st letter capitalize
- Title : Every word 1st alph. capitalize
- Upper
- Lower
- Swapcase : Lower to upper, upper to lower each letter in string

Does not change the orignal string

In [67]:
```
msg = 'welcome back bro'
print(msg.capitalize())
```

```
Welcome back bro
```

In [69]:
```
msg = 'welcome back bro'
print(msg.title())
```

```
Welcome Back Bro
```

In [70]:
```
msg
# no change in the original string
```

Out[70]:
```
'welcome back bro'
```

In [ ]:

In [72]:
```
msg1 = 'welcome back bro'
msg1.upper()
```

Out[72]:
```
'WELCOME BACK BRO'
```

In [73]:
```
msg2 = 'WELCOME BACK BRO'
msg2.lower()
```

Out[73]:
```
'welcome back bro'
```

In [74]:
```
msg1 # no change in the original string
```

Out[74]:
```
'welcome back bro'
```

In [75]:
```
msg2 # no change in the original string
```

Out[75]:
```
'WELCOME BACK BRO'
```

In [ ]:

In [76]:
```
'WeLcOMe BAcK brO'.swapcase()
```

Out[76]:
```
'wElComE baCk BRo'
```

In [ ]:

Count

- Find frequency of any sub-string into any string

In [82]:
```
x = 'wElComE baCk BRoe'
x.count('E')
```

Out[82]:
```
2
```

In [83]:
```
x.count('e')
```
Out[83]:
```
1
```

In [84]:
```
x.count(' ')
```
Out[84]:
```
2
```

In [ ]:
```
```

Find/Index

- returns the 1st occurance of the char in str
- If the particular substing/char is not found int the string, then find() returns -1

Major difference between find and index is

- In find(),If the particular substing/char is not found int the string, then find() returns -1

whereas

- In index(), u will receive an error stating 'substring not found'

Note :

- Prefer find() instead of index()

In [87]:
```
x = 'wElComE baCk BRoe warns'
x.find('w')
```
Out[87]:
```
0
```

In [88]:
```
x.find('baCk')
```
Out[88]:
```
8
```

In [91]:
```
x.find('')
```
Out[91]:
```
0
```

In [89]:
```
x.find('ip')
```
Out[89]:
```
-1
```

In [ ]:
```
```

In [92]:
```
x = 'wElComE baCk BRoe warns'
x.index('w')
```
Out[92]:
```
0
```

```
In [93]: x.index('ip')

         ---------------------------------------------------------------------------
         ValueError                                Traceback (most recent call l
         ast)
         Input In [93], in <module>
         ----> 1 x.index('ip')

         ValueError: substring not found


In [ ]:
```

endswith/startswith

```
In [95]: msg = 'You cant defeat me'
         msg.endswith('me')

Out[95]: True

In [96]: msg.endswith('defeat me')

Out[96]: True

In [97]: msg.endswith('me ')

Out[97]: False

In [98]: msg.endswith(' me')

Out[98]: True

In [103]: msg.endswith('')

Out[103]: True

In [ ]:

In [99]: msg = 'You cant defeat me'
         msg.startswith('me')

Out[99]: False

In [100]: msg.startswith('y')

Out[100]: False

In [101]: msg = 'You cant defeat me'
          msg.startswith('')

Out[101]: True

In [102]: msg = 'You cant defeat me'
          msg.startswith('Y')

Out[102]: True
```

```
In [ ]:
```

format

- The format() method returns the formatted string.
- The format() method formats the specified value(s) and insert them inside the string's placeholder.
- The placeholder is defined using curly brackets {}

Ex. Login page - display welcome msg

```
In [104]: 'hey my name is {} and I am diving deep into {}'.format('Harsh','Python')
Out[104]: 'hey my name is Harsh and I am diving deep into Python'
```

```
In [112]: 'hey my name is {1} and I am diving deep into {0}'.format('Forest','Pytho
Out[112]: 'hey my name is Python and I am diving deep into Forest'
```

```
In [113]: 'hey my name is {x} and I am diving deep into {y}'.format(x='Amazon Fores
Out[113]: 'hey my name is Amazon Forest and I am diving deep into Py Python'
```

```
In [114]: 'hey my name is {x} and I am diving deep into {x}'.format(x='Amazon Fores
Out[114]: 'hey my name is Amazon Forest and I am diving deep into Amazon Forest'
```

```
In [115]: 'hey my name is {y} and I am diving deep into {x}'.format(x='Amazon Fores
Out[115]: 'hey my name is Py Python and I am diving deep into Amazon Forest'
```

```
In [2]: 'I have 2 {x} and 1 {y}'.format(x='berry',y='apple',z='kiwi')
Out[2]: 'I have 2 berry and 1 apple'
```

```
In [ ]:
```

```
In [110]: name = 'Harsh'
          age = 21
          f'My name is : {name},  my age is : {age}'
Out[110]: 'My name is : Harsh,  my age is : 21'
```

```
In [ ]:
```

fucntions return's True or False

these functions mainly ask questions

- isaplha
- isalnum
- isdecimal
- isdigit
- isidentifier

```
In [4]: 'FAL234'.isalnum()

Out[4]: True

In [5]: 'FAL234'.isalpha()

Out[5]: False

In [6]: 'aaAx'.isalpha()

Out[6]: True

In [7]: '3232A'.isdecimal()

Out[7]: False

In [9]: '3232'.isdecimal()

Out[9]: True

In [10]: '3232A'.isalnum()

Out[10]: True

In [11]: '33'.isdigit()

Out[11]: True

In [12]: '3232A'.isdigit()

Out[12]: False

In [13]: 'Hello world'.isidentifier()

Out[13]: False

In [14]: 'Hello_world'.isidentifier()

Out[14]: True

In [ ]:
```

split fucntion

- converts the string into the list
- It does the opposote of join()

In [15]:
```
"Fear leads to anger; anger leads to hatred; hatred leads to conflict; co
```
Out[15]:
```
['Fear',
 'leads',
 'to',
 'anger;',
 'anger',
 'leads',
 'to',
 'hatred;',
 'hatred',
 'leads',
 'to',
 'conflict;',
 'conflict',
 'leads',
 'to',
 'suffering.']
```

In [16]:
```
"Fear leads to anger; anger leads to hatred; hatred leads to conflict; co
```
Out[16]:
```
['Fear leads ',
 ' anger; anger leads ',
 ' hatred; hatred leads ',
 ' conflict; conflict leads ',
 ' suffering.']
```

In [18]:
```
"Fear leads to anger; anger leads to hatred; hatred leads to conflict; co
```
Out[18]:
```
['Fear leads to anger; anger leads to hatred; hatred leads to conflict;
conflict leads to suffering.']
```

In [ ]:
```

```

join fucntion

- It converts the list into string
- It does the opposite of split

In [21]:
```
# using split()
'life is beautiful yet its cruel yet its beauty lies inside'.split()
```
Out[21]:
```
['life',
 'is',
 'beautiful',
 'yet',
 'its',
 'cruel',
 'yet',
 'its',
 'beauty',
 'lies',
 'inside']
```

```
In [22]: # using join()

         ''.join(['life',
          'is',
          'beautiful',
          'yet',
          'its',
          'cruel',
          'yet',
          'its',
          'beauty',
          'lies',
          'inside'])

Out[22]: 'lifeisbeautifulyetitscruelyetitsbeautyliesinside'
```

```
In [23]: ' '.join(['life',
          'is',
          'beautiful',
          'yet',
          'its',
          'cruel',
          'yet',
          'its',
          'beauty',
          'lies',
          'inside'])

Out[23]: 'life is beautiful yet its cruel yet its beauty lies inside'
```

```
In [25]: '__'.join(['life',
          'is',
          'beautiful',
          'yet',
          'its',
          'cruel',
          'yet',
          'its',
          'beauty',
          'lies',
          'inside'])

Out[25]: 'life__is__beautiful__yet__its__cruel__yet__its__beauty__lies__inside'
```

```
In [26]: '-'.join(['life',
          'is',
          'beautiful',
          'yet',
          'its',
          'cruel',
          'yet',
          'its',
          'beauty',
          'lies',
          'inside'])

Out[26]: 'life-is-beautiful-yet-its-cruel-yet-its-beauty-lies-inside'
```

```
In [ ]:
```

Replace fuctn

- The replace() method returns a copy of the string where the old substring is replaced with the new substring.
- The original string is unchanged.
- If the old substring is not found, it returns the copy of the original string.

```
In [27]: 'Nory was a Catholic because her mother was a Catholic, and Nory’s mothe

Out[27]: 'Rin was a Catholic because her mother was a Catholic, and Rin’s mother
         was a Catholic because her father was a Catholic'

In [29]: 'Rin was a Catholic because her mother was a Catholic, and Rin’s mother w

Out[29]: 'Rin was a Catholic because her mother was a Catholic, and Rin’s mother
         was a Catholic because her father was a Catholic'

In [30]: 'Rin was a Catholic because her mother was a Catholic, and Rin’s mother w

Out[30]: 'Rin was a Indian because her mother was a Indian, and Rin’s mother was
         a Indian because her father was a Indian'

In [ ]:
```

Strip Fucntn

- The strip() method removes characters from both left and right based on the argument (a string specifying the set of characters to be removed).

or

- Python string method strip() returns a copy of the string in which all chars have been stripped from the beginning and the end of the string (default whitespace characters).

Note: If the chars argument is not provided, all leading and trailing whitespaces are removed from the string.

Ex. Real life - user registeration page

- user enter name
- db mai isi format mai store hoga jo dangerous hai is lia strip use karte hai

```
In [31]: name = '                  arav                  '
         name.strip()

Out[31]: 'arav'

In [ ]:
```