Python

```
10.5//2  #Floor operator discard the decimal
```

```
5.0
```

```
round(10.222,2)
```

```
10.22
```

```
5%2      #display the reminder
```

```
1
```

```
a=[1,1,2,3,1,3,2,3,1,2,3,2,2]

if a.count(1)>a.count(2):
  print('count of 1 is more')

else:
  print('count of 2 is more')
```

```
count of 2 is more
```

multi line string

```
x='''djkjojoj
jojojojoj
jojojo'''

print(x)
```

```
djkjojoj
jojojojoj
jojojo
```

```
'''djkjojoj
jojojojoj
jojojo'''
```

```
'djkjojoj\njojojojoj\njojojo'
```

```
#THIS IS SINGLE LINE COMMENT

#single line string

h='sunil rathod this is single line'

print(h)
```

```
sunil rathod this is single line
```

Rules for defining variables

-Variable name should **not start with the number**. ex. 1num, 2x

-Variable name should **not have blank space** in it. ex. num 1, first name

-Variable name should not contain any Special Character except '_' (underscore) ex. @num, x#

-a and A are both different variables. ex. num1 and Num1 is different.

-variable name should not match with any python's reserved keywords. ex. is, in, and, if,else,for,while,or

-Always try to keep you varibale name short and informative.

- Exmaple

```
account_balance=
```

Naming Conventions

- packages : utilities
- modules : db_utilies
- Classes : BankAccount
- Function : open_acount
- variable : account_id
- constant : MIN_APR

```
age=18
```

```
print(age)
```

```
18
```

```
type(age)
```

```
int
```

Variable Display method

```
name="sunil"
```

```
print("my name is ",name)                    #by comma( ,)
```

```
my name is  sunil
```

```
print("my name is {}".format(name))          #by format function
```

```
my name is sunil
```

imp by f function

```
print(f"my name is {name}")                   # by f function
```

```
my name is sunil
```

GLOBAL AND LOCAL VARIABLE

```
y=10                  #Global variable
def automobile():
     global z          #conver local variable as global
     z=30
     x='car'               #local variable
     print(x)


automobile()
print(z)
print(y)
```

```
car
30
10
```

How to accept a value from User

```
name = input("Enter your name : ")

print(name)
```

```
Enter your name : sun
sun
```

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

print(age)
```

```
Enter your age : 28
28
```

Data types in Python

- integer - a =10
- float - b = 10.9 (decimal values)
- string - c = 'IEC' (collection of characters)
- boolean - True, False

- Natural Number =1,2,3,...
- Whole Number =0,1,2.....
- Integer Number = -2,-1,0,1,2.....
- Odd and even Number
- Prime Number

Data Structures and Its Types

Types of Data Structures (Sequences) :Collection of element

- List ["m",1,2]
- Tuple ("k",1,4)
- Sets {"m",451,4}
- Dictionary {"id":1,"name":"sunil"}

1 LISTS

- Syntax - var_name = [value1, value2, value3,...., valueN]
  - the values are enclosed in square brackets [1,3,2]
  - lists follows indexing and the indices starts with 0 to N-1
  - the values stored in the list are called elements
  - lists can versatile elements. List can store multiple datatypes in a single variable
- 1.order important.indexing and slicing important 2.duplicate allowed 3.denotaed by [ ]
- List is a collection of multiple elements of different (or same) data types.

```
a=[1,2,3,4,5,'sun','moon']

type(a)
```

```
list
```

len function

- len() function is used to get the length (size) of the list.

```
len(a)
```

```
7
```

Index Function

```
a.index("sun")
```

```
5
```

Access a element from the list

```
a=[1,2,3,4,5,'sun','moon']

#Positive Index 0....len(list)-1.
#Negative Index   -1....-len(list)


a[0]                                        # the index of 1st element is 0
```

```
1
```

```
a[1]
```

```
2
```

```
a[1]=True     #Replace the element
a
```

```
[1, True, 3, 4, 5, 'sun', 'moon']
```

```
a[8]      #IndexError: list index out of range
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-24-59b20334f494> in <cell line: 1>()
----> 1 a[8]      #IndexError: list index out of range

IndexError: list index out of range
```

SEARCH STACK OVERFLOW

```
for i in a:
    print(i)
```

```
1
2
3
4
5
sun
moon
```

```
[i for i in a]    #list comprehension
```

```
[1, 2, 3, 4, 5, 'sun', 'moon']
```

Add a new element to the list

- append()
- extend()

- insert()

```
a=[1, 2, 3, 4, 5]

a.append(100)
a
```

```
[1, 2, 3, 4, 5, 100]
```

```
a.extend([105, 106, 107])  # add multiple elements at the end of the list

a
```

```
[1, 2, 3, 4, 5, 100, 105, 106, 107]
```

```
b=[7,8]

a.extend(b)

a
```

```
[1, 2, 3, 4, 5, 100, 105, 106, 107, 7, 8]
```

```
a.insert(7,1000)       # a.insert(index, value)  - adds a new element at the given index



a
```

```
[1, 2, 3, 4, 5, 100, 105, 1000, 1000, 106, 107, 7, 8]
```

```
a[1:1] = ["two"]                    # adding a new element at index 1 using slicing
a
```

```
[1, 'two', 'two', 2, 3, 4, 5, 105, 106, 1000, 107, 7, 8, 105, 106, 107, 7, 8]
```

```
a[2:2] = [100, 101, 102]           # adding multiple elements  using slicing

a
```

```
[1,
 'two',
 100,
 101,
 102,
 'two',
 2,
 3,
 4,
 5,
 105,
 106,
 1000,
 107,
 7,
 8,
 105,
 106,
 107,
 7,
 8]
```

Delete elements from the list¶

- del
- pop()
- pop(index)
- remove(value)
- clear()

```
a=[10,20,30,40,50,60,70,80]
```

del

```
del a[0]   # del keyword
```

```
del a[2:5]  # deleting multiple elements from the list using slcing comcept
```

```
a
```

```
[20, 30, 70, 80]
```

pop

```
a.pop()   # deletes the last element of the list
```

```
80
```

```
a.pop(2)  # a.pop(index) - deletes the value at the specified index
```

```
70
```

remove

```
a.remove(20)   # a.remove(value)  - deletes the value
```

```
a
```

```
[30]
```

clear

```
a.clear()  # deletes all th elements of the list
```

```
a
```

```
[]
```

Slicing - Accessing a subset of a list¶

- slicing enables us to access multiple elements at a time
- Syntax - list_name[start: end: step] - where ends at end-1 , skips step-1 elements
- Syntax - for reversing a list - list_name[::-1]

```
index = ["zero", "one", "two", "three", "four", "five"]
```

```
index
```

```
['zero', 'one', 'two', 'three', 'four', 'five']
```

```
index[0]
```

```
'zero'
```

```
index[0:]
```

```
['zero', 'one', 'two', 'three', 'four', 'five']
```

```
index[3:]
```

```
['three', 'four', 'five']
```

```
index[1:]  # index[start: ]  - starts from index 1 and ends at end of the list
```

```
['one', 'two', 'three', 'four', 'five']
```

```
index[:4]
```

```
['zero', 'one', 'two', 'three']
```

```
index[:1]
```

```
['zero']
```

```
# index[:stop] where stop is (stop - 1)  - starts with 0 and ends at (stop - 1)
```

```
index[:2]
```

```
['zero', 'one']
```

```
index[2:5]
```

```
['two', 'three', 'four']
```

```
index[::1]
```

```
['zero', 'one', 'two', 'three', 'four', 'five']
```

```
index[0:5:2]
```

```
['zero', 'two', 'four']
```

- index - positive - accessing elements from the Left hand side of list
- index - negative - accessing elements from the righ hand side of list

```
index[-1]
```

```
'five'
```

```
index[-4:-1]
```

```
['two', 'three', 'four']
```

```
index[-4:]     #access upto last element
```

```
['two', 'three', 'four', 'five']
```

```
# Display the list in reverse order
```

```
index[::-1]
```

```
['five', 'four', 'three', 'two', 'one', 'zero']
```

```
index[::-2]
```

```
['five', 'three', 'one']
```

```
index[-3:-1]
```

```
['three', 'four']
```

```
index[-5:-1:2]
```

```
['one', 'three']
```

```
del index[2:5]  # deleting multiple elements from the list
```

```
index[1:1] = ["two"]  # adding a new element at index 1 using slicing
```

```
index[2:2] = [100, 101, 102]  # adding multiple elements  using slicing
```

```
a = [1,3,1,1,4,5,6,7,7,7,7]
```

count

```
a.count(7) # frequency of an element in the list
```

```
4
```

reverse

```
a.reverse()  #revesrse list
```

```
a
```

```
[7, 7, 7, 7, 6, 5, 4, 1, 1, 3, 1]
```

sort

```
a.sort()   # sorts the values in ascending order
```

```
a
```

```
[1, 1, 1, 3, 4, 5, 6, 7, 7, 7, 7]
```

```
a.sort(reverse = True)  # sorts the values in descending order
```

```
a
```

```
[7, 7, 7, 7, 6, 5, 4, 3, 1, 1, 1]
```

```
x = [1,2,3]
y = x            # Deep Copy if any value changed in x that will be reflect in y also

#x[1] = 20
del x[1]

print("value of x: ", x)
print("value of y : ", y)
print("id of x :", id(x))
print("id of y :", id(y))
```

```
value of x:  [1, 3]
value of y :  [1, 3]
id of x : 139517529127616
id of y : 139517529127616
```

```
x = [1,2,3]
y = x.copy()
# Shallow Copy - the value change/modified in x will not be reflected in y

#x[1] = 20
del x[1]
```

```
for i in my_list:
   print(i)
```

```
50
50
110
2
3
sunil
pune
divya
4
5
sunil
sunil
```

```
if "sunil" in my_list:
  print("found item ")
else:
  print("item not found")
```

```
found item
```

min max and sum

```
l=[20,14,5,235,45,65,78,14,2,5,7]
```

```
max(l)
```

```
2
```

```
min(l)
```

```
2
```

```
sum(l)
```

```
490
```

Nested List

```
X=[1,2,3,['HI','SUNIL'],["ram","shita",[1,2,3]]]
```

```
X[3][1]     #access element
```

```
'SUNIL'
```

```
X[4][2][0]
```

```
1
```

```
y=X.copy()
```

```
y
```

```
[1, 2, 3, ['HI', 'SUNIL'], ['ram', 'shita', [1, 2, 3]]]
```

```
y=X
```

```
y
```

```
[1, 2, 3, ['HI', 'SUNIL'], ['ram', 'shita', [1, 2, 3]]]
```

```
matrix=[
      [0,0,0],
      [1,1,1],
      [2,2,2]
]
```

```
matrix
```

```
    [[0, 0, 0], [1, 1, 1], [2, 2, 2]]
```

```
matrix[0][2]
```

```
    0
```

```
for j in range(len(matrix)):
            print(matrix[j][0])
```

```
    0
    1
    2
```

```
for i in range(len(matrix)):
      for j in range(len(matrix)):
            print(matrix[i][j])
```

```
    0
    0
    0
    1
    1
    1
    2
    2
    2
```

convert list of word into string

join

```
word_list = ['allx', 'love', 'gfg', 'xit', 'is', 'best']
```

```
result_string = ' '.join(word_list)      #imp

for i in word_list:
  print(i)
```

```
print(result_string)
```

```
    allx
    love
    gfg
    xit
    is
    best
    allxlovegfgxitisbest
```

TUPLE

- Tuples - It is same as List but Tuple is immutable.Collection of multiple values. Tuples are versatile/ hetrogenous i.e. it can store various datatypes in single variable. Comma seperated values/elements.
- Syntax -(1,2)

```
a = 1,2,3,4,5
a
```

```
(1, 2, 3, 4, 5)
```

```
type(a)
```

```
tuple
```

```
a = (1,2,3,4,5)
a
```

```
(1, 2, 3, 4, 5)
```

```
type(a)
```

```
tuple
```

Tuples are Immutable

- We cannot add, delete or modify tuples

```
col = (1,1.4, -7, "A", True)
```

```
col[::-1]
```

```
(True, 'A', -7, 1.4, 1)
```

```
col.index(1.4)
```

```
1
```

```
freq = (2,3,1,1,1,15)
```

```
freq.count(1)
```

```
3
```

Nested Tuples

```
t = (1,2,3, ("A", "B"), 4 ,5)
```

```
t[3]
```

```
('A', 'B')
```

```
t[3][1]
```

```
'B'
```

```
x=[(1,2),(3,4),(5,6),(7,8)]
```

```
for a,b in x:
  print(a)
```

```
1
3
5
7
```

.....

Sets

- Collection of comma seperated values enclosed in curly braces
- Syntax - {value1, value2,.....valueN}
- ORDER NOT IMPORTANT AND DUPLICATE NOT ALLOWED
- Can not access the element using INDEX

```
A = {1,3,5,7,9,10}
B = {2,4,6,8, 10 }
```

```
A
```

```
{1, 3, 5, 7, 9, 10}
```

```
B
```

```
{2, 4, 6, 8, 10}
```

```
for i in A:
   print(i)
```

```
1
3
5
7
9
10
```

```
A.add(11)
```

```
A
```

```
{1, 3, 5, 7, 9, 10, 11}
```

```
A.union(B)
```

```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}
```

```
A.intersection(B)
```

```
{10}
```

```
A.difference(B)
```

```
{1, 3, 5, 7, 9, 11}
```

```
A.discard(9) # removes a element from the set
```

```
A
```

```
{1, 3, 5, 7, 10, 11}
```

```
my_list=[1,1,2,2,3,4,5,6,8]
```

```
x=set(my_list)       #convert list into set
```

```
x
```

```
{1, 2, 3, 4, 5, 6, 8}
```

```
# sets donot allow duplicate elements
# We cannot declare set inside a set
```

Dictionary

- collection of key value pairs

- Syntax - {key1:value1, key2:value2, key3:value3.....keyN:valueN}
- Dictionary are mutable
- Dictionary **donot support indexing**
- We can access values of the dictionary using the keys
- Duplicate key not allowed but duplicate value allowed.

```
d = {"name":"Vrushali", "age": 24, "city": "Pune"}
```

```
d
```

```
{'name': 'Vrushali', 'age': 24, 'city': 'Pune'}
```

```
len(d)
```

```
3
```

```
d['name']   # we can access the values using keys
```

```
'Vrushali'
```

```
d["name"]="sunil"     #update values
```

```
d
```

```
{'name': 'sunil', 'age': 24, 'city': 'Pune'}
```

```
d["mobile_no"]=7038426854            #add new key and value pair
```

```
d
```

```
{'name': 'sunil', 'age': 24, 'city': 'Pune', 'mobile_no': 7038426854}
```

```
d.items()
```

```
dict_items([('name', 'Vrushali'), ('age', 24), ('city', 'Pune')])
```

```
d.keys()
```

```
dict_keys(['name', 'age', 'city'])
```

```
d.values()
```

```
dict_values(['Vrushali', 24, 'Pune'])
```

```
d.pop("age")
```

```
24
```

```
d
```

```
{'name': 'Vrushali', 'city': 'Pune'}
```

```
del d["id"]  # deletes the key value pair
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-142-247ce595e0b2> in <cell line: 1>()
----> 1 del d["id"]  # deletes the key value pair

KeyError: 'id'
```

SEARCH STACK OVERFLOW

```
d
```

```
{'name': 'Vrushali', 'city': 'Pune', 'age': 24}
```

```
for i in d.keys():

  print(i)
```

```
name
age
city
```

```
a=[]
for i in d.keys():

  a.append(i)

print(a)
```

```
['name', 'age', 'city']
```

```
for i in d.values():
  print(i)
```

```
Vrushali
mumbai
24
```

```
for i in d.items():
  print(i)
```

```
('name', 'Vrushali')
('city', 'mumbai')
('age', 24)
```

```
d["name"][0:5]
```

```
'Vrush'
```

```
for k,v in  d.items():
  print(f"{k} = {v}")
```

```
name = Vrushali
age = 24
city = Pune
```

sorted

```
sorted(d.items())
```

```
[('age', 24), ('city', 'Pune'), ('name', 'Vrushali')]
```

```
sorted(d.keys())
```

```
['age', 'city', 'name']
```

```
sorted(d.values())   #value contain inr and strig then error
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-79-5b1f5d2187d5> in <cell line: 1>()
----> 1 sorted(d.values())

TypeError: '<' not supported between instances of 'int' and 'str'

SEARCH STACK OVERFLOW
```

```
sorted(d.items(), reverse=True)
```

```
[('name', 'Vrushali'), ('city', 'Pune'), ('age', 24)]
```

Nested dict

```
mydict={
    'child1':{
        'name':'raj',
        'year':'2011'
            },
    'child':{
        'name':'sunil',
        'year':'2020'
    }
}
```

```
print(mydict)
```

```
{'child1': {'name': 'raj', 'year': '2011'}, 'child': {'name': 'sunil', 'year': '2020'}}
```

```
mydict['child1']['name']          #access the nested element
```

```
'raj'
```

Convert dict into list

```
ds = {"name":"Vrushali", "age": 24, "city": "Pune"}
```

```
list(ds.items())
```

```
[('name', 'Vrushali'), ('age', 24), ('city', 'Pune')]
```

Example

```
transactions=[{'items':'wedget','type':'sale','quantity':10},
              {'items':'wedget','type':'sale','quantity':13},
              {'items':'wedget','type':'refund','quantity':2},
              {'items':'licence','type':'sale','quantity':50},
              {'items':'licence','type':'sale','quantity':10},
              {'items':'licence','type':'refund','quantity':4}

              ]
```

```
transactions
```

```
[{'items': 'wedget', 'type': 'sale', 'quantity': 10},
 {'items': 'wedget', 'type': 'sale', 'quantity': 13},
 {'items': 'wedget', 'type': 'refund', 'quantity': 2},
 {'items': 'licence', 'type': 'sale', 'quantity': 50},
 {'items': 'licence', 'type': 'sale', 'quantity': 10},
 {'items': 'licence', 'type': 'refund', 'quantity': 4}]
```

```
transactions[1]['items']
```

```
'wedget'
```

```
total_sold={}

for tras in transactions:
      item=tras["items"]
      is_Sale=True if tras['type']== 'sale' else False
      quantity=tras['quantity']




      if item in total_sold:
         total_sold[item]+=quantity
      else:
        total_sold[item]=quantity

print(total_sold)
```

```
{'wedget': 25, 'licence': 64}
```

```
net_Sale={}

for tras in transactions:
      item=tras["items"]
      is_Sale=True if tras['type']== 'sale' else False
      quantity=tras['quantity']


      if not is_Sale:
        quantity=-quantity

      if item in net_Sale:
         net_Sale[item]+=quantity
      else:
        net_Sale[item]=quantity

print(net_Sale)
```

```
{'wedget': 21, 'licence': 56}
```

Merge two dict

```
d1={"a":1,"b":2}

d2={"c":3,"d":4}

d1.update(d2)     #imp


d1
```

```
{'a': 1, 'b': 2, 'c': 3, 'd': 4}
```

dictionary comprehension

```
sale={
    "a":10,
    "b":20,
    "c":30,
    "d":40

}

{key for key,value in sale.items() if value>30}
```

```
{'d'}
```

```
square_dict = {x: x**2   for x in range(1, 6)}


print(square_dict)
```

```
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

Operators

- Arithmetics Operators

```
+, -, *, /, % (modulous), // (floor), ** (exponential operator)
```

- Logical Operators

```
AND, OR, NOT
```

- Comparison Operators

```
==, > , < , >= , <= , != (not equal to)
```

- Membership operator - to check whether an element belongs to the objet(list, dict..etc)

```
in, not in
```

- Identical Operator - checks whether both the object belongs to the same location or not

```
is, not is
```

- Bitwise Operator (& , ! , |)
- Boolean Operator

```
True , False
```

Arathmatic operators

- Addition (+)- Subtraction (-)
- Product (*)
- Division (/)
- Modulus (%)
- Flooor(//)

```
3%2
```

```
1
```

```
1025//10
```

```
102
```

```
-72//10
```

```
-8
```

Assignment operators

- = - Assign the right value to the left. ex. x = 2
- += - Adds the right side value to the left. ex. x += 2

- -= - Subtract the right side value from the left side value. ex. x-=2
- *= - Product the right side value with the left side value. ex. x*=2
- /= - Divide the left side value with right side value. ex. x /= 2
- %= - Mod the left side value with the right side value. ex. x %= 2

```
N=10  #assign
N==45  #compare
```

```
False
```

Comparison Operators

- == equals to
- < less than ex. 2<3 - True
- > greater than ex. 4>1 - True
- <= less than equals to ex. age <= 16
- >= greater than equals to ex. age >= 18
- != not equals to ex. 2 != 3 - True

Logical operators

- AND &
- OR |
- NOT !

Membership operators

- in - Returns True if the specified element is the member of specified group.
- NOT in - Returns True if the specified element is not the member of specified group

Identity operators

- is - Returns True if the identity of both elements matches.
- is not - Returns True if the identity of both elements does not matches.

```
#The identity operators compare the memory locations of two object
```

```
a=10
b=10

a is b
```

```
True
```

```
x=15
y=10

x is y
```

```
False
```

```
h=10
y=10.0

h==y    # == compare value
```

```
True
```

```
h is y  # is   compare object
```

```
False
```

Conditional statements

If-else

'if' else condition is used to make the desicion as per the specified condition.
If the condition is true, 'if' block will get executed and if the condition is false, 'else' block will get executed.

Syntax:

```
if condition:
    Execute code           #(when if condition is True)
else:
    Execute code           #(when if condition is False)
```

```
price=200

if price >300:
  print("greater then 300")
else:
  print("less than 300")
```

```
less than 300
```

```
if 2 != 2:
  print('hi')
else:
  print('bye')
```

```
bye
```

Q1. Write a program to check if the user is eligible to vote or not.

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

if age >= 18:
  print("You can vote")
else:
  print("You can't vote")
```

```
Enter your age = 15
You can't vote
```

Q2. Write a program to check if the number is even or odd.

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

if num % 2 == 0:
  print("even")
else:
  print("odd")
```

```
Enter the number = 45
odd
```

Q3. Practice

```
for i in range(1,20):
  if i%2==0 and i<=10:
    print(i)
    if i==10:
      break
  else:
    print(f"{i} is odd number")
```

```
1 is odd number
2
3 is odd number
4
5 is odd number
6
7 is odd number
8
9 is odd number
10
```

Q4. Write a program to tell if the student is pass or fail.

Take the marks (out of 120) from user and print if he is pass or fail.

```
>=35% - Pass
<35% - Fail
```

```
marks = 30

percent = (marks*100)/120
print(percent)

if percent >= 35:
  print("Pass")
else:
  print("Fail")
```

```
25.0
Fail
```

Q5. Write a program to tell if the student is pass or fail.

Take the marks of 5 subjects from student. Each paper was of 120 marks. Tell the student if he is pass or fail.

```
>=35% - Pass
<35% - Fail
```

```
s1 = 12
s2 = 65
s3 = 12
s4 = 12
s5 = 66

marks = s1 + s2 + s3 + s4 + s5
total_marks = 120 * 5

if (marks*100)/total_marks >= 35:
  print("Pass")
else:
  print("Fail")
```

```
Fail
```

if-elif

```
Syntax:

if condition:
    some code if true
elif condition2:
    some code if true
else:
    some code if both conditions get false
```

```
a=10
b=25

if a>b:
  print("a is greater than b")
elif a==b:
  print('a is equal to b')
else:
  print("b is greater than a")
```

```
b is greater than a
```

```
try :
    a=10
    b=2
    c=3

    if a>b and a>c:
      print(a,'is greater than b and c')
    elif b>a and b>c:
      print(b,'is greater than a add c')
    else:
      print(c,'is greater than a and b')
except:
  print('enter only numer')
```

```
10 is greater than b and c
```

```
name = ''

if name == 'Treat':
    print("Treat le k aa!!!!")
elif name == 'Oreo':
    print("Oreo le k aa!!")
elif name == 'Dark Fantisy':
    print("Fantasy le k aa!")
else:
    print("Bounce le k aaja :(")
```

```
Bounce le k aaja :(
```

Q6. Write a python program to calculate the total onroad price of a bike.

```
price <= 100000 -> 10% tax
price <= 200000 -> 15% tax
price <= 300000 -> 20% tax
price > 300000 -> 30% tax

insurence of 10000 should be added seperately.
RTO processing fee 8000.
Expected output:

Price = 100000 rs
Tax = 10% (10000 rs)
```

```
Insurence = 10000 rs
RTO fee = 8000 rs
Onroad Price = 128000 rs
```

```
bp = int(input("Enter bike price = "))
tax = 0

if bp <= 100000:
    tax = 10
elif bp <= 200000:
    tax = 15
elif bp <= 300000:
    tax = 20
else:
    tax = 30

on_road_price = bp + ((tax*bp)/100) + 10000 + 8000

print(f"Price = {bp} rs")
print(f"Tax = {tax}% ({((tax*bp)/100)} rs))")
print("Insurence = 10000 rs")
print("RTO fee = 8000 rs")
print(f"Onroad Price = {on_road_price} rs")
```

```
Enter bike price = 50000
Price = 50000 rs
Tax = 10% (5000.0 rs))
Insurence = 10000 rs
RTO fee = 8000 rs
Onroad Price = 73000.0 rs
```

Nested if

```
Syntax:

if condition:
    if condition:
        some code if true
    else:
        some code if false
else:
    if condition:
        some code if true
    else:
        some code if false
```

```
x = 10

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

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

```
x is positive
x is even
```

```
name = 'Treat'
flavour = 'Mango'

if name == 'Treat':
    if flavour == 'Mango':
        print("Mango wala treat le k aa!!!!!")
    else:
        print("Jo honga vo flavour ka Treat la")
else:

    print("jo mile vo leke aa: ")
```

```
jo mile vo leke aa:(
```

nested if-else

Q7. Write a program to get greatest of 3 numbers.

a = 4
b = 2
c = 8
Expected output:

8 is greater

```
a = 14
b = 9
c = 8

if a > b:
    if c > a:
        print('c is greater')
    else:
        print('a is greater')
else:
    if c > b:
        print('c is greater')
    else:
        print('b is greater')
```

```
price=10

if price<1000:
  if price<500:
    print("50 quantity want")
  else:
     print("only 10 quantity want")
else:
  print("Price is too high..")
```

```
50 quantity want
```

```
account_enabled=True
balance=1000
withdraw=2000

if account_enabled and withdraw<=balance:
  print("authorized")
else:
  if not account_enabled:
    print("account disabled")
  else:
    print("insufficient fund")
```

```
insufficient fund
```

if-elif-else

```
grade=int(input("Enter grade :"))

if grade>=90:
  print("A++")
elif grade>=80:
  print("A")
elif grade>=70:
  print("B")
elif grade>35:
  print("C")
else:
  print(F)
```

```
Enter grade :56
C
```

Q8. Write a program to calculate the electricity bill as per units.

```
first 10 units -> free from government
<=50 -> 2rs/unit
>50 and <=100 -> 3rs/unit
>100 and <=150 -> 4rs/unit
>150 and <=200 -> 5.5rs/unit
>200 -> 7rs/unit
Expected output:

Units used = 160
Your bill is = 485
```

```
units = int(input("Enter units = "))
amount = 0
if units <= 50:
    amount = (units - 10)*2
elif units <= 100:
    amount = 80 + ((units-50)*3)
elif units <= 150:
    amount = 230 + ((units - 100)*4)
elif units <= 200:
    amount = 430 + ((units - 150)*5.5)
else:
    amount = 705 + ((units - 200)*7)

print(f"Units used = {units}")
print(f"Your bill is = {amount}")
```

```
Enter units = 55
Units used = 55
Your bill is = 95
```

Loops

- Loops are used to run the same set of code multiple times till the condition gets satisfied.

for loop Syntax: FOR loop is used for iterating over a sequences such as lists, tuples, strings etc.

```
for i in series:
    some code
```

```
list(range(5))
```

```
[0, 1, 2, 3, 4]
```

```
tuple(range(5))
```

```
(0, 1, 2, 3, 4)
```

```
my=['a','b']

for i in my:
  y=i+i
  print(y)
```

```
aa
bb
```

```
for i in range(1, 11):   #range
    print(i)
```

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

```
for i in "Hello":
    print(i)
```

```
H
e
l
l
o
```

Q1. Write a program to print the table of 2.

```
n = 2
for i in range(1, 11):
    print(n*i)
```

```
2
4
6
8
10
12
14
16
18
20
```

Q2. Write a program to print the tables from 1 to 5

```
for i in range(1, 3):
    for j in range(1, 11):
        print(i*j)
```

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

```
6
8
10
12
14
16
18
20
```

Q3. Write a program to print the table of users choice.

```
n = int(input("Enter the number = "))

for i in range(1, 11):
    print(n*i)
```

```
Enter the number = 45
45
90
135
180
225
270
315
360
405
450
```

Q4. Write a program to print the table of users choice till the multiple of users choice.

```
n = int(input("Enter the number = "))
m = int(input("Enter the multiple = "))

for i in range(1, m+1):
    print(n*i)
```

```
Enter the number = 5
Enter the multiple = 2
5
10
```

Q5. Write a program to print the table from users choice to users choice till the multiple of users choice.

```
f = int(input("Enter from table = "))
t = int(input("Enter to table = "))
m = int(input("Enter multiple = "))

for i in range(f, t+1):
    for j in range(1, m+1):
        print(i*j)
```

```
Enter from table = 5
Enter to table = 6
Enter multiple = 2
5
10
6
12
```

Q6. Write a program to print all the even numbers from 1 to 100.

```
for i in range(1, 50):
    if i % 2 == 0:
        print(i)
```

```
2
4
6
```

```
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40
42
44
46
48
```

Q7. Write a program to print the tables of all odd numbers from 1 to 3.

```
for i in range(1, 4):
    if i % 2 != 0:
        for j in range(1, 11):
            print(i*j)
```

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

```
a=[
    [1,24],
    [45,65,85],
    [35,45,68],
    [85,6,1]
]
```

```
for i in a:
  for e in i:
    print(e)
```

```
1
24
45
65
85
35
45
68
85
6
1
```

enumerate

- where the first element in the pair is the index, and the second element is the value from the iterable.

- enumerate is a convenient tool when you need both the index and the value while iterating over a sequence.

```
a=["sunil","mi","forever","ish"]

for i in enumerate(a):
  print(i)
```

```
(0, 'sunil')
(1, 'mi')
(2, 'forever')
(3, 'ish')
```

While loop

- With the help of while loop we can execute a set of statement as long as a condition is true

```
Syntax:


while condition:
    some code
```

```
i = 1
while i < 5:
    print(i)
    i += 1
```

```
1
2
3
4
```

```
i=10
while i >5:
  print(i)
  i=i-1
```

```
10
9
8
7
6
```

```
while True:
    a = input("Enter fav movie name = ")
    if a == 'exit':
        print("Bye")
        break
    else:
        print(a)
```

```
Enter fav movie name = radhe
radhe
Enter fav movie name = exit
Bye
```

```
min_lenth=4

while True:
    name=input("Enter name :")
    if (len(name)>min_lenth) and (name.isalpha()):
        break
print(name)
```

```
Enter name :sunl655
Enter name :sun
Enter name :sunil
sunil
```

```
#continue

for num in range(1,11):
  if num%2==0:
    continue
  else:
    print(num)
```

```
1
3
5
7
9
```

```
#pass :if condition is true and when we are not writing the body.then we use the pass

if "a" in "aaa":
  pass
```

```
i=1                          #break we can stop the loop when if condition is true
while i<10:
  print(i)
  if i==5:
    break       #break
  i=i+1
```

```
1
2
3
4
5
```

```
#Continue :we can skip the current iteration and continue with the next

i=0
while i<10:
  i=i+1
  if i==4:
    continue    #continue
  print(i)
```

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

```
#continue

for num in range(1,11):
  if num%2==0:
    continue
  else:
    print(num)
```

```
1
3
5
7
9
```

Q8. Write a program using while loop to print the table of 2.

```
i = 1
while i<11:
    print(i*2)
    i+=1
```

```
2
4
6
8
10
12
14
16
18
20
```

Q9. Write a program using while loop to print the tables from 1 to 10.

```
i = 1
while i<3:
    j = 1
    while j<11:
        print(i*j)
        j += 1
    i+=1
```

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

Q10. Take the user input and print it till the user enters the odd number.

```
# by using break
while True:
    a = int(input("Enter the number = "))
    if a % 2 == 0:
        print(a)
    else:
        print("bye")
        break
```

```
Enter the number = 12
12
Enter the number = 4
4
Enter the number = 5
bye
```

```
# without using break

status = True
while status:
    a = int(input("Enter the number = "))
    if a % 2 == 0:
        print(a)
    else:
        print("bye")
        status = False
```

```
Enter the number = 14
14
Enter the number = 54
54
Enter the number = 5
bye
```

```
#.....................................................................................................
```

solve using any loop

```
#Q11 write a program to below pattern

i=1
while i<=5:
  j=1
  while j<=i:
    print("*",end=" ")
    j=j+1
  print()
  i=i+1
```

```
*
* *
* * *
* * * *
* * * * *
```

```
i=1
while i<=5:
  j=1
  while j<=i:
    print(i,end=" ")
    j=j+1
  print()
  i=i+1
```

```
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
```

```
i=1
while i<=5:
  j=1
  while j<=i:
    print(i,end=" ")
    j=j+1
  print()
  i=i+1
```

```
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
```

```
i=5
while i>0:
   j=1
   while j<=i:
    print('*',end='')

    j=j+1
   print()
   i=i-1
```

```
*****
****
***
**
*
```

2D List

```
a = [[1, 2, 3, 4, 5, 6, 7, 8, 9, 10],[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]]

print(a)
```

```
[[1, 2, 3, 4, 5, 6, 7, 8, 9, 10], [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]]
```

```
print(a[1][6])
```

```
14
```

```
v
```

```
b = [[],[],[]]

b[0].append(1)
print(b)
```

```
[[1], [], []]
```

```
b.append(2)
print(b)
```

```
[[1], [], [], 2]
```

```
b.append([2, 3])
print(b)
```

```
[[1], [], [], 2, [2, 3]]
```

Q1. Write a program to create a list of table from 1 to 10.

```
tables = []
for i in range(1, 11):
    temp = []
    for j in range(1, 11):
        temp.append(i*j)
    tables.append(temp)

for i in tables:
    print(i)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
[3, 6, 9, 12, 15, 18, 21, 24, 27, 30]
[4, 8, 12, 16, 20, 24, 28, 32, 36, 40]
[5, 10, 15, 20, 25, 30, 35, 40, 45, 50]
[6, 12, 18, 24, 30, 36, 42, 48, 54, 60]
[7, 14, 21, 28, 35, 42, 49, 56, 63, 70]
[8, 16, 24, 32, 40, 48, 56, 64, 72, 80]
```

```
[9, 18, 27, 36, 45, 54, 63, 72, 81, 90]
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
```

Iterable and Iterators

- an Iterable is something that can be iterated over like Tuple, List, Dict
- Iterator : is iterate the next element

List Comprihension

- A list comprehension is used to generate a list object

```
# x=[expression for loop  if condition]

a = [x for x in range(1, 11)]

print(a)
```

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

```
a = [x**2 for x in range(1, 11)]
print(a)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

```
a = [x-1 for x in range(1, 11)]
print(a)
```

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

```
country=['india','indo','japan','jermany','dubai','pakistan']
x=[a.upper() for a in country if a.startswith('p')]

print(x)
```

```
['PAKISTAN']
```

```
x=[("HI") for a in range(1,10) if a%2==0 ]

x=[ "even" if a%2==0 else "odd" for a in range(1,11)]          #use if else at start

print(x)
```

```
['odd', 'even', 'odd', 'even', 'odd', 'even', 'odd', 'even', 'odd', 'even']
```

```
v=[('Sunil') if i%2==0 else ('Divya') for i in range(1,10) ]


print(v)
```

```
['Divya', 'Sunil', 'Divya', 'Sunil', 'Divya', 'Sunil', 'Divya', 'Sunil', 'Divya']
```

```
c=[a for a in range(1,10)]

print(c)
```

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

```
[[i,j] for i in range(5) for j in range(10) if j%2==0 if j%4==0]
```

```
[[0, 0],
 [0, 4],
 [0, 8],
 [1, 0],
 [1, 4],
```

```
 [1, 8],
 [2, 0],
 [2, 4],
 [2, 8],
 [3, 0],
 [3, 4],
 [3, 8],
 [4, 0],
 [4, 4],
 [4, 8]]
```

```
num=(1,2,3,4,5,6)   #Tuple
```

```
sq=[i*2 for i in num]
```

```
sq
```

```
[2, 4, 6, 8, 10, 12]
```

```
sen="Python is awesome language".split(" ")
```

```
filter=[item for item in sen if len(item)>5 ]
```

```
filter
```

```
['Python', 'awesome', 'language']
```

Q1. write a program to square all the elements of given list.

```
a = [2, 4, 7, 8, 3, 6]
```

```
a = [x**2 for x in a]
```

```
print(a)
```

```
[4, 16, 49, 64, 9, 36]
```

Q2. Write a program to filter all the even numbers from the given list.

```
a = [2, 4, 7, 8, 3, 6]
```

```
b = [x for x in a if x % 2 == 0]
print(b)
```

```
[2, 4, 8, 6]
```

Q3. Write a program to square all the even number in the list.

```
a=[i**2 for i in range(1,4)]
```

```
print(a)
```

```
[1, 4, 9]
```

```
c=[i for i in range(10)]
```

```
print(c)
```

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

```
numbers = range(1, 11)
result = [x**2 if x % 2 == 0 else x**3 for x in numbers]
print(result)
```

```
[1, 4, 27, 16, 125, 36, 343, 64, 729, 100]
```

```
a=[2,4,8,9]

c=[i**2 for i in a]

print(c)
```

```
[4, 16, 64, 81]
```

Type Casting

```
#convert string to int

a='3'

b=int(a)

print(b)
type(b)
```

```
3
int
```

```
#int to string

a=2

b=str(a)

print(b)
type(b)
```

```
2
str
```

```
#int to float
a=10

b=float(a)

print(b)
type(b)
```

```
10.0
float
```

```
#list to tuple

l=[1,2,3]

t=tuple(l)

type(t)
```

```
tuple
```

```
#tuple to list

t=(1,2,3)

l=list(t)

type(l)
```

```
list
```

```
#list to set

l=[1,2,3]

s=set(l)

type(s)
```

```
set
```

STRING FUNCTION

```
s="Hi i know python spark sql hive sqoop"
```

upper()

```
s.upper()
```

```
'HI I KNOW PYTHON SPARK SQL HIVE SQOOP'
```

lower()

```
s.lower()
```

```
'hi i know python spark sql hive sqoop'
```

swapcase()

```
s.swapcase()
```

```
'hI I KNOW PYTHON SPARK SQL HIVE SQOOP'
```

title()

```
s.title()
```

```
'Hi I Know Python Spark Sql Hive Sqoop'
```

capitalize()

```
s.capitalize()
```

```
'Hi i know python spark sql hive sqoop'
```

Case in sensetive Comparision

```
#abc==ABC


"ABC"=="abc"
```

```
False
```

```
s="SUNIL"
c="sunil"

s.casefold()==c.casefold()    #for case in-sensitive comparision
```

```
True
```

split()

```
s.split()  #output in list
```

```
['Hi', 'i', 'know', 'python', 'spark', 'sql', 'hive', 'sqoop']
```

replace

```
s.replace("Hi","DATA engineer")
```

```
'DATA engineer i know python spark sql hive sqoop'
```

```
a="hi you are \"sunil rathod\""

a
```

```
'hi you are "sunil rathod"'
```

```
s="  sunil rathod  "
```

strip

```
s.strip()  #strip  white spaces
```

```
'sunil rathod'
```

```
s.rstrip()
```

```
'  sunil rathod'
```

```
s.lstrip()
```

```
'sunil rathod  '
```

```
s.startswith("s")
```

```
False
```

```
s.startswith(" ")
```

```
True
```

```
s.endswith(" ")
```

```
True
```

remove specific word,character or symbol at star oe end of string

```
a="##sunil.rathod@gmail.com##"

a.strip('#')
```

```
'sunil.rathod@gmail.com'
```

Validation

```
s.isupper()
```

```
False
```

```
s.islower()
```

```
True
```

```
s.isdigit()
```

```
False
```

```
s.isalpha()
```

```
False
```

Join

```
a=["sunil",".","rathod","@","gmail",".","com"]

''.join(a)
```

```
'sunil.rathod@gmail.com'
```

```
a=" hii i am sunil rathod i am from pune"

",".join(a)   #without using split string
```

```
' ,h,i,i, ,i, ,a,m, ,s,u,n,i,l, ,r,a,t,h,o,d, ,i, ,a,m, ,f,r,o,m, ,p,u,n,e'
```

```
a=" hii i am sunil rathod i am from pune"
b=a.split()


c="_".join(b)

print(c)
```

```
hii_i_am_sunil_rathod_i_am_from_pune
```

```
a = "hii i am sunil rathod i am from pune"
b = a.split()

b = [word.upper() for word in b]

c = "_".join(b)

print(c)
```

```
HII_I_AM_SUNIL_RATHOD_I_AM_FROM_PUNE
```

```
sen="Rules for defining the variable"
```

```
len(sen)
```

```
31
```

```
sen[len(sen)-1]    #last character
```

```
'e'
```

```
sen[len(sen)]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-42-05e9fa428be5> in <cell line: 1>()
----> 1 sen[len(sen)]

IndexError: string index out of range

SEARCH STACK OVERFLOW
```

```
sen[::-1]
```

```
'elbairav eht gninifed rof seluR'
```

in

```
validation=input("enter word :")
a="sunil.rathod@gmail.com"


if validation in a:
     print(validation)
else:
  print("word not avaiable")
```

```
enter word :com
com
```

```
b=a.split()
for i in b:
  if 'gmail' in i:
    print("gmail")
  else:
    print(i)
```

```
gmail
```

Function

- A function is a reusable block of code that is used to perform a specific operation
- 1)Build in function print(),sum(),type()..etc
- 2)user define function
- 3)Anonymous Function
- Function are the set of code written to achieve the specific output.

- def Funtion_name():

```
print()
return
yeild
```

```
def sample(a):     #parameter
  print(a)

sample(10)          #argumnet
```

```
    10
```

```
def is_even(x):
  if x%2==0:
    return True          #when we return any value,then we can save it in variable
  else:
    return False


save_output=is_even(11)     #we can save return value in variable


save_output
```

```
    False
```

```
.................
```

```
def find_max(a,b,c):
  max=a
  if b>max:
    max=b
  if c>max:
    max=c
  return f"max value = {max}"

find_max(20,45,35)
```

```
    'max value = 45'
```

```
#.............

def find_avg(*value):

    return sum(value) /len(value)


find_avg(10,20)
```

```
    15.0
```

```
def calculate_average(numbers):

    if not numbers:
        return 0.0

    total = sum(numbers)
    average = total / len(numbers)
    return average


numbers_list = [10,30]
result = calculate_average(numbers_list)

print(f"Average: {result}")
```

```
    Average: 20.0
```

```
.....................
```

```
#nested Function


def fun1(a):
    print(a)
    def fun2(b):
      c=10
      print(f"a={a}, b={b}, c={c}")

    fun2(3)


fun1(5)
```

```
    5
    a=5, b=3, c=10
```

```
.....
```

```
#Write a function to swap the value of two variable

def swap():
  a=int(input("Enter number 1 :"))
  b=int(input("Enter number 2 :"))

  print('before swap')


  print(f'a={a}')
  print(f'b={b}')

  a=b
  b=a

  print('after swap')
  print(f'a={a}')
  print(f'b={b}')


swap()
```

```
    Enter number 1 :5
    Enter number 2 :4
    before swap
    a=5
    b=4
    after swap
    a=4
    b=4
```

```
3
```

```
    3
```

```
def add(a,b):
  return a+b


add(4,8)
```

```
    12
```

```
def is_even(a):        #a is parameter
  if a%2==0:
    return 'even'
  else:
    return 'odd'

is_even(10)              #10 is argument
```

```
    'even'
```

```
s="sunil rathod and divya g are best friends"


from posixpath import split
def findword(s):
  return 'divya' in s


findword(s)
```

```
True
```

```
#count word
s='hi hi hi sunil sunil i i am sunil'
def countword(s):
  count=0
  for word in s.lower().split():
    if word=="sunil":
       count=count+1
  return count


countword(s)
```

```
3
```

Types of argument

- default
- positional
- keyword

```
def power(a=1,b=1):     #default argument
    return a**b

power(2,3)
```

```
8
```

```
def power(a,b):
  return a**b

power(10,12)                    #postional argument
```

```
1000000000000
```

```
def power(a,b):
  return a**b

power(b=3,a=2)            #keyword argument
```

```
8
```

```
#Function with return

def add(a,b):
  c=a+b
  return c           #return single value


add(10,45)
```

```
55
```

```
def multi(a,b):
  c=a+b
  d=a*b
  return c,d          #return multiple value


c,d=multi(10,5)



c
```

```
    15
```

```
d
```

```
    50
```

args and kwargs

```
#args receive unlimited number of input from user


def multiple(*args):      #use when we don't know the number of input come,store value in tuple
  product=1
  for i in args:
    product=product*i
  return product

multiple(1,2,3,4,5)
```

```
    120
```

```
def b(*s):
  return sum(s)

b(1,2,4,5,4,8,9,7,8,9)
```

```
    57
```

kwargs value store in key:value pair

```
def display(**dis):                       #value store in key and value
    for (key,value) in dis.items():
      print(key,'=',value)


display(id='1',name='sunil')
```

```
    id = 1
    name = sunil
```

Nested function

```
def f():
  print("f")
  def g():
    print("g")
  g()


f()
```

```
    f
    g


d=add(4,5)       #return single value

d+5
```

```
    14
```

```
def operation(a,b):
  c=a+b
  d=a-b
  return c,d          #return multiple value

f,s=operation(4,5)      #return multiple value

print(f)
```

```
    9
```

```
print(s)
```

```
    -1
```

```
#arbitrary argument

def myf(*ab):
  print(ab)
  return ab[1]

myf('sunil','rathod','sunny')
```

```
    ('sunil', 'rathod', 'sunny')
    'rathod'
```

MAKE CALCULATOR USING FUNCTION

```
def calculator(x,y,operation):

    if operation=='+':
      return x+y
    elif operation=='-':
      return x-y
    elif operation=='*':
      return x*y
    elif operation=='/':
      return x/y


try:
  x=int(input('Enter number 1 : '))
  y=int(input('Enter number 2 : '))
  operation=input('Enter operation :')

  output=calculator(x,y,operation)
  print(output)
except Exception as e :
    print(f"please check error : {e}")
    print('Enter correct input')
```

```
    Enter number 1 : 45
    Enter number 2 : 0
    Enter operation :/
    please check error : division by zero
    Enter correct input
```

Lambda *Function*

```
#A lambda function is small anonymous function,defined without name and without using def keyword
#lambda function can take any number of argument,but can have one expression.
#lambda Function used with Higher Order Function.

lambda arguments:expression

lambda a,b:a+b
```

```
<function __main__.<lambda>(a, b)>
```

```
add=lambda a,b:a+b

add(5,6)
```

```
11
```

```
x=lambda a:a**2

print(x(8))
```

```
64
```

```
#odd or even

x=lambda x:'even' if x%2==0 else 'odd'

x(15)
```

```
'odd'
```

```
#Imp topic in lambda function: Map(),Filter(),reduce()

#  * define multiple argument

c=lambda *a:sum(a)

c(10,45,78,98,78)
```

```
309
```

```
def my_fun(x):               #lambda within user defined function
  return (lambda y:x*y)

z=my_fun(4)

print(z(2))
```

```
8
```

```
#MAP

#apply operation using map function on each element in sequences.


num=[4,5,3,6]

doulbe=list(map(lambda x:x+x,num))

doulbe
```

```
[8, 10, 6, 12]
```

```
strings=["python","sunil","rathod","data"]


capital=list(map(lambda x:x.upper(),strings))

capital
```

```
['PYTHON', 'SUNIL', 'RATHOD', 'DATA']
```

```
l=[45,1,2,4,5,6,66]

l.sort()
l
```

```
[1, 2, 4, 5, 6, 45, 66]
```

```
attendance=[("5A",85),("6A",45),("7A",75),("8A",95),("9A",47),("10A",65)]

sorted(attendance)
```

```
[('10A', 65), ('5A', 85), ('6A', 45), ('7A', 75), ('8A', 95), ('9A', 47)]
```

```
sorted(attendance, key=lambda x:x[1])        #vvimp
```

```
[('6A', 45), ('9A', 47), ('10A', 65), ('7A', 75), ('5A', 85), ('8A', 95)]
```

```
#Filter ..........................................................................

h=[45,75,7,88,99,65,44,12]

fil=list(filter(lambda x:x>40,h))

fil
```

```
[45, 75, 88, 99, 65, 44]
```

```
country=["india","indo","japan","pakistan","america","Us","srilanka"]

fil=list(filter(lambda x:len(x)>3,country))

fil
```

```
['india', 'indo', 'japan', 'pakistan', 'america', 'srilanka']
```

```
#Reduce ..........................................................................

from functools import reduce

num=[45,78,98,78,75]

sum=reduce(lambda x,y:x+y,num)

sum
```

```
374
```

```
max_value=reduce(lambda x,y:max(x,y),num)
max_value
```

```
98
```

```
min_value=reduce(lambda x,y:min(x,y),num)
min_value
```

```
45
```

```
maxe=reduce(lambda x,y: x if x>y else y,num)

maxe
```

```
98
```

```
score=[[1,35,42],[2,32,75],[3,30,82],[4,33,75],[5,35,60]]
abv_attendance=34


new_marks=map(lambda x: x[2]+2 if x[1]>abv_attendance  else x[2]-2,score)

list(new_marks)
```

```
[44, 73, 80, 73, 62]
```

```
sales=[{"country":"india","sale":150},{"country":"pak","sale":10},{"country":"japan","sale":500},{"country":"inc


country_key=map(lambda x:x["country"],sales)

list(country_key)
```

```
['india', 'pak', 'japan', 'indo']
```

```
value=map(lambda x:x["sale"],sales)
list(value)
```

```
[150, 10, 500, 47]
```

```
fil=filter(lambda x:x["country"]=="india",sales)

list(fil)
```

```
[{'country': 'india', 'sale': 150}]
```

```
high_sale=filter(lambda x:x["sale"]>200,sales)
list(high_sale)
```

```
[{'country': 'japan', 'sale': 500}]
```

Map(lambda,series)

- The map function in Python is used to apply a given function to each item in an iterable (such as a list, tuple, or other sequence) and return a new iterable with the results.

```
mylist=[2,4,6,8,10,11,13]

z=list(map(lambda x:x**2 if x%2==0 else x,mylist))

print(z)
```

```
[4, 16, 36, 64, 100, 11, 13]
```

```
mylist=[2,4,6,8,10,11,13]

z=list(map(lambda x:x**2 if x>=10 else x/2,mylist))

z
```

```
[1.0, 2.0, 3.0, 4.0, 100, 121, 169]
```

Filter(lambda,series)

```
# Example: Filtering students with scores above 70
students = [
    {"name": "Alice", "score": 85},
    {"name": "Bob", "score": 60},
    {"name": "Charlie", "score": 75}
]

passed_students = list(filter(lambda student: student["score"] > 70, students))


passed_students
```

```
[{'name': 'Alice', 'score': 85}, {'name': 'Charlie', 'score': 75}]
```

…

```
my_list=[1,2,3,4,5,6,7,8]

new_list=list(filter(lambda a:(a%2==0),my_list ))        #filter

print(new_list)
```

```
[2, 4, 6, 8]
```

```
x=list(filter(lambda x:x%2==0,mylist))

print(x)
```

```
[2, 4, 6, 8, 10]
```

```
h=[2,6,8,9,7,5,3,11,12,13,14,15]

print(list(map(lambda  x:x**2,filter(lambda x:x%2==0,h))))
```

```
[4, 36, 64, 144, 196]
```

```
print(list(map(lambda  x:x**2,filter(lambda x:x%2==0,h))))
```

```
[4, 36, 64, 144, 196]
```

Reduce function()

```
from IPython.core.interactiveshell import functools
from functools import reduce

#It will make the sum of element from list.

c=[1,2,3,4,5]

add=reduce(lambda a,b:a+b,c)

add
```

```
15
```

```
a
```

```
10
```

```
mylist=[10,20,30,40]


from functools import reduce

z=reduce(lambda a,x:a+x,mylist,0)     # it will make the sum

print(z)
```

```
100
```

```
l1=[2,5,7,4,2,3]

l2=[3,4,6,1,6,9]


print(reduce(lambda a,x:a+x,l1)+reduce(lambda b,y:b+y,l2))
```

```
52
```

```
l1=[2,5,7,4,2,3]

l2=[3,4,6,1,6,9]


print(reduce(lambda a,x:a+x,l1+l2))
```

```
52
```

```
l1=[2,5,7,4,2,3]

l2=[3,4,6,1,6,9]

l1.extend(l2)


l1+l2
```

```
[2, 5, 7, 4, 2, 3, 3, 4, 6, 1, 6, 9, 3, 4, 6, 1, 6, 9]
```

```
a=[[2,5,3],
[3,4,6],
[1,6,9]]


b=reduce(lambda x,y:x+y,a)


d=reduce(lambda x,y:x+y,b)

d
```

```
39
```

```
a = [
    [2, 5, 3],
    [3, 4, 6],
    [1, 6, 9]
]


column_sums = [sum(column) for column in zip(*a)]

print(column_sums)
```

```
    [6, 15, 18]
```

```
a = [
    [2, 5, 3],
    [3, 4, 6],
    [1, 6, 9]
]


print(list(map(lambda x:reduce(lambda x,y:x+y,x,0),a)))
```

```
    [10, 13, 16]
```

```
# Method 1: Using str.replace()

input_string = "This, is, a, string, with, commas"
output_string = input_string.replace(",", "")

sep=output_string.split()

#print(output_string)

for i in sep:
  print(i)
```

```
    This
    is
    a
    string
    with
    commas
```

Decorator

- Decorator is a concept in which we can add the external functionly to our function without changing it's source code.
- A function is an instance of the Object type.<>
- You can store the function in a variable.<>
- You can pass the function as a parameter to another function.<>
- You can return the function from a function.

```
def print_hi():

  print('hi')

store_function_as_variablr=print_hi

store_function_as_variablr()
```

```
    hi
```

Inner function

```
def a():
    print("hey")
    def b():
        print("hi")
    return b

z = a()
z()
```

```
    hey
    hi
```

```
#
```

```
from types import WrapperDescriptorType
#decorator
```

```
def securuty_code(func):
   def wrapper():
       print('pre security')
       func()
       print('post security')
   return wrapper
```

```
@securuty_code
def kyc():
  print('kyc')
```

```
kyc()
```

```
pre security
kyc
post security
```

```
....
```

```
import time
```

```
def time_cal(func):
    def wrapper():
        start = time.time()
        func()
        end = time.time()
        print(end-start)
    return wrapper
```

```
@time_cal
def myloop():
    sum = 0
    for i in range(10):
        sum =sum+i
        print(sum)
```

```
myloop()
```

```
0
1
3
6
10
15
21
28
36
45
0.0002391338348388672
```

Generator

```
l=[1,2,3]
for i in l:
  print(i)
```

```
1
2
3
```

```
def funk(l):
  for i in l:
    return i     #return 1st element only

a=funk([1,2,3])

a
```

```
1
```

```
#gererator

def funk(l):
  for i in l:
     yield i

a=funk([1,2,3])

print(next(a))
```

```
1
```

```
print(next(a))
```

```
2
```

```
print(next(a))
```

```
3
```

Iterator

Iterable and Iterators

- an Iterable is something that can be iterated over. like Tuple, List, Dict
- Iterator : is iterate the next element

```
mylist=[1,2,3,4]

a=iter(mylist)
```

```
print(next(a))
```

```
1
```

```
print(next(a))
```

```
2
```

```
print(next(a))
```

```
3
```

```
iterator=iter(mylist)

try:
   while True:

      element=next(iterator)
      print(element)

except:
  pass
```

```
1
2
3
4
```

```
from re import split
a='Hum sath sath ha, , , ,'

b=a.split()

print(b)
```

```
['Hum', 'sath', 'sath', 'ha,', ',', ',', ',']
```

```
for i in b:
  print(i)
```

```
Hum
sath
sath
ha,
,
,
,
```

```
class square:
  side=0

  def area(self):
    return self.side**2,(self.side+self.side)*2

    #return (self.side+self.side)*2


a=square()

a.side=12

c,d=a.area()




print(c)
print(d)
```

```
144
48
```

OOP'S Object oriented Programing Language

Constructor

- Constructor is a special function that get's automatically called when object of class get's created.
- the main purpose is to create and initialize the object

```
def __init__(self,name,age,salary):      #Constructor, (self) is reference argument
        self.emp_name=name
        self.emp_age=age
        self.emp_salary=salary
```

Classes and object

- Class is the blueprint of any real world object. Classes are the user defined data types.

```
class employee:
     def __init__(self,name,age,salary):         #Constructor, (self) is reference argument
         self.emp_name=name
         self.emp_age=age
         self.emp_salary=salary


     def emp_detail(self):
        print("Name of employee is :",self.emp_name)
        print("Age of employee is :",self.emp_age)
        print("Salary of employee is :",self.emp_salary)


     def msg(self):
      print("Thank you ",self.emp_name,"!!!!!")



e1=employee('Divya',25,10000)

e1.emp_detail()
e1.msg()
```

.....

```
class insan:
    hairs = ''
    eyes = ''
    gender = ''
    dimag = ''

    def create_insan(self):
        print(f"Hairs = {self.hairs}")
        print(f"Eyes = {self.eyes}")
        print(f"Gender = {self.gender}")
        print(f"Dimag = {self.dimag}")

x= insan()

x.hairs = 'black'
x.eyes = 'black'
x.gender = 'male'
x.dimag = 'bohot'

x.create_insan()
```

```
Hairs = black
Eyes = black
Gender = male
Dimag = bohot
```

```
class circle:
    radius = 0

    def area(self):
        return 3.14 * (self.radius**2)
```

```
c1 = circle()
c1.radius = 2
print(c1.area())
```

```
class rectangle:
    length = 0
    breadth = 0

    def area(self):
        return self.length * self.breadth

    def par(self):
        return (2*self.length)+(2*self.breadth)

r = rectangle()

r.length = 2
r.breadth = 4

print(r.area())
print(r.par())
```

constructors

```
class rectangle:
    length = 0
    breadth = 0

    def __init__(self, l, b):
        self.length = l
        self.breadth = b
        self.area = l * b

    def rarea(self):
        return self.length * self.breadth

o = rectangle(20, 10)

o.rarea()
```

```
create a class bank

name
opening balance
pin

check_balance()
deposite()
withdrawl()
```

```
class Bank:
    name = ''
    balance = 0
    pin = 0

    def __init__(self, n, o, p):
        self.name = n
        self.balance = o
        self.pin = p

    def check_balance(self):
        print(f"Current balance = {self.balance}")

    def deposite(self, amt):
        self.balance += amt
        print(f"{amt}rs has been succesfully credited to your account!")

    def withdrawl(self, amt, pin):
        if pin == self.pin:
            self.balance -= amt
            print(f"{amt}rs has been succesfully debited to your account!")
        else:
            print("Incorrect pin!"

idris = Bank('Idris Burhani', 20000, 1234)

idris.check_balance()

idris.deposite(30000)

idris.withdrawl(10000, 1234)

idris.balance = 1
```

Encapsulation

- Encapsulation is used to restrict the access of our class properties from outside the class.
- Public
- Protected
- Private
- (_) protected
- (_ __) private
- In encapsulation variable make private
- By usining getter and setter method we cant access the private variable

```
class A:
    name = 'Umer'
    __last_name = 'Purkar'

o = A()
print(o.name)
# print(o.__last_name)

o.__last_name = 'bhai'
print(o.__last_name)
```

```
class Bank:
    __name = ''
    __balance = 0
    __pin = 0

    def __init__(self, n, o, p):
        self.__name = n
        self.__balance = o
        self.__pin = p

    def check_balance(self):
        print(f"Current balance = {self.__balance}")

    def deposite(self, amt):
        self.__balance += amt
        print(f"{amt}rs has been succesfully credited to your account!")

    def withdrawl(self, amt, pin):
        if pin == self.__pin:
            self.__balance -= amt
            print(f"{amt}rs has been succesfully debited to your account!")
        else:
            print("Incorrect pin!")

    # getter method
    def getName(self):
        return self.__name

    # setter method
    def setName(self, n, p):
        if p == self.__pin:
            self.__name = n
            print(f"You name was succesfully changed to {self.__name}")
        else:
            print("Incorrect Pin!")

    def changePin(self, new, old):
        if old == self.__pin:
            self.__pin = new
            print("You pin was succesfully changed!")
        else:
            print('Incorrect pin!')

o = Bank('Neha', 30000, 9876)

o.check_balance()

o.deposite(10000)

o.withdrawl(5000, 4567)

o.__balance = 1

print(o.__balance)

print(o.getName())

o.setName('Neha Muley', 9876)

o.changePin(4567, 9876)
```

Inheritance

- Inheritance is a concept where one class can access the properties of other class. Here the class which inherits the properties is called as child class (sub class) and the class which is being inherited is called as parent class (base class).
- Child class can access all the properties of parent class including data members and member function.

Single Inheritance

- one to one
- When there is only one parent and one child, it is called as single inheritance.

```
class A:
    num1 = 2
    def a_hello(self):
        print("I am from class A")

    def samem(self):
        print("I am from class A same function")

class B(A):
    num2 = 4
    def b_hello(self):
        print("I am from class B")


    def samem(self):
        print("I am from class B same function")

        super().samem()   #call samem method using 'super'

o = B()

print(o.num2)
print(o.num1)

o.b_hello()
o.a_hello()


o.samem()
```

```
4
2
I am from class B
I am from class A
I am from class B same function
I am from class A same function
```

```
class A:
    num1 = 0

    def __init__(self, a):
        self.num1 = a

    def a_hello(self):
        print("I am from class A")

class B(A):
    num2 = 0

    def __init__(self, a, b):
        self.num2 = a
        super().__init__(b)

    def b_hello(self):
        print("I am from class B")
```

```
o = B(2, 4)

print(o.num2)
print(o.num1)
```

Multilevel Inheritance

- one to one to one
- Where this is grand parent - parent - child relationship between more than two classes, it is called as multilevel inheritance. Here child class can access the properties of parent and grand parent both. It is like a chain of classes and there can me n number of classes in this chain.

```
class A:
    num1 = 2

    def a_hello(self):
        print("I am from class A")

class B(A):
    num2 = 4

    def b_hello(self):
        print("I am from class B")

class C(B):
    num3 = 6

    def c_hello(self):
        print("I am from class C")

o = C()

print(o.num3)
print(o.num2)
print(o.num1)
o.a_hello()
o.b_hello()
o.c_hello()
```

```
class A:
    num1 = 0

    def __init__(self, a):
        self.num1 = a

    def a_hello(self):
        print("I am from class A")

class B(A):
    num2 = 0

    def __init__(self, a, b):
        self.num2 = a
        super().__init__(b)

    def b_hello(self):
        print("I am from class B")

class C(B):
    num3 = 0

    def __init__(self, a, b, c):
        self.num3 = a
        super().__init__(b, c)

    def c_hello(self):
        print("I am from class C")
```

```
o = C(2, 4, 6)

print(o.num3)
print(o.num2)
print(o.num1)
```

Multiple Inheritance

- many to one
- Where there are more than one parents and only one child, it is called as multiple inheritance. Here child class can access the properties from all the inherited parents.

```
class A:
    num1 = 2

class B:
    num2 = 4

class C(A, B):
    num3 = 6


o = C()

print(o.num3)
print(o.num2)
print(o.num1)
```

```
6
4
2
```

```
class A:
    num1 = 0

    def __init__(self, a, b):
        self.num1 = a
        super().__init__(b)

class B:
    num2 = 0

    def __init__(self, a):
        self.num2 = a

class C(A, B):
    num3 = 0

    def __init__(self, a, b, c):
        self.num3 = a
        super().__init__(b, c)


o = C(2, 4, 6)

print(o.num3)
print(o.num2)
print(o.num1)
```

Hierarchical Inheritance

- one to many
- When there is only one parent and multiple childs is called as hierarchical inheritancec.

```
class A:
    num1 = 2

class B(A):
    num2 = 4

class C(A):
    num3 = 6

o = C()
o2 = B()

print(o.num1)
print(o2.num1)
```

OVERRIDING

```
from typing_extensions import Self
class A:
   def hello(self):
    print("I am in class A")

class B(A):
   def hello(self):
    print("I am in class B")


class C(B):
   def hello(Self):
    print("I am in class C")

   def hi(self):
        self.hello()
        super().hello()
        super(B,self).hello()
```

```
o=C()

o.hi()
```

```
    I am in class C
    I am in class B
    I am in class A
```

Abstraction

```
class animal:
     def speak(self):
          pass
```

```
class dog(animal):
    def speak(self):
       print('bhau')
```

```
class cat(animal):
    def speak(self):
       print('mau')


tomy=dog()

tomy.speak()
```

```
    bhau
```

```
kitty=cat()

kitty.speak()
```

```
    mau
```

absratct class and abstract method

```
from abc import ABC,abstractmethod

class database_connection(ABC):
    def
```

CONNECT MYSQL WITH PYTHON

```
pip install mysql-connector
```

```
    Requirement already satisfied: mysql-connector in /usr/local/lib/python3.10/dist-packages (2.2.9)
```

```
import mysql.connector
```

```
connection = mysql.connector.connect(
    host= 'localhost',
    user= 'root',
    password= 'Sunil@1995',
    database= 'bank'


)
```

Module in Python

- a module is a file containing Python code. This code can define functions, classes, and variables, or it can execute statements when the module is imported.
- Modules provide a way to organize and encapsulate code into separate files, making it more maintainable and reusable.

```
# math_operations.py      #save this file .py

def add(x, y):
    return x + y

def subtract(x, y):
    return x - y
```

```
# main.py

import math_operations                  #import the module

result = math_operations.add(5, 3)
print("Addition result:", result)

result = math_operations.subtract(8, 2)
print("Subtraction result:", result)
```

Multi Precessing

- Multiprocessing in Python refers to the ability to run multiple processes concurrently, taking advantage of multiple CPU cores and parallelism to perform tasks more efficiently.

Multitreading

- Multithreading allows you to perform multiple tasks simultaneously
- multithreading can be used to process large datasets more efficiently by distributing the workload across multiple threads
- Tread class method

```
run()
start()
join()
isAlive
setName()
getName()
```

```
#USING MAIN TREAD
from time import sleep
from threading import Thread          #Using Multitreasing

class A(Thread):
  def run(self):
        for i in range(5):
            print("sunil")
            sleep(1)

class B(Thread):                      #Using Multitreasing
  def run(self):
        for i in range(5):
            print("Rathod")
            sleep(1)
```

```
t1=A()
t2=B()

#t1.run()
#t2.run()

t1.start()           #Using MultitreasingF
t2.start()

t1.join()
t2.join()

print("main thread ")
```

```
sunil
Rathod
sunil
Rathod
sunil
Rathod
sunil
Rathod
sunil
Rathod
main thread
```

```
from time import sleep
from threading import Thread          #Using Multitreasing

class A(Thread):
  def run(self):
        for i in range(5):
            print(i)
            sleep(1.5)

class B(Thread):                      #Using Multitreasing
  def run(self):
        for i in range(5):
            print(i)
            sleep(1.5)


a=A()
b=B()

a.start()
b.start()

a.join()
b.join()
```

```
0
0
1
1
2
2
3
3
4
4
```

ExCEPTION HANDLING

- NameError:
- ValueError:int("abc") # Cannot convert "abc" to an integer
- FileNotFoundError:
- ZeroDivisionError: Raised when you try to divide by zero.
- IndexError: Raised when you try to access an index that is out of range in a list or another iterable.
- Exception is nothing but run time errors and it occurs due to incorrect implemention of logic
- try
- except
- else

```
try:
   2/0
   print("no error ")
except Exception as e:
  print(e)
```

```
division by zero
```

```
try:
    x=int(input('enter number :'))
    print(x**
          2)
except Exception as e:
    print(e)
    print("enter number only")


else:                               #if try block condition true it will execute
  print("it execute only when try condion true")
finally:                            #always execute the block
  print("always execute")
```

```
enter number2
4
it execute only when try condion true
always execute
```

```
while True:
    try :
       n=int(input("please enter integer"))
       print(n**2)
       break

    except Exception as e:
         print(e)
         print("no valid integer ! try again")

print("Thanks !!!")
```

```
please enter integer2
4
Thanks !!!
```

File Handling

- Read, Write, Create. Append
- Text File
- CSV File
- Binary File

```
- create      : create   (x)
- Reading     :read mode (r)
- Writing      :write mode (w)
- Appending   :append mode (a)
```

```
#   file_object=open("file path","mode")

#file mode=  r=read  , w=write,  a=append,

#file operation  : create,read,write,copy,delete, close
```

```
f=open("filepath","x")

print("file create successfully...")

f=open("filepath","w")

f.write("sunil is writing in file")

f=open("filepath","r")

print(f.read(5))

f.close()

f=open("/content/note.txt",'r')

print(f.read())        #to read file

f.close()

f1=open("/content/note.txt",'w')

f1.write("hello sunil ")        #write in file
```

example

```
Fopen=open('/content/Final interviw question.txt','r')   #read

Fopen.read(500)

from os import write
Fwrite=open('/content/Practice.txt','w')   #write

Fwrite.write('hii sunil i am writing')


Fwrite=open('/content/Practice.txt','r')

Fwrite.read()

#create file (x)

fcreate=open("/content/newfile.txt","x")

print("file create successfully...")

fcreate=open('/content/newfile.txt','w')   #write

fcreate.write('i am writing in new file')

fcreate=open('/content/newfile.txt','r')

fcreate.read()

Fopen=open('/content/Final interviw question.txt','a')   #append

from os import close
Fopen.close()            #close file

1+91
```

```
92
```

Python date time

```
import datetime

x=datetime.datetime.now()

print(x)
```

```
2023-11-05 01:31:04.074309
```

```
y=x.strftime("%Y")

print(y)
```

```
2023
```

```
print(datetime.datetime(2023,1,2))
```

```
2023-01-02 00:00:00
```

```
from datetime import datetime


print(datetime.now().strftime("today is %A,%B %d,%Y"))   #  %A==Day    %B==Month   %d==Date   %Y==Year
```

```
today is Monday,December 04,2023
```

```
print(datetime.now().strftime("today is %d"))
```

```
today is 04
```

DATE

```
from datetime import date
date.today()
```

```
datetime.date(2023, 12, 4)
```

```
a=date.today()
a.year  #year
a.day   #day
a.month #mont
```

```
12
```

```
date.today().strftime("%A")   #strftime   date formatting
```

```
'Monday'
```

```
date.today().strftime("%a")
```

```
'Mon'
```

```
date.today().strftime("%B")
```

```
'December'
```

```
date.today().strftime("%b")
```

```
'Dec'
```

```
date.today().strftime("%d")
```

```
'04'
```

```
date.today().strftime("%D")
```

```
'12/04/23'
```

```
date.today().strftime("%Y")
```

```
'2023'
```

```
date.today().strftime("%y")
```

```
'23'
```

TIME

```
from datetime import datetime


a=datetime.now()
a
```

```
datetime.datetime(2023, 12, 4, 9, 22, 50, 234701)
```

```
a.strftime("%H %M %S")
```

```
'09 22 50'
```

convert string to date

```
A="12102023"

date_object = datetime.strptime(A, "%d%m%Y").date()

date_object
```

```
datetime.date(2023, 10, 12)
```

Python logging

- use to debbug the program
- it easier to diagnose issues, monitor the application's behavior, and understand what's happening during its execution.

Logging level

#DEBUG (level 10):

- Detailed diagnostic information, typically used for debugging. Messages that help you understand the flow of your code and diagnose issues.

INFO (level 20):

- General information messages that confirm the normal operation of the application. Useful for tracking the program's progress and significant events.

WARNING (level 30):

- Indicate potential issues or situations that might lead to problems. Not necessarily errors but noteworthy conditions. Warnings often signify that something unusual happened or that there may be a problem in the near future.

ERROR (level 40):

- Indicates that a specific error occurred, but the application can continue to run. These messages should be used for exceptions and errors that are handled gracefully.

CRITICAL (level 50):

- Denotes a very severe error that may lead to the program's termination. These messages indicate a critical failure that cannot be ignored.

```
from re import DEBUG
import logging

logging.basicConfig(filename='demo.log,level=logging.DEBUG,filemode='w',formst=)  #imp
logging.disable()

def namecheck(name):
  if len(name)<2:
    logging.debug('checking for name length')
    return("invalid name")
  elif name.isspace():
     logging.debug('checking for name space')
     return("invalid name")
  else:
     logging.debug('failed all check')
     return("invalid name")


print(namecheck('1'))
```

```
    invalid name
```

CONNECT MYSQL DB WITH PYTHON

```
pip install mysql-connector-python
```

```
import mysql.connector
```

```
connection = mysql.connector.connect(
    host= 'localhost',
    user= 'root',
    password= 'Sunil@1995',
    database= 'bank'


)
```

```
cur=connection.cursor()
```

Insert data¶

```
query='insert into studends(s_name,marks) values (%s,%s)'
val=('A',90)
```

```
cur.execute(query,val)
```

```
connection.commit()
```

```
query='insert into studends(s_name,marks) values (%s,%s)'
val=[('B',90),('C',88),('D',45)]
```

```
cur.executemany(query,val)

connection.commit()
```

Featch data

```
query='select * from studends'

cur.execute(query)

data=cur.fetchall()

print(data)

for i in data:
    print(i)

for i in data:
    print(f'{i[1]} - {i[2]}')
```

update

```
query='update studends set marks=00 where roll_no=4'

cur.execute(query)

connection.commit()
```

Bank Project

```
import mysql.connector

class My_connection:
    __host = ''
    __user = ''
    __pass = ''
    __database = ''

    def __init__(self, h, u, p, d):
        self.__host = h
        self.__user = u
        self.__pass = p
        self.__database = d

    def connect_db(self):
        try:
            con = mysql.connector.connect(
                host = self.__host,
                user = self.__user,
                password = self.__pass,
                database = self.__database,

            )
            return con
        except:
            pass




class Bank(My_connection):

    def __init__(self):
        try:
            super().__init__('localhost', 'root','Sunil@1995', 'bank')
            self.__con = super().connect_db()
            self.__cur = self.__con.cursor()
        except:
            print("Try again later!")


    def create_account(self,id,name,balance,pin):
        query = 'insert into bank_detail (id,name, balance, pin) values(%s, %s, %s, %s)'
        val = (id,name,balance,pin)
        try:
            self.__cur.execute(query, val)
            self.__con.commit()
            acc_no = self.__cur.lastrowid
            print(f"Your account was succesfully created, your account no is {id}")
        except:
            print("Try again later!")




    def check_balance(self, a, p):
        query = f'select balance, pin from bank_detail where id = {a}'
        try:
            self.__cur.execute(query)
            data = self.__cur.fetchall()
            if p == data[0][1]:
                print(f"Current balance = {data[0][0]}")
            else:
                print("Incorrect Pin!")
        except:
            print("Try again later!")




    def deposite(self, id,amt):
```

```
            query = f'update bank_detail set balance=balance + {amt} where id={id}'



            print(f"{amt}rs has been succesfully credited to your account!")







    def withdrawl(self,id,amt, pin):

        try:
            query = f'select balance, pin from bank_detail where id = {id}'
            self.__cur.execute(query)
            data = self.__cur.fetchall()

            if pin == data[0][1]:
                   query = f'update bank_detail set balance=balance - {amt} where id={id}
                   self.__cur.execute(query)
                   self.__con.commit()

                   print(f"{amt} rs has been succesfully debited to your account!")


o=Bank()
                print("Incorrect Pin!")
o.create_account(6,'Div',5000,2254)
```