

welcome

welcome

shift+enter=run the cell

dd=delete the cell

Keywords

In [1]:

```
import keyword  
print(keyword.kwlist)
```

```
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
```

In [2]:

```
false=10
```

In [3]:

```
false
```

Out[3]:

10

identifier

name given to entity like class,functions,variables etc

In [4]:

```
1var=20
```

Input In [4]
1var=20
^

SyntaxError: invalid decimal literal

In []:

```
var1=20
```

In []:

```
var1
```

In []:

```
var@=30
```

In []:

```
var_=30
```

In []:

```
var_
```

In []:

```
finally=40
```

In []:

```
Finally=40
```

In []:

```
Finally
```

In []:

```
from=50
```

In []:

```
From=50
```

In []:

```
From
```

comments in python :used to explain the code for more readability

In []:

```
print('python')      #define python
```

In []:

```
#single Line comment  
#multi line comment
```

In []:

```
"this  
is python  
session""
```

In []:

```
'''welcome  
to  
the  
india'''
```

statement

In []:

```
val5=10
```

In []:

```
p1=20+30  
p1
```

In []:

```
p2=20+30\  
+40+50\  
+70+80  
p2
```

In []:

```
p2=20+30\  
+40+50\  
+70+80  
p2
```

indentation

In []:

```
x=10  
if x==10:  
    print('x is equal to 10')
```

docstrings

In []:

```
def square(num):
    """square function will return the square of a number"""
    return num**2
```

In []:

```
square(2)
```

In []:

```
square.__doc__
```

variables

In []:

```
a=5
```

In []:

```
a
```

In []:

```
p=50
q=25
r=q
```

In []:

```
print(id(p))
```

In []:

```
print(id(q))
```

data type

int float bool/boolean complex

In []:

```
a=1254789631422
```

In []:

```
type(a)
```

In []:

```
b=10.2  
b
```

In []:

```
type(b)
```

In []:

```
bool1=True
```

In []:

```
bool1
```

In []:

```
print(bool(-2))
```

In []:

```
x=2+3j
```

In []:

```
type(x)
```

Strings

In []:

```
str1="hello"
```

In []:

```
type(str1)
```

In []:

```
len(str1)
```

In []:

```
str2=" hello python "
```

In []:

```
str2[0]
```

In []:

```
len(str2)
```

indexing

In []:

```
str3='welcome'
```

in python we always start indexing from 0

In []:

```
str3[0]
```

In []:

```
str3[-7]
```

Slicing

In []:

```
str3[3:6]
```

In []:

```
str3[3:7]
```

In []:

```
str3[0:]
```

In []:

```
str3[3:]
```

In []:

```
str4="bangalore"
```

In []:

```
str4[2:5]
```

In []:

```
str4[3:]
```

In []:

```
str5='data science'
```

In []:

```
str5[5:]
```

In []:

```
str5[2:8]
```

In []:

```
str6='heloo'
```

In []:

```
str[3]='l'
```

In []:

```
del str6
```

In []:

```
str6
```

string concatenation

In []:

```
s1='data'  
s2='science' #data science
```

In []:

```
print(s1,s2,"dvnksdvnvdkndvn")
```

In []:

```
print(s1+' - '+s2)
```

String Membership

In []:

```
mystr='hello everyone' #in,not in membership operator
```

In []:

```
print('hello' in mystr)
```

In []:

```
print('python' in mystr)
```

In []:

```
print('eve' in mystr)
```

In []:

```
print('-eve' in mystr)
```

string partitioning

In []:

```
mystr1='natural language with python and R and java'  
mystr1
```

In []:

```
l=mystr1.partition('with')  
l
```

In []:

```
mystr1.capitalize()
```

In []:

```
mystr1.upper()
```

In []:

```
mystr1.lower()
```

In []:

```
mystr1.count('a')
```

In []:

```
mystr2="hello"  
mystr2
```

In []:

```
mystr2.strip()
```

In []:

```
mystr2.rstrip()
```

In []:

```
mystr2.lstrip()
```

21 july 2023

In []:

```
str6="heloo"
print(id(str6))
```

In []:

```
x=str6.replace('o','l')
print(id(x))
```

In []:

```
str7='good morning'
```

In []:

```
str7.replace("good", "beautiful")
```

In []:

```
str8='one two three four five six seven'
```

In []:

```
str8
```

In []:

```
str8.split()
```

variable assignment

In []:

```
x=10
```

In [5]:

```
x,y,z=20,25.5,'hello'
```

In [6]:

```
print(x)
print(y)
print(z)
```

```
20
25.5
hello
```

Operators

Arthmetic

Assignment

comparition /relational

logical

membership

identity

bitwise

Arthmetic =====

Addition

Substraction

Multiplication

Division

Floor division

Modulos

Exponentional

In [7]:

```
print(10+2)
print(10-2)
print(10*2)
```

```
12
8
20
```

In [8]:

```
print(15/2)
print(15//2)
print(15%2)
```

7.5
7
1

In [9]:

```
print(2**2)
```

4

Assignment

In [10]:

```
x=5
```

In [11]:

```
y=10
y+=2      #y=y+2
```

In [12]:

```
print(y)
```

12

In [13]:

```
z=15
z-=10      #z=z-10
```

In [14]:

```
print(z)
```

5

comparision /relational

== Equal

!= not equal

#> greater than

< less than

<= less than equal to

#>= greater than equal to

In [15]:

```
x=10  
y=20  
print(x==y)  
print(x<y)  
print(x>y)
```

```
False  
True  
False
```

Logical operators

and

or

not

In [16]:

```
x=10  
y=20  
print((x<5) and (y<10))  
print((x<5) or (y<10))  
print((x==10) and (y==20))  
print((x==10) or (y==20))
```

```
False  
False  
True  
True
```

identity operators

In [17]:

```
print((10) is (10.0))
```

```
False
```

```
<>:1: SyntaxWarning: "is" with a literal. Did you mean "=="?  
<>:1: SyntaxWarning: "is" with a literal. Did you mean "=="?  
C:\Users\Mukund\AppData\Local\Temp\ipykernel_1312\436760226.py:1: SyntaxWa  
rning: "is" with a literal. Did you mean "=="?  
    print((10) is (10.0))
```

In [18]:

```
print('hello')
x=10
if (x>5):
    print('welcome')
print(hdsvvhsvn)
print('python')
```

hello
welcome

```
-----
-
NameError: name 'hdsvvhsvn' is not defined
Traceback (most recent call last)
Input In [18], in <cell line: 5>()
  3 if (x>5):
  4     print('welcome')
----> 5 print(hdsvvhsvn)
  6 print('python')
```

NameError: name 'hdsvvhsvn' is not defined

Type Casting

In []:

```
a='2'
b=2
```

In []:

```
type(a)
```

In []:

```
type(b)
```

Auto type casting

Forced Type casting

In []:

```
4+3.23+False #4.00+3.23+0.00
```

In []:

```
3+2.3+False+True #3.00+2.3+0.00+1.00
```

In []:

```
2+3+'data'+4.5
```

In []:

```
12+15+'10'+2
```

In []:

```
print(int(True))
```

In []:

```
print(int(bool("data")))
```

In []:

```
a='10'
```

In []:

```
print(int(a))
```

In []:

```
b='data123'
```

In []:

```
print(int(b))
```

In []:

```
z=3.2  
print(int(z))
```

In []:

```
x=3  
print(float(x))2
```

In []:

```
print(float(False))
```

Python KT session 24 July 23

List

In []:

```
list1=[10,20,30,40,50] #homogenous same kind of data
print(list1)
print('hello')
```

In []:

```
type(list1)
```

In []:

```
list2=[60,20.5,'happy'] #hetrogenous multiple kind of data
list2
```

In []:

```
type(list2)
```

In []:

```
list3=[[1,2,3],[4,5,6]] #nested List
```

In []:

```
type(list3)
```

Properties of List

In []:

```
#List store element in sequential order
#List store hetrogenous element
#List allow duplicate value
#List are mutable or changeable
```

In []:

```
list4=[1,1,20,30,40]
id(list4)
```

In []:

```
list4
```

In []:

```
list4[1]=5
```

In []:

```
list4
id(list4)
```

In []:

```
list5=[10,50,7,80,90,5,20,100]
```

In []:

```
list5[-4]
```

In []:

```
list5[-7]
```

In []:

```
list5[-5:-2]      #start index position:
```

In []:

```
list6=[[1,2,3],[4,5,6]]
```

In []:

```
list6[1][1]
```

In []:

```
list7=[10,50,7,80,90,5,20,100]
```

In []:

```
list7.append(99)
```

In []:

```
list7
```

In []:

```
list7.append(105)
```

In []:

```
list7
```

In []:

```
list7.pop()
```

In []:

```
list7
```

In []:

```
list7.pop()
```

In []:

```
list7
```

In []:

```
list7.pop(2)
```

In []:

```
list7
```

In []:

```
list7.insert(0,5)
```

In []:

```
list7
```

In []:

```
list7.reverse()
```

In []:

```
list7
```

In []:

```
list7.sort()
```

In []:

```
list7
```

In []:

```
list7.sort(reverse=True)
```

In []:

```
list7
```

Tuple

In []:

```
tup1=(10,20,30,40)  
tup1
```

In []:

```
type(tup1)
```

In []:

```
tup2=50,60,40,70  
tup2
```

In []:

```
type(tup2)
```

Properties of tuple

In []:

```
#tuple store element in sequential order  
#tuple allow duplicate values  
#tuple is immutable or unchangeable
```

In []:

```
tup3=(1,1,5,7,8)
```

In []:

```
tup3
```

In []:

```
tup3[1]=9
```

In []:

```
tup4=(10,20,30,50,70,80,90)
```

In []:

```
tup4[3:5]
```

In []:

```
tup4.index(80)
```

In []:

```
tup4.count(50)
```

Set

In []:

```
set1={10,20,60}  
set1
```

In []:

```
type(set1)
```

In []:

```
set2={60,50,10,20,80,10,50}  
len(set2)
```

In []:

```
set2
```

Properties of Set

In []:

```
#set store element in random order  
#set didn't allow duplicate values  
#set is immutable or mutable  
#indexing and slicing is not possible in set
```

In []:

```
list10=[] #empty list  
type(list10)
```

In []:

```
tup10=() #empty tuple  
type(tup10)
```

In []:

```
set3=set() #empty set  
type(set3)
```

In []:

```
set4={10,20,50,70,80}
```

In []:

```
set4[1:3] #indexing /slicing is not possible
```

In []:

```
set4.remove(70)
```

In []:

```
set4
```

In []:

```
set4.discard(80)
```

In []:

```
set4
```

In []:

```
set5={10,20,50,70,80,90,100}
```

In []:

```
set5.remove(105)
```

In []:

```
set5.discard(109)
```

Dictionary

In []:

```
dict1={} #empty dictionary  
type(dict1)
```

In []:

```
dict2={'name':'rohit','age':25}
```

In []:

```
dict2
```

In []:

```
dict2.keys()
```

In []:

```
dict2.values()
```

In []:

```
dict2.items()
```

In []:

```
dict2['location']='chennai'
```

In []:

```
dict2
```

In []:

```
dict2['location']='mumbai'
```

In []:

```
dict2
```

In []:

```
dict3={1:'one',2:'two','A':[ 'shankar','balaji','girish']}
```

In []:

```
dict3
```

In []:

```
dict3['A'][1][2]
```

In []:

```
dict4={0:{'one':1,'two':2},1:{'city':'banglore','area':'whitefield'},'three':[ 'Mumbai','P'
```

In []:

```
dict4
```

In []:

```
dict4[1]['area']
```

In []:

```
dict4[0]['two']
```

In []:

```
dict4['three'][0]
```

In []:

```
dict4['three'][1][2]
```

In []:

```
dict4[1]['city']
```

In []:

```
dict4.pop('three')
```

Conditional Statement

In []:

```
#if  
#if---else  
#if--elif--elif....  
#nested if
```

In []:

```
x=10  
if x==10:           #true  
    print('matched')  
print('hello')
```

In []:

```
x=20  
if x==10:           #false  
    print('matched')  
print('hello')
```

In []:

```
y=15  
if y>20:  
    print('welcome')  
print('python')
```

In []:

```
y=15  
if y<20:  
    print('welcome')  
print('python')
```

In []:

```
#write down the code as a output "Logged in" when password is 12345
```

In []:

```
y=12345
if y==12345:
    print('logged in')
```

In []:

```
x=10
y=20
print('hello')
if(x+y==30 and x>y):      #true
    print('yes')
else:
    print('no')
print('welcome')
```

In []:

```
x=13
if x%2==0:                  #false
    print('x is even')
else:
    print('x is odd')
```

In []:

```
y=12
if y%3==0:                  #cond true
    print('divisible')
else:
    print('not divisible')
```

In []:

```
x=2
if x==1:                      #false
    print('python')
elif x==2:                     #true
    print('jupyter')
elif x==3:
    print('ML')
else:
    print("AI")
```

In []:

```
x=10
if x==1:
    print('python')
elif x==2:
    print('jupyter')
elif x==3:
    print('ML')
else:
    print("AI")
```

In []:

```
light_color='pink'
if light_color=='red':
    print('stop')
elif light_color=='green':
    print('go')
elif light_color=='orange':
    print('slow')
else:
    print('invalid color')
```

In []:

```
light_color='green'
x=10
if light_color=='red':
    if x==10:
        print('hello')
    print('stop')
elif light_color=='green':
    print('go')
elif light_color=='orange':
    print('slow')
else:
    print('invalid color')
```

In []:

```
light_color='red'
x=11
if light_color=='red':
    if x==10:
        print('hello')
    print('stop')
elif light_color=='green':
    print('go')
elif light_color=='orange':
    print('slow')
else:
    print('invalid color')
```

input method

In []:

```
light_color=input("Enter the color of the traffic light(red/green/orange): ")

if light_color=='red':
    print('stop')
elif light_color=='green':
    print('go')
elif light_color=='orange':
    print('slow')
else:
    print('invalid color')
```

Python KT 25 July 23

In [20]:

```
name=input("enter your name : ") #input method always return string value
```

```
enter your name : mukund
```

In [21]:

```
age=input('enter the age :')
print(type(age))
```

```
enter the age :20
<class 'str'>
```

In [22]:

```
age=int(input('enter the age :')) #you need to convert the data type using type casting
print(type(age))
```

```
enter the age :30
<class 'int'>
```

Control Statement

Loop----while and for

In [23]:

```
i=1          #initializer
while i<=5:    #expression  #true
    print(i)
    i=i+1
```

```
1
2
3
4
5
```

In [28]:

```
i=5
while i>=2:
    print(i)
    i=i-1
```

5
4
3
2

In [48]:

```
name='hello'
i=0
while i<=4:
    print(name[i],end=' ')
    i=i+2
```

h l o

In [46]:

```
name='umesh'
i=0
while i<=4:
    print(name[i],end=" ")
    i=i+1
```

u m e s h

In [35]:

```
print('hello','welcome',sep="*")
```

hello*welcome

In [40]:

```
print('hello')
print('welcome')
```

hello
welcome

In [45]:

```
print('hello',end=" ")
print('welcome')
```

hello welcome

For loop

In [50]:

```
x=[10,20,45,37,30,40,50]
for i in x:
    if i%2==0:
        print(i)
```

10
20
30
40
50

In [57]:

```
x=[10,40,21,35,32]
for i in x:
    if i%2!=0:
        print(i)
```

21
35

In [58]:

```
name='welocme'
for i in name:
    if i in "aeiou":
        print(i)
```

e
o
e

Nested For

In [61]:

```
a=['science','math','english']
b=[10,20,30]
for i in a:    #science
    for j in b:    #10
        print(j,i)
```

10 science
20 science
30 science
10 math
20 math
30 math
10 english
20 english
30 english

Range

In [60]:

```
#to generate range of numbers
```

In [63]:

```
x1=list(range(100))
print(x1)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 3
9, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57,
58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 7
6, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94,
95, 96, 97, 98, 99]
```

In [64]:

```
x2=list(range(25)) #range will generate from 0 and end with n-1
print(x2)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
21, 22, 23, 24]
```

In [65]:

```
x3=list(range(1,10))
print(x3)
```

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

In [66]:

```
x3=list(range(5,20)) #it will start from intilizer and end with n-1(2 variable)
print(x3)
```

```
[5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

In [67]:

```
x3=list(range(2,25,2))
print(x3)
```

```
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24]
```

In [73]:

```
x3=list(range(10,30,17))
print(x3)
```

```
[10, 27]
```

In [76]:

```
#1-100 included
x3=list(range(1,101))
print(x3)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 2
1, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39,
40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 5
8, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76,
77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 9
5, 96, 97, 98, 99, 100]
```

In [80]:

```
x3=list(range(101,5,-1))
print(x3)
```

```
[101, 100, 99, 98, 97, 96, 95, 94, 93, 92, 91, 90, 89, 88, 87, 86, 85, 84,
83, 82, 81, 80, 79, 78, 77, 76, 75, 74, 73, 72, 71, 70, 69, 68, 67, 66, 6
5, 64, 63, 62, 61, 60, 59, 58, 57, 56, 55, 54, 53, 52, 51, 50, 49, 48, 47,
46, 45, 44, 43, 42, 41, 40, 39, 38, 37, 36, 35, 34, 33, 32, 31, 30, 29, 2
8, 27, 26, 25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10,
9, 8, 7, 6]
```

Packing and unpacking

In [86]:

```
marks=72,85,91,59          #packing
rohit,balaji,gayathri,gomathi=marks  #unpacking
print(rohit)
print(balaji)
print(gayathri)
print(gomathi)
print(type(marks))
```

```
72
85
91
59
<class 'tuple'>
```

In [88]:

```
x=10,20,30
a,b,c=x

print(a)
print(b)
print(c)
print(type(x))
```

```
10
20
30
<class 'tuple'>
```

Functions

In [89]:

```
#block of code which perform specific task for us  
#once we will assign/create function we can call n number of times  
#parameter  
#argument
```

In []:

```
#build in  
#user defined  
#anonymous
```

In [90]:

```
x=[39,17,5,9,25]  
print(len(x))    #no. of element  
print(type(x))   #datatype  
print(max(x))    #max no.  
print(min(x))    #min no.  
print(sum(x))    #sum no.
```

```
5  
<class 'list'>  
39  
5  
95
```

In [91]:

```
def myfun1(a,b):      #def definition to define user defined functions  
    return a+b  
res=myfun1(10,20)     #calling the function  
print(res)  
res*5
```

```
30
```

In [92]:

```
def myfun1(a,b):  
    print(a+b)  
myfun1(10,20)
```

```
30
```

In []:

```
#return to store the output .....
```

In [94]:

```
def square(n): #parameter variable define under the parenthesis
    n=n*n
    return n #store the output
res1=square(10) #argument assigning the value to the parameter
print(res1)
print(res1+5)
```

100
105

Types of Parameter/Argument

In [95]:

```
#positional
#keyword
#default
#Arbitrary / variable number of argument / args
#Keyword Arbitrary / keyword variable number of argument / kwargs
```

In [97]:

```
#positional
def add(x,y):
    print(x)
    print(y)

add(50,40)
```

50
40

In [98]:

```
#keyword
def add(x,y):
    print(x)
    print(y)

add(y=50,x=40)
```

40
50

In [107]:

```
#default
def add(x,y=20,z=50):
    print(x)
    print(y)
    print(z)

add(10)
```

```
10
20
50
```

In [109]:

```
#Arbitrary / variable number of argument / args
def fun2(x):
    print(x)

fun2()
```

```
-----
-
TypeError                                     Traceback (most recent call last)
t)
Input In [109], in <cell line: 5>()
  2 def fun2(x):
  3     print(x)
----> 5 fun2()

TypeError: fun2() missing 1 required positional argument: 'x'
```

In [110]:

```
def fun2(x):
    print(x)

fun2(10,20)
```

```
-----
-
TypeError                                     Traceback (most recent call last)
t)
Input In [110], in <cell line: 4>()
  1 def fun2(x):
  2     print(x)
----> 4 fun2(10,20)

TypeError: fun2() takes 1 positional argument but 2 were given
```

In [115]:

```
def fun2(*x):    #args
    print(x)

fun2()
fun2(10,20)
fun2(1,2,3)
fun2(99,80,70,60,70,50)
```

```
()  
(10, 20)  
(1, 2, 3)  
(99, 80, 70, 60, 70, 50)
```

In [123]:

```
def fun3(x,*y,z):
    print(x)
    print(y)
    print(z)

#fun3(10,20,30,40,60)
#fun3(10,20,30,40)
fun3(10,20,30)
```

```
-----  
-  
TypeError
```

```
Traceback (most recent call last)
```

```
t)
Input In [123], in <cell line: 8>()
    4     print(z)
    5 #fun3(10,20,30,40,60)
    6 #fun3(10,20,30,40)
----> 8 fun3(10,20,30)
```

```
TypeError: fun3() missing 1 required keyword-only argument: 'z'
```

In [131]:

```
def fun3(x,*y,z=100):
    print(x)
    print(y)
    print(z)

fun3(10,20,30,z='hello')
```

```
10
(20, 30)
hello
```

In [130]:

```
x=10
print(x)
x=20
print(x)
```

```
10
20
```

In [133]:

```
#Keyword Arbitrary / keyword variable number of argument / kwargs
def fun5(**kwargs):
    print(kwargs)

fun5(name="rohit",location="mumbai",age=25)

{'name': 'rohit', 'location': 'mumbai', 'age': 25}
```

In [135]:

```
dict12={'name':'rohit','location':'mumbai','age':25}
def fun6(**kwargs):
    print(kwargs)

fun6()

{}
```

In []:

```
#anonymous function
# function without a name
# Lambda is called inliner
#special function map and filter
```

In [136]:

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

res=add(15,25)
print(res)
```

```
40
```

In [137]:

```
add=lambda x,y:x+y  #syntax  Lambda input parameter : return value
res=add(15,25)
res
```

Out[137]:

```
40
```

In [138]:

```
fun6=lambda x:'even' if x%2==0 else "odd"
res1=fun6(10)
print(res1)
```

even

In [139]:

```
fun6=lambda x:'even' if x%2==0 else "odd"
res1=fun6(13)
print(res1)
```

odd

In [140]:

```
fun7=lambda y:'vowel' if y in "aeiou" else 'not vowel'
res2=fun7('a')
print(res2)
```

vowel

In [141]:

```
fun7=lambda y:'vowel' if y in "aeiou" else 'not vowel'
res2=fun7('b')
print(res2)
```

not vowel

In [145]:

```
str15='hello python'
print('h'in str15)
print(' 'in str15)
```

True

True

In [153]:

```
#map and filter
num=[1,2,3,4,5,6,7,8,9,10] #print the output in such a way that it should be multiplication
mul_num=set(map(lambda x:x*2,num)) #map will take 2 parameter 1:function 2: iterable
print(mul_num)
```

{2, 4, 6, 8, 10, 12, 14, 16, 18, 20}

In [155]:

```
num=[1,2,3,4,5,6,7,8,9,10]
even_num=list(filter(lambda x:x%2==0,num)) #filter will take 2 parameter 1:function(cond)
print(even_num)
```

[2, 4, 6, 8, 10]

In [156]:

```
x=[1,2,3,4,5]
def fun6(x):
    return x+1

res=list(map(fun6,x))
print(res)
```

[2, 3, 4, 5, 6]

In [157]:

```
def fun7(x):
    return x**2

res=list(map(fun7,x))
print(res)
```

[1, 4, 9, 16, 25]

In [158]:

```
res=list(map(lambda x:x**2,x))  #
print(res)
```

[1, 4, 9, 16, 25]

In []:

```
addition=lambda *args:sum(args)
res=addition(10,20,30,40,50)
print(res)
```

Packages and module

In [161]:

```
#Import moduleName
#import module name as alias name
#import module name with specific code
#import module with *
```

In [165]:

```
from datetime import date,datetime
#from one import add

today_date=date.today()
print(today_date)
now_time=datetime.now()
print(now_time)
```

2023-07-25

2023-07-25 16:23:49.942593

In [168]:

```
from datetime import date,datetime,timedelta
current_date = datetime.now()
yesterday = current_date - timedelta(days=1)
yesterday
```

Out[168]:

```
datetime.datetime(2023, 7, 24, 16, 28, 37, 514424)
```

break continue & pass

In [166]:

```
for i in range(10):
    if i==7:          #condition is true it will break the loop
        break
    print(i,end=" ")
```

```
0 1 2 3 4 5 6
```

In [167]:

```
for i in range(10):
    if i==7:
        continue      #condition will met it skip again loop will continue
    print(i,end=" ")
```

```
0 1 2 3 4 5 6 8 9
```

In [171]:

```
def my_fun():
    pass
    print('hello')

my_fun()
```

```
hello
```

In [172]:

```
def my_fun():
    pass
```

Local Variable & Global Variable

In [173]:

```
def m1(a):      #local variable which declared inside of the function
    p=50         #local variable
```

```
m1(10)
```

In [174]:

```
x=40      #global variable variable --outside of the function
y=35      #global variable
def m1(a):    #local
    p=50      #Local

m1(10)
```

In [175]:

```
#scope
#local variable: inside the function only
#global variable :scope is everywhere
```

In [181]:

```
def f1():
    a=10
    print(a)

def f2():
    b=20
    print(b)

f1()
f2()
print(b)
```

```
10
20
20
```

In [182]:

```
x=10      #global
def f1():
    x=50      #Local

print(x)
```

```
10
```

In [183]:

```
x=10      #global
def f1():
    x=50      #local
    print(x)

f1()
```

```
50
```

In [1]:

```
a={1,2,3,4,5}  
b={4,5,6,7,8}  
a-b
```

Out[1]:

```
{1, 2, 3}
```

In [2]:

```
a.difference(b)
```

Out[2]:

```
{1, 2, 3}
```

In [4]:

```
a.difference_update(b)  
a
```

Out[4]:

```
{1, 2, 3}
```

Python KT 27 July 2023

exception handling

In [1]:

```
#types of error  
#python mainly has 3 types of error
```

In [4]:

```
#Syntax error  
for i in range(10)
```

Input In [4]
for i in range(10)

SyntaxError: expected ':'

In [6]:

```
#Logical error  
radius=5  
area_of_circle=2*3.14*radius  
area_of_circle
```

Out[6]:

```
31.400000000000002
```

In [7]:

```
#runtime error (exception)
res=10/0
res
```

```
-----
-
ZeroDivisionError                                Traceback (most recent call last)
t)
Input In [7], in <cell line: 2>()
    1 #runtime error
----> 2 res=10/0
    3 res
```

ZeroDivisionError: division by zero

In [8]:

```
#types of exceptions
#ZeroDivisionError
```

In [9]:

```
#type error
res='10'/2
res
```

```
-----
-
TypeError                                     Traceback (most recent call last)
t)
Input In [9], in <cell line: 2>()
    1 #type error
----> 2 res='10'/2
    3 res
```

TypeError: unsupported operand type(s) for /: 'str' and 'int'

In [10]:

```
#value error
num=int('hello')
num
```

```
-----
-
ValueError                                    Traceback (most recent call last)
t)
Input In [10], in <cell line: 2>()
    1 #value error
----> 2 num=int('hello')
    3 num
```

ValueError: invalid literal for int() with base 10: 'hello'

In [11]:

```
#index error
my_list=[1,2,3]
my_list[5]
```

```
- IndexError
Input In [11], in <cell line: 3>()
    1 #index error
    2 my_list=[1,2,3]
----> 3 my_list[5]
```

IndexError: list index out of range

In [12]:

```
#key error
my_dict={'a':1,'b':2}
my_dict['c']
```

```
- KeyError
Input In [12], in <cell line: 3>()
    1 #key error
    2 my_dict={'a':1,'b':2}
----> 3 my_dict['c']
```

KeyError: 'c'

In [13]:

```
#attribute error
my_str='hello'
my_str.intersection()
```

```
- AttributeError
Input In [13], in <cell line: 3>()
    1 #attribute error
    2 my_str='hello'
----> 3 my_str.intersection()
```

AttributeError: 'str' object has no attribute 'intersection'

In [14]:

```
#file not found error
```

In [15]:

```
#ModuleNotFoundError : import wrong module name
```

In [17]:

```
def m2(x,y):
    res=x/y
    print(res)

m2(10,2)
```

5.0

In [18]:

```
def m2(x,y):
    res=x/y
    print(res)

m2(10,0)
```

```
-
```

ZeroDivisionError Traceback (most recent call last)
t)
Input In [18], in <cell line: 5>()
 2 res=x/y
 3 print(res)
----> 5 m2(10,0)

Input In [18], in m2(x, y)
 1 def m2(x,y):
----> 2 res=x/y
 3 print(res)

ZeroDivisionError: division by zero

In [19]:

```
def m2(x,y):
    res=x/y
    print(res)

m2('10',2)
```

-

TypeError Traceback (most recent call last)

Input In [19], in <cell line: 5>()
 2 res=x/y
 3 print(res)
----> 5 m2('10',2)

Input In [19], in m2(x, y)
 1 def m2(x,y):
----> 2 res=x/y
 3 print(res)

TypeError: unsupported operand type(s) for /: 'str' and 'int'

In [22]:

```
def m2(x,y):
    try:                      #in case of valid input try block will execute
        res=x/y
        print(res)
    except ZeroDivisionError:
        print('division by zero is not allowed')
    except TypeError:
        print('invalid data type for division')

m2('10',2)
m2(10,0)
m2(10,5)
```

invalid data type for division
 division by zero is not allowed
 2.0

In [33]:

```
def list_element(list16,index):
    value=list16[index]
    print('element at index',index,':',value)

list16=[1,2,3,4,5]
list_element(list16,2)

element at index 2 : 3
```

In [34]:

```
def list_element(list16,index):
    value=list16[index]
    print('element at index',index,':',value)

list16=[1,2,3,4,5]
list_element(list16,5)
```

-

IndexError Traceback (most recent call last)

t)
Input In [34], in <cell line: 6>()
 3 print('element at index',index,':',value)
 5 list16=[1,2,3,4,5]
----> 6 list_element(list16,5)

Input In [34], in list_element(list16, index)
 1 def list_element(list16,index):
----> 2 value=list16[index]
 3 print('element at index',index,':',value)

IndexError: list index out of range

In [35]:

```
def list_element(list16,index):
    try:
        value=list16[index]
        print('element at index',index,':',value)
    except IndexError:
        print('the list does not have an element at index',index)

list16=[1,2,3,4,5]
list_element(list16,5)
```

the list does not have an element at index 5

In [36]:

```
def list_element(list16,index):
    try:
        value=list16[index]
        print('element at index',index,':',value)
    except:           #generic exception
        print('the list does not have an element at index',index)

list16=[1,2,3,4,5]
list_element(list16,5)
```

the list does not have an element at index 5

In [37]:

```
def m2(x,y):
    try:
        res=x/y
        print(res)
    except:
        print('division by zero is not allowed')
    except:
        print('invalid data type for division')

#we can't use multiple except block without mentioning error name
#one try block can have max one generic except block
m2('10',2)
m2(10,0)
m2(10,5)
```

Input In [37]

```
except:  
^
```

SyntaxError: default 'except:' must be last

In []:

```
# we can handle exception through try and except block
# in a code multiple exception can occur
# we can handle them by using 2 ways
# multiple try-except block
# using single try but multiple except block
```

In [39]:

```
#in case of exception only except block will execute
try:
    print(10/0)
except:
    print('exception handled')
else:
    print("else is excuted")
```

exception handled

In [40]:

```
#in case of no exception try and else block will execute
try:
    print(10/2)
except:
    print('exception handled')
else:
    print("else is excuted")
```

5.0
else is excuted

In [42]:

```
def m2(x,y):
    try:
        res=x/y
        print(res)
    except ZeroDivisionError:
        print('division by zero is not allowed')
    except TypeError:
        print('invalid data type for division')
    finally:
        print('this block will always execute')

m2('10',2)
```

invalid data type for division
this block will always execute

In [43]:

```
def m2(x,y):
    try:
        res=x/y
        print(res)
    except ZeroDivisionError:
        print('division by zero is not allowed')
    except TypeError:
        print('invalid data type for division')
    finally:
        print('this block will always execute')
```

m2(10,0)

division by zero is not allowed
this block will always execute

In [45]:

```
def m2(x,y):
    try:
        res=x/y
        print(res)
    except ZeroDivisionError:
        print('division by zero is not allowed')
    except TypeError:
        print('invalid data type for division')
    else:
        print('hello')
    finally:
        print('this block will always execute')
```

m2(10,2)

5.0
hello
this block will always execute

List Comprehension

In [46]:

```
#precise way to create a List
```

In [47]:

```
num=[1,2,3,4,5]
square=[]
for i in num:
    square.append(i**2)

square
```

Out[47]:

```
[1, 4, 9, 16, 25]
```

In [48]:

```
num=[1,2,3,4,5]
square=[i**2 for i in num] #syntax: [append code/return value for Loop ]
square
```

Out[48]:

```
[1, 4, 9, 16, 25]
```

In [49]:

```
numbers=[1,2,3,4,5,6,7,8,9,10]
even_numbers=[]
for i in numbers:
    if i%2==0:
        even_numbers.append(i)

even_numbers
```

Out[49]:

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

In [50]:

```
numbers=[1,2,3,4,5,6,7,8,9,10]
even_numbers=[i for i in numbers if i%2==0] #syntax: [append code/return value for Loop
even_numbers
```

Out[50]:

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

In [51]:

```
#in case of if-else for Loop will come at Last
```

In [52]:

```
# [1st return value if condn else 2nd return value for Loop ]
```

In [53]:

```
list1=[1,2,3,4,5,6]
res=['even' if i%2==0 else 'odd' for i in list1]
res
```

Out[53]:

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

extract vowels from string using list comprehension

In [57]:

```
s1 = ['apple', 'banana', 'orange', 'mango', 'papaya']
v = [i for i in s1 if i[0] in 'aeiou']
v
```

Out[57]:

```
['apple', 'orange']
```

Python KT 28 July 2023

File Handling

OOPs

The main OOPS concepta are:-

1 : Class

2 : Object

3 : Encapsulation

4 : Inheritance

5 : Polymorphism

6 : Abstraction

In [1]:

```
#class and object
#class is a virtual entity
#object is real entity
#every object has some properties and functionality
#properties====variable
#functionality====method
#class tell us how our object look alike
```

In [14]:

```
class House: #define class
    def __init__(self):      #constructor
        print(self)
        self.no_of_rooms=5
        self.color="white"
        self.location='banglore'
        self.floor=2
        self.area=100      #variable attribute properties

    def shelter(self):      #method
        print('living')

h1=House()      #object or instance
print(h1)
print(h1.no_of_rooms)
print(h1.color)
print(h1.location)
print(h1.floor)
h1.shelter()
h2=House()
print(h2)
print(h2.color)
```

```
<__main__.House object at 0x00000291B3570340>
<__main__.House object at 0x00000291B3570340>
5
white
banglore
2
living
```

In [15]:

```
#init method is called as constructor
#it will get automatically called once we create the object
#init method will assign object address to self
```

In [17]:

```
class Test:
    def __init__(self):
        self.a=40

t1=Test()
t2=Test()
t3=Test()
t1.a=t2.a+t3.a
t2.a=t1.a-20
t3.a=t2.a+t1.a
print(t1.a)
print(t2.a)
print(t3.a)
```

80
60
140

instance variable and class variable

In [24]:

```
class Circle:
    pi=3.14      #class variable
    def __init__(self, radius):
        self.radius=radius      #instance variable

    def area(self):
        print(self.pi*(self.radius)**2)

c1=Circle(10)
c1.area()
print(c1.pi)  #object
print(Circle.pi)
#print(Circle.radius) #class name
```

314.0
3.14
3.14

In [23]:

```
#types of variable
#instance and class
#variable define inside the class and outside of all method
#varibale define inside the object or self as prefix
#instance variable can be access through object
#we can access class variable through object and class
#we cann't access instance variable through class name
```

instance method ,class method and static method

In [34]:

```

class Human:
    def eat(self):          #instance method
        print('human can eat')

    @classmethod             #decorator
    def move(cls):           #class method
        print('human can move')

    @staticmethod            #decorator
    def walk():              #static method
        print('human can walk')

h1=Human()
h1.eat()
h1.move()
Human.walk()
Human.move()
#Human.eat()

```

human can eat
 human can move
 human can walk
 human can move

In [35]:

```

#instance method having self as parameter
#we can call instance method using object
#using class name we can't call instance method
#we define class method using cls as parameter
#we can call class method using object and class name
#we should always mentioned @classmethod decorator above the method name
#we always mentioned @staticmethod as decorator in above of the method
#no need to pass any parameter to the static method
#we can call static method with the help of class name and not with object

```

Encapsulation

In [37]:

```
#binding logical data together in one class
```

In [38]:

```

doctor_name="xyz"
patient_name='abc'
exp=10
patient_age=27

def admit():
    print('admitting patient to the hospital')

def surgery():
    print('performing operations')

```

In [39]:

```
class Doctor:
    def __init__(self):
        self.doctor_name='xyz'
        self.exp=10

    def surgery(self):
        print('performing surgery')

class patient:
    def __init__(self):
        self.patient_name='abc'
        self.patient_age=27

    def admit(self):
        print('admitting patient to the hospital')
```

In [40]:

```
#data hiding
```

In [41]:

```
class A:
    def m1(self):
        print('hi')

class B:
    def m2(self):
        print('hello')
```

In [45]:

```
class A:                      #parent class /base class/super class
    def m1(self):
        print('hi')

class B(A):                  #child class/derived class/sub class
    def m2(self):
        print('hello')

a=A()
a.m1()
#a.m2  #becuase m2 is define in class B
b=B()
b.m2()
b.m1()  #because we have inherited A
```

```
hi
hello
hi
```

In [46]:

```
#types of inheritance
#single Level inheritance
#multilevel inheritance
#hierarchical inheritance
#multiple inheritance
#hybrid or diamond inheritance
```

In [47]:

```
#single Level inheritance
#one parent class and one child class
class A:                      #parent class /base class/super class
    def m1(self):
        print('hi')

class B(A):                    #child class/derived class/sub class
    def m2(self):
        print('hello')
```

In [57]:

```
#multilevel inheritance
class A:                      #parent class /base class/super class
    def m1(self):
        print('hi')

class B(A):                    #child class/derived class/sub class
    def m2(self):
        print('hello')

class C(B):                    #child class/derived class/sub class
    def m3(self):
        print('welcome')

a=A()
#a.m3()
b=B()
b.m1()
b.m2()
#b.m3()
c=C()
c.m3()
c.m2()
c.m1()
```

```
hi
hello
welcome
hello
hi
```

In [68]:

```
#hierarchical inheritance
class A:                      #parent class /base class/super class
    def m1(self):
        print('hi')

class B(A):                    #child class/derived class/sub class
    def m2(self):
        print('hello')

class C(A):                    #child class/derived class/sub class
    def m3(self):
        print('welcome')

a=A()
a.m1()
#a.m3() a.m2()
b=B()
b.m1()
b.m2()
#b.m3()
c=C()
c.m1()
c.m3()
#c.m2()
```

```
hi
hi
hello
hi
welcome
```

In [77]:

```
#multiple inheritance
class A:                      #parent class /base class/super class
    def m1(self):
        print('hi')

class B:                      #child class/derived class/sub class
    def m2(self):
        print('hello')

class C(A,B):                  #child class/derived class/sub class
    def m3(self):
        print('welcome')

a=A()
a.m1()
#a.m3() a.m2()
b=B()
b.m2()
#b.m1()
#b.m3()
c=C()
c.m3()
c.m1()
c.m2()
```

```
hi
hello
welcome
hi
hello
```

In [87]:

```
#hybrid or diamond inheritance
class A:                      #parent class /base class/super class
    def m1(self):
        print('hi')

class B(A):                   #child class/derived class/sub class
    def m2(self):
        print('hello')

class C(A):                   #child class/derived class/sub class
    def m3(self):
        print('welcome')

class D(B,C):                 #child class/derived class/sub class
    def m4(self):
        print('welcome')

a=A()
b=B()
c=C()
d=D()
a.m1()
#a.m4()
#b.m4()
d.m1()
```

```
hi
hi
```

In []: