

From Scratch

August 11, 2023

```
[1]: print('Hello World')
```

Hello World

1 1. Variable

```
[2]: a=10
```

```
[3]: b=20
```

```
[4]: print(a,b)
     print(id(a),id(b)) # memory addr
```

10 20
140319984563616 140319984563936

```
[5]: print('a')
     print('a',a,b)
```

a
a 10 20

```
[6]: c='hello'
     print(c)
```

hello

```
[7]: a1=20
     1a=20
```

```
File "<ipython-input-7-38e24ea1db44>", line 2
1a=20
  ^
SyntaxError: invalid syntax
```

1.0.1 String concatenation

APPENDING TWO OR MULTIPLE VARIABLE

```
[8]: a= 'Hello'  
     b='World'  
     a+ ' '+b
```

```
[8]: 'Hello World'
```

```
[9]: c=20  
     a+c
```

```
↳ -----  
  
      TypeError                                Traceback (most recent call↳  
↳last)                                         
  
      <ipython-input-9-0df1bald4cf2> in <module>  
          1 c=20  
      ----> 2 a+c  
  
      TypeError: can only concatenate str (not "int") to str
```

1.1 Arithmetic operators (+ - * / // % **)

```
[10]: a=20  
  
      b=5  
  
      print('a'+b')  
  
      print(a+b)  
  
      print(a*b)  
  
      print(a/b)  
  
      print(a-b)  
  
      print(a%b)    # Modulus will return remainder  
  
      print(a**b)   # Exponenets 20*20*20.... 5 times
```

```
print(a//b) # Floor division

print(10%7)

print(5**3)

print(50//7) # should be 7.142857....
```

```
ab
25
100
4.0
15
0
3200000
4
3
125
7
```

1.2 Assignment operator (= += -=)

```
[11]: x=5
      print(x)
```

```
5
```

```
[12]: x+=7
      print(x)
```

```
12
```

```
[13]: x-=8
      print(x)
```

```
4
```

1.2.1 Comparison operator (== , != , > , < , >= , <=)

```
[14]: a=8
      b=10
      a==b
```

```
[14]: False
```

```
[15]: a!=b
```

```
[15]: True
```

```
[16]: a>b
```

```
[16]: False
```

```
[17]: a<b
```

```
[17]: True
```

```
[18]: a>=b
```

```
[18]: False
```

```
[19]: a<=b
```

```
[19]: True
```

1.2.2 logical operator (and or not)

```
[20]: a=5  
a<8 and a>6
```

```
[20]: False
```

```
[21]: a==5 and a<6
```

```
[21]: True
```

```
[22]: x=10  
  
x==10 and x>9 and x<20
```

```
[22]: True
```

```
[23]: a<8 or b>10
```

```
[23]: True
```

```
[24]: x=10  
y=20  
  
print(x!=y or x>y and x<=y)
```

True

```
[25]: not (x!=y or x>y and x<=y)
```

[25]: False

```
[26]: x!=y
```

[26]: True

```
[27]: not x!=y
```

[27]: False

1.3 Membership operator (in , not in)

```
[28]: X='Hello'  
print('e' in X)
```

True

```
[29]: print('L' in X)
```

False

```
[30]: print('L' not in X)
```

True

```
[31]: l=[10,20,30,29]  
print(40 in l)
```

False

```
[32]: print(30 not in l)
```

False

1.4 Identity operator (is , is not)

```
[33]: x=10  
y=10  
x is y
```

[33]: True

```
[34]: x == y
```

```
[34]: True
```

```
[35]: a=3  
      b=6  
      a is b
```

```
[35]: False
```

```
[36]: a==b
```

```
[36]: False
```

```
[37]: x is not y
```

```
[37]: False
```

```
[38]: x!=y
```

```
[38]: False
```

1.5 bitwise operator (& , | , ^) or (and , or , Xor)

1.6 Truth Table

1= True 0= False

A	B	A&B	A B	A^B
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

```
[39]: x=10 # bin  
      print(bin(10))  
      y=8  
      print(bin(8))
```

```
0b1010
```

```
0b1000
```

```
[40]: print(x&y, bin(x&y))
```

```
8 0b1000
```

```
[41]: print(x|y,bin(x|y))
```

```
10 0b1010
```

```
[42]: print(x^y, bin(x^y))
```

```
2 0b10
```

```
[43]: a=18
      b=9
      if a%b=0:
          print('Divisible')
      else:
          print('NA')
```

```
File "<ipython-input-43-f77ffebfa341>", line 3
if a%b=0:
    ^
SyntaxError: invalid syntax
```

2 Data types in python

Data types list:

- . Numeric
 - Integer
 - Float
 - Complex Numbers
- . Sequence type
 - String
 - List (Mutable)
 - Tuple
- . Dictionary (Mutable)
- . Set

2.0.1 Integer

```
[44]: a= 10  
      type(a)
```

```
[44]: int
```

2.0.2 Float

```
[45]: a=2.5  
      type(a)
```

```
[45]: float
```

2.0.3 Complex

```
[46]: x=1+3j  
      type(x)
```

```
[46]: complex
```

2.0.4 String

```
[47]: ## ' ', " ", "  
  
      x='Apple'  
      type(x)
```

```
[47]: str
```

```
[48]: a='Hello@123'  
      type(a)
```

```
[48]: str
```

```
[49]: A=''  
      Hello  
      Welcome to India  
      ''  
      print(A)
```

Hello

Welcome to India

2.0.5 List

```
[50]: A=['stable', 10, 20]
      type(A)
```

[50]: list

```
[51]: A.append(10)
      A
```

[51]: ['stable', 10, 20, 10]

```
[52]: A.remove(10)
```

```
[53]: X=[10,'Apple', 'Orange', 30]
      X
```

[53]: [10, 'Apple', 'Orange', 30]

```
[54]: X[2]='Grapes'
```

```
[55]: X
```

[55]: [10, 'Apple', 'Grapes', 30]

```
[56]: X.append(100)
```

```
[57]: X
```

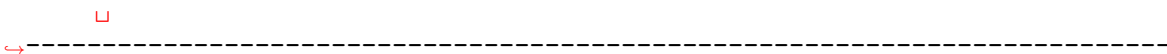
[57]: [10, 'Apple', 'Grapes', 30, 100]

Tuple

```
[58]: s=(10,20,'Hello')
      type(s)
```

[58]: tuple

```
[59]: s.append(10)
```



```
AttributeError                                Traceback (most recent call_
↳last)
```

```
<ipython-input-59-bc62c451aa5b> in <module>
----> 1 s.append(10)
```

```
AttributeError: 'tuple' object has no attribute 'append'
```

```
[60]: s[0]
      s[2]
```

```
[60]: 'Hello'
```

2.0.6 Dictionary

```
[61]: Empty_Dist={}
      type(Empty_Dist)
```

```
[61]: dict
```

```
[62]: Siblings={'Zaman':27, 'Yasir': 24, 'Misba': 21}
      type(Siblings)
```

```
[62]: dict
```

```
[63]: print(Siblings['Zaman'])
```

```
27
```

```
[64]: Siblings
```

```
[64]: {'Zaman': 27, 'Yasir': 24, 'Misba': 21}
```

2.0.7 Set (an unordered collection)

```
[65]: S1={1,2,3,6,7,6,1,1} ## only values are there
      type(S1)
```

```
[65]: set
```

```
[66]: S1 #remove duplicates values
```

```
[66]: {1, 2, 3, 6, 7}
```

3 Getting user input & Type casting

Key points: ### . inout() ### . int() ### . float() ### . e val()

```
[67]: ### input casting
```

```
a=input('Enter the value:-')
a
```

Enter the value:- 12

```
[67]: '12'
```

```
[68]: b=input('Enter the value2:-')
      print(a+b)
```

Enter the value2:- 23

1223

```
[69]: ## type casting
```

```
a=int(input('Enter the value:-'))
```

Enter the value:- 13

```
[70]: b=int(input('Enter the value2:-'))
```

Enter the value2:- 45

```
[71]: a+b
```

```
[71]: 58
```

float casting a=float(input('Enter the value:-')) b=float(input('Enter the value2:-'))

```
[72]: a=float(input('Enter the value:-'))
      b=float(input('Enter the value2:-'))
```

Enter the value:- 1

Enter the value2:- 2

```
[73]: a+b
```

```
[73]: 3.0
```

```
[74]: ## e val casting
```

```
a=eval(input('Enter the value:-'))
b=eval(input('Enter the value2:-'))
```

```
Enter the value:- 45
Enter the value2:- 12
```

```
[75]: a+b
```

```
[75]: 57
```

```
[76]: x=eval(input('Enter the value:-'))
      z=eval(input('Enter the value2:-'))
      x+z
```

```
Enter the value:- 100
Enter the value2:- 12
```

```
[76]: 112
```

4 Conditional statement in Python

Key points: ### . If ### . If Else ### . If Elif Else

1. If statement

```
if(conditional statement):
    (statement to be executed)
```

```
[77]: #eg.
      a=10
      if a%2==0:
          print('a is an even number')
```

```
a is an even number
```

```
[78]: z=100
      if z%10==0:
          print("10 is factor of z")
```

```
10 is factor of z
```

```
[82]: Z=eval(input("Enter the value:-"))
      if Z%2!=0:
          print("No is odd")
```

```
Enter the value:- 45
```

```
No is odd
```

```
[81]: A=eval(input("Enter your number1:-"))
      if A%2==0:
          print("No is even")
```

Enter your number1:- 34

No is even

2. If Else statement

```
if(conditional statement):
    (statement to be executed)
else:
    (alternate statement)
```

```
[83]: #eg
      A=eval(input("Enter the value:-"))
      if A%2==0:
          print("No. is Even")
      else:
          print("No. is Odd")
```

Enter the value:- 13

No. is Odd

```
[84]: Z=eval(input("Enter the value:-"))
      if Z%2==0:
          print("No. is Even")
      else:
          print("No. is Odd")
```

Enter the value:- 45

No. is Odd

3. If Else Elif statement

```
if(conditional #1):
    (statement 1)

elif(condition 2):
    (statement# 2)

elif(cond 3):
    (3rd statement)

else:
    (alternate statement)
```

```
[85]: #eg.  
M=eval(input("Enter your marks:-"))  
if M>=60:  
    print("first div")  
elif 48<M<60:  
    print("Second div")  
elif 33<M<48:  
    print("Third div")  
else:  
    print("Student is Fail")
```

Enter your marks:- 78

first div

4.0.1 How to build simple Calculator in Python using conditions

```
[86]: print('''  
+ Add  
- Subtract  
* Multiply  
/ Divide''')
```

+ Add
- Subtract
* Multiply
/ Divide

```
[87]: A=int(input("Enter first val:1"))  
B=int(input("Enter second val:2"))  
opr=input("Enter the operator")  
if opr=="+":  
    print(A+B)  
elif opr=="-":  
    print(A-B)  
elif opr=="*":  
    print(A*B)  
elif opr=="/":  
    print(A/B)  
else:  
    print("invalid operation")
```

Enter first val:1 56

Enter second val:2 78

Enter the operator +

134

```
[88]: X=eval(input("Enter first val:1"))
      Y=eval(input("Enter second val:2"))
      opr=input("Enter the operator")
      if opr=="+":
          print(X+Y)
      elif opr=="-":
          print(X-Y)
      elif opr=="*":
          print(X*Y)
      elif opr=="/":
          print(X/Y)
      else:
          print("invalid operation")
```

```
Enter first val:1 14
Enter second val:2 72
Enter the operator *
1008
```

```
[89]: ## If we are not using IF Elif Else then,

A=eval(input("Enter first val:1"))
B=eval(input("Enter second val:2"))
opr1=input("Enter the operator")
if opr1=="+":
    print(A+B)
if opr1=="-":
    print(A-B)
if opr1=="*":
    print(A*B)
if opr1=="/":
    print(A/B)
if opr1!='+' and opr1!='-' and opr1!='*' and opr1!='/' :
    print("Invalid Operation")
```

```
Enter first val:1 13
Enter second val:2 67
Enter the operator -
-54
```

5 For Loop with Range() in Python

It is used for repeatation and iteration

Range function which is used for number it is the most important function for creating a For Loop condition

```
[90]: range(5) #start from 0 and #conditoin<5 increment 1
```

```
[90]: range(0, 5)
```

```
[91]: range(1,6) #start=1 #conditon<6 #increment 1 #range(i, j) produces i, i+1, i+2, ..., j-1.
```

```
[91]: range(1, 6)
```

```
[92]: range(1,6,2) #start=1 #condition<6 #increment 2
```

```
[92]: range(1, 6, 2)
```

```
[93]: #help(range)
```

```
[94]: for n in range(5):  
      print(n)
```

```
0  
1  
2  
3  
4
```

```
[95]: for n in range(5):  
      print('Hello world')
```

```
Hello world  
Hello world  
Hello world  
Hello world  
Hello world
```

```
[96]: for N in range(5,15):  
      print(N)
```

```
5  
6  
7  
8  
9  
10  
11  
12  
13  
14
```



```
[97]: for N in range(5,40,5):  
      print(N)
```

```
5  
10  
15  
20  
25  
30  
35
```

```
[98]: for N in range(1,11):  
      print('2*',N,'=',2*N)
```

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

```
[99]: for z in range(1,15):  
      print('5*',z,'=',5*z)
```

```
5* 1 = 5  
5* 2 = 10  
5* 3 = 15  
5* 4 = 20  
5* 5 = 25  
5* 6 = 30  
5* 7 = 35  
5* 8 = 40  
5* 9 = 45  
5* 10 = 50  
5* 11 = 55  
5* 12 = 60  
5* 13 = 65  
5* 14 = 70
```

```
[100]: ## Range for reverse case  
      range(10,0)  
      range(10,0,-1)
```

```
[100]: range(10, 0, -1)
```

```
[101]: for n in range(10,0,-1):  
        print(n)
```

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

```
[102]: for n in range(10,2,-2):  
        print(n)
```

```
10  
8  
6  
4
```

```
[103]: ## Reverse looping  
for a in range(10,0,-1):  
    print("2*",a,'=',2*a)
```

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

6 While loop

```
.start  
.condition
```

```
[104]: 4-6
```

```
[104]: -2
```

```
[105]: i=1
while i<=10:
    print(i, '.hello world')
    i=i+1

print(i)
```

```
1 .hello world
2 .hello world
3 .hello world
4 .hello world
5 .hello world
6 .hello world
7 .hello world
8 .hello world
9 .hello world
10 .hello world
11
```

```
[106]: ## Reverse while loop
```

```
[107]: a=10
while a>=1:
    print(a, 'Reverse loop')
    a=a-1

a
```

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

```
[107]: 0
```

7 20. String Indexing & String slicing in Python

```
[108]: w="String Indexing"  #For forward 0-.... and reverse -1 to..  
w[-7]
```

```
[108]: 'n'
```

```
[109]: print(w[0:7]) # slicing
```

String

```
[110]: print(len(w))
```

15

```
[111]: print(w[0: :2])
```

Srn neig

```
[112]: print(w[-1])
```

g

```
[113]: print(w[-1::-1])
```

gnixednI gnirtS

```
[114]: print(w[-1::-2])
```

gien nrS

8 21. String Iteration in Python

```
[115]: Z='My name is Hasan Rizvi'  
t=len(Z)  
t
```

```
[115]: 22
```

```
[116]: Z[2:18:2]
```

```
[116]: ' aeI aa '
```

```
[117]: for a in range(t):  
        print(a, Z[a])
```

```
0 M
1 y
2
3 n
4 a
5 m
6 e
7
8 i
9 s
10
11 H
12 a
13 s
14 a
15 n
16
17 R
18 i
19 z
20 v
21 i
```

```
[118]: ## taking reverse
Z= 'My name is Hasan Rizvi'
Z=Z[-1::-1]
Z
```

```
[118]: 'ivziR nasaH si eman yM'
```

```
[119]: A=len(Z)
A
```

```
[119]: 22
```

```
[120]: for a in range(A):
        print(a,Z[a])
```

```
0 i
1 v
2 z
3 i
4 R
5
6 n
7 a
8 s
9 a
```

```
10 H
11
12 s
13 i
14
15 e
16 m
17 a
18 n
19
20 y
21 M
```

```
[121]: for a in range(A):
        print(Z[a])
```

```
i
v
z
i
R
```

```
n
a
s
a
H
```

```
s
i
```

```
e
m
a
n
```

```
y
M
```

```
[122]: ## taking reverse
S= 'My name is Hasan Rizvi'
t=len(S)
t
```

```
[122]: 22
```

```
[123]: for a in range(t-1,-1,-1):  
        print(a,S[a])
```

```
21 i  
20 v  
19 z  
18 i  
17 R  
16  
15 n  
14 a  
13 s  
12 a  
11 H  
10  
9 s  
8 i  
7  
6 e  
5 m  
4 a  
3 n  
2  
1 y  
0 M
```

```
[124]: for a in range(t-1,-1,-1):  
        print(S[a])
```

```
i  
v  
z  
i  
R  
  
n  
a  
s  
a  
H  
  
s  
i  
  
e  
m  
a  
n
```

y
M

```
[125]: # iteration without using range  
S='My name is Hasan Rizvi'
```

```
[126]: for a in S:  
        print(a)
```

M
y

n
a
m
e

i
s

H
a
s
a
n

R
i
z
v
i

```
[127]: S=S[::-1]  
for a in S:  
    print(a,a)
```

i i
v v
z z
i i
R R

n n
a a
s s
a a
H H


```
s s
i i

e e
m m
a a
n n

y y
M M
```

9 22. Lower, Upper, Title and capitalize String functions in Python

```
[128]: S='syed HaSAn sAdIq rizvi'
      n=S.lower()
      n
```

```
[128]: 'syed hasan sadiq rizvi'
```

```
[129]: n=S.upper()
      n
```

```
[129]: 'SYED HASAN SADIQ RIZVI'
```

```
[130]: n=S.title()
      n
```

```
[130]: 'Syed Hasan Sadiq Rizvi'
```

```
[131]: n=S.capitalize()
      n
```

```
[131]: 'Syed hasan sadiq rizvi'
```

10 23. Find, Index, Isalpha, Isdigit & Isalnum string functions in Python

```
[132]: #find
      S='Welcome'
      print(S.find('l'))
```

2

```
[133]: print(S.find('e'))
```

1

```
[134]: print(S.find('el'))
```

1

```
[135]: print(S.find('e',2)) #start from index# 2
```

6

```
[136]: print(S.find('a')) #if the value is not present it will give -1
```

-1





```
[137]: # Index  
A='Samsung Apple Grapes'  
print(A.index('p'))
```

9

```
[138]: print(A.index('p',11))
```

17

```
[139]: print(A.index('z'))
```

```
↳  -----  
↳  -----  
  
ValueError                                Traceback (most recent call↳   
↳  last)  
  
    <ipython-input-139-938e4b9013b6> in <module>  
----> 1 print(A.index('z'))  
  
ValueError: substring not found
```

```
[140]: # isalpha  
A='Samsung Apple Nokia'  
print(A.isalpha())
```

False

```
[141]: # isalpha
z='Samsung'
print(z.isalpha())
```

True

```
[142]: # isdigit
A='12357'
print(A.isdigit())
```

True

```
[143]: # isdigit
A='Samsung Apple Nokia'
print(A.isdigit())
```

False

```
[144]: # isalnum
A='Samsung123'
print(A.isalnum())
```

True

```
[145]: # isalnum
A='Apple 123'
print(A.isalnum())
```

False

```
[146]: # isalnum
A='Samsung'
print(A.isalnum())
```

True

11 24. Ord[] and Chr[] functions in Python

- . Chr: it converts int to chr and return string
- . Ord: it converts chr to int and return int

```
[147]: y=chr(65)
print(type(y),y)
```

<class 'str'> A

```
[148]: a=70
       print(chr(a))
```

F

```
[149]: a='A'
       print(ord(a))
```

65

```
[150]: a='Z'
       print(ord(a))
```

90

```
[151]: a='f'
       print(ord(a))
```

102

12 25. String format method in Python

It gives an option to enter any value in between string

```
[152]: txt1= "My name is {fname} {lname}".format(fname='Hasan',lname='Rizvi')
       txt1
```

```
[152]: 'My name is Hasan Rizvi'
```

```
[153]: txt2= "My name is {0} {1}".format('Hasan','Rizvi')
       txt2
```

```
[153]: 'My name is Hasan Rizvi'
```

```
[154]: txt3="My name is {} {}".format('Hasan','Rizvi')
       txt3
```

```
[154]: 'My name is Hasan Rizvi'
```

```
[155]: w= "Welcome to {} world of {} to {} lovers".format('the','coding','all coding')
       w
```

```
[155]: 'Welcome to the world of coding to all coding lovers'
```

```
[156]: txt2= "My name is {1} {0}".format('Hasan','Rizvi')
       txt2
```

```
[156]: 'My name is Rizvi Hasan'
```

```
[157]: txt1= "My name is {a} {b}".format(a='Hasan',b='Rizvi')
txt1
```

```
[157]: 'My name is Hasan Rizvi'
```

```
[158]: txt1= "My name is {a:^15} {b} and I'm a Data Scientist".
↳format(a='Hasan',b='Rizvi')
txt1
```

```
[158]: "My name is      Hasan      Rizvi and I'm a Data Scientist"
```

```
[159]: txt1= "My name is {a:>15} {b} and I'm a Data Scientist".
↳format(a='Hasan',b='Rizvi')
txt1
```

```
[159]: "My name is              Hasan Rizvi and I'm a Data Scientist"
```

```
[160]: txt1= "My name is {a:<15} {b} and I'm a Data Scientist".
↳format(a='Hasan',b='Rizvi')
txt1
```

```
[160]: "My name is Hasan              Rizvi and I'm a Data Scientist"
```

```
[161]: txt1= "My name is {a:10} {b:^10} and I'm a Data Scientist".
↳format(a='Hasan',b='Rizvi')
txt1
```

```
[161]: "My name is Hasan      Rizvi      and I'm a Data Scientist"
```

13 26. List In Python

- . it is mutable
- . , seperated
- . [...]
- . a=[12,'app']

```
[162]: l=[12,'appl',1,2,3]
```

```
[163]: print(type(l))
```

```
<class 'list'>
```

```
[164]: print(l[3])
```

2

```
[165]: l2=[1,2,3,'A',[4,5,6,'B']]
```

```
[166]: print(l2[1])
```

2

```
[167]: print(l2[4][3])
```

B

```
[168]: print(l2[3])
```

A

```
[169]: print(l2[-1][2])
```

6

```
[170]: #Slicing  
print(l2[0:2])
```

[1, 2]

```
[171]: l3=[1,2,3,3,4,5,'A',[1,2]]  
print(l3[0:4])
```

[1, 2, 3, 3]

```
[172]: print(l3[3:])
```

[3, 4, 5, 'A', [1, 2]]

```
[173]: l3
```

```
[173]: [1, 2, 3, 3, 4, 5, 'A', [1, 2]]
```

```
[174]: print(l3[1:6:2]) ## step slicing
```

[2, 3, 5]

```
[175]: print(l3[-1::-2])
```

[[1, 2], 5, 3, 2]

```
[176]: print(l3[-2:-5:-1])
```

['A', 5, 4]

```
[177]: print(l3[-1::-1])
```

```
[[1, 2], 'A', 5, 4, 3, 3, 2, 1]
```

14 27. List Iteration in Python

. getting a data of list at once

```
[178]: l=[10,2,3,4,5,6]
      t=len(l)
      t
```

```
[178]: 6
```

```
[179]: for a in range(t):
      print(l[a])
```

```
10
2
3
4
5
6
```

```
[180]: # without using range
      S=[1,2,3,4,5,5]
      for a in S:
          print(S[a])
```

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

```
[181]: #Reverse list
```

```
n=[2,3,1,4,5]
l=len(n)
1
```

```
[181]: 5
```

```
[182]: for a in range(l-1,-1,-1):
      print(n[a])
```

5
4
1
3
2

15 28. Del, Pop, Remove, clear and update function in List

```
[183]: l=[12,23,24,25,26]  
1
```

```
[183]: [12, 23, 24, 25, 26]
```

```
[184]: del l[3]
```

```
[185]: 1
```

```
[185]: [12, 23, 24, 26]
```

```
[186]: del l[0]
```

```
[187]: 1
```

```
[187]: [23, 24, 26]
```

```
[188]: l2=[10,20,30,40,50]  
l2.pop(4)
```

```
[188]: 50
```

```
[189]: l2
```

```
[189]: [10, 20, 30, 40]
```

```
[190]: l2.pop(0)
```

```
[190]: 10
```

```
[191]: l2
```

```
[191]: [20, 30, 40]
```

```
[192]: l3=[1,2,3,4,12,3]  
l3.remove(4)
```

```
[193]: l3
```


[193]: [1, 2, 3, 12, 3]

```
[194]: 13.remove(12)
```

```
[195]: 13
```

[195]: [1, 2, 3, 3]

```
[196]: 13.clear()
```

```
[197]: 13
```

[197]: []

```
[198]: 14=[1,2,3,4,5,6]
```

```
[199]: 14
```

[199]: [1, 2, 3, 4, 5, 6]

```
[200]: 14.clear()
```

```
[201]: 14
```

[201]: []

```
[202]: # update  
A=[20,12,23,45,56]  
A[0]=34 #change the value of 0 index with 34
```

```
[203]: A
```

[203]: [34, 12, 23, 45, 56]

15.0.1 More functions of list:

- . insert()
- . append()
- . extends()

```
[204]: l=[12,13,14,15]  
l.insert(4,16)  #(index, value need to be added)
```

```
[205]: l
```

[205]: [12, 13, 14, 15, 16]

```
[206]: l.append(20) # value of will be added
1
```

```
[206]: [12, 13, 14, 15, 16, 20]
```

```
[207]: m=[17,18]
l.append(m)
```

```
[208]: 1
```

```
[208]: [12, 13, 14, 15, 16, 20, [17, 18]]
```

```
[209]: 1
```

```
[209]: [12, 13, 14, 15, 16, 20, [17, 18]]
```

```
l.extend(m) l
```

16 29. Python list comprehension- Elegant way to crate lists

```
[210]: l=[]
for a in range(1,101):
    l.append(a)
```

```
[211]: print(l)
```

```
[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, 39, 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, 76, 77, 78, 79, 80, 81, 82,
83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]
```

```
[212]: n=[m for m in range(1,101)] ## using list comprehension
print(n)
```

```
[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, 39, 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, 76, 77, 78, 79, 80, 81, 82,
83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]
```

```
[213]: n=[h for z in range(1,101)] ## using list comprehension
print(n)
```

↳ -----

NameError Traceback (most recent call↳
↳last)

```
<ipython-input-213-663493c4badc> in <module>
----> 1 n=[h for z in range(1,101)]  ## using list comprehension
      2 print(n)
```

```
<ipython-input-213-663493c4badc> in <listcomp>(.0)
----> 1 n=[h for z in range(1,101)]  ## using list comprehension
      2 print(n)
```

NameError: name 'h' is not defined

```
[214]: n=[m for m in range(1,101) if m%2==0]  ## using list comprehension with↳
↳condition
print(n)
```

```
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42,
44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82,
84, 86, 88, 90, 92, 94, 96, 98, 100]
```

```
[215]: #Que. create a table of 5 using for loop and list
z=[a for a in range(5,51) if a%5==0]
print(z)
```

```
[5, 10, 15, 20, 25, 30, 35, 40, 45, 50]
```

```
[216]: #Que. create a table of 8 using for loop and list
t8=[a for a in range(8,81) if a%8==0]
print(t8)
```

```
[8, 16, 24, 32, 40, 48, 56, 64, 72, 80]
```

```
[217]: s="Hello_World"
t=[a for a in s]
print(t)
```

```
['H', 'e', 'l', 'l', 'o', '_', 'W', 'o', 'r', 'l', 'd']
```

17 30. Count, Max, Min, Sort, Reverse & Index list function

```
[218]: l=[1,2,2,3,4,5,2,4,6,8,8,1,4,5,3]
      l.count(2)
```

```
[218]: 3
```

```
[219]: l.count(3)
```

```
[219]: 2
```

```
[220]: m=max(l)
      m
```

```
[220]: 8
```

```
[221]: l=["Hello","World","Welcome","Zebra"]
      a=max(l)
      a
```

```
[221]: 'Zebra'
```

```
[222]: b=min(l)
      b
```

```
[222]: 'Hello'
```

```
[223]: l=[1,2,2,3,4,5,2,4,6,8,8,1,4,5,3]
      l.sort()
```

```
[224]: l
```

```
[224]: [1, 1, 2, 2, 2, 3, 3, 4, 4, 4, 5, 5, 6, 8, 8]
```

```
[225]: a=[23,1,3,4,53,2,5,78,44,3,90,32]
      a.sort()
      a
```

```
[225]: [1, 2, 3, 3, 4, 5, 23, 32, 44, 53, 78, 90]
```

```
[226]: a.reverse()
      a
```

```
[226]: [90, 78, 53, 44, 32, 23, 5, 4, 3, 3, 2, 1]
```

```
[227]: l.reverse()
      l
```

```
[227]: [8, 8, 6, 5, 5, 4, 4, 4, 3, 3, 2, 2, 2, 1, 1]
```

```
[228]: l.index(5)
```

```
[228]: 3
```

```
[229]: a=[23,1,3,4,53,2,5,78,44,3,90,32]
a.index(78)
```

```
[229]: 7
```

18 31. Zip function-iterate over 2+ lists at the same time

```
[230]: l1=[1,2,3,4]
l2=[5,6,7,8]
for a,b in zip(l1,l2):
    print(a,b)
```

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

```
[231]: a=[1,2,3,4,5]
b=['Jan','Feb','Mar','April']
for i,j in zip(a,b):
    print(i,j)
```

```
1 Jan
2 Feb
3 Mar
4 April
```

```
[232]: ## without using zip
l=[10,20,30,40]
l1=[3,4,77,88]
```

```
[233]: t=len(l)
t
```

```
[233]: 4
```

```
[234]: for h in range(t):
        print(l[h],l1[h])
```

```
10 3
20 4
30 77
40 88
```

19 32. Python program to convert string to a list

```
[235]: n=input('enter the value')
      n
```

enter the value 14

```
[235]: '14'
```

```
[236]: l=n.split()
```

```
[237]: l
```

```
[237]: ['14']
```

```
[238]: # Que. Convert multiple input from user into list.

l=[]
for a in range(1,4):
    n=input('Enter the value' + str(a)+":-")
    l.append(n)
```

Enter the value1:- 78

Enter the value2:- 90

Enter the value3:- 45

```
[239]: print(l)
```

```
['78', '90', '45']
```

20 34. Implement a stack and queue using a list data type

20.1 Stack using list in python

- . Stack is a Linear data structure.
 - . Stores items in a LIFO or FILO
 - eg. plates in the wedding's etc.

Stack Operation:-

- Push: Inserting an elements.

- Pop: Deletion an element(last element)
- Peek:Display the last element
- Display: Display list
- Exit

```
[240]: l=[]
while True:
    c=int(input('''
    1 Push Elements
    2 Pop Elements
    3 Peek Element
    4 Display Stack
    5 Exit
    '''))
    if c==1:
        n=input("Enter the value:-");
        l.append(n)
        print(l)

    elif c==2:
        if len(l)==0:
            print("Empty Stack")
        else:
            p=l.pop()
            print(p)
            print(l)

    elif c==3:
        if len(l)==0:
            print("empty stack")
        else:
            print("Last stack value",l[-1])

    elif c==4:
        print("Display Stack",l)

    elif c==5:
        break;
    else:
        print('invalid operation')
```

```
1 Push Elements
2 Pop Elements
3 Peek Element
4 Display Stack
5 Exit
1
```

Enter the value:- 56

['56']

- 1 Push Elements
- 2 Pop Elements
- 3 Peek Element
- 4 Display Stack
- 5 Exit

1

Enter the value:- 89

['56', '89']

- 1 Push Elements
- 2 Pop Elements
- 3 Peek Element
- 4 Display Stack
- 5 Exit

145

invalid operation

- 1 Push Elements
- 2 Pop Elements
- 3 Peek Element
- 4 Display Stack
- 5 Exit

1

Enter the value:- 45

['56', '89', '45']

- 1 Push Elements
- 2 Pop Elements
- 3 Peek Element
- 4 Display Stack
- 5 Exit

2

45

['56', '89']

- 1 Push Elements
- 2 Pop Elements
- 3 Peek Element
- 4 Display Stack


```

5 Exit
3
Last stack value 89

1 Push Elements
2 Pop Elements
3 Peek Element
4 Display Stack
5 Exit
4
Display Stack ['56', '89']

1 Push Elements
2 Pop Elements
3 Peek Element
4 Display Stack
5 Exit
5

```

20.2 Queue using list in Python

- . Queue is a linear data structure
- . Stores item in a FIFO(first in first out) manner
- eg. Ticket counter

Queue operations:-

1. Enqueue: Adds an item to the queue.
2. Dequeue: Removes an item to the queue.
3. Front: Get the front item from queue.
4. Rear: Get the last item from

```

[241]: l=[]
while True:
    c=int(input('
        1 Push Elements
        2 Pop first Element
        3 Front Element
        4 Last Element
        5 Display Elements
        6 Exit
        '))
    if c==1:
        n=input("Enter an Element");
        l.append(n)
        print(l)

```

```

elif c==2:
    if len(l)==0:
        print("Empty Queue")
    else:
        del l[0]
        print("removing first element",l)

elif c==3:
    if len(l)==0:
        print("Empty Queue")
    else:
        print("First element is:-",l[0])

elif c==4:
    if len(l)==0:
        print('Empty queue')
    else:
        print("Last element is:-",l[-1])

elif c==5:
    if len(l)==0:
        print("Empty Queue")
    else:
        print("Elements are:-",l)

elif c==6:
    break;

```

```

1 Push Elements
2 Pop first Element
3 Front Element
4 Last Element
5 Display Elements
6 Exit
1
Enter an Element 57
['57']

```

```

1 Push Elements
2 Pop first Element
3 Front Element
4 Last Element

```

```
        5 Display Elements
        6 Exit
        1
Enter an Element 1
['57', '1']
```

```
        1 Push Elements
        2 Pop first Element
        3 Front Element
        4 Last Element
        5 Display Elements
        6 Exit
        1
Enter an Element 12
['57', '1', '12']
```

```
        1 Push Elements
        2 Pop first Element
        3 Front Element
        4 Last Element
        5 Display Elements
        6 Exit
        1
Enter an Element 14
['57', '1', '12', '14']
```

```
        1 Push Elements
        2 Pop first Element
        3 Front Element
        4 Last Element
        5 Display Elements
        6 Exit
        2
removing first element ['1', '12', '14']
```

```
        1 Push Elements
        2 Pop first Element
        3 Front Element
        4 Last Element
        5 Display Elements
        6 Exit
        3
First element is:- 1
```

```

1 Push Elements
2 Pop first Element
3 Front Element
4 Last Element
5 Display Elements
6 Exit
4

```

Last element is:- 14

```

1 Push Elements
2 Pop first Element
3 Front Element
4 Last Element
5 Display Elements
6 Exit
5

```

Elements are:- ['1', '12', '14']

```

1 Push Elements
2 Pop first Element
3 Front Element
4 Last Element
5 Display Elements
6 Exit
6

```

20.3 35. Dictionary in Python

```

. Mutable
. Unorderd
. Key | Value
. {}
eg. d={'Name':'Python'}

```

```

[250]: d={'Name':'Python','Fees':3500,'Duration':'3 Months'}
type(d)
d

```

```

[250]: {'Name': 'Python', 'Fees': 3500, 'Duration': '3 Months'}

```

```

[251]: d['Name']

```

```

[251]: 'Python'

```

```
[252]: d['Fees']
```

```
[252]: 3500
```

```
[253]: for n in d:  
        print(n)  
        print(d[n])
```

```
Name  
Python  
Fees  
3500  
Duration  
3 Months
```

20.4 36. Dictionary function in Python

- . get()
- . keys()
- . values()
- . items()
- . pop()
- . dict()
- . update()
- . clear()
- . del

```
[254]: d={'Name': 'Python',  
        'Fees': 3000,  
        'Duration': '2 Months'  
        }
```

```
[255]: n=d.get('Name')  
n
```

```
[255]: 'Python'
```

```
[256]: f=d.get('Fees')  
f
```

```
[256]: 3000
```

```
[257]: for a in d.keys():  
        print(a)
```

```
Name  
Fees
```

Duration

```
[258]: for b in d.values():  
        print(b)
```

Python

3000

2 Months

```
[259]: for z in d.items():  
        print(z)
```

('Name', 'Python')

('Fees', 3000)

('Duration', '2 Months')

```
[260]: for a,b in d.items():  
        print(a,': ',b)
```

Name : Python

Fees : 3000

Duration : 2 Months

```
[261]: del d['Fees']
```

```
[262]: d
```

```
[262]: {'Name': 'Python', 'Duration': '2 Months'}
```

```
[263]: a=d.pop('Name')  
d
```

```
[263]: {'Duration': '2 Months'}
```

```
[264]: a
```

```
[264]: 'Python'
```

```
[265]: d=dict(name='Python', Fees=3000)
```

```
[266]: d
```

```
[266]: {'name': 'Python', 'Fees': 3000}
```

```
[267]: d.update({'Fees':3500})  
d
```

```
[267]: {'name': 'Python', 'Fees': 3500}
```

```
[268]: d.clear()
```

```
[269]: d
```

```
[269]: {}
```

```
[270]: ## inserting new key in dictionary
d={'Name':'Python','Duration':'2 Months'
}
for a,b in d.items():
    print(a,':',b)
```

```
Name : Python
Duration : 2 Months
```

```
[271]: ## Adding Fees in d
d['Fees']=8000
```

```
[272]: d
```

```
[272]: {'Name': 'Python', 'Duration': '2 Months', 'Fees': 8000}
```

20.5 37. Python Nested Dictionary

```
[273]: course={'python':{'Fees':3000,'Duration':'2 Months'},
              'C':{'Fees':2000,'Duration':'1.5 Months'},
              'Java':{'Fees':1500,'Duration':'1 Month'}
}
print(course)
```

```
{'python': {'Fees': 3000, 'Duration': '2 Months'}, 'C': {'Fees': 2000,
'Duration': '1.5 Months'}, 'Java': {'Fees': 1500, 'Duration': '1 Month'}}
```

```
[274]: for a,b in course.items():
        print(a,b)
```

```
python {'Fees': 3000, 'Duration': '2 Months'}
C {'Fees': 2000, 'Duration': '1.5 Months'}
Java {'Fees': 1500, 'Duration': '1 Month'}
```

```
[275]: print(course['C'])
```

```
{'Fees': 2000, 'Duration': '1.5 Months'}
```

```
[276]: print(course['C']['Fees'])
```

```
2000
```

```
[277]: for a,b in course.items():  
        print(a,b['Fees'],b['Duration'])
```

```
python 3000 2 Months  
C 2000 1.5 Months  
Java 1500 1 Month
```

```
[278]: course['C']['Fees']=1800
```

```
[279]: course
```

```
[279]: {'python': {'Fees': 3000, 'Duration': '2 Months'},  
        'C': {'Fees': 1800, 'Duration': '1.5 Months'},  
        'Java': {'Fees': 1500, 'Duration': '1 Month'}}
```

20.6 38. Tuple in Python

- . Ordered data type
- . Immutable
- . ()

eg. a=(20,30,40,50)

```
[280]: a=(20,30,40,50)  
a
```

```
[280]: (20, 30, 40, 50)
```

```
[281]: type(a)
```

```
[281]: tuple
```

```
[282]: a[1]
```

```
[282]: 30
```

```
[283]: b=('app',10,30)  
type(b)
```

```
[283]: tuple
```

```
[284]: for c in a:  
        print(c)
```

```
20  
30
```


40
50

```
[285]: for c in range(1):  
        print(c)
```

```
↳ -----  
  
TypeError                                Traceback (most recent call↳  
↳last)  
  
  <ipython-input-285-7413bba6c627> in <module>  
----> 1 for c in range(1):  
      2     print(c)  
  
TypeError: 'list' object cannot be interpreted as an integer
```

```
[ ]: for c in range(1):  
      print(a[c])
```

```
[ ]: t=(10,20,30,40)  
     l=len(t)
```

```
[286]: for a in range(1):  
        print(t[a])
```

```
↳ -----  
  
TypeError                                Traceback (most recent call↳  
↳last)  
  
  <ipython-input-286-c91307ca034e> in <module>  
----> 1 for a in range(1):  
      2     print(t[a])  
  
TypeError: 'list' object cannot be interpreted as an integer
```

20.6.1 Functions used in tuple

- . Min
- . Max
- . Count
- . Index
- . Sum

```
[287]: l=(1,2,1,3,2,4,5,2,6,2,9)
      type(l)
```

[287]: tuple

```
[288]: m=min(l)
      m
```

[288]: 1

```
[289]: ma=max(l)
      ma
```

[289]: 9

```
[290]: c=l.count(2)
      c
```

[290]: 4

```
[291]: I=l.index(3)
      I
```

[291]: 3

```
[292]: i1=l.index(9)
      i1
```

[292]: 10

```
[293]: S=sum(l)
      S
```

[293]: 37

```
[294]: x=sum(l,30)
      x
```

[294]: 67

20.7 39. Set in python

- . Unordered
- . Mutable
- . Unindex
- . {} / set()
- . unique element

```
[295]: s={10,20,30}  
type(s)
```

```
[295]: set
```

```
[296]: s
```

```
[296]: {10, 20, 30}
```

```
[297]: for a in s:  
       print(a)
```

```
10  
20  
30
```

20.7.1 function in set

- . set()
- . add()
- . pop()
- . remove()
- . clear()
- . discard()
- . update()

```
[298]: l=[10,20,30]  
s=set(l)  
s
```

```
[298]: {10, 20, 30}
```

```
[299]: type(s)
```

```
[299]: set
```

```
[300]: s.add(40)
```

```
[301]: s
```

```
[301]: {10, 20, 30, 40}
```

```
[302]: s.pop()
```

```
[302]: 40
```

```
[303]: s
```

```
[303]: {10, 20, 30}
```

```
[304]: s.remove(20)
s
```

```
[304]: {10, 30}
```

```
[305]: s.discard(10)
```

```
[306]: s
```

```
[306]: {30}
```

```
[307]: s.clear()
```

```
[308]: print(s)
```

```
set()
```

```
[309]: l={1,2,3,4}
```

```
[310]: a=(5,6,7)
```

```
[311]: l.update(a)
```

```
[312]: l
```

```
[312]: {1, 2, 3, 4, 5, 6, 7}
```

20.8 40. User defined Functions in Python

Function: it is a block of statements which can be used repetively in a program. . Simple function . Function with arguments . Return type function

```
[313]: def abc(): # defining simple function
        print("This is my simple function")
```

```
[314]: abc()
```

This is my simple function

```
[315]: def sum(a,b): # function with arguments
        c=a+b
        print("The sum of two no. is-",c)
```

```
[316]: a=15
        b=-1
        sum(a,b)
```

The sum of two no. is- 14

```
[317]: def mul(x,y=10): # function with arguments
        z=x*y
        print("The multiplication of two no. is-",z)
```

```
[318]: mul(110,2)
```

The multiplication of two no. is- 220

```
[319]: mul(5)
```

The multiplication of two no. is- 50

```
[320]: def divi(X,Y):
        Z=X/Y
        return Z
```

```
[321]: out=divi(20,10)
```

```
[322]: print(out)
```

2.0

```
[323]: def squ(x):
        return x*x
```

```
[324]: s=squ(5)
```

```
[325]: s
```

```
[325]: 25
```

```
[326]: def squ(x):
        return x*x,x*3
```

```
[327]: S=squ(5)
        S
```

```
[327]: (25, 15)
```

```
[328]: def add_numbers(number_1, number_2):  
        sum = number_1 + number_2  
        return sum
```

```
[329]: result = add_numbers(5, 3)
```

```
[330]: print("The sum is:", result)
```

The sum is: 8

Make a function in Python which take two numbers from user and perform add, sub mul, div

```
[331]: def My_Mini_Calculator():  
        number_1 = float(input("Enter the first number: "))  
        number_2 = float(input("Enter the second number: "))  
  
        # Addition  
        result = number_1 + number_2  
        print("The sum is:", result)  
  
        # Subtraction  
        result = number_1 - number_2  
        print("The difference is:", result)  
  
        # Multiplication  
        result = number_1 * number_2  
        print("The product is:", result)  
  
        # Division  
        result = number_1 / number_2  
        print("The quotient is:", result)  
  
        # Call the function  
        My_Mini_Calculator()
```

Enter the first number: 13

Enter the second number: 15

The sum is: 28.0

The difference is: -2.0

The product is: 195.0

The quotient is: 0.8666666666666667

```
[332]: def My_Mini_Calculator():  
        number_1 = float(input("Enter the first number: "))
```

```

number_2 = float(input("Enter the second number: "))

# Addition
add_result = number_1 + number_2

# Subtraction
diff_result = number_1 - number_2

# Multiplication
mult_result = number_1 * number_2

# Division
div_result = number_1 / number_2

return {
    "Addition": add_result,
    "Subtraction": diff_result,
    "Multiplication": mult_result,
    "Division": div_result,
}

# Call the function
results = My_Mini_Calculator()

print("The sum is:", results["Addition"])
print("The difference is:", results["Subtraction"])
print("The product is:", results["Multiplication"])
print("The quotient is:", results["Division"])

```

```

Enter the first number: 67
Enter the second number: 13

The sum is: 80.0
The difference is: 54.0
The product is: 871.0
The quotient is: 5.153846153846154

```

You can create a custom function in Python to accomplish the following:

Ask the user to enter 1st and last names, be careful about case sensitivity.

Print the output, need only to capitalize the first letter of the first name and last name.

```

[333]: def format_name():
        first_name = input("Enter your first name: ")
        last_name = input("Enter your last name: ")

        # Capitalize the first letter of the first name
        formatted_first_name = first_name.capitalize()

```

```

# Capitalize the first letter of the last name
formatted_last_name = last_name.capitalize()

# Print the formatted name
print("Candidate name is:", formatted_first_name, formatted_last_name)

# Call the function
format_name()

```

Enter your first name: Hasan

Enter your last name: Rizvi

Candidate name is: Hasan Rizvi

Create a Python programme that includes a user-defined function called `print_even_numbers()`. The function should prompt the user to set a range limit before printing all even numbers up to that limit.

```

[334]: def print_even_numbers():
        n = int(input("Enter the range limit: "))
        even_numbers = []
        for num in range(1, n + 1):
            if num % 2 == 0:
                even_numbers.append(num)

        print("Even numbers up to", n, "are:", even_numbers)

# Call the function
print_even_numbers()

```

Enter the range limit: 20

Even numbers up to 20 are: [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]

20.9 Lambda Function

here are some key points about lambda functions in Python, each point is a one-liner:

Lambda functions are small, anonymous functions defined with the `lambda` keyword.

They can take any number of arguments but can only have one expression.

The expression is evaluated and returned when the function is called.

Lambda functions do not require a return statement, the expression is implicitly returned.

They can be used wherever function objects are required, like inside functions like `map()`, `filter()`, and `reduce()`.

You can't include statements like loops, if, or else in lambda functions; only expressions are allowed.

Lambda functions are useful for small tasks that are not reused throughout your code.

They can be assigned to variables and used like regular functions.

```
[335]: # lambda arguments: expression
```

```
add = lambda x, y: x + y
print(add(5, 3)) # Output: 8
```

8

```
[336]: # Define a lambda function that takes three arguments and returns their sum
```

```
add_three_numbers = lambda x, y, z: x + y + z
```

```
# Use the lambda function
```

```
result = add_three_numbers(5, 3, 8)
```

```
print(result) # Output: 16
```

16

```
[337]: square = lambda x: x ** 2
```

```
print(square(5)) # Output: 25
```

25

```
[338]: concat = lambda x, y: x + y
```

```
print(concat("Hello", " World")) # Output: "Hello World"
```

Hello World

```
[339]: even_or_odd = lambda x: 'even' if x%2==0 else 'odd'
```

```
print(even_or_odd(1520)) # Output: 'odd'
```

even

```
[340]: Multily=lambda x,y: x*y
```

```
print(Multily(12,5))
```

60

```
[341]: add = lambda a, b: a + b
```

```
num1 = int(input("Enter first number: "))
```

```
num2 = int(input("Enter second number: "))
```

```
print(add(num1, num2))
```

Enter first number: 12

Enter second number: 72

84

```
[342]: add = lambda: int(input("Enter first number: ")) + int(input("Enter second_
↪number: "))
print(add())
```

```
Enter first number: 14
Enter second number: 90

104
```

```
[353]: lambda_function = lambda x: [i ** 2 for i in x]
print(lambda_function([1, 2, 3, 4, 5])) # Output: [1, 4, 9, 16, 25]
```

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

```
[354]: cube=lambda x: [i**3 for i in x]
print(cube([1,2,3,4,5]))
```

```
[1, 8, 27, 64, 125]
```

Let's take a real-world use case involving sorting:

Consider you have a list of dictionaries, where each dictionary represents a product with a name and a price. You want to sort this list of products based on the price.

```
[355]: # A list of dictionaries is created, where each dictionary represents a product.
↪
# Each product has two properties: 'name' and 'price'.

products = [
    {'name': 'Laptop', 'price': 50000},
    {'name': 'Mobile', 'price': 30000},
    {'name': 'Ipad', 'price': 45000},
    {'name': 'Desktop', 'price': 35000},
]

sorted_products = sorted(products, key=lambda x: x['price'])

for product in sorted_products:
    print(product)
```

```
{'name': 'Mobile', 'price': 30000}
{'name': 'Desktop', 'price': 35000}
{'name': 'Ipad', 'price': 45000}
{'name': 'Laptop', 'price': 50000}
```

20.10 Recursion

Recursion refers to the programming technique in which a function defines itself by calling itself. Here are a few highlights:

Recursion occurs when a function calls itself while providing a different input for each consecutive call.

Base Case: To prevent unbounded recursion, every recursive function should have a base case, which is a condition under which it stops invoking itself.

The execution context or state of each recursive call is different. The current values of a function's parameters and variables are placed onto the call stack when it calls itself.

Problem-Solving: Recursion is frequently used to solve problems that can be divided into smaller, simpler problems of the same type.

Memory: Because recursive functions must retain stack frames for all recursive calls, they consume more memory than iterative versions.

Efficiency: Because of the complexity of maintaining the stack, recursive functions can be less efficient and can result in a stack overflow for deep recursion.

Readability: Despite potential inefficiencies, recursion can result in easier-to-read and write code for certain issues, such as traversing tree-like data structures, sorting algorithms (such as quicksort and mergesort), solving Tower of Hanoi, Fibonacci series, and so on.

Remember that recursion is a tool, and whether or not to utilise it depends on the particular problem you're attempting to solve. Some issues lend themselves nicely to recursive solutions, while others may benefit from an iterative approach.

```
[356]: def recursive_sum(n):  
        # Base case  
        if n == 1:  
            return 1  
        # Recursive case  
        else:  
            return n + recursive_sum(n - 1)  
  
print(recursive_sum(12))
```

78

```
[357]: def factorial(n):  
        # Base case: 1! = 1  
        if n == 1:  
            return 1  
        # Recursive case: n! = n * (n-1)!  
        else:  
            return n * factorial(n-1)
```

The Fibonacci sequence is a set of numbers in which each number is the sum of the two numbers before it. It usually starts with 0 and 1. In certain variants, it begins with two 1's.

Here's the beginning of the Fibonacci sequence:

0, 1, 1, 2, 3, 5, 8, 13, 21, ...

You can see the pattern:

$$0 + 1 = 1$$

$$1 + 1 = 2$$

$$1 + 2 = 3$$

$$2 + 3 = 5$$

$$3 + 5 = 8$$

...

Each number is the sum of the two numbers before it.

```
[358]: def fibonacci(n):  
        if n<=1:  
            return n  
        else:  
            return fibonacci(n-1)+fibonacci(n-2)  
print(fibonacci(6))
```

8

```
[359]: def my_fibonacci(n):  
        a, b = 0, 1  
        for i in range(n):  
            print(a)  
            a, b = b, a + b  
  
my_fibonacci(7)
```

0

1

1

2

3

5

8

```
[360]: # We need to have a counter  
cnt = 1 # This is a counter, because to explain each step. Nothing more than it.  
  
def my_fibonacci(n):  
    global cnt  
  
    if n <= 0:  
        return []  
    elif n == 1:
```

```

        return [0]
    elif n == 2:
        return [0, 1]
    else:
        fibonacci_series = my_fibonacci(n - 1)
        print(f"Step {cnt}: Our Fibonacci series up to n-1 terms =_
→{fibonacci_series}")
        cnt += 1
        next_term = fibonacci_series[-1] + fibonacci_series[-2]
        print(f"Step {cnt}: Next term to add = {next_term}")
        cnt += 1
        fibonacci_series.append(next_term)
        print(f"Step {cnt}: Our Fibonacci series after adding next term =_
→{fibonacci_series}")
        cnt += 1
        return fibonacci_series

print("So my final Fibonacci series is ---> ", my_fibonacci(4))

```

Step 1: Our Fibonacci series up to n-1 terms = [0, 1]
 Step 2: Next term to add = 1
 Step 3: Our Fibonacci series after adding next term = [0, 1, 1]
 Step 4: Our Fibonacci series up to n-1 terms = [0, 1, 1]
 Step 5: Next term to add = 2
 Step 6: Our Fibonacci series after adding next term = [0, 1, 1, 2]
 So my final Fibonacci series is ---> [0, 1, 1, 2]

21 Exception handling

It is a mechanism in programming to handle runtime errors, which are known as exceptions.

It's a process where you define a block of code that will be executed if an error occurs when the program is running. This allows the program to continue running (or terminate gracefully) even if an error occurs.

Without exception handling, an error occurring in a program would cause the program to immediately stop. This can be very undesirable, especially in production software, as it can lead to a poor user experience or even data loss.

In Python, you use try and except blocks. The try block contains the code that might raise an exception, and the except block contains the code that will be executed if an exception is raised.

The else block allows you run code without errors.

The finally block executes code regardless of the try-and-except blocks.

Some of the most common types of exceptions are:

ZeroDivisionError: Raised when the second argument of a division or modulo operation is zero.

TypeError: Raised when an operation or function is applied to an object of inappropriate type.

ValueError: Raised when a built-in operation or function receives an argument that has the right type but an inappropriate value.

IndexError: Raised when a sequence subscript is out of range.

KeyError: Raised when a dictionary key is not found.

FileNotFoundError: Raised when a file or directory is requested but doesn't exist.

IOError: Raised when an I/O operation (such as a print statement, the built-in open() function or a method of a file object) fails for an I/O-related reason.

ImportError: Raised when an import statement fails to find the module definition or when a from ... import fails to find a name that is to be imported.

MemoryError: Raised when an operation runs out of memory.

OverflowError: Raised when the result of an arithmetic operation is too large to be expressed by the normal number format.

AttributeError: Raised when an attribute reference or assignment fails.

SyntaxError: Raised when the parser encounters a syntax error.

IndentationError: Raised when there is incorrect indentation.

NameError: Raised when a local or global name is not found.

|

Role of Try and Except:

try block: The code within the try block contains the statements that may potentially raise an exception. It allows you to specify the section of code that you want to monitor for exceptions.

except block: If an exception occurs within the try block, the corresponding except block(s) are executed. The except block allows you to define the actions or code that should be executed when a specific exception is raised. You can have multiple except blocks to handle different types of exceptions.

The else block allows you run code without errors.

The finally block executes code regardless of the try-and-except blocks.

Use the raise keyword to throw (or raise) an exception.

22 User-defined Exceptions

By deriving a new class from the default Exception class in Python, we can define our own exception types.

```
[361]: class MyCustomError(Exception):  
        pass
```

In the above code, `MyCustomError` is derived from the built-in `Exception` class. You can use this in your code by using the `raise` statement.

```
[362]: raise MyCustomError("This is a custom error")
```

```
MyCustomError                                Traceback (most recent call↳  
↳last)  
  
    <ipython-input-362-95d924fa6573> in <module>  
----> 1 raise MyCustomError("This is a custom error")  
  
MyCustomError: This is a custom error
```

```
[363]: # define user-defined exceptions
class WrongAge(Exception):
    "Raised when the input value is less than 100"
    pass
```

```
[364]: # you need to guess this number
n = 18

try:
    input_num = int(input("Enter a age: "))
    if input_num < n:
        raise WrongAge # calling your custom exception
    else:
        print("You can work")
except WrongAge:
    print("Invalid Age: You are not allowed to work")
```

Enter a age: 24

You can work

23 Logging

Logging is a technique for monitoring events that take place when some software is in use.

For the creation, operation, and debugging of software, logging is crucial.

There are very little odds that you would find the source of the issue if your programme fails and you don't have any logging records.

Additionally, it will take a lot of time to identify the cause.

```
[375]: # first import the logging library
import logging

""" In the code above, we first import the logging module, then we call the
    basicConfig method to configure the logging.

    We set the level to DEBUG, which means that all logs of level
    DEBUG and above will be tracked."""

logging.basicConfig(level=logging.DEBUG)

# Logging Level: severity of the event being logged
# Least severe to most severe
logging.debug('This is a debug message')
logging.info('This is an info message')
logging.warning('This is a warning message')
logging.error('This is an error message')
logging.critical('This is a critical message')
```

```
CRITICAL:root:This is a critical message
```

Some programmers utilise the idea of "Printing" the statements to check whether they were correctly performed or if an error had occurred.

However, printing is not a smart move. For basic scripts, it might be the answer to your problems, however the printing solution will fall short for complex programmes.

A built-in Python package called logging enables publishing status messages to files or other output streams. The file may provide details about which portion of the code is run and any issues that have come up.

Here are the different log levels in increasing order of severity:

DEBUG: Detailed information, typically of interest only when diagnosing problems.

INFO: Confirmation that things are working as expected.

WARNING: An indication that something unexpected happened, or may happen in the future (e.g. 'disk space low'). The software is still working as expected.

ERROR: More serious problem that prevented the software from performing a function.

CRITICAL: A very serious error, indicating that the program itself may be unable to continue running.

23.1 Debug

```
[376]: import logging

logging.basicConfig(level=logging.DEBUG)

def add(x, y):
    logging.debug('Variables are %s and %s', x, y)
    return x + y

add(1, 2)
```

[376]: 3

23.2 Info

```
[377]: import logging

logging.basicConfig(level=logging.INFO)

def login(user):
    logging.info('User %s logged in', user)

login('Admin User')
```

23.3 Warning

```
[378]: import logging

logging.basicConfig(level=logging.WARNING)

def MyBalance(amount):
    if amount < 40000:
        logging.warning('Sorry you have Low balance: %s', amount)

MyBalance(10000)
```

23.4 Error

```
[379]: import logging

logging.basicConfig(level=logging.ERROR)
```

```
def LetUsDivide(n, d):
    try:
        result = n / d
    except ZeroDivisionError:
        logging.error('You are trying to divide by zero, which is not allowed')
    else:
        return result

LetUsDivide(4, 0)
```

23.5 Critical Errors

```
[380]: import logging

logging.basicConfig(level=logging.CRITICAL)

def LetUsCheckSystem(sys):
    if sys != 'OK':
        logging.critical('System failure: %s', sys)

LetUsCheckSystem('You need to handle the issue now')
```

CRITICAL:root:System failure: You need to handle the issue now

23.6 Save to a file

```
[381]: import os

# Specify the directory and file
dir_path = r'C:\Users\Dell\Desktop\June\Latest\iNeuron\Sessions\17_18June2023'
log_file = 'system.txt'
full_path = os.path.join(dir_path, log_file)

# Check if the directory exists and create it if necessary
os.makedirs(dir_path, exist_ok=True)

# Try writing a test message to the file
with open(full_path, 'w') as f:
    f.write('This is a test message')
```

```
[382]: import os
print(os.getcwd())
```

/home/labsuser/WS Cube Tech

```
[383]: import os
import logging

# Specify the directory and file
dir_path = r'C:\Users\Dell\Desktop\June\Latest\iNeuron\Sessions\17_18June2023'
log_file = 'system.txt'
full_path = os.path.join(dir_path, log_file)

# Check if the directory exists and create it if necessary
os.makedirs(dir_path, exist_ok=True)

# Set up logging
# Get a logger instance (this will fetch the root logger)
logger = logging.getLogger()

# Set the level of the logger to CRITICAL
# This means it will handle events of level CRITICAL and above
logger.setLevel(logging.CRITICAL)

# Create a FileHandler instance to write logs to a file
handler = logging.FileHandler(full_path)

# Set the format of the logs using a Formatter
# This format includes the log timestamp, log level and log message
handler.setFormatter(logging.Formatter('%(asctime)s:%(levelname)s:%(message)s'))

# Add the handler to the logger
# This connects the logger to the handler so that logs get written to the file
logger.addHandler(handler)

def LetUsCheckSystem(sys):
    if sys != 'OK':
        logging.critical('System failure: %s', sys)

LetUsCheckSystem('You need to handle the issue now')
handler.close()
```

CRITICAL:root:System failure: You need to handle the issue now

```
[384]: #import pdb

def addition(a, b):
    pdb.set_trace() # Set a breakpoint here
    result = a + b
    return result
```

```
print(addition(5, 7))
```

```

↳ -----
NameError                                Traceback (most recent call↳
↳last)

  <ipython-input-384-592f3c84c557> in <module>
      6     return result
      7
----> 8 print(addition(5, 7))

  <ipython-input-384-592f3c84c557> in addition(a, b)
      2
      3 def addition(a, b):
----> 4     pdb.set_trace() # Set a breakpoint here
      5     result = a + b
      6     return result

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

23.7 41. Module In python

Math module

- math.ceil(a)
- math.fabs(x): Returns absolute value
- math.factorial(a)
- math.floor(a)
- math.fsum(a)

```
[385]: import math
x=150.5251
print(math.ceil(x))
```

151

```
[386]: y=-10
print(math.fabs(y))
```

10.0

```
[387]: x=4  
print(math.factorial(x))
```

24

```
[388]: print(math.factorial(6))
```

720

```
[389]: z=12.526  
print(math.floor(z))
```

12

```
[390]: Z=[10,25,52,52]  
print(math.fsum(Z))
```

139.0

23.8 42. Random module in python

- . Randint()
- . Randrange()
- . Randchoice()

```
[391]: import random
```

```
[392]: n=random.randint(0,10)  
print(n)
```

4

```
[393]: n=random.randrange(1,25)
```

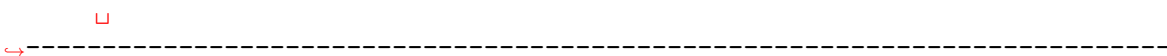
```
[394]: n
```

```
[394]: 7
```

```
[395]: l=[12,10,15,20]  
random.choice(l)
```

```
[395]: 10
```

```
[396]: print(lc)
```



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

```
<ipython-input-396-ffa623b2e61f> in <module>
----> 1 print(lc)
```

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

```
[397]: random.shuffle(l)
      print(l)
```

```
[15, 20, 10, 12]
```

23.9 43. DateTime Module in Python

```
[398]: import datetime
      x=datetime.datetime.now()
```

```
[399]: print(x)
```

```
2023-08-11 03:32:08.831010
```

```
[400]: print(datetime.datetime(2023,8,9))
```

```
2023-08-09 00:00:00
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

