```
print("\"Welcome to ExcelR!\"")
```

```
"Welcome to ExcelR!"
```

```
"hello"
```

```
'hello'
```

```
120/12*2-4
```

```
16.0
```

```
(5-2)*10
```

```
30
```

```
marks=80
marks
```

```
80
```

```
type(marks)
```

```
int
```

```
5**4
```

```
625
```

```
price=50.5
price
type(price) # only shows output of last line
```

```
50.5
```

```
price=50.5
print(price) # to see intermediate line's output use print() function
(type(price))
```

```
50.5
float
```

```
name="Snehal"
type(name)
```

```
str
```

```
dir(str) # all methods that can be applied on strings
# __ dunder/magic methods (double underscore methods which python uses internally)
```

```
['__add__',
 '__class__',
 '__contains__',
 '__delattr__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__getnewargs__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__iter__',
 '__le__',
 '__len__',
 '__lt__',
 '__mod__',
 '__mul__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
```

```
 '__rmod__',
 '__rmul__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 'capitalize',
 'casefold',
 'center',
 'count',
 'encode',
 'endswith',
 'expandtabs',
 'find',
 'format',
 'format_map',
 'index',
 'isalnum',
 'isalpha',
 'isascii',
 'isdecimal',
 'isdigit',
 'isidentifier',
 'islower',
 'isnumeric',
 'isprintable',
 'isspace',
 'istitle',
 'isupper',
 'join',
 'ljust',
```

```
str1="welcome to Data Science."
str1.capitalize()
```

```
'Welcome to data science.'
```

```
str1
```

```
'welcome to Data Science.'
```

```
str1.upper()
```

```
'WELCOME TO DATA SCIENCE.'
```

```
str1.lower()
```

```
'welcome to data science.'
```

```
str1.title()
```

```
'Welcome To Data Science.'
```

```
food="bIrYAni"
name="sNEHAL"
print(food.swapcase())
name.swapcase()
```

```
BiRyaNI
'Snehal'
```

```
s1="My name is Snehal."
print(s1.split()) # list of strings is created
s2="Mumbai-Pune-Mumbai"
s2.split("-")
```

```
['My', 'name', 'is', 'Snehal.']
['Mumbai', 'Pune', 'Mumbai']
```

```
mail="snehal@excelr.com"
mail.split("@")
```

```
['snehal', 'excelr .com']
```

```
s1
```

```
'My name is Snehal.'
```

```
s1.replace("Snehal","Jyoti")
```

```
'My name is Jyoti.'
```

```
# single line comment
'''
multiline comment
'''
```

```
'\nmultiline comment \n'
```

```
city="Satara"
city.count("a")
```

```
3
```

```
"Satara".count("a")
```

```
3
```

```
name="Raj"
name*4
```

```
'RajRajRajRaj'
```

```
n1=5
n1*7
```

```
35
```

```
name="Rajesh"
surname="Kulkarni"
name + surname # concatenates without space
```

```
'RajeshKulkarni'
```

```
name + " " +surname
```



```
'Rajesh Kulkarni'
```

```
# Conditional Statement
a=30
b=50
if a>b:
  print("a is greater")
  print("In True block")
  c=a+b
  print(c)
else:
  print('b is greater')
  print("In False block")
  c=a*b
  print(c)
```

```
b is greater
In False block
1500
```

```
a=1200
b=300
if(a>b):
  print(a,"is greater")
else:
  print("Greatest number is:",b)
```

```
Greatest number is: 300
value -100
```

```
salary=15000
if (salary>20000):
  tax=0.25
  print("Hello from true if")
else:
```

```
  tax=0.20
  print("Hello from false if / else")
print("Bye... outside of if block")
net_sal=salary-tax*salary
print("Net Salary=",net_sal)
# change value of salary as 15000 and execute again
```

```
Hello from false if / else
Bye... outside of if block
Net Salary= 12000.0
```

```
range(5,10)
```

```
range(5, 10)
```

```
# For Loop
for i in range(5,10): # Print numbers from 5 to 9. Default increment is 1.
  print(i)
print("Bye")
```

```
5
6
7
8
9
Bye
```

```
# print numbers 1 to 10


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

```
1
3
5
7
9
```

```
# Q1. print all even numbers from 1 to 20
# Q2. print all odd numbers from 1 to 20


# 'break' statement terminates the loop when condition is satisfied
for i in range(1,11):
    if i==5:
       print("Same")
       break
    else:
        print(i)
```

```
1
2
3
4
Same
```

```
for num in range(11): # 0 to 10
  print(num, num*num) # num and its square
```

```
0 0
1 1
2 4
3 9
4 16
5 25
6 36
7 49
8 64
9 81
10 100
```

```
# range(start,stop,step)
for i in range(1,30,5):
  #print(i)
  print(i,"@")
```

```
1 @
6 @
11 @
```

```
16 @
21 @
26 @
```

```
subject="Science"
for letter in subject:
  print(letter)
```

```
S
c
i
e
n
c
e
```

```
# print name 3 times on first line, 4 times on 2nd, 5 - 3rd, 6 - 4th line. Range(3,7)

name="Snehal"
for i in range(3,7):
  print(name*i)
```

```
SnehalSnehalSnehal
SnehalSnehalSnehalSnehal
SnehalSnehalSnehalSnehalSnehal
SnehalSnehalSnehalSnehalSnehalSnehal
```

```
# print each letter of movie on separate line
# change case of each letter (upper to small and small to upper) of movie on single line with seperator ~
movie="Drishyam"
for letter in movie:
  #print(letter)
  #print(letter.swapcase())
  print(letter.swapcase(),end="  ") #try upper(), lower(),"-","#",etc. Default end parameter is 'new line'
```

```
d  R  I  S  H  Y  A  M
```

```
# Print all even and odd numbers between given range using For Loop
for i in range(1, 11):
    if i % 2 == 0:
        print('Even Number:', i)
    else:
        print('Odd Number:', i)
```

```
Odd Number: 1
Even Number: 2
Odd Number: 3
Even Number: 4
Odd Number: 5
Even Number: 6
Odd Number: 7
Even Number: 8
Odd Number: 9
Even Number: 10
```

```
# Print all even and odd numbers between given range using For Loop
print('Even Numbers:')
for i in range(1, 11):
  if i % 2 == 0:
    print(i)

print('Odd Numbers:')
for i in range(1,11):
  if i%2!=0: # != not equal to
    print(i)
```

```
Even Numbers:
2
4
6
8
10
Odd Numbers:
1
3
5
7
9
```

```
status1=True
type(status1)
status2=False
type(status2)
```

```
bool
```

```
# print each letter 4 times with separator as space
for letter in "Data Science":
  print(letter*4,end=" ")
```

```
DDDD aaaa tttt aaaa      SSSS cccc iiii eeee nnnn cccc eeee
```

```
len("My name")
```

```
7
```

```
len("Snehal")
```

```
6
```

```
# Data Structure: 1.LIST
# Lists are mutable.
# created using square brackets []
marks=[40,55,70,60,80,85,78]
marks
```

```
[40, 55, 70, 60, 80, 85, 78]
```

```
type(marks)
```

```
list
```

```
dir(list)
```

```
['__add__',
 '__class__',
 '__class_getitem__',
 '__contains__',
 '__delattr__',
 '__delitem__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__gt__',
 '__hash__',
 '__iadd__',
 '__imul__',
 '__init__',
 '__init_subclass__',
 '__iter__',
 '__le__',
 '__len__',
 '__lt__',
 '__mul__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__reversed__',
 '__rmul__',
 '__setattr__',
 '__setitem__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 'append',
 'clear',
 'copy',
 'count',
 'extend',
 'index',
 'insert',
 'pop',
 'remove',
 'reverse',
 'sort']
```

```
len(marks)
```

```
7
```

```
new_marks=sorted(marks) # creates new sorted list. Original marks list remains unchanged
new_marks
```

```
[40, 55, 60, 70, 78, 80, 85]
```

```
marks
```

```
[40, 55, 70, 60, 80, 85, 78]
```

```
marks.sort() # original list is sorted in ascending order by default
marks
```

```
[40, 55, 60, 70, 78, 80, 85]
```

```
marks.append(35) # Append will add element in the last only.
marks
```

```
[40, 55, 60, 70, 78, 80, 85, 35]
```

```
marks.reverse()
```

```
marks
```

```
[35, 85, 80, 78, 70, 60, 55, 40]
```

```
marks2=[42,64,50,78,87,80]
```

```
marks2.sort(reverse=True) # sorts in descending order when reverse=True
marks2
```

```
[87, 80, 78, 64, 50, 42]
```

```
marks3=[40,60,50,70,80,85]
marks3.sort(reverse=False)
```

```
marks3
```

```
[40, 50, 60, 70, 80, 85]
```

```
marks
```

```
[35, 85, 80, 78, 70, 60, 55, 40]
```

```
90 in marks # checks whether 90 is present in list or not. IF yes returns True, else False
```

```
False
```

```
85 in marks
```

```
True
```

```
list1=[10,20,30]
list2=[40,50,60]
list1 + list2 # concatenation of two lists
```

```
[10, 20, 30, 40, 50, 60]
```

```
(list1+list2)*3 # repeats 3 times
```

```
[10, 20, 30, 40, 50, 60, 10, 20, 30, 40, 50, 60, 10, 20, 30, 40, 50, 60]
```

```
# List of Strings
list3=["Mango","Apple","Banana","Pineapple","Custard Apple","Strawberry"]
list3
```

```
['Mango', 'Apple', 'Banana', 'Pineapple', 'Custard Apple', 'Strawberry']
```

```
# Display fruits by changing case of all fruits to uppercase
for fruit in list3:
  #print(fruit.upper()) # in sigle line: print(fruit.upper(),end=" ")
  print(fruit.upper(),end=" ")
  #print(fruit.replace("u","a"), end=" ")
```

```
MANGO APPLE BANANA PINEAPPLE CUSTARD APPLE STRAWBERRY
```

```
# In Python Indexing starts with 0
list3
```

```
['Mango', 'Apple', 'Banana', 'Pineapple', 'Custard Apple', 'Strawberry']
```

```
list3[0]
```

```
'Mango'
```

```
list3.count("Banana")
```

```
1
```

```
list3[6] # raise erro as index is upto 5 only (0 to 5)
```

```
list3[5]
```

```
'Strawberry'
```

```
# Slicing
list3[0:4] # 0,1,2,3
```

```
['Mango', 'Apple', 'Banana', 'Pineapple']
```

```
# from Apple to Strawberry i.e. till end
list3[1:]# or list3[1:6]
```

```
['Apple', 'Banana', 'Pineapple', 'Custard Apple', 'Strawberry']
```

```
# from beginning to Pinaapple
list3[:4]
```

```
['Mango', 'Apple', 'Banana', 'Pineapple']
```

```
list3
```

```
['Mango', 'Apple', 'Banana', 'Pineapple', 'Custard Apple', 'Strawberry']
```

```
# Negative indexing: -1 refers to last element
list3[-1]
```

```
'Strawberry'
```

```
list3[-3]
```

```
'Pineapple'
```

```
list3[-6]
```

```
'Mango'
```

```
# start stop step in slicing
print(list3)
list3[::2] # 0,2,4,6
```

```
['Mango', 'Apple', 'Banana', 'Pineapple', 'Custard Apple', 'Strawberry']
['Mango', 'Banana', 'Custard Apple']
```

```
n1=[1:31] # to create list of 1 to 30 - will raise error
```

```
n1=list(range(1,31))
print(n1,end=" ")
```

```
[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]
```

```
n1[::4]
```

```
[1, 5, 9, 13, 17, 21, 25, 29]
```

```
# print nos from index 2 to 26 with step value of 5
n1[2:27:5]
```

```
[3, 8, 13, 18, 23]
```

```
print(list3, end=" ")
```

```
['Mango', 'Apple', 'Banana', 'Pineapple', 'Custard Apple', 'Strawberry']
```

```
list3[::-1]
```

```
['Strawberry', 'Custard Apple', 'Pineapple', 'Banana', 'Apple', 'Mango']
```

```
list3[::-2] # step value is -2
```

```
[]
```

```
# use of 'in' keyword: search substring in given word
'the' in 'Netherland'
```

```
True
```

```
'The' in 'Netherland'
```

```
False
```

```
'therer' in 'Netherland'
```

```
False
```

```
list3
```

```
['Mango', 'Apple', 'Banana', 'Pineapple', 'Custard Apple', 'Strawberry']
```

```
list3.append("Mulberry")
```

```
list3
```

```
['Mango',
 'Apple',
 'Banana',
 'Pineapple',
 'Custard Apple',
 'Strawberry',
 'Mulberry']
```

```
list3.remove("Mulberry") # removes mentioned element
```

```
list3
```

```
['Mango', 'Apple', 'Banana', 'Pineapple', 'Custard Apple', 'Strawberry']
```

```
# display fruits which contain substring as 'berry' using for loop
for fruit in list3:
  if 'berry' in fruit:
    print(fruit)
```

```
Strawberry
Mulberry
```

```
# display fruits which contain substring as 'pple' using for loop
for fruit in list3:
  if 'pple' in fruit:
    print(fruit)
```

```
Apple
Pineapple
Custard Apple
```

```
list3.append("Black berry")
```

```
list3
```

```
['Mango',
 'Apple',
 'Banana',
 'Pineapple',
 'Custard Apple',
 'Strawberry',
 'Mulberry',
 'Black berry']
```

```
# Create list4 of fruits from list3 having only 'berry' as substring and store remaining fruits in list5
list4=[]
list5=[]
for fruit in list3:
  if 'berry' in fruit:
    list4.append(fruit)
  else:
    list5.append(fruit)
print(list4,end=" ")
print(list5,end=" ")
```

```
['Strawberry', 'Mulberry', 'Black berry'] ['Mango', 'Apple', 'Banana', 'Pineapple', 'Custard Apple']
```

```
# print squares of all numbers of list1 to list2
l1=[2,3,4,5,6]
l2=[]
for i in l1:
  l2.append(i**2) # OR l2.append(i*i)
print(l1,end=" ")
print(l2,end=" ")
```

```
[2, 3, 4, 5, 6] [4, 9, 16, 25, 36]
```

```
# List comprehension - makes your code compact
l1=[2,3,4,5,6]
l2=[i**2 for i in l1]
print(l1,end=" ")
print(l2,end=" ")
```

```
[2, 3, 4, 5, 6] [4, 9, 16, 25, 36]
```

```
list3
#list3.append("Black berry")
```

```
# extract fruit names containing substring 'berry' using List comprehension
list3
```

```
['Mango',
 'Apple',
 'Banana',
 'Pineapple',
 'Custard Apple',
 'Strawberry',
 'Mulberry',
 'Black berry']
```

```
newlist=[fruit for fruit in list3 if 'berry' in fruit]
```

```
print(newlist, end=" ")
```

```
['Strawberry', 'Mulberry', 'Black berry']
```

```
newlist1=[fruit for fruit in list3 if 'berry' not in fruit]
```

```
newlist1
```

```
['Mango', 'Apple', 'Banana', 'Pineapple', 'Custard Apple']
```

```
city=["Pune","Mumbai","Satara","Thane","Solapur","Kolhapur"]
```

```
newlistpur=[i for i in city if "pur" not in i]
```

```
newlistpur
```

```
['Pune', 'Mumbai', 'Satara', 'Thane']
```

```
list3
```

```
['Mango',
 'Apple',
 'Banana',
 'Pineapple',
 'Custard Apple',
 'Strawberry',
 'Mulberry',
 'Black berry']
```

```
list3.index('Mango')
```

```
0
```

```
list3.index('Black berry')
```

```
7
```

```
list3.index('Orange') # will raise error as item is not present in the list
```

```
# index 2 item was banana which is replaced by Orange
# Lists are mutable - you can replace items of list
list3[2]="Orange"
list3
```

```
['Mango',
 'Apple',
 'Orange',
 'Pineapple',
 'Custard Apple',
 'Strawberry',
 'Mulberry',
 'Black berry']
```

```
# Data Structure: 2.TUPLE
# Tuples are not mutable i.e. immutable
# created using round brackets ()
t1=(1,2,3,4,5)
t1
```

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

```
type(t1)
```

```
tuple
```

```
t1[0]
```

```
1
```

```
t1[0]=7 # will raise an error as tuples are immutable
```

```
t3=t1 + (6,7,8,9,10)
```

```
t3
```

```
(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

```
dir(t1)
```

```
t1.count(1)# in bracket we have passed an element of tuple (not the index)
```

```
1
```

```
t1.index(4) # in bracket we have passed an element of tuple (not the index)
```

```
3
```

```
t2=(1,1,1,2,2,6,6,6,6,6)
```

```
t2.count(6)
```

```
5
```

```
# indexing and slicing with tuple is same as list
# extracting elements of tuple
t1[2]
```

```
3
```

```
t1
```

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

```
t1[1:5]
```

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

```
t1[3:]
```

```
(4, 5)
```

```
# Data Structure: 3.Dictionary
# Dictionaries are used to store data values in "key:value pairs".
# A dictionary is a collection which is ordered, changeable and do not allow duplicates.
# Dictionaries are written with curly brackets {"Key":"Value"}, and have keys and values.
# Here, keys are unique identifiers that are associated with each value.:

capital_dict={"Nepal": "Kathmandu", "Italy": "Rome", "England": "London"}
print(capital_dict)
```

```
{'Nepal': 'Kathmandu', 'Italy': 'Rome', 'England': 'London'}
```

```
type(capital_dict)
```

```
dict
```

```
dir(capital_dict)
```

```
# access element of dictionary
capital_dict["England"]
```

```
'London'
```

```
# keys() method extracts the keys of the dictionary and returns the list of keys as a view object.
keys=capital_dict.keys()
keys
```

```
dict_keys(['Nepal', 'Italy', 'England'])
```

```
capital_dict.keys()
```

```
dict_keys(['Nepal', 'Italy', 'England'])
```

```
# The values() method returns a view object that displays a list of all the values in the dictionary.
values=capital_dict.values()
values
```

```
dict_values(['Kathmandu', 'Rome', 'London'])
```

```
# The items() method returns a view object that displays a list of dictionary's (key, value) tuple pairs.
items=capital_dict.items()
items
```

```
dict_items([('Nepal', 'Kathmandu'), ('Italy', 'Rome'), ('England', 'London')])
```

```
# The update() method updates the dictionary with the elements from another dictionary object or from an iterable of key/value pairs.
add_capital={"India": "New Delhi","abc":"xyz"}
capital_dict.update(add_capital)
```

```
capital_dict
```

```
capital_dict.update({"Australia":"Sydney"}) # capital of Australia is Canberra
capital_dict
```

```
# The pop() method removes and returns an element from a dictionary having the given key.
capital_dict.pop("Australia")
```

```
'Sydney'
```

```
capital_dict
```

```
{'Nepal': 'Kathmandu',
 'Italy': 'Rome',
 'England': 'London',
 'India': 'New Delhi',
 'abc': 'xyz'}
```

```
capital_dict['abc']=("Mumbai")
```

```
capital_dict
```

```
{'Nepal': 'Kathmandu',
 'Italy': 'Rome',
 'England': 'London',
 'India': 'New Delhi',
 'abc': 'Mumbai'}
```

```
capital_dict.popitem() # removes last key:value pair from dictionary
```

```
('abc', 'Mumbai')
```

```
capital_dict
```

```
{'Nepal': 'Kathmandu',
 'Italy': 'Rome',
 'England': 'London',
 'India': 'New Delhi'}
```

```
# Dictionary Values can be of different data type
mylist=["Mango","Orange","Strawberry","Banana","Mulberry","Black berry"]
```

```
mytuple=(20,53,45,78,89,65,97)
```

```
mydict={"A":mylist,
        "B":mytuple,
        "C":capital_dict
       }
mydict
```

```
{'A': ['Mango', 'Orange', 'Strawberry', 'Banana', 'Mulberry', 'Black berry'],
 'B': (20, 53, 45, 78, 89, 65, 97),
 'C': {'Nepal': 'Kathmandu', 'Italy': 'Rome', 'England': 'London'}}
```

```
mydict['A']
```

```
['Mango', 'Orange', 'Strawberry', 'Banana', 'Mulberry', 'Black berry']
```

```
mydict['C']
```

```
{'Nepal': 'Kathmandu', 'Italy': 'Rome', 'England': 'London'}
```

```
mydict['A'][3] #index 0 1 2 3, 3rd index element = banana
```

```
'Banana'
```

```
mydict['C']['Italy']
```

```
'Rome'
```

```
# Math functions: can import specific functions or all using *
sqrt(25) # raise error as you have not imported
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-10-8faec0ab57b9> in <cell line: 2>()
      1 # Math functions: can import specific functions or all using *
----> 2 sqrt(25) # raise error as you have not imported
```

```
from math import sqrt
sqrt(25)
```

```
5.0
```

```
factorial(5)
```

```
from math import factorial
factorial(5)
```

```
120
```

```
sin(90)
```

```
from math import sin,cos,tan
```

```
tan(45)
```

```
1.6197751905438615
```

```
from math import *
listnew=[5,2,3]
prod(listnew)
```

```
30
```

```
# OR you can say 'import math'
# But while calling any function you have to use math.Function_Name()
# The math.prod() method returns the product of the elements from the given iterable.
import math # or 'import math as mt', mt.sqrt(64 )
sequence = (2, 2,5,6) # iterable where elements are given
print(math.prod(sequence))
```

```
120
```

```
import math
list4=[2,3,4]
math.prod(list4)
```

```
24
```

```
dir(math) # all functions of math library
```