```
def circle(r):
    area = 3.14 * r **2
    circumference = 2 * 3.14 * r
    return area, circumference


circle(5)
(78.5, 31.400000000000002)
x = 7
circle(x)
(153.86, 43.96)
r = 3
circle(r)
(28.26, 18.84)

result = circle(5)
print("area = ", result[0], "and circumference = ", result[1])
area =  78.5 and circumference =  31.400000000000002
```

```
def func(a,b,c):
    print(a, b, c)


# Positional mapping
func(1,2,3)
1 2 3

# keyword mapping
func(c=3, b=2, a=1)
1 2 3

# Hybird mapping
func(1, c=3, b=2)
1 2 3
```

Recursive Functions:

- Python also excepts function recursion, which means a defined function can call itself.
- Recursion is a common mathematical and programming concept. It means that a function calls itself. This has the benefit of meaning that you can loop through data to reach a result.

Components of a recursive function:

Base Case:

- Indicates the stopping condition
- Could be more than one

Recursive Call

- Moves the execution towards the base case
- Could be more than one

```
def factorial(n):
    """
        Compute the factorial of non-negative number using recursion
    """
    if n>0:
        result = n * factorial(n-1)
    else:
        result = 1
    return result

factorial(5)
120
factorial(9)
362880
factorial(0)
1
```

Data Structures:

Data structures are ways of organizing and storing collection of data, allowing for efficient access, manipulation and retrieval.

They are essentially containers that can hold multiple values, potentially of different data types in a structured manner

- Lists:

Multiple items can be stored in a single variable. For creation of list we use square brackets [].

- Accessed by offset position.
- Ordered collection of arbitrary objects.
- Variable-length, heterogeneous and arbitrarily nestable.
- Mutable mapping.

List (indexes):

List = [0, 1, 2, 3, 4, 5]

0	1	2	3	4	5

List[0] = 0

List[1] = 1

List[2] = 2

List[3] = 3

List[4] = 4

List[5] = 5

List[0:] = [0,1,2,3,4,5]

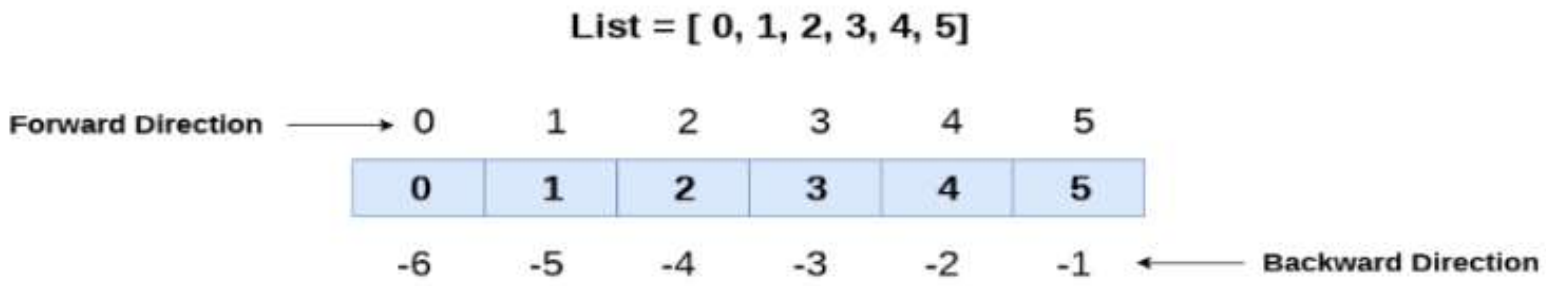
List[:] = [0,1,2,3,4,5]

List[2:4] = [2, 3]

List[1:3] = [1, 2]

List[:4] = [0, 1, 2, 3]

List (indexes):



Operators	Description	Example
Repetition	The repetition operator enable the list elements to be repeated multiple times.	L1 * 2 **Output:** [1,2,3,4,1,2,3,4]
Concatenation	It concatenates the lists mentioned on either sides of the operator	L1 + L2 **Output**: [1,2,3,4,5,6,7,8]
Membership	It returns TRUE if a particular item exists in a particular list, otherwise FALSE	print(2 in L1) **Output:** True
Iteration	The for loop is used to iterate over the list elements.	L1 = [1,2,3,4] for item in L1: print(item) **Output:** 1 2 3 4

List (Change Item Values)

```
L = ['apple', 'mango', 'blackberry', '1', '2', '3', 'a', 'b', 'c']
L[1]
'mango'
L[1] = 'peach'
L
['apple', 'peach', 'blackberry', '1', '2', '3', 'a', 'b', 'c']
L[1:3]
['peach', 'blackberry']
L[1:3] = ['watermelon']
L
['apple', 'watermelon', '1', '2', '3', 'a', 'b', 'c']
```

List (Insert, append, pop):

```
L
['apple', 'watermelon', '1', '2', '3', 'a', 'b', 'c']
L.insert(2, "apricot")
L
['apple', 'watermelon', 'apricot', '1', '2', '3', 'a', 'b', 'c']
L.append("orange")
L
['apple', 'watermelon', 'apricot', '1', '2', '3', 'a', 'b', 'c', 'orange']
L.pop()
'orange'
L
['apple', 'watermelon', 'apricot', '1', '2', '3', 'a', 'b', 'c']
L.pop(1)
'watermelon'
L
['apple', 'apricot', '1', '2', '3', 'a', 'b', 'c']
```

- List (Extend, clear, del)

```
L
['apple', 'apricot', '1', '2', '3', 'a', 'b', 'c']
L.extend(['d', 'e', 'f'])
L
['apple', 'apricot', '1', '2', '3', 'a', 'b', 'c', 'd', 'e', 'f']
del L[0]
L
['apricot', '1', '2', '3', 'a', 'b', 'c', 'd', 'e', 'f']
del L
L
Traceback (most recent call last):
  File "<pyshell#408>", line 1, in <module>
    L
NameError: name 'L' is not defined

L = ['apricot', '1', '2', '3']
L
['apricot', '1', '2', '3']
L.clear()
L
[]
```

- List (count, index, remove, reverse, sort):

```
L
['apricot', '1', '2', '3', '1']
L.count('1')
2
L.count('apricot')
1
L.index('1')
1
L.index('3')
3
L
['apricot', '1', '2', '3', '1']
L.remove('1')
L
['apricot', '2', '3', '1']
L.reverse()
L
['1', '3', '2', 'apricot']
L.sort()
L
['1', '2', '3', 'apricot']
```

```
# list sorting on string items
L = ['banana', 'apple', 'cherry']
L.sort()
L
['apple', 'banana', 'cherry']
L.sort(reverse=True)
L
['cherry', 'banana', 'apple']

L = ["Banana", "apple", "Cherry"]
L.sort()
L
['Banana', 'Cherry', 'apple']
L.sort(key=str.lower)
L
['apple', 'Banana', 'Cherry']
# sort by length
L = ["Python", "C", "Java", "Javascript"]
L.sort(key=len)
L
['C', 'Java', 'Python', 'Javascript']

L.sort(key=len, reverse=True)
L
['Javascript', 'Python', 'Java', 'C']
```

Tasks on List:

1. Write commands to perform operations on list L = [5,6,8,9,2,1] to convert it into

   [1,2,5,6,8,9]

   [1,2,6,8,9]

2. Write a code to get first and second best scores from the list:

   L = [85, 86,83,23,45,84,1,2,0]

3. From given list:

   gadgets = ["Mobile", "Laptop", 100, "Camera", 310.28, "Speakers", 27.00, "Television", 1000, "Laptop Case", "Camera Lens"]

   a) Create separate list of strings and numbers
   b) Sort the string list in ascending order
   c) Sort the string list in descending order
   d) Sort the number list in ascending order
   e) Sort the number list in descending order

- Tuple:

- A Tuple is a container used to hold a series of comma-separated values between parenthesis (), such as (x,y) coordinates.
- Tuples are like list, except they are immutable.
- A Tuple is a collection which is ordered and unchangeable.
- The parenthesis is optional, but it is a good practice to use.
- An empty tuple can be created.

```
T = 2,3,4,5
print(T)
(2, 3, 4, 5)
print(type(T))
<class 'tuple'>

T = (2,3,4,5)
print(T)
(2, 3, 4, 5)
print(type(T))
<class 'tuple'>

T = ("SMIT")
print(type(T))
<class 'str'>
T = ("SMIT",)
print(type(T))
<class 'tuple'>
```

- Tuple Properties:

A tuple has following properties:

1. Positive and negative indexes.
2. Slicing.
3. Updating.

```
T = ("apple", "cherry", "banana")
L = list(T)
L
['apple', 'cherry', 'banana']
print(type(L))
<class 'list'>

Tuple = tuple(L)
Tuple
('apple', 'cherry', 'banana')
print(type(Tuple))
<class 'tuple'>
```

- Tuple (Unpacking):

```
fruits = ("apple", "banana", "cherry")
(green, yellow, red) = fruits
print(green)
apple
print(yellow)
banana
print(red)
cherry
```

- Tuple (Using Asterisk):

```
fruits = ("apple", "banana", "cherry", "apricot", "mango")
(green, yellow, *red) = fruits
print(green)
apple
print(yellow)
banana
print(red)
['cherry', 'apricot', 'mango']

# Try this statement  (green, *yellow, red) = fruits

(*green, *yellow, red) = fruits
SyntaxError: multiple starred expressions in assignment
```

- Tuple (Built-in Functions):

➢ count() → Returns the number of times a specified value occurs in a tuple.

➢ index() → Searches the tuple for a specified value and returns the position of where it was found.

```
fruits
('apple', 'banana', 'cherry', 'apricot', 'mango')
fruits.count("apple")
1
fruits.index("apricot")
3
```