Sequence objects

- collection of elements - str, range()
- Container sequences/Objects - list, tuple, dict, set

`Operations on Generic Sequences`

```
- Membership - in | not in
- Iteration - for-loop`
```

`Operations on Ordered/Indexed Sequences`

```
- Indexing - obj[index_pos]
- Slicing - obj[start : stop]
- Concatenation - `+` operator
- Repeatition - `*` operator
```

`Functions on Generic Sequences`

```
 - len() - gives the number of elements in the sequence
 - max() - gives the largest element in the sequence
 - min() - gives the smallest element in the sequence
 - sum() - applicable to numeric sequences, returns the sum of all
elements in the sequence
 - math.prod() - applicable to numeric sequences, returns the
product of all elements in the sequence
 - sorted() - sorts the elements in the sequence in ASC order and
returns a list object
```

Python Sequences and Containers

Object	Container Object	Sequence Type	Element Type	Enclosed in	Immutability	Duplicates
str()	No	ordered/indexed	characters	"" or ''	Yes	Yes
tuple()	Yes	ordered/indexed	mixed data (heterogeneous)	()	Yes	Yes
list()	Yes	ordered/indexed	mixed data (heterogeneous)	[]	No	Yes
set()	Yes	unordered	heterogeneous (immutable objects)	{}	No	No

Object	Container Object	Sequence Type	Element Type	Enclosed in	Immutability	Duplicates
dict()	Yes	unordered	Key - immutable Value - any type	{}	No	Key - No Value - Yes

Note - Strings

In [16]:
```
strg = "It's a nice day"  # combination of single and double quotes
strg
```

Out[16]:
```
"It's a nice day"
```

In [12]:
```
'"Hi" how are you'
```

Out[12]:
```
'"Hi" how are you'
```

In [14]:
```
r"C:\USers\newfolder"  #- raw string
```

Out[14]:
```
'C:\\USers\\newfolder'
```

In [17]:
```
a = 10
b = 20
c = 50
print("Addition of", a, "and", b, "is", c)
```

```
Addition of 10 and 20 is 50
```

In [19]:
```
amt = 500
print("Your current balance is", amt)
```

```
Your currenbt balance is 500
```

In [20]:
```
input("Your current balance is" + str(amt))
```

Out[20]:
```
'x'
```

In [21]:
```
input(f"Your current balance is {amt}")  # formatted string
```

Out[21]:
```
'x'
```

In [ ]:
```

```

Lists in Python

Lists -

- are Python containers
- are an ordered sequence of mixed data

- enclose elements in a pair of square brackets, separated by commas
- mutable

Empty List

In [ ]:
```
lst = []

lst = list()
```

Note - bool() empty list - False

Create a List

In [1]:
```
friends = [ 'Ross', 'Monica', 'Joey', 'Chandler']
```

Retrive elements from List

Indexing and slicing

Ex. Extract first element from the list

In [2]:
```
friends[0]
```

Out[2]:
```
'Ross'
```

Ex. Extract second last element from the list

In [3]:
```
friends[-2]
```

Out[3]:
```
'Joey'
```

Ex. Extract first 3 elements from the list

In [4]:
```
friends[0:3]
```

Out[4]:
```
['Ross', 'Monica', 'Joey']
```

Note - Any sequence can be converted to list objects using list()

Add elements to List

lst.append(`object`) - appends element to the end of the list

Ex. Append 'Phoebe' to the end of the list

In [6]:
```
friends.append("Phoebe")
print(friends)
```

```
['Ross', 'Monica', 'Joey', 'Chandler', 'Phoebe', 'Phoebe']
```

lst.insert(`object`) - inserts element at the mentioned index location

Ex. Insert 'Rachel' at index position 4

```
In [7]: friends.insert(4, "Rachel")
        print(friends)
```

```
['Ross', 'Monica', 'Joey', 'Chandler', 'Rachel', 'Phoebe', 'Phoebe']
```

lst.extend(`seq-obj`) - always take a sequnce as parameter, appends all the elements from sequence to end of the list.

Ex. Extend tuple of new_friends to the end of the friends list

```
In [8]: new_friends = ("Jack", "Rosie", "George", "Jasmine")
        friends.extend(new_friends)
        print(friends)
```

```
['Ross', 'Monica', 'Joey', 'Chandler', 'Rachel', 'Phoebe', 'Phoebe', 'Jack', 'Rosi
e', 'George', 'Jasmine']
```

Modify elements in List

Ex. Replace 'Rosie' with "Amy"

```
In [9]: friends.index("Rosie")
```

```
Out[9]: 8
```

```
In [10]: friends[8] = "Amy"
         print(friends)
```

```
['Ross', 'Monica', 'Joey', 'Chandler', 'Rachel', 'Phoebe', 'Phoebe', 'Jack', 'Amy',
'George', 'Jasmine']
```

Remove elements from a list

lst.pop() - deletes the last element from the list, also returns the deleted value

```
In [22]: friends.pop()
```

```
Out[22]: 'Jasmine'
```

```
In [23]: print(friends)
```

```
['Ross', 'Monica', 'Joey', 'Chandler', 'Rachel', 'Phoebe', 'Phoebe', 'Jack', 'Amy',
'George']
```

lst.pop(`index-value`) - deletes the element at the mentioned `index-value` from the list

```
In [24]: friends.pop(2)
         print(friends)
```

```
['Ross', 'Monica', 'Chandler', 'Rachel', 'Phoebe', 'Phoebe', 'Jack', 'Amy', 'Georg
e']
```

lst.remove(`obj`) - removes the mention `obj` from the list

In [25]:
```
friends.remove("Ross")  # returns None object
print(friends)
```

```
['Monica', 'Chandler', 'Rachel', 'Phoebe', 'Phoebe', 'Jack', 'Amy', 'George']
```

Using slicing operator to remove multiple element from a list

In [26]:
```
del friends[0:3]
print(friends)
```

```
['Phoebe', 'Phoebe', 'Jack', 'Amy', 'George']
```

Operations on Lists

- Iteration
- Membership
- Concatenation
- Repetition

In [27]:
```
list_1 = [1, 2, 3, 4]
```

Conacatenation

In [28]:
```
list_1 + [5, 6, 7]
```

Out[28]:
```
[1, 2, 3, 4, 5, 6, 7]
```

Repetition

In [29]:
```
list_1 * 3
```

Out[29]:
```
[1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4]
```

Built-in functions on Lists

- **len()** - returns length of the list
- **min(), max()** - returns minimum and maximum element from the list
- **sorted()** - sorts the elements of the list and returns a list
- **sum()** - applicable to only numeric lists, returns summation of all the elements int the list

In [30]:
```
numbers = [3, 4, 2, 6, 5, 1]
```

In [31]:
```
len(numbers)
```

Out[31]: 6

In [32]:
```
min(numbers)
```

Out[32]: 1

In [33]:
```
sorted(list_1)
```

Out[33]: [1, 2, 3, 4]

In [34]:
```
sum(numbers)
```

Out[34]: 21

Ex. WAP to create a list of marks of students in 5 subjects (out of 100) and print the percentage

In [37]:
```
no_of_subjects = int(input("Enter number of subjects - "))
marks = []
for i in range(1, no_of_subjects+1):
    m = int(input(f"Enter marks for subject {i}"))
    marks.append(m)
percentage = sum(marks)/no_of_subjects
print(f"Percentage - {round(percentage, 2)}")
```

```
Percentage - 71.67
```

List Methods

lst.index(`object`) - returns the index position of first occurence of the `object`

In [38]:
```
friends = ['Ross', 'Monica', 'Rachel', 'Joey', 'Phoebe']
friends.index("Joey")
```

Out[38]: 3

lst.count(`object`) - returns the `count` of number of times the object is repeated in the list

In [39]:
```
friends.count("Joey")
```

Out[39]: 1

lst.sort() - sorts the list object

In [40]:
```
friends = ['Ross', 'Monica', 'Rachel', 'Joey', 'Phoebe']
friends.sort()
```

In [41]:
```
result = friends.sort()  # returns None and modifies original
print(result)
```

```
None
```

In [42]:
```
result = sorted(friends)  # returns a list
print(result)
```

```
['Joey', 'Monica', 'Phoebe', 'Rachel', 'Ross']
```

lst.reverse() - reverses the sequence of elements in the list

In [43]:
```
friends = ['Ross', 'Monica', 'Rachel', 'Joey', 'Phoebe']
friends.reverse()  # original is modified
print(friends)
```

```
['Phoebe', 'Joey', 'Rachel', 'Monica', 'Ross']
```

In [44]:
```
friends[::-1]
```

Out[44]:
```
['Ross', 'Monica', 'Rachel', 'Joey', 'Phoebe']
```

In [46]:
```
list(reversed(friends))
```

Out[46]:
```
['Ross', 'Monica', 'Rachel', 'Joey', 'Phoebe']
```

In [48]:
```
list(reversed("abcd"))
```

Out[48]:
```
['d', 'c', 'b', 'a']
```

In [49]:
```
"".join(reversed("abcd"))
```

Out[49]:
```
'dcba'
```

Creating a copy of a list

(Shallow copying and Deep Copying)

In Python, Assignment statements do not copy objects, they create bindings between a target and an object. When we use = operator user thinks that this creates a new object; well, it doesn't. It only creates a new variable that shares the reference of the original object. Sometimes a user wants to work with mutable objects, in order to do that user looks for a way to create "real copies" or "clones" of these objects. Or, sometimes a user wants copies that user can modify without automatically modifying the original at the same time, in order to do that we create copies of objects.

A copy is sometimes needed so one can change one copy without changing the other. In Python, there are two ways to create copies :

- Deep copy
- Shallow copy

In [50]:
```
lst1 = [2, 8, ["Jane", "Thomas", "Jack"]]
lst2 = lst1
```

```
lst1[0] = 10
print(lst1,"\n", lst2)
```

```
[10, 8, ['Jane', 'Thomas', 'Jack']]
 [10, 8, ['Jane', 'Thomas', 'Jack']]
```

Shallow Copy

In [51]:
```
lst1 = [2, 8, ["Jane", "Thomas", "Jack"]]
lst2 = lst1.copy() # lst1[:]
lst1[0] = 10
print(lst1,"\n", lst2)
```

```
[10, 8, ['Jane', 'Thomas', 'Jack']]
 [2, 8, ['Jane', 'Thomas', 'Jack']]
```

Deep Copy

In [54]:
```
import copy
lst1 = [2, 8, ["Jane", "Thomas", "Jack"]]
lst2 = copy.deepcopy(lst1)
lst1[2][0] = "George"
print(lst1,"\n", lst2)
```

```
[2, 8, ['George', 'Thomas', 'Jack']]
 [2, 8, ['Jane', 'Thomas', 'Jack']]
```

Ex. Write the program to multiply the two lists

In [55]:
```
num_list_1 = [1, 2, 3, 4]
num_list_2 = [0, 5, 2, 1]
```

Utility Functions

zip(`lst1` , `lst2` , ...)

- takes two or more ordered sequences
- combines the elements index-wise in the form of tuples
- returns a sequence of tuples

In [56]:
```
zip(num_list_1, num_list_2)
```

Out[56]:
```
<zip at 0x147a813f1c0>
```

In [57]:
```
list(zip(num_list_1, num_list_2))
```

Out[57]:
```
[(1, 0), (2, 5), (3, 2), (4, 1)]
```

In [58]:
```
for i in zip(num_list_1, num_list_2) :
    print(i)
```

```
(1, 0)
(2, 5)
(3, 2)
(4, 1)
```

In [62]:
```
for i in zip(num_list_1, num_list_2) :
    print(i[0] * i[1], end = " ")
```

```
0 10 6 4
```

In [61]:
```
for i,j in zip(num_list_1, num_list_2) :  # application of unpacking of tuples
    print(i*j)
```

```
0
10
6
4
```

enumerate(`seq-obj` , `start=0`) - adds counter to an iterable and returns a sequence of tuples (the enumerate object).

Ex. WAP to multiply the elements of the list with counter starting from 1

In [64]:
```
lst = [10, 20, 30, 40, 50]
enumerate(lst)
```

Out[64]:
```
<enumerate at 0x147a7f43150>
```

In [65]:
```
list(enumerate(lst))
```

Out[65]:
```
[(0, 10), (1, 20), (2, 30), (3, 40), (4, 50)]
```

In [66]:
```
list(enumerate(lst, start = 1))
```

Out[66]:
```
[(1, 10), (2, 20), (3, 30), (4, 40), (5, 50)]
```

In [68]:
```
for i, j in enumerate(lst, start = 1):
    print(i*j, end = " ")
```

```
10 40 90 160 250
```

Ex. WAP to calculate percentage of each student and print in tablular format assuming student ID as 101, 102, 103

In [73]:
```
marks = [(50, 60, 75), (80, 76, 98), (55, 74, 67), (74, 83, 92), (85, 45, 97)]

print("-" * 30)
print(f"Student ID \t Percentage")
print("-" * 30)

for counter, mar in enumerate(marks, start = 101) :
    print(f"{counter} \t \t {round(sum(mar)/len(mar))}%")
```

```
-----------------------------
Student ID       Percentage
-----------------------------
101              62%
102              85%
103              65%
104              83%
105              76%
```

Sets in Python

Sets -

- are Python containers
- are an unordered sequence of mixed data (immutable objects)
- encloses elements in a pair of curly brackets, separated by commas
- mutable
- do not allow duplicates
- allow set operations on data

Creating a set

In [74]:
```
s = {10, 20, 30, 40, 50}
print(s)
```

```
{50, 20, 40, 10, 30}
```

Empty Set

In [75]:
```
set()
```

Out[75]:
```
set()
```

Note - bool() of empty set is always False

Add elements to Set

set.add(`obj`)

- adds a new element to the set

In [76]:
```
s.add("abcd")
print(s)
```

```
{50, 20, 'abcd', 40, 10, 30}
```

set.update(`seq`)

- takes a sequence object as a parameter and adds all the elemnts from the sequence to the set

In [77]:
```
s.update([1, 2, 3, 4])
print(s)
```

```
{1, 2, 3, 4, 'abcd', 10, 20, 30, 40, 50}
```

Remove elememnt from sets

pop()

- removes a random element from the set

In [78]:
```
s.pop()
print(s)
```

```
{2, 3, 4, 'abcd', 10, 20, 30, 40, 50}
```

remove(`obj`)

- removes a specified elemnt from the set, givevs error if the element is not present in the set

In [79]:
```
s.remove("abcd")
print(s)
```

```
{2, 3, 4, 10, 20, 30, 40, 50}
```

discard(`obj`)

- removes the specified element from the set, it will not give any error if element is not present.

In [80]:
```
s.remove("abcd")
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[80], line 1
----> 1 s.remove("abcd")

KeyError: 'abcd'
```

In [82]:
```
print(s.discard("abcd"))
print(s)
```

```
None
{2, 3, 4, 10, 20, 30, 40, 50}
```

Built-in functions on Sets

- **len()** - returns length of the sets
- **min(), max()** - returns minimum and maximum element from the set
- **sorted()** - sorts the elements of the set and returns a list
- **sum()** - applicable to only numeric sets, returns summation of all the elements int the set

In [83]:
```
set_a = {10, 20, 30, 40, 50, 20}
```

In [84]:
```
len(set_a)
```

Out[84]: 5

In [85]:
```
min(set_a)
```

Out[85]: 10

In [86]:
```
sum(set_a)
```

Out[86]: 150

In [87]:
```
sorted(set_a)
```

Out[87]: [10, 20, 30, 40, 50]

Operations on Sets

- Iteration
- Membership
- Set Operations
  - Union | Intersection | Difference | Symmetric Difference
  - Disjoint sets
  - Subsets and Supersets

In [89]:
```
# STrictly avoided
lst = [1, 2, 3, 4]
for i in range(len(lst)):
    print(lst[i])
```

```
1
2
3
4
```

In [90]:
```
for i in range(len(s)) :
    print(s[i])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[90], line 2
      1 for i in range(len(s)) :
----> 2     print(s[i])

TypeError: 'set' object is not subscriptable
```

In [91]:
```
for i in s :
    print(i)
```

```
2
3
4
10
20
30
40
50
```

Union | Intersection | Difference | Symmetric Difference

In [92]:
```
set1 = {1, 2, 3, 4, 5}
set2 = {4, 5, 6, 7, 8}
```

In [94]:
```
set1 | set2 # union operator
set1.union(set2)  # union set method
```

Out[94]: {1, 2, 3, 4, 5, 6, 7, 8}

In [95]:
```
set1 & set2 # intersection
set1.intersection(set2)
```

Out[95]: {4, 5}

In [96]:
```
set1 ^ set2
set1.symmetric_difference(set2)
```

Out[96]: {1, 2, 3, 6, 7, 8}

In [97]:
```
set1 - set2
set1.difference(set2)
```

Out[97]: {1, 2, 3}

Disjoint set

- if the two sets have no common elements

In [98]:
```
set1 = {1, 2, 3, 4, 5}
set2 = {6, 7, 8, 9, 10}

set1.isdisjoint(set2)
```

Out[98]: True

Subset | Superset

- If all elemenets of set1 are present in set2 then,
  - set1 is subset of set2
  - set2 will be superset of set1

In [99]:
```
set1 = {1, 2, 3, 4, 5}
set2 = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

set1.issubset(set2)
```

Out[99]: True

In [100…
```
set2.issuperset(set1)
```

Out[100… True

Examples -

Set of members drinking tea and coffee

In [101…
```
drinks_coffee = {"Jane", "Jack", "Sam", "George", "Dori"}

drinks_tea = {"Jack", "Frank", "Cody", "Dori", "Bill"}
```

Ex. Are there any members who drink tea and coffee both? (Yes/No). Display their names

In [102…
```
if drinks_coffee.isdisjoint(drinks_tea) :
    print("No common members")
else:
    print("Yes,", drinks_coffee.intersection(drinks_tea))
```

```
Yes, {'Dori', 'Jack'}
```

Ex. Does all the members who drink tea also drink coffee? (Yes/No). Display their names of memebers who drink tea but not coffee

In [103…
```
if drinks_tea.issubset(drinks_coffee) :
    print("Yes")
else:
    print("No,", drinks_tea.difference(drinks_coffee))
```

```
No, {'Bill', 'Frank', 'Cody'}
```

Ex. WAP to check if password entered by user is satisfying following condions or not -

1. length of password must be >= 8
2. username and password must not be same
3. Password must contain atleast 1 digit, 1 capital alphabet, 1 small alphabet and 1 special character like "!@#$%^&*"

In [117…
```
import string
username = "Jane"
password = "Jane@1234"

result = []

result.append(len(password) >= 8)
result.append(username != password)
result.append(set(password).intersection(set(string.ascii_lowercase)))
result.append(set(password).intersection(set(string.ascii_uppercase)))
result.append(set(password).intersection(set(string.digits)))
result.append(set(password).intersection(set("!@#$%^&*")))
print(result)
if all(result):
    print("Valid")
else:
    print("Invalid")
```

```
[True, True, {'e', 'a', 'n'}, {'J'}, {'3', '4', '2', '1'}, {'@'}]
Valid
```

In [105…
```
import string
print(dir(string))
```

```
['Formatter', 'Template', '_ChainMap', '__all__', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__', '_re', '_sentinel_dict', '_string', 'ascii_letters', 'ascii_lowercase', 'ascii_uppercase', 'capwords', 'digits', 'hexdigits', 'octdigits', 'printable', 'punctuation', 'whitespace']
```

In [106…
```
string.ascii_lowercase
```

Out[106…
```
'abcdefghijklmnopqrstuvwxyz'
```

- **`all()`** - returns True if all the elements in the list are True
- **`any()`** - returns True if any one element in the list is True

In [114…
```
all([1, 2, 3, 4, 5, 0])
```

Out[114…
```
False
```

In [115…
```
any([1, 2, 3, 4, 5, 0])
```

Out[115…
```
True
```

Dictionary in Python

Dictionaries are -

- are Python containers

- are an unordered(not indexing) sequence of mixed data, maintains the order, deletes in LIFO
- Encloses elements in a pair of curly brackets
- elements are stored in the form of {key : value} pairs separated by commas
- keys are always unique and immutable
- values need not be unique and can be of any type
- mutable

Empty Dictionary

In [118…
```
{}

dict()
```

Out[118…
```
{}
```

Note - bool() of empty dict is always False

Creating a dictionary

Ex. Create a dictionary consisting of country names and their currencies

In [120…
```
countries = {"India" : "INR", "USA" : "USD"}
print(countries)
```

```
{'India': 'INR', 'USA': 'USD'}
```

In [ ]:
```
[{"Name" : "Jane",
     "Salary" : 60000},
    {"Name" : "Jack",
     "Salary" : 70000},
]
```

In [ ]:
```
{"Name" : ["Jane", "Jack"],
"Salary" : [ 60000, 70000]}
```

Retriving elements from a Dictionary

Ex. Print currency for "India"

In [121…
```
countries["India"]
```

Out[121…
```
'INR'
```

Ex. Print currency for "Japan"

In [122…
```
countries["Japan"]
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[122], line 1
----> 1 countries["Japan"]

KeyError: 'Japan'
```

dict.get() - returns the value of the item with the specified key

dictionary.get(keyname, value)

- keyname - Required. The keyname of the item you want to return the value from
- value - Optional. A value to return if the specified key does not exist. Default value None

In [123…
```
countries.get("India")
```

Out[123…
```
'INR'
```

In [124…
```
print(countries.get("Japan"))
```

```
None
```

In [125…
```
print(countries.get("Japan", "Not Present"))
```

```
Not Present
```

Adding new element to dictionary

Ex. Add Japan and its currency to dictionary

In [127…
```
countries["Japan"] = "Yen"
print(countries)
```

```
{'India': 'INR', 'USA': 'USD', 'Japan': 'Yen'}
```

Modifying dictionary

Ex. Modify the currency for USA as "$"

In [130…
```
countries["USA"] = "$"
print(countries)
```

```
{'India': 'INR', 'USA': '$', 'Japan': 'Yen'}
```

Updating a dictionary

dict.update(`new_dict`) - inserts the specified items to the dictionary

Ex. Add contents from new_country dictionary to countries

In [132…
```
new_dict = {"Indonesia" : "IDR", "Singapore" : "SGD", "Thailand" : "Bhat"}
countries.update(new_dict)
```

```
print(countries)
```

```
{'India': 'INR', 'USA': '$', 'Japan': 'Yen', 'Indonesia': 'IDR', 'Singapore': 'SGD',
'Thailand': 'Bhat'}
```

Remove element from dictionary

dict.pop(key)

- removes the specified key an its value from the dictionary

In [133…
```
countries.pop("India")
print(countries)
```

```
{'USA': '$', 'Japan': 'Yen', 'Indonesia': 'IDR', 'Singapore': 'SGD', 'Thailand': 'Bh
at'}
```

dict.popitem()

- randomly removes a key-value pair from dictionary

In [134…
```
countries.popitem()
```

Out[134…
```
('Thailand', 'Bhat')
```

In [135…
```
countries
```

Out[135…
```
{'USA': '$', 'Japan': 'Yen', 'Indonesia': 'IDR', 'Singapore': 'SGD'}
```

dict.clear()

- removes all the pairs from the dictionary

In [ ]:
```
countries.clear()
```

Ex. Write a program to retrieve the capital of 'Germany' from a given dictionary

In [136…
```
europe = { 'Spain': { 'Capital':'Madrid', 'Population':4.77 },
           'France': { 'Capital':'Paris', 'Population':6.7 },
           'Germany': { 'Capital':'Berlin', 'Population':8.28 },
           'Norway': { 'Capital':'Oslo', 'Population':0.533 }}
europe["Germany"]["Capital"]
```

Out[136…
```
'Berlin'
```

Functions/Operations on Dictionary

- Iteration - for loop, dict.keys(), dict.values(), dict.items()
- Membership
- len(), sorted()

In [139…
```
employees = {'Jane': 70000, 'Rosie': 90000, 'Mary': 40000, 'Sam': 55000, 'George':
for i in employees :
    print(i, " - ", employees[i])
```

```
Jane  -  70000
Rosie  -  90000
Mary  -  40000
Sam  -  55000
George  -  76000
```

In [140…
```
employees.keys()
```

Out[140…
```
dict_keys(['Jane', 'Rosie', 'Mary', 'Sam', 'George'])
```

In [141…
```
employees.values()
```

Out[141…
```
dict_values([70000, 90000, 40000, 55000, 76000])
```

In [142…
```
employees.items()
```

Out[142…
```
dict_items([('Jane', 70000), ('Rosie', 90000), ('Mary', 40000), ('Sam', 55000),
('George', 76000)])
```

In [145…
```
for key, val in employees.items():
    print(key, " - ", val)
```

```
Jane  -  70000
Rosie  -  90000
Mary  -  40000
Sam  -  55000
George  -  76000
```

In [146…
```
sorted(employees)
```

Out[146…
```
['George', 'Jane', 'Mary', 'Rosie', 'Sam']
```

Creating dictionaries using sequences

Ex. WAP to create a dictionary where keys are employee codes starting from 101 and its values are the employee names

In [149…
```
names = ['Jane', 'Rosie', 'Mary', 'Sam', 'George']
dict(enumerate(names, start = 101))
```

Out[149…
```
{101: 'Jane', 102: 'Rosie', 103: 'Mary', 104: 'Sam', 105: 'George'}
```

Ex. WAP to create a dictionary combining the following two lists where name is key and marks as value

In [156…
```
names = ['Jane', 'Rosie', 'Mary', 'Sam', 'George']
salary = [70000, 90000, 40000, 55000, 76000]

dict(zip(names, salary))
```

Out[156…
```
{'Jane': 70000, 'Rosie': 90000, 'Mary': 40000, 'Sam': 55000, 'George': 76000}
```

Ex. WAP to create a dict of employes with emp_id as keys starting from 101 and a tuple of names and their ages.

In [157…
```
names = ['Jane', 'Rosie', 'Mary', 'Sam', 'George']
ages = [40, 30, 44, 25, 56]
dict(enumerate(zip(names, ages), start = 101))
```

Out[157…
```
{101: ('Jane', 40),
 102: ('Rosie', 30),
 103: ('Mary', 44),
 104: ('Sam', 25),
 105: ('George', 56)}
```

In [155…
```
data = [{"Name" : "Jane",
     "Salary" : 60000},
    {"Name" : "Jack",
     "Salary" : 70000}]
```

```
Name
Salary
Name
Salary
```

Out[155…
```
[]
```

Ex. WAP to print the sum of squares of numbers from 1-5

In [158…
```
# reducing the sequence to a single value/object
total = 0
for i in range(1, 6):
    total += i**2
total
```

Out[158…
```
55
```

Ex. WAP to generate a list of squares of numbers from 1-10

In [159…
```
# comprehension - generate a new mutable DS based on expression applied to each ele
squares = []
for i in range(1, 6):
    squares.append(i**2)
squares
```

Out[159…
```
[1, 4, 9, 16, 25]
```

Comprehensions in Python

`Comprehensions` are an elegant way to define and create mutable data structures like lists, sets, dictionary based on existing sequences Syntax –

`[<expression> for <var> in <sequence> if <condition>]`

1. Identify the sequence
2. Identify condition if any
3. Expression
4. Mutable datastructure

Ex. WAP to generate a list of squares of number in range of 1-10

In [160…
```
[i**2 for i in range(1, 11)]
```

Out[160…
```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Ex. WAP to create a list squares of even number in range of 1-10

In [161…
```
[i**2 for i in range(1, 11) if i % 2 == 0]
```

Out[161…
```
[4, 16, 36, 64, 100]
```

Ex. WAP to create a dict of number from 1-10 as keys and their squares as values

In [162…
```
{i : i**2 for i in range(1, 11)}
```

Out[162…
```
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
```

Ex. WAP to create a dict of numbers from 1-5 as keys and values will be the type of the number as even or odd

In [163…
```
{i : "even" if i % 2 == 0 else "odd" for i in range(1, 6)}
```

Out[163…
```
{1: 'odd', 2: 'even', 3: 'odd', 4: 'even', 5: 'odd'}
```

Ex. WAP to add 7% service tax to all the values in the "sales" list

In [164…
```
sales = [290, 500, 800, 650]
[i * 1.07 for i in sales]
```

Out[164…
```
[310.3, 535.0, 856.0, 695.5]
```

Ex. WAP to sum all the values in the "sales" tuple

In [166…
```
sales = ("$290", "$500", "$800", "$650")
sum([int(i.replace("$", "")) for i in sales])
```

Out[166…
```
2240
```

Ex. WAP to extract all the digits from the string using list comprehension

In [169…
```
string = "Current assets is worth 2450000 and current liabilities stands at 1230000
[int(ch) for ch in string if ch.isdigit()]
```

Out[169…
```
[2, 4, 5, 0, 0, 0, 0, 1, 2, 3, 0, 0, 0, 0]
```

Ex. Extract all the numbers from the string using list comprehension

In [170…
```
string = "Current assets is worth 2450000 and current liabilities stands at 1230000
[int(ch) for ch in string.split() if ch.isdigit()]
```

Out[170…
```
[2450000, 1230000]
```

Ex. WAP to create a dict of names and the total marks(percentage) of each student.

In [175…
```
names = ['Jane', 'Rosie', 'Mary', 'Sam', 'George']
marks = ([70, 65, 32], [90, 76, 98], [40, 55, 78], [50, 87, 67], [76, 72, 89])

result = dict(zip(names, [round(sum(m)/len(m)) for m in marks]))
result
```

Out[175…
```
{'Jane': 56, 'Rosie': 88, 'Mary': 58, 'Sam': 68, 'George': 79}
```

Ex. Are there any students with percentage more 70? yes/no

In [179…
```
any([i>70 for i in result.values()])
```

Out[179…
```
True
```

Ex. Names of students getting more than 70

In [176…
```
[name for name in result if result[name] > 70 ]
```

Out[176…
```
['Rosie', 'George']
```

Functions in Python

A function is set of statements that take input in the form of parameters, performs computation on the input and returns a result in the form of return statement

Syntax –

def `function-name` (`parameters if any`):

 # function code

`return` statement

Note : It is a best practice to avoid usage of input() and print() functions in a function definition

WAF to calculate factorial of a number

In [188…
```
def factorial(num):
    if type(num) == int :
```

```
        fact = 1
        for i in range(num, 1, -1) :
            fact *= i
        return fact
    else :
        return "Invalid"
```

In [189…
```
factorial(5)
```

Out[189…
```
120
```

In [190…
```
print(factorial("abcd"))
```

```
Invalid
```

In [197…
```
def func(num) :
    return num**2, num**3  # packing of tuples
```

In [198…
```
a, b = func(5)  # unpacking of tuples
```

Function Arguments

- Required Positional Arguments
- Default Arguments
- Variable length Arguments
- Key-word Arguments
- Variable-length Keyword Arguments

Required Positional Argument

In [204…
```
def demo(name, age) :
    print(f"Name - {name} | Age - {age}")
demo("Jack", 30)
demo(30, "Jack")
demo ("Jack")
```

```
Name - Jack | Age - 30
Name - 30 | Age - Jack
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[204], line 5
      3 demo("Jack", 30)
      4 demo(30, "Jack")
----> 5 demo ("Jack")

TypeError: demo() missing 1 required positional argument: 'age'
```

In [205…
```
len()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[205], line 1
----> 1 len()

TypeError: len() takes exactly one argument (0 given)
```

In [209…
```
lst = [10, 20, 30, 40, 50]
lst.insert(5)  # insert 5
lst
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[209], line 2
      1 lst = [10, 20, 30, 40, 50]
----> 2 lst.insert(5)  # insert 5
      3 lst

TypeError: insert expected 2 arguments, got 1
```

In [206…
```
lst = [10, 20, 30, 40, 50]
lst.insert(5, 1)  # insert 5 at index 1
lst
```

Out[206…
```
[10, 20, 30, 40, 50, 1]
```

In [208…
```
lst = [10, 20, 30, 40, 50]
lst.insert("abcd", 1)  # insert "abcd" at index 1
lst
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[208], line 2
      1 lst = [10, 20, 30, 40, 50]
----> 2 lst.insert("abcd", 1)  # insert "abcd" at index 1
      3 lst

TypeError: 'str' object cannot be interpreted as an integer
```

In [207…
```
help(lst.insert)
```

```
Help on built-in function insert:

insert(index, object, /) method of builtins.list instance
    Insert object before index.
```

Default Argument

In [214…
```
def demo(name, age = 25) :
    print(f"Name - {name} | Age - {age}")
demo("Jack", 30)
demo(30, "Jack")
demo("Jack")
```

```
Name - Jack | Age - 30
Name - 30 | Age - Jack
Name - Jack | Age - 25
```

Example - str.replace()

In [210…
```
strg = "mississippi"
strg.replace("i", "*")
```

Out[210…
```
'm*ss*ss*pp*'
```

In [212…
```
strg = "mississippi"
strg.replace("i", "*", 2)
```

Out[212…
```
'm*ss*ssippi'
```

In [211…
```
help(strg.replace)
```

```
Help on built-in function replace:

replace(old, new, count=-1, /) method of builtins.str instance
    Return a copy with all occurrences of substring old replaced by new.

      count
        Maximum number of occurrences to replace.
        -1 (the default value) means replace all occurrences.

    If the optional argument count is given, only the first count occurrences are
    replaced.
```

In [215…
```
help(enumerate)
```

```
Help on class enumerate in module builtins:

class enumerate(object)
 |  enumerate(iterable, start=0)
 |
 |  Return an enumerate object.
 |
 |    iterable
 |      an object supporting iteration
 |
 |  The enumerate object yields pairs containing a count (from start, which
 |  defaults to zero) and a value yielded by the iterable argument.
 |
 |  enumerate is useful for obtaining an indexed list:
 |      (0, seq[0]), (1, seq[1]), (2, seq[2]), ...
 |
 |  Methods defined here:
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __iter__(self, /)
 |      Implement iter(self).
 |
 |  __next__(self, /)
 |      Implement next(self).
 |
 |  __reduce__(...)
 |      Return state information for pickling.
 |
 |  ----------------------------------------------------------------------
 |  Class methods defined here:
 |
 |  __class_getitem__(...)
 |      See PEP 585
 |
 |  ----------------------------------------------------------------------
 |  Static methods defined here:
 |
 |  __new__(*args, **kwargs)
 |      Create and return a new object.  See help(type) for accurate signature.
```

Variable-length Argument

In [218…

```python
def demo(name, *args, age = 18) :
    print(f"Name - {name} | Age - {age} | Args - {args}")

demo("Jane", 60, 70, 80, 90, 19)
```

```
Name - Jane | Age - 18 | Args - (60, 70, 80, 90, 19)
```

Key-word Argument

In [219…

```python
demo("Jane", 60, 70, 80, 90, age = 19)
```

```
Name - Jane | Age - 19 | Args - (60, 70, 80, 90)
```

Variable-length Keyword Argument

In [222…
```python
def demo(name, *args, age = 18, **kwargs) :
    print(f"Name - {name} | Age - {age} | Args - {args} | Kwargs - {kwargs}")

demo("Jane", 60, 70, 80, 90, age = 19, gender = "F", mob = 98765434)
```

```
Name - Jane | Age - 19 | Args - (60, 70, 80, 90) | Kwargs - {'gender': 'F', 'mob': 9
8765434}
```

Example on unpacking of tuples/dict wrt function call

- Writing data to a file

In [228…
```python
def write_to_file(name, age, gender) :
    with open("details.txt", "a") as file :
        file.write(f"{name} | {age} | {gender}\n")
        print(f"Data for {name} added to the file.")
```

In [230…
```python
data = [["Rosie", "F", 35], ["Jack", "M", 29], ["George", "M", 32]]
for lst in data :
    write_to_file(*lst)
```

```
Data for Rosie added to the file.
Data for Jack added to the file.
Data for George added to the file.
```

In [233…
```python
data = [
    {"name" : "Rosie", "gender" :"F", "age" :30},
    {"name" : "Jack", "gender": "M", "age" :35},
    {"name" : "Sam", "gender": "M", "age" :32},
]
for dct in data :
    write_to_file(**dct)
```

```
Data for Rosie added to the file.
Data for Jack added to the file.
Data for Sam added to the file.
```

In [234…
```python
help(sorted)
```

```
Help on built-in function sorted in module builtins:

sorted(iterable, /, *, key=None, reverse=False)
    Return a new list containing all items from the iterable in ascending order.

    A custom key function can be supplied to customize the sort order, and the
    reverse flag can be set to request the result in descending order.
```

In [235…
```python
help(str.replace)
```

```
Help on method_descriptor:

replace(self, old, new, count=-1, /) unbound builtins.str method
    Return a copy with all occurrences of substring old replaced by new.

      count
        Maximum number of occurrences to replace.
        -1 (the default value) means replace all occurrences.

    If the optional argument count is given, only the first count occurrences are
    replaced.
```

- `/` - All the arguments before / must be posiotnal only
- `*` - All the arguments after * must be key-word only

In [237…
```python
def demo(name, age) :
    print(f"Name - {name} | Age - {age}")

demo("Jane", 30)  # positional only
demo("Jane", age = 30)  # one positional and 1 keyword
demo(age = 30, name = "Jane") # key-word only
```

```
Name - Jane | Age - 30
Name - Jane | Age - 30
Name - Jane | Age - 30
```

In [238…
```python
def demo(name,*, age) :  # name can be either but age must be key-word argument
    print(f"Name - {name} | Age - {age}")

demo("Jane", age = 30)
demo(age = 30, name = "Jane")
demo("Jane", 30)
```

```
Name - Jane | Age - 30
Name - Jane | Age - 30
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[238], line 6
      4 demo("Jane", age = 30)
      5 demo(age = 30, name = "Jane")
----> 6 demo("Jane", 30)

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

In [239…
```python
def demo(name, /, age) :  # name must be positional only but age can be either
    print(f"Name - {name} | Age - {age}")

demo("Jane", 30)
demo("Jane", age = 30)
demo(age = 30, name = "Jane")
```

```
Name - Jane | Age - 30
Name - Jane | Age - 30
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[239], line 6
      4 demo("Jane", 30)
      5 demo("Jane", age = 30)
----> 6 demo(age = 30, name = "Jane")

TypeError: demo() got some positional-only arguments passed as keyword arguments: 'n
ame'
```

Scope of variables

In [242…
```
def outer():
    # a = 15  # enclosed scope
    def inner():
        # a = 20  # local variable
        print(a)
    inner()
a = 10  # global
outer()
```

```
10
```

In [255…
```
from math import pi  # built-in scope
def outer():
    pi = 15  # enclosed scope
    def inner():
        # pi = 20  # local variable
        print("inner func", pi)
    inner()
# pi = 10  # global
outer()
print("main", pi)
```

```
inner func 15
main 3.141592653589793
```

In [248…
```
sum = 0
for i in range(1, 10):
    sum+= i
sum
```

Out[248… 45

In [249…
```
sum(range(1, 10))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[249], line 1
----> 1 sum(range(1, 10))

TypeError: 'int' object is not callable
```

In [250…
```
print(dir(__builtins__))
```

```
['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException', 'BaseExcept
ionGroup', 'BlockingIOError', 'BrokenPipeError', 'BufferError', 'BytesWarning', 'Chi
ldProcessError', 'ConnectionAbortedError', 'ConnectionError', 'ConnectionRefusedErro
r', 'ConnectionResetError', 'DeprecationWarning', 'EOFError', 'Ellipsis', 'EncodingW
arning', 'EnvironmentError', 'Exception', 'ExceptionGroup', 'False', 'FileExistsErro
r', 'FileNotFoundError', 'FloatingPointError', 'FutureWarning', 'GeneratorExit', 'IO
Error', 'ImportError', 'ImportWarning', 'IndentationError', 'IndexError', 'Interrupt
edError', 'IsADirectoryError', 'KeyError', 'KeyboardInterrupt', 'LookupError', 'Memo
ryError', 'ModuleNotFoundError', 'NameError', 'None', 'NotADirectoryError', 'NotImpl
emented', 'NotImplementedError', 'OSError', 'OverflowError', 'PendingDeprecationWarn
ing', 'PermissionError', 'ProcessLookupError', 'RecursionError', 'ReferenceError',
'ResourceWarning', 'RuntimeError', 'RuntimeWarning', 'StopAsyncIteration', 'StopIter
ation', 'SyntaxError', 'SyntaxWarning', 'SystemError', 'SystemExit', 'TabError', 'Ti
meoutError', 'True', 'TypeError', 'UnboundLocalError', 'UnicodeDecodeError', 'Unicod
eEncodeError', 'UnicodeError', 'UnicodeTranslateError', 'UnicodeWarning', 'UserWarni
ng', 'ValueError', 'Warning', 'WindowsError', 'ZeroDivisionError', '__IPYTHON__', '_
_build_class__', '__debug__', '__doc__', '__import__', '__loader__', '__name__', '__
package__', '__spec__', 'abs', 'aiter', 'all', 'anext', 'any', 'ascii', 'bin', 'boo
l', 'breakpoint', 'bytearray', 'bytes', 'callable', 'chr', 'classmethod', 'compile',
'complex', 'copyright', 'credits', 'delattr', 'dict', 'dir', 'display', 'divmod', 'e
numerate', 'eval', 'exec', 'execfile', 'filter', 'float', 'format', 'frozenset', 'ge
t_ipython', 'getattr', 'globals', 'hasattr', 'hash', 'help', 'hex', 'id', 'input',
'int', 'isinstance', 'issubclass', 'iter', 'len', 'license', 'list', 'locals', 'ma
p', 'max', 'memoryview', 'min', 'next', 'object', 'oct', 'open', 'ord', 'pow', 'prin
t', 'property', 'range', 'repr', 'reversed', 'round', 'runfile', 'set', 'setattr',
'slice', 'sorted', 'staticmethod', 'str', 'sum', 'super', 'tuple', 'type', 'vars',
'zip']
```

In [253…

```
import math
print(dir(math))
```

```
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'a
sin', 'asinh', 'atan', 'atan2', 'atanh', 'cbrt', 'ceil', 'comb', 'copysign', 'cos',
'cosh', 'degrees', 'dist', 'e', 'erf', 'erfc', 'exp', 'exp2', 'expm1', 'fabs', 'fact
orial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose',
'isfinite', 'isinf', 'isnan', 'isqrt', 'lcm', 'ldexp', 'lgamma', 'log', 'log10', 'lo
g1p', 'log2', 'modf', 'nan', 'nextafter', 'perm', 'pi', 'pow', 'prod', 'radians', 'r
emainder', 'sin', 'sinh', 'sqrt', 'sumprod', 'tan', 'tanh', 'tau', 'trunc', 'ulp']
```

In [254…

```
help(math.pi)
```

```
Help on float object:

class float(object)
 |  float(x=0, /)
 |
 |  Convert a string or number to a floating point number, if possible.
 |
 |  Methods defined here:
 |
 |  __abs__(self, /)
 |      abs(self)
 |
 |  __add__(self, value, /)
 |      Return self+value.
 |
 |  __bool__(self, /)
 |      True if self else False
 |
 |  __ceil__(self, /)
 |      Return the ceiling as an Integral.
 |
 |  __divmod__(self, value, /)
 |      Return divmod(self, value).
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __float__(self, /)
 |      float(self)
 |
 |  __floor__(self, /)
 |      Return the floor as an Integral.
 |
 |  __floordiv__(self, value, /)
 |      Return self//value.
 |
 |  __format__(self, format_spec, /)
 |      Formats the float according to format_spec.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __getnewargs__(self, /)
 |
 |  __gt__(self, value, /)
 |      Return self>value.
 |
 |  __hash__(self, /)
 |      Return hash(self).
 |
 |  __int__(self, /)
 |      int(self)
 |
```

```
|  __le__(self, value, /)
|      Return self<=value.
|
|  __lt__(self, value, /)
|      Return self<value.
|
|  __mod__(self, value, /)
|      Return self%value.
|
|  __mul__(self, value, /)
|      Return self*value.
|
|  __ne__(self, value, /)
|      Return self!=value.
|
|  __neg__(self, /)
|      -self
|
|  __pos__(self, /)
|      +self
|
|  __pow__(self, value, mod=None, /)
|      Return pow(self, value, mod).
|
|  __radd__(self, value, /)
|      Return value+self.
|
|  __rdivmod__(self, value, /)
|      Return divmod(value, self).
|
|  __repr__(self, /)
|      Return repr(self).
|
|  __rfloordiv__(self, value, /)
|      Return value//self.
|
|  __rmod__(self, value, /)
|      Return value%self.
|
|  __rmul__(self, value, /)
|      Return value*self.
|
|  __round__(self, ndigits=None, /)
|      Return the Integral closest to x, rounding half toward even.
|
|      When an argument is passed, work like built-in round(x, ndigits).
|
|  __rpow__(self, value, mod=None, /)
|      Return pow(value, self, mod).
|
|  __rsub__(self, value, /)
|      Return value-self.
|
|  __rtruediv__(self, value, /)
|      Return value/self.
|
```

```
 |  __sub__(self, value, /)
 |      Return self-value.
 |
 |  __truediv__(self, value, /)
 |      Return self/value.
 |
 |  __trunc__(self, /)
 |      Return the Integral closest to x between 0 and x.
 |
 |  as_integer_ratio(self, /)
 |      Return a pair of integers, whose ratio is exactly equal to the original floa
t.
 |
 |      The ratio is in lowest terms and has a positive denominator.  Raise
 |      OverflowError on infinities and a ValueError on NaNs.
 |
 |      >>> (10.0).as_integer_ratio()
 |      (10, 1)
 |      >>> (0.0).as_integer_ratio()
 |      (0, 1)
 |      >>> (-.25).as_integer_ratio()
 |      (-1, 4)
 |
 |  conjugate(self, /)
 |      Return self, the complex conjugate of any float.
 |
 |  hex(self, /)
 |      Return a hexadecimal representation of a floating-point number.
 |
 |      >>> (-0.1).hex()
 |      '-0x1.999999999999ap-4'
 |      >>> 3.14159.hex()
 |      '0x1.921f9f01b866ep+1'
 |
 |  is_integer(self, /)
 |      Return True if the float is an integer.
 |
 |  ----------------------------------------------------------------------
 |  Class methods defined here:
 |
 |  __getformat__(typestr, /)
 |      You probably don't want to use this function.
 |
 |        typestr
 |          Must be 'double' or 'float'.
 |
 |      It exists mainly to be used in Python's test suite.
 |
 |      This function returns whichever of 'unknown', 'IEEE, big-endian' or 'IEEE,
 |      little-endian' best describes the format of floating point numbers used by t
he
 |      C type named by typestr.
 |
 |  fromhex(string, /)
 |      Create a floating-point number from a hexadecimal string.
 |
```

```
|       >>> float.fromhex('0x1.ffffp10')
|       2047.984375
|       >>> float.fromhex('-0x1p-1074')
|       -5e-324
|
|  ----------------------------------------------------------------------
|  Static methods defined here:
|
|  __new__(*args, **kwargs)
|      Create and return a new object.  See help(type) for accurate signature.
|
|  ----------------------------------------------------------------------
|  Data descriptors defined here:
|
|  imag
|      the imaginary part of a complex number
|
|  real
|      the real part of a complex number
```

Lambda Function

- one-liner functions
- defined using lambda keyword
- do have a name
- function objects are generally stored in variables

In [256…
```
add = lambda x, y : x + y
```

In [257…
```
add(2, 3)
```

Out[257…
```
5
```

In [259…
```
len  # - function object
```

Out[259…
```
<function len(obj, /)>
```

In [260…
```
var = len
var("abcd")
```

Out[260…
```
4
```

Ex. Write a lambda function to check if a number is even or odd

In [261…
```
even_odd = lambda num : "even" if num % 2 == 0 else "odd"
```

In [262…
```
even_odd(5)
```

Out[262…
```
'odd'
```

Applications of Function object

In [264…
```
lst = ["flight", "car", "train", "bike"]
sorted(lst)  # sorts alphabetically
```

Out[264…
```
['bike', 'car', 'flight', 'train']
```

In [265…
```
# sort by number of characters in the list
sorted(lst, key = len)
```

Out[265…
```
['car', 'bike', 'train', 'flight']
```

In [266…
```
sorted(lst, key = lambda strg : strg[-1])
```

Out[266…
```
['bike', 'train', 'car', 'flight']
```

In [267…
```
max(lst)
```

Out[267…
```
'train'
```

In [268…
```
max(lst, key = len)
```

Out[268…
```
'flight'
```

In [269…
```
max(lst, key = min)
```

Out[269…
```
'flight'
```

In [270…
```
employess = {'Jane': 70000, 'Rosie': 90000, 'Mary': 40000, 'Sam': 55000, 'George':
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: