1. String function

1. Capitalize

In [1]:
```
# 1. Capitalize
s = 'hello python'
s = s.capitalize()
s
```

Out[1]: 'Hello python'

2. Upper Case

In [2]:
```
s = 'hello python'
s = s.upper()
print(s)
```

```
HELLO PYTHON
```

3. lower case

In [3]:
```
a = 'Hello Python'
a=a.lower()
print(a)
```

```
hello python
```

4. title()

In [4]:
```
name = 'vaibhav nehete'
name = name.title()
name
```

Out[4]: 'Vaibhav Nehete'

In [1]:
```
name = '1van bat'
name = name.title()
name

```

Out[1]: '1Van Bat'

5. casefold()

In [5]:
```
string = 'Python is ß Language # @'
string1 = string.casefold()
print(string1)

```

```
python is ss language # @
```

6. swapcase()

In [6]:
```
string = 'Python is programming Language, PYTHON is used for MAchine Learnin

string.swapcase()
```

Out[6]:
```
'pYTHON IS PROGRAMMING lANGUAGE, python IS USED FOR maCHINE lEARNING'
```

7. strip()

In [7]:
```
s = '  Data Science   '
s.strip()
```

Out[7]:
```
'Data Science'
```

8. lstrip()

In [8]:
```
# remove leading whitespace
s = '  Data Science'
s.lstrip()
```

Out[8]:
```
'Data Science'
```

9. rstrip()

In [9]:
```
# remove leading whitespace
s = 'Data Science   '
s.rstrip()
```

Out[9]:
```
'Data Science'
```

In [ ]:
```

```

In [ ]:
```

```

10. replace()

In [10]:
```
str1 = 'Hello Python and 12 data science Python'
str1.replace('Python','Java').replace('Hello','welocme to')
```

Out[10]:
```
'welocme to Java and 12 data science Java'
```

In [11]:
```
string = 'Python is Programming Language, Python is used for MAchine Learnin
# string.replace(old,new,[count])
string.replace('P','K',4)
```

Out[11]:
```
'Kython is Krogramming Language, Kython is used for MAchine Learning:'
```

11. count()

In [12]:
```
s = 'hello python language'
s.count('h')
```

Out[12]:
```
2
```

In [13]:
```
a = 'hello python'
a.count('l',0,3)
```

Out[13]:
```
1
```

12. index()

In [14]:
```
# index(substring,[start,end index])
s = 'welcome to python class'
s.index('python',1,-1)

```

Out[14]:
```
11
```

13. endswith()

In [15]:
```
# Return boolean value(True / False)

s = 'welcome to python to class.'
s.endswith('.')
```

Out[15]:
```
True
```

In [ ]:
```

```

14. startswith()

In [16]:
```
s = 'welcome to python to class.'
s.startswith('welcome')
```

Out[16]: True

15. split()

In [17]:
```
a = "We are Python learning Python Language"
a.split()
```

Out[17]: ['We', 'are', 'Python', 'learning', 'Python', 'Language']

In [18]:
```
a = "We are Python learning Python Language"
a.split('Python')
```

Out[18]: ['We are ', ' learning ', ' Language']

In [19]:
```
path = '/home/rahul/Desktop/newrepo.png'
file_name = path.split('/')[-1]
file_name
```

Out[19]: 'newrepo.png'

16. find()

In [20]:
```
txt = "Hello, welcome to my world."
x = txt.find("e")
print(x)
```

```
1
```

In [21]:
```
txt = "Hello, welcome to my world."
x = txt.find("e", 5, 10)
print(x)
```

```
8
```

In [22]:
```
txt = "Hello, welcome to my world."
print(txt.find("q"))
```

```
-1
```

In [ ]:
```

```

17. isalnum()

In [23]:
```
s = 'python'
s.isalnum()
```

Out[23]: True

In [24]:
```
s = 'python class'
s.isalnum()
```

Out[24]: False

18. isalpha()

In [25]:
```
s = 'Pyhton'
s.isalpha()
```

Out[25]: True

19. isdecimal()

In [26]:
```
s = '123345'
s.isdecimal()
```

Out[26]: True

In [27]:
```
a = '940-042-3030'
a.isdecimal()
```

Out[27]: False

In [28]:
```
num = '123.45'
num.isdecimal()
```

Out[28]: False

20. isdigit()

In [29]:
```
n = '123234'
n.isdigit()
```

Out[29]: True

In [30]:
```
n = '123.234'
n.isdigit()
```

Out[30]: False

21. islower()

In [31]:
```
b = 'data science'
b.islower()
```

Out[31]: True

22. isupper()

In [32]:
```
a = 'PYTHON-344'
a.isupper()
```

Out[32]: True

23. isspace()

In [33]:
```
s = '   '
s.isspace()
```

Out[33]: True

In [34]:
```
s1 = ' 1   '
s1.isspace()
```

Out[34]: False

24. istitle()

In [35]:
```
s = '22Python Class$$'
s.istitle()
```

Out[35]: True

In [36]:
```
s = 'Python class'
s.istitle()
```

Out[36]: False

25. isnumeric()

In [37]:
```
s = '12345'
s.isnumeric()
```

Out[37]: True

In [38]:
```
s = '123.45'
s.isnumeric()
```

Out[38]: False

26. center()

In [39]:
```
s = 'python class'
s.center(20,'X')
```

Out[39]: 'XXXXpython classXXXX'

27. join()

In [40]:
```
myTuple = ("John", "Peter", "Vicky")
x = "#".join(myTuple)
print(x)
```

```
John#Peter#Vicky
```

In [41]:
```
myDict = {"name": "John", "country": "Norway"}
mySeparator = "TEST"
x = mySeparator.join(myDict)
print(x)
```

```
nameTESTcountry
```

28.zfill()

In [42]:
```
txt = "50"

x = txt.zfill(10)

print(x)
```

```
0000000050
```

In [ ]:
```

```

29. rfind()

In [43]:
```
txt = "Mi casa, su casa."
x = txt.rfind("casa")
print(x)
```

```
12
```

30. rpartition()

In [44]:
```
txt = "I could eat bananas all day, bananas are my favorite fruit"

x = txt.rpartition("bananas")

print(x)
```

```
('I could eat bananas all day, ', 'bananas', ' are my favorite fruit')
```

2. List function

1. append()

In [1]:
```
1 fruits = ["apple", "banana", "cherry"]
2 fruits.append("orange")
3 print(fruits)
```

```
['apple', 'banana', 'cherry', 'orange']
```

In [2]:
```
1 a = ["apple", "banana", "cherry"]
2 b = ["Ford", "BMW", "Volvo"]
3 a.append(b)
4 print(a)
```

```
['apple', 'banana', 'cherry', ['Ford', 'BMW', 'Volvo']]
```

2. extend()

In [3]:
```
1 fruits = ['apple', 'banana', 'cherry']
2 cars = ['Ford', 'BMW', 'Volvo']
3 fruits.extend(cars)
4 print(fruits)
```

```
['apple', 'banana', 'cherry', 'Ford', 'BMW', 'Volvo']
```

In [4]:
```
1 fruits = ['apple', 'banana', 'cherry']
2 points = (1, 4, 5, 9)
3 fruits.extend(points)
4 print(fruits)
```

```
['apple', 'banana', 'cherry', 1, 4, 5, 9]
```

In [5]:
```
# languages list
languages = ['French']

# languages tuple
languages_tuple = ('Spanish', 'Portuguese')

# languages set
languages_set = {'Chinese', 'Japanese'}

# appending language_tuple elements to language
languages.extend(languages_tuple)

print('New Language List:', languages)

# appending language_set elements to language
languages.extend(languages_set)

print('Newer Languages List:', languages)
```

```
New Language List: ['French', 'Spanish', 'Portuguese']
Newer Languages List: ['French', 'Spanish', 'Portuguese', 'Japanese', 'Chines
e']
```

In [6]:
```
a1 = [1, 2]
a2 = [1, 2]
b = (3, 4)

# a1 = [1, 2, 3, 4]
a1.extend(b)
print(a1)

# a2 = [1, 2, (3, 4)]
a2.append(b)
print(a2)
```

```
[1, 2, 3, 4]
[1, 2, (3, 4)]
```

3. sort()

In [7]:
```
cars = ['Ford', 'BMW', 'Volvo']
cars.sort()
print(cars)
```

```
['BMW', 'Ford', 'Volvo']
```

In [8]:
```
cars = ['Ford', 'BMW', 'Volvo']
cars.sort(reverse=True)
print(cars)
```

```
['Volvo', 'Ford', 'BMW']
```

In [9]:
```
# A function that returns the length of the value:
def myFunc(e):
    return len(e)

cars = ['Ford', 'Mitsubishi', 'BMW', 'VW']

cars.sort(reverse=True, key=myFunc)
print(cars)
```

```
['Mitsubishi', 'Ford', 'BMW', 'VW']
```

In [10]:
```
# A function that returns the 'year' value:
def myFunc(e):
    return e['year']

cars = [
  {'car': 'Ford', 'year': 2005},
  {'car': 'Mitsubishi', 'year': 2000},
  {'car': 'BMW', 'year': 2019},
  {'car': 'VW', 'year': 2011}
]

cars.sort(key=myFunc)
print(cars)
```

```
[{'car': 'Mitsubishi', 'year': 2000}, {'car': 'Ford', 'year': 2005}, {'car': 'V
W', 'year': 2011}, {'car': 'BMW', 'year': 2019}]
```

4. sorted()

```
sorted() is the built in fuction
```

In [11]:
```
a = ("b", "g", "a", "d", "f", "c", "h", "e")
x = sorted(a)
print(x)
```

```
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
```

In [12]:
```
a = (1, 11, 2)
x = sorted(a)
print(x)
```

```
[1, 2, 11]
```

In [13]:
```
a = ("h", "b", "a", "c", "f", "d", "e", "g")
x = sorted(a, reverse=True)
print(x)
```

```
['h', 'g', 'f', 'e', 'd', 'c', 'b', 'a']
```

5. reverse()

In [14]:
```
fruits = ['apple', 'banana', 'cherry']
fruits.reverse()
print(fruits)
```

```
['cherry', 'banana', 'apple']
```

6. insert()

In [15]:
```
fruits = ['apple', 'banana', 'cherry']
fruits.insert(1, "orange")
print(fruits)
```

```
['apple', 'orange', 'banana', 'cherry']
```

7. pop()

In [16]:
```
fruits = ['apple', 'banana', 'cherry']
fruits.pop(1)       #index
print(fruits)
```

```
['apple', 'cherry']
```

8. remove()

In [17]:
```
fruits = ['apple', 'banana', 'cherry']
fruits.remove("banana")
print(fruits)
```

```
['apple', 'cherry']
```

9. count()

In [18]:
```
fruits = ["apple", "banana", "cherry"]
x = fruits.count("cherry")
print(x)
```

```
1
```

In [19]:
```
points = [1, 4, 2, 9, 7, 8, 9, 3, 1]
x = points.count(9)
print(x)
```

```
2
```

10. clear()

In [20]:
```
fruits = ['apple', 'banana', 'cherry', 'orange']
fruits.clear()
print(fruits)
```

```
[]
```

11. copy()

In [21]:
```
fruits = ['apple', 'banana', 'cherry', 'orange']
x = fruits.copy()
print(x)
```

...

12. index()

In [22]:
```
fruits = ['apple', 'banana', 'cherry']
x = fruits.index("cherry")
print(x)
```

```
2
```

List comprehension:-

What Is List Comprehension Python?

```
Some of the programming languages have a syntactic construct called list comprehension for creating lists on the
basis of existing lists.Python is one such language. In other words,list comprehensions are used for converting
one list into another list or creating a new list from other iterables.

A list comprehension consists of:
Input sequence
A variable to store members of input sequence
Predicate expression
Output expression that produces the output list based on the input sequence and also satisfies the predicate
```

Syntax of List comprehension Python

```
[ expression for item in list if conditional ]
```

```
# When compared to a normal Python Lists syntax, the above syntax is equivalent to

for item in list:
if conditional:
expression
```

In [2]:
```
x = [i for i in range(15)]
print (x)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
```

In [3]:
```
cubes = [x**3 for x in range(5)]
print (cubes)
```

```
[0, 1, 8, 27, 64]
```

In [4]:
```
list1 = [3,4,5]
new_multiplied_list = [item*2 for item in list1]
print (new_multiplied_list)
```

```
[6, 8, 10]
```

In [ ]:
```
listOfWords = ["this","is","python","tutorial","from","intellipaat"]
new_list = [ word[0] for word in listOfWords ]
print (new_list)
```

```
# List Comprehension Python Vs. For Loop in Python
Whenever we want to repeat a block of code for a fixed number of times, the first way of doing it is using ‘for loops’.
List comprehensions are also capable of doing the same that too with a better approach than for loops,
since list comprehensions are more compact. Hence, it provides a very good alternative to for loops.
```

In [ ]:
```
letters = []
for letter in ‘Intellipaat’:
letters.append(letter)
print(letters)
```

In [5]:
```
letters = [ letter for letter in 'Intellipaat' ]
print(letters)
```

```
['I', 'n', 't', 'e', 'l', 'l', 'i', 'p', 'a', 'a', 't']
```

```
List Comprehension Python Vs. Python Lambda Functions
In Python, we also use Lambda functions to modify and manipulate lists.
Lambda functions are also known as anonymous functions.
Lambda functions are usually used with various built-in functions such as map() filter(), and reduce() to work on lists.
```

In [ ]:
```
#1. Map() with Lambda Function
```

In [6]:
```
letters = list(map(lambda x: x, 'intellipaat'))
print(letters)
```

```
['i', 'n', 't', 'e', 'l', 'l', 'i', 'p', 'a', 'a', 't']
```

In [8]:
```
new_list=[ x for x in 'intellipaat']
print(new_list)
```

```
['i', 'n', 't', 'e', 'l', 'l', 'i', 'p', 'a', 'a', 't']
```

In [ ]:
```
# Filter() with Lambda Function
```

In [9]:
```
list1 = [1,2,3,4,5,6,7,8,9,10]
list1 = list(map(int, list1))
new_list= filter(lambda x: x%2, list1)
print(list(new_list))
```

```
[1, 3, 5, 7, 9]
```

In [10]:
```
list1 = [1,2,3,4,5,6,7,8,9,10]
print(list1)
new_list = [ x for x in list1 if x%2 ]
print(new_list)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 3, 5, 7, 9]
```

In [11]:
```
# Reduce() with Lambda Function
```

In [12]:
```
from functools import reduce
list1 = [1,2,3,4,5,6]
new_list = reduce(lambda x,y: x+y, list1)
print(new_list)
```

```
21
```

```
Note: Python list comprehensions only work with one variable, so the use of Y here is not allowed.
      Hence, to perform the above task, we can use an aggregation function, such as, sum().
```

In [13]:
```
list1 = [1,2,3,4,5,6]
new_list = sum([x for x in list1])
print(new_list)
```

```
21
```

In [ ]:
```
# Conditionals in Python List Comprehension
```

In [15]:
```
new_list = [x for x in range(20) if x%2==0]
print(new_list)
```

```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

In [17]:
```
# Using Nested IF with Python List Comprehension

new_list = [x for x in range(50) if x %2==0 if x%5==0]
print(new_list)
```

```
[0, 10, 20, 30, 40]
```

In [18]:
```
# Using if else statement with Python List comprehension

even_odd = [f"{x} is even" if x%2==0 else f"{x} is odd" for x in range (10)]
print(even_odd)
```

```
['0 is even', '1 is odd', '2 is even', '3 is odd', '4 is even', '5 is odd', '6
is even', '7 is odd', '8 is even', '9 is odd']
```

In [22]:
```
for x in range(4,7):
    for y in range(1,11):
        print(f"{x}*{y}={x*y}")
```

```
4*1=4
4*2=8
4*3=12
4*4=16
4*5=20
4*6=24
4*7=28
4*8=32
4*9=36
4*10=40
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
6*1=6
6*2=12
6*3=18
6*4=24
6*5=30
6*6=36
6*7=42
6*8=48
6*9=54
6*10=60
```

In [23]:
```
table = [[x*y for y in range(1,11)] for x in range(4,7)]
print(table)
```

```
[[4, 8, 12, 16, 20, 24, 28, 32, 36, 40], [5, 10, 15, 20, 25, 30, 35, 40, 45, 5
0], [6, 12, 18, 24, 30, 36, 42, 48, 54, 60]]
```

4. Dictionary function

1. get()

In [25]:
```
car = {"brand": "Ford","model": "Mustang","year": 1964}
x = car.get("model")
print(x)
```

```
Mustang
```

In [26]:
```
car = {"brand": "Ford","model": "Mustang","year": 1964}
x = car.get("car",117)    # Try to return the value of an item that do not e
print(x)
```

```
117
```

2. values()

In [27]:
```
car = {"brand": "Ford","model": "Mustang","year": 1964}
x = car.values()
print(x)
```

```
dict_values(['Ford', 'Mustang', 1964])
```

In [28]:
```
# When a values is changed in the dictionary, the view object also gets upda

car = {"brand": "Ford","model": "Mustang","year": 1964}
x = car.values()
car["year"] = 2018
print(x)
```

```
dict_values(['Ford', 'Mustang', 2018])
```

3. keys()

In [29]:
```
car = {"brand": "Ford","model": "Mustang","year": 1964}
x = car.keys()
print(x)
```

```
dict_keys(['brand', 'model', 'year'])
```

In [30]:
```
# When an item is added in the dictionary, the view object also gets updated
car = {"brand": "Ford","model": "Mustang","year": 1964}
x = car.keys()
car["color"] = "white"
print(x)
```

```
dict_keys(['brand', 'model', 'year', 'color'])
```

4. items()

In [31]:
```
car = {"brand": "Ford","model": "Mustang","year": 1964}
x = car.items()
print(x)
```

```
dict_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 1964)])
```

In [32]:
```
# When an item in the dictionary changes value, the view object also gets up
car = {"brand": "Ford","model": "Mustang","year": 1964}
x = car.items()
car["color"] = "white"
print(x)
```

```
dict_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 1964), ('color',
'white')])
```

5. fromkeys()

In [33]:
```
x = ('key1', 'key2', 'key3')
y = 0
thisdict = dict.fromkeys(x, y)
print(thisdict)
```

```
{'key1': 0, 'key2': 0, 'key3': 0}
```

In [34]:
```
# Same example as above, but without specifying the value:
x = ('key1', 'key2', 'key3')
thisdict = dict.fromkeys(x)
print(thisdict)
```

```
{'key1': None, 'key2': None, 'key3': None}
```

In [ ]:
```

```

In [ ]:
```

```

6. setdefault()

In [35]:
```
car = {"brand": "Ford","model": "Mustang","year": 1964}
x = car.setdefault("model", "Bronco")
print(x)
```

```
Mustang
```

In [36]:
```
# Get the value of the "color" item, if the "color" item does not exist, ins
car = {"brand": "Ford","model": "Mustang","year": 1964}
x = car.setdefault("color", "white")
print(x)
```

```
white
```

7. copy()

In [37]:
```
car = {"brand": "Ford","model": "Mustang","year": 1964}
x = car.copy()
print(x)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

8. clear()

In [38]:
```
car = {"brand": "Ford","model": "Mustang","year": 1964}
car.clear()
print(car)
```

```
{}
```

9. pop()

In [39]:
```
car = {"brand": "Ford","model": "Mustang","year": 1964}
car.pop("model")
print(car)
```

```
{'brand': 'Ford', 'year': 1964}
```

In [40]:
```
car = {"brand": "Ford","model": "Mustang","year": 1964}
x = car.pop("model")
print(x)
```

```
Mustang
```

10.popitem()

In [41]:
```
car = {"brand": "Ford","model": "Mustang","year": 1964}
car.popitem()
print(car)
```

```
{'brand': 'Ford', 'model': 'Mustang'}
```

In [42]:
```
car = {"brand": "Ford","model": "Mustang","year": 1964}
x = car.popitem()
print(x)
```

```
('year', 1964)
```

11. update()

In [43]:
```
car = {"brand": "Ford","model": "Mustang","year": 1964}
car.update({"color": "White"})
print(car)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'White'}
```

12. zip()

In [44]:
```
a = ("John", "Charles", "Mike")
b = ("Jenny", "Christy", "Monica")

x = zip(a, b)
print(tuple(x))
```

```
(('John', 'Jenny'), ('Charles', 'Christy'), ('Mike', 'Monica'))
```

In [45]:
```
a = ("John", "Charles", "Mike")
b = ("Jenny", "Christy", "Monica")

x = zip(a, b)
print(list(x))
```

```
[('John', 'Jenny'), ('Charles', 'Christy'), ('Mike', 'Monica')]
```

In [46]:
```
a = ["John", "Charles", "Mike"]
b = ["Jenny", "Christy", "Monica"]

x = zip(a, b)
print(tuple(x))
```

```
(('John', 'Jenny'), ('Charles', 'Christy'), ('Mike', 'Monica'))
```

In [47]:
```
a = ["John", "Charles", "Mike"]
b = ["Jenny", "Christy", "Monica"]

x = zip(a, b)
print(list(x))
```

```
[('John', 'Jenny'), ('Charles', 'Christy'), ('Mike', 'Monica')]
```

In [48]:
```
a = ["John", "Charles", "Mike"]
b = ["Jenny", "Christy", "Monica"]

x = zip(a, b)
print(dict(x))
```

```
{'John': 'Jenny', 'Charles': 'Christy', 'Mike': 'Monica'}
```

3. Tuple function

1. count()

In [1]:
```
thistuple = (1, 3, 7, 8, 7, 5, 4, 6, 8, 5)

x = thistuple.count(5)

print(x)
```

```
2
```

2. index()

In [2]:
```
thistuple = (1, 3, 7, 8, 7, 5, 4, 6, 8, 5)

x = thistuple.index(8)

print(x)
```

```
3
```

3. len()

In [3]:
```
tup = (1, 3, 7, 8, 7, 5, 4, 6, 8, 5)
len(tup)
```

Out[3]:
```
10
```

In [ ]:
```

```

4. min()

In [4]:
```
tup = (1, 3, 7, 8, 7, 5, 4, 6, 8, 5)
min(tup)
```

Out[4]:
```
1
```

In [ ]:
```

```

5. max()

In [5]:
```
tup = (1, 3, 7, 8, 7, 5, 4, 6, 8, 5)
max(tup)
```

Out[5]: 8

6. copy()

In [6]:
```
tup = (1, 3, 7, 8, 7, 5, 4, 6, 8, 5)
a=tup.copy()
a
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-6-df5d3c1732c1> in <module>
      1 tup = (1, 3, 7, 8, 7, 5, 4, 6, 8, 5)
----> 2 a=tup.copy()
      3 a

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

5. Set function

1. add

In [27]:
```
fruits = {"apple", "banana", "cherry"}
fruits.add("orange")
print(fruits)
```

```
{'cherry', 'orange', 'apple', 'banana'}
```

In [28]:
```
fruits = {"apple", "banana", "cherry"}
fruits.add("apple")
print(fruits)
```

```
{'cherry', 'apple', 'banana'}
```

2. update()

In [29]:
```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}
x.update(y)
print(x)
```

```
{'cherry', 'banana', 'google', 'microsoft', 'apple'}
```

3. copy()

In [30]:
```
fruits = {"apple", "banana", "cherry"}
x = fruits.copy()
print(x)
```

```
{'cherry', 'apple', 'banana'}
```

4. pop()

In [31]:
```
fruits = {"apple", "banana", "cherry"}
x = fruits.pop()
print(x)
```

```
cherry
```

In [32]:
```
1 fruits = {"apple", "banana", "cherry"}
2 fruits.pop()
3 print(fruits)
```

```
{'apple', 'banana'}
```

5. remove()

In [33]:
```
1 fruits = {"apple", "banana", "cherry"}
2 fruits.remove("banana")
3 print(fruits)
```

```
{'cherry', 'apple'}
```

6. discard()

In [34]:
```
1 fruits = {"apple", "banana", "cherry"}
2 fruits.discard("banana")
3 print(fruits)
```

```
{'cherry', 'apple'}
```

7. clear()

In [35]:
```
1 fruits = {"apple", "banana", "cherry"}
2 fruits.clear()
3 print(fruits)
```

```
set()
```

8. union()

In [36]:
```
1 x = {"apple", "banana", "cherry"}
2 y = {"google", "microsoft", "apple"}
3 z = x.union(y)
4 print(z)
```

```
{'cherry', 'banana', 'google', 'microsoft', 'apple'}
```

In [37]:
```
x = {"a", "b", "c"}
y = {"f", "d", "a"}
z = {"c", "d", "e"}
result = x.union(y, z)
print(result)
```

```
{'b', 'f', 'e', 'd', 'a', 'c'}
```

9. intersection()

In [38]:
```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}
z = x.intersection(y)
print(z)
```

```
{'apple'}
```

In [39]:
```
x = {"a", "b", "c"}
y = {"c", "d", "e"}
z = {"f", "g", "c"}
result = x.intersection(y, z)
print(result)
```

```
{'c'}
```

10. intersection_update()

In [40]:
```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}
x.intersection_update(y)
print(x)
```

```
{'apple'}
```

In [53]:
```
x = {"a", "b", "c"}
y = {"c", "d", "e"}
z = {"f", "g", "c"}
x.intersection_update(y, z)
print(x)
```

```
{'c'}
```

In [ ]:
```

```

In [ ]:
```

```

11. difference()

In [42]:
```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}
z = x.difference(y)
print(z)
```

```
{'cherry', 'banana'}
```

In [43]:
```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}
z = y.difference(x)
print(z)
```

```
{'google', 'microsoft'}
```

12. difference_update()

In [44]:
```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}
x.difference_update(y)
print(x)
```

```
{'cherry', 'banana'}
```

13. symmetric_difference()

In [45]:
```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}
z = x.symmetric_difference(y)
print(z)
```

```
{'google', 'microsoft', 'cherry', 'banana'}
```

14. symmetric_difference_update()

In [46]:
```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}
x.symmetric_difference_update(y)
print(x)
```

```
{'cherry', 'google', 'banana', 'microsoft'}
```

In [ ]:
```

```

15. isdisjoint()

In [47]:
```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "facebook"}
z = x.isdisjoint(y)
print(z)
```

```
True
```

In [48]:
```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}
z = x.isdisjoint(y)
print(z)
```

```
False
```

16. issubset()

In [49]:
```
x = {"a", "b", "c"}
y = {"f", "e", "d", "c", "b", "a"}
z = x.issubset(y)
print(z)
```

```
True
```

In [50]:
```
x = {"a", "b", "c"}
y = {"f", "e", "d", "c", "b"}
z = x.issubset(y)
print(z)
```

```
False
```

17. issuperset()

In [51]:
```
x = {"f", "e", "d", "c", "b", "a"}
y = {"a", "b", "c"}
z = x.issuperset(y)
print(z)
```

```
True
```

In [52]:
```
x = {"f", "e", "d", "c", "b"}
y = {"a", "b", "c"}
z = x.issuperset(y)
print(z)
```

```
False
```

In [1]:
```
# calculate the average of 10,20,20
a,b,c=10,20,30
avg=(a+b+c)/3
print(avg)
```

```
20.0
```

In [6]:
```
def average(n1,n2,n3):
    print('Average of 3 numbers')
    print("n1 = {}, n2 = {}, n3 = {}".format(n1,n2,n3))
    avg = (n1+n2+n3)/3
    print(avg)

n1 = 10 ; n2 = 20 ; n3 = 30
average(n1,n2,n3)
```

```
Average of 3 numbers
n1 = 10, n2 = 20, n3 = 30
20.0
```

In [8]:
```
average(12,14,12)
```

```
Average of 3 numbers
n1 = 12, n2 = 14, n3 = 12
12.666666666666666
```

In [9]:
```
def add_mul(n1,n2,n3):
    addition= n1+n2+n3
    mul = n1*n2*n3
    print(f'addition of {n1},{n2} and {n3} is {addition}')
    print(f'multiplication of {n1},{n2} and {n3} is {mul}')
n1=12
n2=13
n3=14
add_mul(n1,n2,n3)
```

```
addition of 12,13 and 14 is 39
multiplication of 12,13 and 14 is 2184
```

In [11]:
```
def add_mul(l1):
    add=0
    mul=1
    for i in l1:
        add=add+i
        mul=mul*i
    print('Addition:',add)
    print('Multiplication:',mul)

n5=[2,3,4,5,6,7]

add_mul(n5)
```

```
Addition: 27
Multiplication: 5040
```

In [13]:
```
def get_max_num(l1):
    max1 = 1
    for i in l1:
        if i > max1:
            max1 = i
    return max1

l1=[2,3,4,10,12,1]
get_max_num(l1)
```

Out[13]: 12

In [19]:
```
def get_multiplication(n1,n2,n3):
    mul = n1 * n2 * n3
    print('mul = ',mul)
    return mul


mul = get_multiplication(3,4,5)
print('m = ',mul)
```

```
mul =  60
m =  60
```

In [20]:

```
def fibonacci(n1): # n1 = 'a'
    sum1=a=0
    b=1
    list1=[]
    list1.append(a)
    list1.append(b)
    for i in range(n1-2):
        sum1=a+b
        list1.append(sum1)
        a=b
        b=sum1
    return list1 # return statement

n = int(input("enter sequesnce no: "))

fibo = fibonacci(n)

print('fibonacci series is',fibo)
```

```
enter sequesnce no: 4
fibonacci series is [0, 1, 1, 2]
```

In [ ]:

```
1
```

datetime module

In [1]:
```
import datetime
import time
```

In [2]:
```
# To know how many time taking to program to completed
```

In [3]:
```
start_time = time.time()

l1 = [3,4,5,6]
for i in l1:
    print('i == ',i)

time_complexity = time.time() - start_time

print('Time Complexity is :',time_complexity)
```

```
i ==  3
i ==  4
i ==  5
i ==  6
Time Complexity is : 0.0019898414611816406
```

Current Date and Time

In [4]:
```
current_time = datetime.datetime.now()
print(current_time)
```

```
2021-05-29 03:27:39.060065
```

In [5]:
```
print(datetime.datetime.today())
```

```
2021-05-29 03:27:39.136847
```

In [6]:
```
# to know current month
```

In [7]:
```
datetime.datetime.now().month
```

Out[7]: 5

In [8]:
```
datetime.datetime.now().year
```

Out[8]: 2021

In [9]:
```
1 datetime.datetime.now().second
```

Out[9]: 39

In [10]:
```
1 datetime.datetime.now().date()
```

Out[10]: datetime.date(2021, 5, 29)

In [11]:
```
1 date=datetime.datetime.now().date()
2 print(date)
```

```
2021-05-29
```

In [12]:
```
1 t = datetime.datetime.now().time()
2 print(t)
```

```
03:27:40.034594
```

Time Delta

In [13]:
```
1 current_time = datetime.datetime.today()
2 x = current_time + datetime.timedelta(days=5)
3 print(x)
```

```
2021-06-03 03:27:40.131230
```

In [14]:
```
1 ### WAP to print next 10 days date
2 curr_date  = datetime.datetime.now().date()
3
4 for i in range(1,11):
5     print(curr_date + datetime.timedelta(days = i))
```

```
2021-05-30
2021-05-31
2021-06-01
2021-06-02
2021-06-03
2021-06-04
2021-06-05
2021-06-06
2021-06-07
2021-06-08
```

In [15]:
```
curr_date  = datetime.datetime.now().date()

for i in range(1,11):
    print(curr_date - datetime.timedelta(days= i))
```

```
2021-05-28
2021-05-27
2021-05-26
2021-05-25
2021-05-24
2021-05-23
2021-05-22
2021-05-21
2021-05-20
2021-05-19
```

In [16]:
```
# strftime method  >> str format time
#  The strftime() method returns a string representing date and time using d
x = datetime.datetime(1993,8,28)
# x = datetime.datetime.now()
print(x)
x.strftime('%d')
x.strftime('%m')

# x.strftime('%D')
# x.strftime('%w')  # 5
# x.strftime('%a')  # Weekday Fri
# x.strftime('%A')  # Weekday Friday
# x.strftime('%b')  # Month name, Feb
# x.strftime('%B')  # Month name, February
# x.strftime('%y')   # 21
# x.strftime('%Y')   # 2021
# x.strftime('%j')   # 50
# x.strftime('%U')   #  Week Number of year
```

```
1993-08-28 00:00:00
```

Out[16]:
```
'08'
```

In [17]:

```python
from datetime import datetime

now = datetime.now() # current date and time

year = now.strftime("%Y")
print("year:", year)

month = now.strftime("%m")
print("month:", month)

day = now.strftime("%d")
print("day:", day)

time = now.strftime("%H:%M:%S")
print("time:", time)

date_time = now.strftime("%m/%d/%Y, %H:%M:%S")
print("date and time:",date_time)
```

```
year: 2021
month: 05
day: 29
time: 03:27:40
date and time: 05/29/2021, 03:27:40
```

Creating string from a timestamp

In [18]:
```
1
2 from datetime import datetime
3
4 timestamp = 1528797322
5
6
7 date_time = datetime.fromtimestamp(timestamp)
8
9 print("Date time object:", date_time)
10
11 d = date_time.strftime("%m/%d/%Y, %H:%M:%S")
12 print("Output 2:", d)
13
14 d = date_time.strftime("%d %b, %Y")
15 print("Output 3:", d)
16
17 d = date_time.strftime("%d %B, %Y")
18 print("Output 4:", d)
19
20 d = date_time.strftime("%I%p")
21 print("Output 5:", d)
```

```
Date time object: 2018-06-12 15:25:22
Output 2: 06/12/2018, 15:25:22
Output 3: 12 Jun, 2018
Output 4: 12 June, 2018
Output 5: 03PM
```

```
1 Locale's appropriate date and time
```

In [19]:
```
1 from datetime import datetime
2
3 timestamp = 1528797322
4 date_time = datetime.fromtimestamp(timestamp)
5
6 d = date_time.strftime("%c")
7 print("Output 1:", d)
8
9 d = date_time.strftime("%x")
10 print("Output 2:", d)
11
12 d = date_time.strftime("%X")
13 print("Output 3:", d)
```

```
Output 1: Tue Jun 12 15:25:22 2018
Output 2: 06/12/18
Output 3: 15:25:22
```

The strptime() method creates a datetime object from the given string.

Note: You cannot create datetime object from every string. The string needs to be in a certain format.

In [20]:
```
# Example 1: string to datetime object

from datetime import datetime

date_string = "21 June, 2018"

print("date_string =", date_string)
print("type of date_string =", type(date_string))

date_object = datetime.strptime(date_string, "%d %B, %Y")

print("date_object =", date_object)
print("type of date_object =", type(date_object))
```

```
date_string = 21 June, 2018
type of date_string = <class 'str'>
date_object = 2018-06-21 00:00:00
type of date_object = <class 'datetime.datetime'>
```

In [21]:
```

from datetime import datetime

dt_string = "12/11/2018 09:15:32"

# Considering date is in dd/mm/yyyy format
dt_object1 = datetime.strptime(dt_string, "%d/%m/%Y %H:%M:%S")
print("dt_object1 =", dt_object1)

# Considering date is in mm/dd/yyyy format
dt_object2 = datetime.strptime(dt_string, "%m/%d/%Y %H:%M:%S")
print("dt_object2 =", dt_object2)
```

```
dt_object1 = 2018-11-12 09:15:32
dt_object2 = 2018-12-11 09:15:32
```

ValueError in strptime()

In [22]:

```
# If the string (first argument) and the format code (second argument) passe
# you will get ValueError. For example:


from datetime import datetime

date_string = "12/11/2018"
date_object = datetime.strptime(date_string, "%d %m %Y")

print("date_object =", date_object)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-22-635eeccc0231> in <module>
      6
      7 date_string = "12/11/2018"
----> 8 date_object = datetime.strptime(date_string, "%d %m %Y")
      9
     10 print("date_object =", date_object)

c:\users\lenovo\appdata\local\programs\python\python38\lib\_strptime.py in _str
ptime_datetime(cls, data_string, format)
    566     """Return a class cls instance based on the input string and the
    567     format string."""
--> 568     tt, fraction, gmtoff_fraction = _strptime(data_string, format)
    569     tzname, gmtoff = tt[-2:]
    570     args = tt[:6] + (fraction,)

c:\users\lenovo\appdata\local\programs\python\python38\lib\_strptime.py in _str
ptime(data_string, format)
    347     found = format_regex.match(data_string)
    348     if not found:
--> 349         raise ValueError("time data %r does not match format %r" %
    350                          (data_string, format))
    351     if len(data_string) != found.end():

ValueError: time data '12/11/2018' does not match format '%d %m %Y'
```

```
Example 3: Current time of a timezone
If you need to find current time of a certain timezone, you can use pytZ
module.
```

In [23]:
```
from datetime import datetime
import pytz

tz_NY = pytz.timezone('America/New_York')
datetime_NY = datetime.now(tz_NY)
print("NY time:", datetime_NY.strftime("%H:%M:%S"))

tz_London = pytz.timezone('Europe/London')
datetime_London = datetime.now(tz_London)
print("London time:", datetime_London.strftime("%H:%M:%S"))
```

```
NY time: 17:58:21
London time: 22:58:21
```

In [24]:
```
# Python timestamp to datetime
from datetime import datetime

timestamp = 1545730073
dt_object = datetime.fromtimestamp(timestamp)

print("dt_object =", dt_object)
print("type(dt_object) =", type(dt_object))
```

```
dt_object = 2018-12-25 14:57:53
type(dt_object) = <class 'datetime.datetime'>
```

In [25]:
```
#  Python datetime to timestamp
# You can get timestamp from a datetime object using datetime.timestamp() me

from datetime import datetime

# current date and time
now = datetime.now()

timestamp = datetime.timestamp(now)
print("timestamp =", timestamp)
```

```
timestamp = 1622239112.861813
```

In [30]:
```
# 2015-1-21
input_date = '21 Jan 2015'

date = datetime.strptime(input_date,'%d %b %Y')
print(date.date())
```

```
2015-01-21
```

In [31]:
```
input_date = '23-Apr-2020'

date = datetime.strptime(input_date,'%d-%b-%Y')
print(date.date())
```

```
2020-04-23
```

In [29]:
```
input_date = '23/04/2020'

date = datetime.datetime.strptime(input_date,'%d/%m/%Y')
print(date.date())
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-29-b2fe05fc1919> in <module>
      1 input_date = '23/04/2020'
      2
----> 3 date = datetime.datetime.strptime(input_date,'%d/%m/%Y')
      4 print(date.date())

AttributeError: type object 'datetime.datetime' has no attribute 'datetime'
```

In [32]:
```
input_date = '04/4/2020'

date = datetime.strptime(input_date,'%d/%m/%Y')  # 2020-04-23
print(date.date())
```

```
2020-04-04
```

In [33]:
```
date_list = ['19/02/2021',
'19 Feb 2021',
'19-Feb-2021',
'19-Feb-21',
'19-02-2021',
'2021-02-19']

def get_date(input_date):

    date_format = ['%d/%m/%Y','%d-%b-%Y','%d-%b-%y','%d %b %Y','%Y-%m-%d','%

    for f in  date_format:
        try:
            return datetime.datetime.strptime(input_date,f).date()
        except:
            continue

for date in date_list:
    print(date ,'>>>', get_date(date))
```

```
19/02/2021 >>> None
19 Feb 2021 >>> None
19-Feb-2021 >>> None
19-Feb-21 >>> None
19-02-2021 >>> None
2021-02-19 >>> None
```

File Handling

Read Write Append Open

Access Mode 'r' >> File read 'w' >> Write 'a' >> Append

1. Write

In [ ]:
```
# If file exists, Overwrite it
# If file does not exists, Then create new file

# open(file_name, access_mode)

fileptr = open('myfile.txt','w')

fileptr.write('Pyhton is programming')

fileptr.close()
```

In [ ]:
```
fileptr = open('myfile3.json','w')

text = '''{'name' :'Python'} '''
fileptr.write(text)

# fileptr.write('')

fileptr.close()
```

In [ ]:
```
fileptr = open('myfile2.txt','w')

text = '''Pyhton is programming language'''
fileptr.write(text)

fileptr.close()
```

2.Append

In [ ]:
```
# Create a file if not exist
#

```

In [ ]:
```
f = open('file.txt','a')

f.write('\n It is a branch of artificial intelligence based on the idea that

f.close()
```

In [ ]:
```
f = open('file.txt','w')

f.write('\nPyhton is programming Language')

f.close()
```

X mode

In [ ]:
```

# If file exists, gives an error
# If file does not exists create new file

f = open('file.txt','x')

f.write('\nPyhton is programming Language')

f.close()
```

3. Read()

In [ ]:
```
fileptr = open('myfile.txt','w')
text = '''Machine learning is a method of data analysis that automates analy
It is a branch of artificial intelligence based on the idea that systems can
identify patterns and make decisions with minimal human intervention.'''

fileptr.write(text)

fileptr.close()
```

In [ ]:
```
f = open('myfile.txt','r')

text = f.read(10)
print(text)

f.close()

```

```
Machine le
```

3.1 readline()

In [ ]:
```
f = open('myfile.txt','r')

text = f.readline()

print(text)
print(f.readline())
f.close()
```

```
Machine learning is a method of data analysis that automates analytical model b
uilding.

It is a branch of artificial intelligence based on the idea that systems can le
arn from data,
```

3.2 readlines()

In [ ]:
```
# Return the list of lines

f = open('myfile.txt','r')

text = f.readlines()
print(text,type(text))
f.close()
```

```
['Machine learning is a method of data analysis that automates analytical model
building.\n', 'It is a branch of artificial intelligence based on the idea that
systems can learn from data, \n', 'identify patterns and make decisions with mi
nimal human intervention.'] <class 'list'>
```

Text Classification image to text using pytesseract

In [ ]:
```
import pytesseract

text = pytesseract.image_to_string('/home/rahul/Desktop/Validation UI/Salary
print(text)

fileptr = open('myfile.txt','w')

fileptr.write(text)

fileptr.close()
```

```
AXIS BANK
he

PARNAVI ELECTRONICS

Joint Holder :- -

SHOP NO 01 BABA MARKET HAROLA

SECTOR 05

NOIDA Customer No :875243714
UTTAR PRADESH Scheme :CURRENT ACCOUNT-NORMAL
201301 Currency :INR

Statement of Account No :917020049438221 for the period (From : 01-08-2017 To
: 16-06-2018)
```

In [ ]:
```
Open Source OCR  : pytesseract

Paid OCR : Vision OCR, AWS Textract

```

In [ ]:
```

```

os module

In [1]:
```
import os
```

In [2]:
```
# get current working directory
```

In [3]:
```
os.getcwd()
```

Out[3]:
```
'C:\\Users\\LENOVO\\Desktop\\os module'
```

In [4]:
```
# change current working directory
```

In [5]:
```
os.chdir('C://')
os.getcwd()
```

Out[5]:
```
'C:\\'
```

In [6]:
```
os.chdir('C:\\Users\\LENOVO\\Desktop\\os module')
```

In [7]:
```
os.getcwd()
```

Out[7]:
```
'C:\\Users\\LENOVO\\Desktop\\os module'
```

In [8]:
```
# create a folder
```

In [9]:
```
import os
```

In [10]:
```
os.mkdir('van')
print('folder created successfully')
```

```
folder created successfully
```

In [11]:
```
# os.mkdir('van')
```

In [12]:
```
# create folder in folder(create subdirectory in folder)
```

In [13]:
```
os.mkdir('van/data')
print('data folder created in van folder')
```

```
data folder created in van folder
```

In [14]:
```
# to create multiple directory
```

In [15]:
```
os.makedirs('sat/cat/bat')
print('folder created successfully')
```

```
folder created successfully
```

In [16]:
```
# remove folder(directory)
```

In [17]:
```
os.rmdir('van/data')
print('only remove data folder')
```

```
only remove data folder
```

In [18]:
```
os.removedirs('sat/cat/bat')
print('all folder are removed')
```

```
all folder are removed
```

In [19]:
```
# to rename directory
```

In [20]:
```
os.rename('van','fan')
print('rename successfully')
```

```
rename successfully
```

In [21]:
```
# To know contents of directory
```

In [22]:
```
os.listdir('.')       # '.' means a current working directory
```

Out[22]:
```
['.ipynb_checkpoints', 'fan', 'os_moudule.ipynb']
```

In [23]:
```
os.listdir('fan')
```

Out[23]:
```
[]
```

In [24]:
```
# to know contents of directori including sub directories
# os.walk(path,topdown=True,oneerror=None,followlinks=false)
```

In [25]:
```
for dirpath,dirname,filenames in os.walk('.'):
    print('current working directory',dirpath)
    print()
    print('directories',dirname)
    print()
    print('files',filenames)
```

```
current working directory .

directories ['.ipynb_checkpoints', 'fan']

files ['os_moudule.ipynb']
current working directory .\.ipynb_checkpoints

directories []

files ['os_moudule-checkpoint.ipynb']
current working directory .\fan

directories []

files []
```

In [26]:
```
# create file
```

In [27]:
```
f=open('python.txt','w')    # file created in cwd
f
```

Out[27]:
```
<_io.TextIOWrapper name='python.txt' mode='w' encoding='cp1252'>
```

```
create a folder if not exist in cwd directory
in that folder creates files
and rename that files
```

In [28]:

```
import os
import shutil
if not os.path.exists('van'):
    os.mkdir('van')
# file = os.listdir('van')
create_files = open('Pract.json','a')
create_files.write("---------Welcome--------")
create_files.close()
create_files1 = open('pract.py','a')
create_files1.write("---------Welcome to python--------")
create_files1.close()

shutil.move('pract.json','van')
shutil.move('pract.py','van')
allfiles=os.listdir('van')
print(allfiles)

allfiles=os.listdir('van')

for i,file in enumerate (allfiles):
    os.rename(os.getcwd() + '\\van\\' + file,os.getcwd() + '\\van\\' + f'pra
    print(i,file)
print(os.listdir('van'))
```

```
['pract.json', 'pract.py']
0 pract.json
1 pract.py
['pract1.txt', 'pract2.txt']
```

Regular Expression

```
A Regular Expression (RE) in a programming language is a special text string used for describing a search pattern.
It is extremely useful for extracting information from text such as code, files, log, spreadsheets or even documents.
While using the Python regular expression the first thing is to recognize is that everything is essentially a character,
and we are writing patterns to match a specific sequence of characters also referred as string.
Ascii or latin letters are those that are on your keyboards and Unicode is used to match the foreign text.
It includes digits and punctuation and all special characters like $#@!%, etc.
```

In [1]:
```
import re
```

In [2]:
```
#Check if the string starts with "The" and ends with "Spain":

txt = "The rain in Spain"
x = re.search("^The.*Spain$", txt)

if x:
  print("YES! We have a match!")
else:
  print("No match")
```

```
YES! We have a match!
```

findall() Function

```
The findall() function returns a list containing all matches.
```

In [3]:
```
txt = "The rain in Spain"
x = re.findall("ai", txt)
print(x)
```

```
['ai', 'ai']
```

```
The list contains the matches in the order they are found.
If no matches are found, an empty list is returned:
```

In [4]:
```
txt = "The rain in Spain"
x = re.findall("Portugal", txt)
print(x)
```

```
[]
```

search() function

```
The search() function searches the string for a match, and returns a Match object if there is a match.
If there is more than one match, only the first occurrence of the match will be returned:
```

In [5]:
```
txt = "The rain in Spain"
x = re.search("\s", txt)
print("The first white-space character is located in position:", x.start())
```

```
The first white-space character is located in position: 3
```

In [6]:
```
# If no matches are found, the value None is returned:

txt = "The rain in Spain"
x = re.search("Portugal", txt)
print(x)
```

```
None
```

The split() Function

```
The split() function returns a list where the string has been split at each match:
```

In [7]:
```
# Split at each white-space character:

txt = "The rain in Spain"
x = re.split("\s", txt)
print(x)
```

```
['The', 'rain', 'in', 'Spain']
```

```
You can control the number of occurrences by specifying the maxsplit parameter:
```

In [8]:
```
# Split the string only at the first occurrence:

txt = "The rain in Spain"
x = re.split("\s", txt, 1)
print(x)
```

```
['The', 'rain in Spain']
```

The sub() Function

```
The sub() function replaces the matches with the text of your choice:
```

In [9]:
```
# Replace every white-space character with the number 9:

txt = "The rain in Spain"
x = re.sub("\s", "9", txt)
print(x)
```

```
The9rain9in9Spain
```

In [10]:
```
# You can control the number of replacements by specifying the count paramet
# Replace the first 2 occurrences:

txt = "The rain in Spain"
x = re.sub("\s", "9", txt, 2)
print(x)
```

```
The9rain9in Spain
```

Match Object

```
A Match Object is an object containing information about the search and
the result.
Note: If there is no match, the value None will be returned, instead of
the Match Object.
```

In [11]:
```
txt = "The rain in Spain"
x = re.search("ai", txt)
print(x) #this will print an object
```

```
<re.Match object; span=(5, 7), match='ai'>
```

```
The Match object has properties and methods used to retrieve information
about the search, and the result:

.span() returns a tuple containing the start-, and end positions of the
match.
```

```
.string returns the string passed into the function
.group() returns the part of the string where there was a match
```

In [12]:
```
# Print the position (start- and end-position) of the first match occurrence
# The regular expression looks for any words that starts with an upper case

txt = "The rain in Spain"
x = re.search(r"\bS\w+", txt)
print(x.span())
```

```
(12, 17)
```

In [13]:
```
txt = "The rain in Spain"
x = re.search(r"\bS\w+", txt)
print(x.string)
```

```
The rain in Spain
```

In [14]:
```
# Print the part of the string where there was a match.
# The regular expression looks for any words that starts with an upper case
txt = "The rain in Spain"
x = re.search(r"\bS\w+", txt)
print(x.group())
```

```
Spain
```

```
Note: If there is no match, the value None will be returned, instead of
the Match Object.
```

In [ ]:
```

```

In [15]:
```
data = """Data science is a "concept to unify statistics, data analysis,
informatics, and their related methods" in order to "understand and analyze
actual phenomena" with data.[3] It uses techniques and theories drawn from m
fields within the context of mathematics, statistics, computer science, info
science, and domain knowledge. Turing award winner Jim Gray imagined data sc
a "fourth paradigm" of science (empirical, theoretical, computational and no
and asserted that "everything about science is changing because of the impac
information technology" and the data deluge"""
```

In [16]:
```
text=re.findall(r'\b[aeiou]\w+',data)
print(text,len(text))
```

```
['is', 'unify', 'analysis', 'informatics', 'and', 'in', 'order', 'understand',
'and', 'analyze', 'actual', 'uses', 'and', 'of', 'information', 'and', 'award',
'imagined', 'as', 'of', 'empirical', 'and', 'and', 'asserted', 'everything', 'a
bout', 'is', 'of', 'impact', 'of', 'information', 'and'] 32
```

In [17]:
```
class id_details:
    def __init__(self,d):
        print("######  Details of the User Are  #######\n")
        self.details = d

    def get_PAN(self):
        PAN_number = re.findall('[A-Z]{5}[\d]{4}[A-Z]{1}',self.details)
        print("PAN Number is :",PAN_number)

    def get_Adhar(self):
        Adhar_number = re.findall('\d{12}',self.details)
        print("Adhar Number is :",Adhar_number)

    def get_DL(self):
        DL_number = re.findall('[A-Z]{2}[\d]{2} [\d]{11}',self.details)
        print("Driving License Number is :",DL_number)

    def get_Passportno(self):
        Passport_number = re.findall('[A-Z]{1} [A-Z]{3} [A-Z]{1}[\d]{7}',sel
        print("Passport Number is :",Passport_number)

    def get_email(self):
        Email_id =  re.findall('[\w._]{2,30}@[a-z]{4,10}[.][a-z]{2,4}',self.
        print("Email Id is :",Email_id)

```

In [18]:
```
d = """Hello My name is Nikesh Agrawal.
My Id details are
Adhar Number - 8112 6645 8731,
Pan Number - ATNPA7920B
Driving License Number - MH35 20150054874
Passport Number -  P IND J8369854
Email Id - agrawal.nikesh60@gmail.com
Mobile Number - 9403132036,8668369580"""

obj = id_details(d)
obj.get_PAN()
obj.get_Adhar()
obj.get_DL()
obj.get_Passportno()
obj.get_email()
# obj.get_mobile

```

```
######  Details of the User Are  #######

PAN Number is : ['ATNPA7920B']
Adhar Number is : []
Driving License Number is : ['MH35 20150054874']
Passport Number is : ['P IND J8369854']
Email Id is : ['agrawal.nikesh60@gmail.com']
```

```
import  pytesseract
# from PIL import Image
```

```
pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-OCR\tesseract.exe'
text = pytesseract.image_to_string(r'D:\RahulK\Velocity\pancard1.jpg')
print(text)
```

In [19]:

```
q=re.sub('\n+',"\n",data)
print(q)
```

```
Data science is a "concept to unify statistics, data analysis,
informatics, and their related methods" in order to "understand and analyze
actual phenomena" with data.[3] It uses techniques and theories drawn from ma
ny
fields within the context of mathematics, statistics, computer science, infor
mation
science, and domain knowledge. Turing award winner Jim Gray imagined data sci
ence as
a "fourth paradigm" of science (empirical, theoretical, computational and now
data-driven)
and asserted that "everything about science is changing because of the impact
of
information technology" and the data deluge
```

In [20]:

```
again = re.findall('\w{1,100}\s',data)
again2 = ' '.join(again)
print(again2)
```

```
Data science is a concept to unify data and their related in order
to understand and analyze actual with It uses techniques and theories
drawn from many fields within the context of computer information and
domain Turing award winner Jim Gray imagined data science as a fourt
h of science computational and now and asserted that everything about
science is changing because of the impact of information and the dat
a
```

In [21]:
```
text_list = data.split('\n')
text = [line for line in text_list if re.sub('[^a-zA-Z0-9/]+',' ',line.strip
text = '\n'.join(text)
print(text)
```

```
Data science is a "concept to unify statistics, data analysis,
informatics, and their related methods" in order to "understand and analyze
actual phenomena" with data.[3] It uses techniques and theories drawn from many
fields within the context of mathematics, statistics, computer science, informa
tion
science, and domain knowledge. Turing award winner Jim Gray imagined data scien
ce as
a "fourth paradigm" of science (empirical, theoretical, computational and now d
ata-driven)
and asserted that "everything about science is changing because of the impact o
f
information technology" and the data deluge
```

```
import pandas as pd

def get_pan_attributes(text):
    new_text = text.split('\n')
    person_name = 'NA';father_name ='NA';PAN_number = 'NA';dob ='NA'

    for index,line in enumerate(new_text):
        if re.findall('INCOME|TAX|DEPARTMENT',line): # or
#             print(index,line)
            person_name = new_text[index + 1]
#             print('person_name',person_name)
            father_name = new_text[index + 2]
#             print('father_name',father_name)

    PAN_number = re.findall('[A-Z]{5}[\d]{4}[A-Z]{1}',text)
    dob = re.findall('\d{2}[/]\d{2}[/]\d{4}',text)
    pan_attributes = {'Person Name':[person_name],'Father Name':
[father_name],
                      'PAN Number':PAN_number,'Date of Birth':dob}

    return pan_attributes

pan_attributes = get_pan_attributes(text)
print(pan_attributes)
df = pd.DataFrame(pan_attributes)
df.to_csv('pan.csv')
```

```
 pdftotext
# pdfreader
# pdfminer
# tabula-py
# camelot
# fuzzywuzzy

pdf >> image >> ocr (non readable pdf)
```

```
9 pdf >> ocr (readable pdf)
```

In [22]:
```
1 output = re.sub("[^A-Za-z0-9,._]\n+","\n",data)
2 print(output)
```

```
Data science is a "concept to unify statistics, data analysis,
informatics, and their related methods" in order to "understand and analyze
actual phenomena" with data.[3] It uses techniques and theories drawn from many
fields within the context of mathematics, statistics, computer science, informa
tion
science, and domain knowledge. Turing award winner Jim Gray imagined data scien
ce as
a "fourth paradigm" of science (empirical, theoretical, computational and now d
ata-driven)
and asserted that "everything about science is changing because of the impact o
f
information technology" and the data deluge
```

In [23]:
```
1 # . >> any char
2 string = 'python'
3 data = re.findall('py...',string)
4 data
```

Out[23]:
```
['pytho']
```

In [24]:
```
1 string = 'pyttttttthon' # zero  or more
2 data = re.findall('pyt*',string)
3 data
```

Out[24]:
```
['pyttttttt']
```

In [25]:
```
1 string = 'pyttttttthon'  # one or more
2 data = re.findall('pyt+',string)
3 data
```

Out[25]:
```
['pyttttttt']
```

In [26]:
```
1 string = 'pyttttttthon'  # Exactly One char
2 data = re.findall('pyt?',string)
3 data
```

Out[26]:
```
['pyt']
```

In [27]:
```
1 string = 'pythonr'  # Exactly One char for zero or one occurrence
2 data = re.findall('honn?',string)
3 data
```

Out[27]:
```
['hon']
```

In [28]:
```
string = 'python class'
data = re.findall('^p',string) # Start with
data
```

Out[28]:
```
['p']
```

In [29]:
```
string = 'python class'
data = re.findall('s$',string) # ends with
data
```

Out[29]:
```
['s']
```

In [30]:
```
string = 'python class '
data = re.findall('^p.*',string)
data
```

Out[30]:
```
['python class ']
```

In [31]:
```
string = 'python class pune'
data = re.findall('^p.*s',string)
data
```

Out[31]:
```
['python class']
```

In [32]:
```
string = 'python class pune'
text = re.findall('python|class|pune',string)
text
```

Out[32]:
```
['python', 'class', 'pune']
```

In [33]:
```
import re

pattern = '^a...s$'
test_string = 'abyss'
result = re.match(pattern, test_string)

if result:
  print("Search successful.")
else:
  print("Search unsuccessful.")

```

```
Search successful.
```

In [34]:
```
# Program to extract numbers from a string

string = 'hello 12 hi 89. Howdy 34'
pattern = '\d+'

result = re.findall(pattern, string)
print(result)

# Output: ['12', '89', '34']
```

```
['12', '89', '34']
```

In [35]:
```
string = 'Twelve:12 Eighty nine:89.'
pattern = '\d+'

result = re.split(pattern, string)
print(result)
```

```
['Twelve:', ' Eighty nine:', '.']
```

In [36]:
```
# Program to remove all whitespaces

# multiline string
string = 'abc 12\
de 23 \n f45 6'

# matches all whitespace characters
pattern = '\s+'

# empty string
replace = ''

new_string = re.sub(pattern, replace, string)
print(new_string)
```

```
abc12de23f456
```

In [37]:
```
# Program to remove all whitespaces

# multiline string
string = 'abc 12\
de 23 \n f45 6'

# matches all whitespace characters
pattern = '\s+'

# empty string
replace = ''

new_string = re.subn(pattern, replace, string)
print(new_string)

```

```
('abc12de23f456', 4)
```

In [38]:
```
string = "Python is fun"

# check if 'Python' is at the beginning
match = re.search('\APython', string)

if match:
  print("pattern found inside the string")
else:
  print("pattern not found")

```

```
pattern found inside the string
```

In [39]:
```
string = '39801 356, 2102 1111'

# Three digit number followed by space followed by two digit number
pattern = '(\d{3}) (\d{2})'

# match variable contains a Match object.
match = re.search(pattern, string)

if match:
  print(match.group())
else:
  print("pattern not found")
```

```
801 35
```

In [40]:
```
string = '\n and \r are escape sequences.'

result = re.findall(r'[\n\r]', string)
print(result)
```

```
['\n', '\r']
```

Time module

```
Python time.time()
The time() function returns the number of seconds passed since epoch.
For Unix system, January 1, 1970, 00:00:00 at UTC is epoch (the point
where time begins).
```

In [1]:
```
import time
seconds = time.time()
print("Seconds since epoch =", seconds)
```

```
Seconds since epoch = 1622240391.013273
```

```
Python time.ctime()
The time.ctime() function takes seconds passed since epoch as an argument
and returns a string representing local time.
```

In [2]:
```
import time

# seconds passed since epoch
seconds = 1545925769.9618232
local_time = time.ctime(seconds)
print("Local time:", local_time)
```

```
Local time: Thu Dec 27 21:19:29 2018
```

```
Python time.sleep()
The sleep() function suspends (delays) execution of the current thread for
the given number of seconds.
```

In [3]:
```
import time

print("This is printed immediately.")
time.sleep(2.4)
print("This is printed after 2.4 seconds.")
```

```
This is printed immediately.
This is printed after 2.4 seconds.
```

```
time.struct_time Class
Several functions in the time module such as gmtime(), asctime() etc.
either take time.struct_time object as an argument or return it.

Here's an example of time.struct_time object.

time.struct_time(tm_year=2018, tm_mon=12, tm_mday=27,
                 tm_hour=6, tm_min=35, tm_sec=17,
                 tm_wday=3, tm_yday=361, tm_isdst=0)

```

```
Index     Attribute        Values
0          tm_year     0000, ...., 2018, ..., 9999
1          tm_mon      1, 2, ..., 12
2          tm_mday     1, 2, ..., 31
3          tm_hour     0, 1, ..., 23
4          tm_min      0, 1, ..., 59
5          tm_sec      0, 1, ..., 61
6          tm_wday     0, 1, ..., 6; Monday is 0
7          tm_yday     1, 2, ..., 366
8          tm_isdst    0, 1 or -1

The values (elements) of the time.struct_time object are accessible using both indices and attributes.


```

```
Python time.localtime()
The localtime() function takes the number of seconds passed since epoch as an argument and
returns struct_time in local time.

```

In [4]:

```
import time

result = time.localtime(1545925769)
print("result:", result)
print("\nyear:", result.tm_year)
print("tm_hour:", result.tm_hour)
```

```
result: time.struct_time(tm_year=2018, tm_mon=12, tm_mday=27, tm_hour=21, tm_mi
n=19, tm_sec=29, tm_wday=3, tm_yday=361, tm_isdst=0)

year: 2018
tm_hour: 21
```

```
Python time.gmtime()

The gmtime() function takes the number of seconds passed since epoch as an argument and returns struct_time in UTC.
```

In [5]:

```
import time

result = time.gmtime(1545925769)
print("result:", result)
print("\nyear:", result.tm_year)
print("tm_hour:", result.tm_hour)
```

```
result: time.struct_time(tm_year=2018, tm_mon=12, tm_mday=27, tm_hour=15, tm_mi
n=49, tm_sec=29, tm_wday=3, tm_yday=361, tm_isdst=0)

year: 2018
tm_hour: 15
```

```
Python time.mktime()
The mktime() function takes struct_time (or a tuple containing 9 elements corresponding to struct_time)
as an argument and returns the seconds passed since epoch in local time.
Basically, it's the inverse function of localtime().
```

In [6]:
```
import time

t = (2018, 12, 28, 8, 44, 4, 4, 362, 0)

local_time = time.mktime(t)
print("Local time:", local_time)
```

```
Local time: 1545966844.0
```

```
Python time.asctime()
The asctime() function takes struct_time (or a tuple containing 9 elements corresponding to struct_time)
as an argument and returns a string representing it. Here's an example:
```

In [7]:
```
import time

t = (2018, 12, 28, 8, 44, 4, 4, 362, 0)

result = time.asctime(t)
print("Result:", result)
```

```
Result: Fri Dec 28 08:44:04 2018
```

```
Python time.strftime()
The strftime() function takes struct_time (or tuple corresponding to it) as an argument and
returns a string representing it based on the format code used. For example,

```

In [8]:
```
import time

named_tuple = time.localtime() # get struct_time
time_string = time.strftime("%m/%d/%Y, %H:%M:%S", named_tuple)

print(time_string)
```

```
05/29/2021, 03:49:54
```

```
Python time.strptime()
The strptime() function parses a string representing time and returns struct_time.
```

In [9]:

```
1 import time
2
3 time_string = "21 June, 2018"
4 result = time.strptime(time_string, "%d %B, %Y")
5
6 print(result)
```

```
time.struct_time(tm_year=2018, tm_mon=6, tm_mday=21, tm_hour=0, tm_min=0, tm_se
c=0, tm_wday=3, tm_yday=172, tm_isdst=-1)
```

web scrapping

```
# It is the process to extract meaningful data/content from websites
```

```
1.Websites with html pages (urllib)

2.Web Scrapping Technology (Regex,bs4)

3.CSV,Excel,MongoDB,MySQL
```

In [1]:
```
from urllib.request import urlopen
import re
import pandas as pd
url = 'https://www.summet.com/dmsi/html/codesamples/addresses.html'

# data = urllib.request.urlopen(url)
data = urlopen(url)
print(data)
```

```
<http.client.HTTPResponse object at 0x000002BD53DC1A90>
```

In [2]:
```
html = data.read()
# print(html)
html_str = html.decode()
# print(html_str)
```

In [3]:
```
text = re.sub('<.*?>',' ',html_str)  # Non Greedy Repetition
print(html_str)
```

```
<html>
<head><title>Sample Addresses!</title></head>
<body>
<h1> A page full of sample addresses for your parsing enjoyment!</h1>
<h2> (All data is random....)</h2>
<ul>
<li>Cecilia Chapman<br/>711-2880 Nulla St.<br/>Mankato Mississippi 96522<br/>(2
57) 563-7401</li>
<li>Iris Watson<br/>P.O. Box 283 8562 Fusce Rd.<br/>Frederick Nebraska 20620<b
r/>(372) 587-2335</li>
<li>Celeste Slater<br/>606-3727 Ullamcorper. Street<br/>Roseville NH 11523<br/>
(786) 713-8616</li>
<li>Theodore Lowe<br/>Ap #867-859 Sit Rd.<br/>Azusa New York 39531<br/>(793) 15
1-6230</li>
<li>Calista Wise<br/>7292 Dictum Av.<br/>San Antonio MI 47096<br/>(492) 709-639
2</li>
<li>Kyla Olsen<br/>Ap #651-8679 Sodales Av.<br/>Tamuning PA 10855<br/>(654) 393
-5734</li>
<li>Forrest Ray<br/>191-103 Integer Rd.<br/>Corona New Mexico 08219<br/>(404) 9
60-3807</li>
<li>Hiroko Potter<br/>P.O. Box 887 2508 Dolor. Av.<br/>Muskegon KY 12482<br/>(3
14) 244-6306</li>
<li>Nyssa Vazquez<br/>511-5762 At Rd.<br/>Chelsea MI 67708<br/>(947) 278-5929</
li>
<li>Lawrence Moreno<br/>935-9940 Tortor. Street<br/>Santa Rosa MN 98804<br/>(68
4) 579-1879</li>
<li>Ina Moran<br/>P.O. Box 929 4189 Nunc Road<br/>Lebanon KY 69409<br/>(389) 73
7-2852</li>
<li>Aaron Hawkins<br/>5587 Nunc. Avenue<br/>Erie Rhode Island 24975<br/>(660) 6
63-4518</li>
<li>Hedy Greene<br/>Ap #696-3279 Viverra. Avenue<br/>Latrobe DE 38100<br/>(608)
265-2215</li>
<li>Melvin Porter<br/>P.O. Box 132 1599 Curabitur Rd.<br/>Bandera South Dakota
45149<br/>(959) 119-8364</li>
<li>Keefe Sellers<br/>347-7666 Iaculis St.<br/>Woodruff SC 49854<br/>(468) 353-
2641</li>
<li>Joan Romero<br/>666-4366 Lacinia Avenue<br/>Idaho Falls Ohio 19253<br/>(24
8) 675-4007</li>
<li>Davis Patrick<br/>P.O. Box 147 2546 Sociosqu Rd.<br/>Bethlehem Utah 02913<b
r/>(939) 353-1107</li>
<li>Leilani Boyer<br/>557-6308 Lacinia Road<br/>San Bernardino ND 09289<br/>(57
0) 873-7090</li>
<li>Colby Bernard<br/>Ap #285-7193 Ullamcorper Avenue<br/>Amesbury HI 93373<br/
>(302) 259-2375</li>
<li>Bryar Pitts<br/>5543 Aliquet St.<br/>Fort Dodge GA 20783<br/>(717) 450-4729
</li>
<li>Rahim Henderson<br/>5037 Diam Rd.<br/>Daly City Ohio 90255<br/>(453) 391-46
50</li>
<li>Noelle Adams<br/>6351 Fringilla Avenue<br/>Gardena Colorado 37547<br/>(559)
104-5475</li>
<li>Lillith Daniel<br/>935-1670 Neque. St.<br/>Centennial Delaware 48432<br/>(3
87) 142-9434</li>
<li>Adria Russell<br/>414-7533 Non Rd.<br/>Miami Beach North Dakota 58563<br/>
```

```
(516) 745-4496</li>
<li>Hilda Haynes<br/>778-9383 Suspendisse Av.<br/>Weirton IN 93479<br/>(326) 67
7-3419</li>
<li>Sheila Mcintosh<br/>P.O. Box 360 4407 Et Rd.<br/>Santa Monica FL 30309<br/>
(746) 679-2470</li>
<li>Rebecca Chambers<br/>P.O. Box 813 5982 Sit Ave<br/>Liberal Vermont 51324<b
r/>(455) 430-0989</li>
<li>Christian Emerson<br/>P.O. Box 886 4118 Arcu St.<br/>Rolling Hills Georgia
92358<br/>(490) 936-4694</li>
<li>Nevada Ware<br/>P.O. Box 597 4156 Tincidunt Ave<br/>Green Bay Indiana 19759
<br/>(985) 834-8285</li>
<li>Margaret Joseph<br/>P.O. Box 508 3919 Gravida St.<br/>Tamuning Washington 5
5797<br/>(662) 661-1446</li>
<li>Edward Nieves<br/>928-3313 Vel Av.<br/>Idaho Falls Rhode Island 37232<br/>
(802) 668-8240</li>
<li>Imani Talley<br/>P.O. Box 262 4978 Sit St.<br/>Yigo Massachusetts 50654<br/
>(477) 768-9247</li>
<li>Bertha Riggs<br/>P.O. Box 206 6639 In St.<br/>Easthampton TN 31626<br/>(79
1) 239-9057</li>
<li>Wallace Ross<br/>313 Pellentesque Ave<br/>Villa Park Hawaii 43526<br/>(832)
109-0213</li>
<li>Chester Bennett<br/>3476 Aliquet. Ave<br/>Minot AZ 95302<br/>(837) 196-3274
</li>
<li>Castor Richardson<br/>P.O. Box 902 3472 Ullamcorper Street<br/>Lynchburg DC
29738<br/>(268) 442-2428</li>
<li>Sonya Jordan<br/>Ap #443-336 Ullamcorper. Street<br/>Visalia VA 54886<br/>
(850) 676-5117</li>
<li>Harrison Mcguire<br/>574-8633 Arcu Street<br/>San Fernando ID 77373<br/>(86
1) 546-5032</li>
<li>Malcolm Long<br/>9291 Proin Road<br/>Lake Charles Maine 11292<br/>(176) 805
-4108</li>
<li>Raymond Levy<br/>Ap #643-7006 Risus St.<br/>Beaumont New Mexico 73585<br/>
(715) 912-6931</li>
<li>Hedley Ingram<br/>737-2580 At Street<br/>Independence Texas 87535<br/>(993)
554-0563</li>
<li>David Mathews<br/>1011 Malesuada Road<br/>Moscow Kentucky 77382<br/>(357) 6
16-5411</li>
<li>Xyla Cash<br/>969-1762 Tincidunt Rd.<br/>Boise CT 35282<br/>(121) 347-0086
</li>
<li>Madeline Gregory<br/>977-4841 Ut Ave<br/>Walla Walla Michigan 82776<br/>(30
4) 506-6314</li>
<li>Griffith Daniels<br/>6818 Eget St.<br/>Tacoma AL 92508<br/>(425) 288-2332</
li>
<li>Anne Beasley<br/>987-4223 Urna St.<br/>Savannah Illinois 85794<br/>(145) 98
7-4962</li>
<li>Chaney Bennett<br/>P.O. Box 721 902 Dolor Rd.<br/>Fremont AK 19408<br/>(18
7) 582-9707</li>
<li>Daniel Bernard<br/>P.O. Box 567 1561 Duis Rd.<br/>Pomona TN 08609<br/>(750)
558-3965</li>
<li>Willow Hunt<br/>Ap #784-1887 Lobortis Ave<br/>Cudahy Ohio 31522<br/>(492) 4
67-3131</li>
<li>Judith Floyd<br/>361-7936 Feugiat St.<br/>Williston Nevada 58521<br/>(774)
914-2510</li>
<li>Seth Farley<br/>6216 Aenean Avenue<br/>Seattle Utah 81202<br/>(888) 106-855
0</li>
<li>Zephania Sanders<br/>3714 Nascetur St.<br/>Hawthorne Louisiana 10626<br/>(5
39) 567-3573</li>
```

```
<li>Calista Merritt<br/>Ap #938-5470 Posuere Ave<br/>Chickasha LA 58520<br/>(693) 337-2849</li>
<li>Craig Williams<br/>P.O. Box 372 5634 Montes Rd.<br/>Springdale MO 57692<br/>(545) 604-9386</li>
<li>Lee Preston<br/>981 Eget Rd.<br/>Clemson GA 04645<br/>(221) 156-5026</li>
<li>Katelyn Cooper<br/>6059 Sollicitudin Road<br/>Burlingame Colorado 26278<br/>(414) 876-0865</li>
<li>Lacy Eaton<br/>1379 Nulla. Av.<br/>Asbury Park Montana 69679<br/>(932) 726-8645</li>
<li>Driscoll Leach<br/>P.O. Box 120 2410 Odio Avenue<br/>Pass Christian Delaware 03869<br/>(726) 710-9826</li>
<li>Merritt Watson<br/>P.O. Box 686 7014 Amet Street<br/>Corona Oklahoma 55246<br/>(622) 594-1662</li>
<li>Nehru Holmes<br/>P.O. Box 547 4764 Sed Road<br/>Grand Rapids CT 87323<br/>(948) 600-8503</li>
<li>Quamar Rivera<br/>427-5827 Ac St.<br/>Schaumburg Arkansas 84872<br/>(605) 900-7508</li>
<li>Hiram Mullins<br/>754-6427 Nunc Ave<br/>Kennewick AL 41329<br/>(716) 977-5775</li>
<li>Kim Fletcher<br/>Ap #345-3847 Metus Road<br/>Independence CO 30135<br/>(368) 239-8275</li>
<li>Rigel Koch<br/>P.O. Box 558 9561 Lacus. Road<br/>Laughlin Hawaii 99602<br/>(725) 342-0650</li>
<li>Jeanette Sharpe<br/>Ap #364-2006 Ipsum Avenue<br/>Wilmington Ohio 91750<br/>(711) 993-5187</li>
<li>Dahlia Lee<br/>1293 Tincidunt Street<br/>Atwater Pennsylvania 76865<br/>(882) 399-5084</li>
<li>Howard Hayden<br/>P.O. Box 847 8019 Facilisis Street<br/>Joliet SC 73490<br/>(287) 755-9948</li>
<li>Hyatt Kramer<br/>1011 Massa Av.<br/>Kent ID 63725<br/>(659) 551-3389</li>
<li>Sonya Ray<br/>Ap #315-8441 Eleifend Street<br/>Fairbanks RI 96892<br/>(275) 730-6868</li>
<li>Cara Whitehead<br/>4005 Praesent St.<br/>Torrance Wyoming 22767<br/>(725) 757-4047</li>
<li>Blythe Carroll<br/>7709 Justo. Ave<br/>Princeton TX 77987<br/>(314) 882-1496</li>
<li>Dale Griffin<br/>P.O. Box 854 8580 In Ave<br/>Revere South Dakota 43841<br/>(639) 360-7590</li>
<li>McKenzie Hernandez<br/>Ap #367-674 Mi Street<br/>Greensboro VT 40684<br/>(168) 222-1592</li>
<li>Haviva Holcomb<br/>P.O. Box 642 3450 In Road<br/>Isle of Palms New York 03828<br/>(896) 303-1164</li>
<li>Ezra Duffy<br/>Ap #782-7348 Dis Rd.<br/>Austin KY 50710<br/>(203) 982-6130</li>
<li>Eleanor Jennings<br/>9631 Semper Ave<br/>Astoria NJ 66309<br/>(906) 217-1470</li>
<li>Remedios Hester<br/>487-5787 Mollis St.<br/>City of Industry Louisiana 67973<br/>(614) 514-1269</li>
<li>Jasper Carney<br/>1195 Lobortis Rd.<br/>New Orleans New Hampshire 71983<br/>(763) 409-5446</li>
<li>Vielka Nielsen<br/>Ap #517-7326 Elementum Rd.<br/>Fort Smith North Dakota 79637<br/>(836) 292-5324</li>
<li>Wilma Pace<br/>Ap #676-6532 Odio Rd.<br/>Darlington CO 06963<br/>(926) 709-3295</li>
<li>Palmer Gay<br/>557-2026 Purus St.<br/>Watertown TN 07367<br/>(963) 356-9268</li>
<li>Lyle Sutton<br/>Ap #250-9843 Elementum St.<br/>South Gate Missouri 68999<b
```

```
r/>(736) 522-8584</li>
<li>Ina Burt<br/>Ap #130-1685 Ut Street<br/>Tyler KS 73510<br/>(410) 483-0352</
li>
<li>Cleo Best<br/>282-8351 Tincidunt Ave<br/>Sedalia Utah 53700<br/>(252) 204-1
434</li>
<li>Hu Park<br/>1429 Netus Rd.<br/>Reedsport NY 48247<br/>(874) 886-4174</li>
<li>Liberty Walton<br/>343-6527 Purus. Avenue<br/>Logan NV 12657<br/>(581) 379-
7573</li>
<li>Aaron Trujillo<br/>Ap #146-3132 Cras Rd.<br/>Kingsport NH 56618<br/>(983) 6
32-8597</li>
<li>Elmo Lopez<br/>Ap #481-7473 Cum Rd.<br/>Yorba Linda South Carolina 28423<b
r/>(295) 983-3476</li>
<li>Emerson Espinoza<br/>Ap #247-5577 Tincidunt St.<br/>Corpus Christi WI 97020
<br/>(873) 392-8802</li>
<li>Daniel Malone<br/>2136 Adipiscing Av.<br/>Lima RI 93490<br/>(360) 669-3923
</li>
<li>Dante Bennett<br/>481-8762 Nulla Street<br/>Dearborn OR 62401<br/>(840) 987
-9449</li>
<li>Sade Higgins<br/>Ap #287-3260 Ut St.<br/>Wilmington OR 05182<br/>(422) 517-
6053</li>
<li>Zorita Anderson<br/>1964 Facilisis Avenue<br/>Bell Gardens Texas 87065<br/>
(126) 940-2753</li>
<li>Jordan Calderon<br/>430-985 Eleifend St.<br/>Duluth Washington 92611<br/>(4
27) 930-5255</li>
<li>Ivor Delgado<br/>Ap #310-1678 Ut Av.<br/>Santa Barbara MT 88317<br/>(689) 7
21-5145</li>
<li>Pascale Patton<br/>P.O. Box 399 4275 Amet Street<br/>West Allis NC 36734<b
r/>(676) 334-2174</li>
<li>Nasim Strong<br/>Ap #630-3889 Nulla. Street<br/>Watervliet Oklahoma 70863<b
r/>(437) 994-5270</li>
<li>Keaton Underwood<br/>Ap #636-8082 Arcu Avenue<br/>Thiensville Maryland 1958
7<br/>(564) 908-6970</li>
<li>Keegan Blair<br/>Ap #761-2515 Egestas. Rd.<br/>Manitowoc TN 07528<br/>(577)
333-6244</li>
<li>Tamara Howe<br/>3415 Lobortis. Avenue<br/>Rocky Mount WA 48580<br/>(655) 84
0-6139</li>
 </ul> </body></html>
```

In [4]:

```
# (372) 587-2335 Mobile Number
# 96522  Pincode
mob_num = re.findall('\(\d{3}\)\s\d{3}-\d{4}',text)
pincode = re.findall(r'\b\d{5}\b',text)

final_data = {'Mobile Number':mob_num,'Pincode':pincode}
print(final_data)
```


```
{'Mobile Number': ['(257) 563-7401', '(372) 587-2335', '(786) 713-8616', '(793)
151-6230', '(492) 709-6392', '(654) 393-5734', '(404) 960-3807', '(314) 244-630
6', '(947) 278-5929', '(684) 579-1879', '(389) 737-2852', '(660) 663-4518', '(6
08) 265-2215', '(959) 119-8364', '(468) 353-2641', '(248) 675-4007', '(939) 353
-1107', '(570) 873-7090', '(302) 259-2375', '(717) 450-4729', '(453) 391-4650',
'(559) 104-5475', '(387) 142-9434', '(516) 745-4496', '(326) 677-3419', '(746)
679-2470', '(455) 430-0989', '(490) 936-4694', '(985) 834-8285', '(662) 661-144
6', '(802) 668-8240', '(477) 768-9247', '(791) 239-9057', '(832) 109-0213', '(8
37) 196-3274', '(268) 442-2428', '(850) 676-5117', '(861) 546-5032', '(176) 805
-4108', '(715) 912-6931', '(993) 554-0563', '(357) 616-5411', '(121) 347-0086',
'(304) 506-6314', '(425) 288-2332', '(145) 987-4962', '(187) 582-9707', '(750)
558-3965', '(492) 467-3131', '(774) 914-2510', '(888) 106-8550', '(539) 567-357
3', '(693) 337-2849', '(545) 604-9386', '(221) 156-5026', '(414) 876-0865', '(9
32) 726-8645', '(726) 710-9826', '(622) 594-1662', '(948) 600-8503', '(605) 900
-7508', '(716) 977-5775', '(368) 239-8275', '(725) 342-0650', '(711) 993-5187',
'(882) 399-5084', '(287) 755-9948', '(659) 551-3389', '(275) 730-6868', '(725)
757-4047', '(314) 882-1496', '(639) 360-7590', '(168) 222-1592', '(896) 303-116
4', '(203) 982-6130', '(906) 217-1470', '(614) 514-1269', '(763) 409-5446', '(8
36) 292-5324', '(926) 709-3295', '(963) 356-9268', '(736) 522-8584', '(410) 483
-0352', '(252) 204-1434', '(874) 886-4174', '(581) 379-7573', '(983) 632-8597',
'(295) 983-3476', '(873) 392-8802', '(360) 669-3923', '(840) 987-9449', '(422)
517-6053', '(126) 940-2753', '(427) 930-5255', '(689) 721-5145', '(676) 334-217
4', '(437) 994-5270', '(564) 908-6970', '(577) 333-6244', '(655) 840-6139'], 'P
incode': ['96522', '20620', '11523', '39531', '47096', '10855', '08219', '1248
2', '67708', '98804', '69409', '24975', '38100', '45149', '49854', '19253', '02
913', '09289', '93373', '20783', '90255', '37547', '48432', '58563', '93479',
'30309', '51324', '92358', '19759', '55797', '37232', '50654', '31626', '4352
6', '95302', '29738', '54886', '77373', '11292', '73585', '87535', '77382', '35
282', '82776', '92508', '85794', '19408', '08609', '31522', '58521', '81202',
'10626', '58520', '57692', '04645', '26278', '69679', '03869', '55246', '8732
3', '84872', '41329', '30135', '99602', '91750', '76865', '73490', '63725', '96
892', '22767', '77987', '43841', '40684', '03828', '50710', '66309', '67973',
'71983', '79637', '06963', '07367', '68999', '73510', '53700', '48247', '1265
7', '56618', '28423', '97020', '93490', '62401', '05182', '87065', '92611', '88
317', '36734', '70863', '19587', '07528', '48580']}
```

In [5]:
```
df = pd.DataFrame(final_data)
print(df)
# df.to_csv('data.csv')
# df.to_excel('data.xlsx')
# df
```

```
     Mobile Number Pincode
0   (257) 563-7401   96522
1   (372) 587-2335   20620
2   (786) 713-8616   11523
3   (793) 151-6230   39531
4   (492) 709-6392   47096
..             ...     ...
95  (676) 334-2174   36734
96  (437) 994-5270   70863
97  (564) 908-6970   19587
98  (577) 333-6244   07528
99  (655) 840-6139   48580

[100 rows x 2 columns]
```

In [7]:
```
df.to_csv('data.csv')
```