Assingnment 1

1.Introduction of Python

Python is the open source programming language that is used in web programming , artificial intelligence , data science and scientific application. it allow programmer to focus on sloving problems.

It relative size and simplified syntax give it an edge over languages.

2.Data Types

A datatype is a classification that specifies which type of value and what type of mathematical , relational and logical operations. it is used to store and retrieve data.

2.1 Variables

2.2 Number

2.3 String

2.4 Lists

2.5 Dictionaries

2.6 Tuples

2.7 Sets

2.1 Variables:

Python is completely object oriented, and not "statically typed". You do not need to declare variables before using them, or declare their type. Every variable in Python is an object.

it is an reserved memory location to store value.

Float

they represent real numbers and are written with a decimal point dividing the integer and fractional parts

In [2]:
```
# floating type
a=232.4
print(a)
```

```
232.4
```

Integer

integer, is a whole number, positive or negative, without decimals, of unlimited length.

In [2]:
```
#integer
a=34
print(a)
```

```
34
```

String

A string in Python is a sequence of characters. It is a derived data type. Strings are immutable. This means that once defined, they cannot be changed.

In [4]:
```
#string
myname="this is my name"
print(myname)
```

```
this is my name
```

Boolean

Boolean is work as yes or no method

In [10]:
```
# Boolean
a=0
```

```
con=False
if a>0:
    con=True
    print(con)
else:
    print(con)
```

```
False
```

2.2 Number

Python supports integers, floating-point numbers and complex numbers. It can store numeric values.

In [13]:
```
34
```

Out[13]:
```
34
```

In [14]:
```
#integer
23+34-3
```

Out[14]:
```
54
```

In [15]:
```
#float
2.3+45.3
```

Out[15]:
```
47.599999999999994
```

In [16]:
```
#Exponent
2**4
```

Out[16]:
```
16
```

2.3 String

String is used to store character or combination of characters.It is a derived data type. Strings are immutable, it can store words, number etc

2.3.1 Acess values in String

```
name="mehtab"
```

In [18]:
```
print(name)
```

```
mehtab
```

In [19]:
```
Fname="mehtab"
Lname="kazmi"
print(Fname,Lname)
```

```
mehtab kazmi
```

In [20]:
```
Fname="Syed"
Sname="Mehtab"
Lname="kazmi"
print(Fname,Sname,Lname)
```

```
Syed Mehtab kazmi
```

In [21]:
```
#character treated as string
c='A'
print(c)
```

```
A
```

2.3.2 Updating string

To change words or characters or modify string according to requirement is called update string

In [34]:
```
name="Syed"
name=name[:1] +" Mehtab"
print(name)
```

```
S Mehtab
```

In [28]:
```
name="Mehtab kazmi "
name=name[:] +"Syed"
print(name)
```

```
Mehtab kazmi Syed
```

In [30]:
```
name="Mehtab kazmi "
name=name[7:] +"Syed"
print(name)
```

```
kazmi Syed
```

2.3.3 Deleting String

Delete command is use to delete whole string or or many strings.

In [36]:
```
name="mehtab"
print(name)

del name
print(name)
```

```
mehtab
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-36-79b9507ed0a9> in <module>
      3
      4 del name
----> 5 print(name)

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

In [40]:
```
Fname="mehtab"
Lname="kazmi"
print(Fname,Lname)

del Fname
print(Lname)
print(Fname,Lname)
```

```
mehtab kazmi
kazmi
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-40-bf6517bc126e> in <module>
      5 del Fname
      6 print(Lname)
----> 7 print(Fname,Lname)

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

2.3.4 String Special Operator

+Operator

(+) Operator is used in to ways.

First is use for adding integer and floating type values.

Secondly used as concatinate operator.

In [42]:
```
variable="mehtab"
print("My name is "+variable)
```

```
My name is mehtab
```

In [79]:
```
a,b=34,43
print(a+b)
```

```
77
```

* Operator

(*) Operator is use to get n time same value reqursively or repeat same string.

In [43]:
```
variable="34"
print(variable*3)
```

```
343434
```

Indexing

For remove , update or add any number or string in List using Index method .

In [45]:
```
variable="Python"
print(variable[0])
```

```
P
```

In [46]:
```
variable="Python"
print(variable[:3])
```

```
Pyt
```

In [47]:
```
variable="Python"
print(variable[3:])
```

```
hon
```

In [48]:
```
variable="Python"
print(variable[1:3])
```

```
yt
```

In [9]:
```
variable="Python"
print('n' in variable)
```

```
True
```

In [10]:
```
variable="Python"
print('N' in variable)
```

```
False
```

In [58]:
```
variable="Python P"
print('P' in variable,"Repeat character P ",variable.count("P")," Time")
```

```
True Repeat character P  2  Time
```

2.3.4 String formatting operator

In [65]:
```
print ("My name is %s and age is %d" % ("Mehtab ",26))
```

```
My name is Mehtab  and age is 26
```

2.4 Lists

List is used to store multiple items in single variable.

It is built-in datatype in python to store collection of data.

It can store different datatype in asingle variable.

List items are ordered, changeable and allow duplicate values.

List created using square barcket.

In [67]:
```
x=["Ali","Ahmed","Raza","Haider"]
print(x)
```

```
['Ali', 'Ahmed', 'Raza', 'Haider']
```

(1)Accessing values in Lists

List are indexed and web can access any item use reference to the index number.

In [68]:
```
x=["Ali","Ahmed","Raza","Haider"]
print(x[1])
```

```
Ahmed
```

In [69]:
```
#print last element
x=["Ali","Ahmed","Raza","Haider"]
print(x[-1])
```

```
Haider
```

In [70]:
```
x=["Ali","Ahmed","Raza","Haider"]
print(x[-3:-1])
```

```
['Ahmed', 'Raza']
```

(2)Updating Lists

To change the value of specific item.

In [71]:
```
x=["Ali","Ahmed","Raza","Haider"]
x[2]="Raheel"
print(x)
```

```
['Ali', 'Ahmed', 'Raheel', 'Haider']
```

In [74]:
```
x=["Ali","Ahmed","Raza","Haider"]
x[1:3]=["Raheel","Razaq"]
print(x)
```

```
['Ali', 'Raheel', 'Razaq', 'Haider']
```

In [75]:
```
#take index 1 and 2 but update only 1 element
x=["Ali","Ahmed","Raza","Haider"]
x[1:3]=["Raheel"]
print(x)
```

```
['Ali', 'Raheel', 'Haider']
```

(3)Delete List element

To delete specific item or delete complete list.

In [78]:
```
x=["Ali","Ahmed","Raza","Haider"]
x.remove("Ali")
print(x)
```

```
['Ahmed', 'Raza', 'Haider']
```

In [79]:
```
#delete 1 element
x=["Ali","Ahmed","Raza","Haider"]
del x[1]
print(x)
```

```
['Ali', 'Raza', 'Haider']
```

In [81]:
```
#delete whole list
x=["Ali","Ahmed","Raza","Haider"]
del x
print(x)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-81-e80249bf9081> in <module>
      2 x=["Ali","Ahmed","Raza","Haider"]
      3 del x
----> 4 print(x)

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

In [83]:
```
#delete whole list
x=["Ali","Ahmed","Raza","Haider"]
del x[2]
print(x)
```

```
['Ali', 'Ahmed', 'Haider']
```

In [84]:
```
x=["Ali","Ahmed","Raza","Haider"]
x.clear()
print(x)
```

```
[]
```

2.5 Dictionaries

Dictionary is used to store data in keys.

Dictionary is ordered, changeable and not allow duplicate values.

Dictionary is written with curly brackets.

(1)Accessing value in Dictionary

You can access the items of a dictionary by referring to its key name

In [2]:
```
info={"name":"Ali", "age":21,"addr":"town"}
print(info["name"])
```

```
Ali
```

In [7]:
```
info={"name":"Ali", "age":21,"addr":"town"}
print(info["age"])
print(info["addr"])
```

```
21
town
```

(2)Updating Dictionary

You can change the value of a specific item by referring to its key name.

In [8]:
```
info={"name":"Ali", "age":21,"addr":"town"}
info["name"]="Ahmed"
print(info)
```

```
{'name': 'Ahmed', 'age': 21, 'addr': 'town'}
```

In [10]:
```
info={"name":"Ali", "age":21,"addr":"town"}
info["name"]="Ahmed"
print(info)

info["dic"]="abc"
print(info)
```

```
{'name': 'Ahmed', 'age': 21, 'addr': 'town'}
{'name': 'Ahmed', 'age': 21, 'addr': 'town', 'dic': 'abc'}
```

In [11]:
```
#override value......
```

```
info={"name":"Ali", "age":21,"addr":"town","age":20}
print(info)
```

```
{'name': 'Ali', 'age': 20, 'addr': 'town'}
```

In [12]:
```
info={"name":"Ali", "age":21,"addr":"town"}
info.update({"age":20})
print(info)
```

```
{'name': 'Ali', 'age': 20, 'addr': 'town'}
```

(3)Delete Dictionary Element

To delete specific item using key name or delete complete Dictionary.

In [14]:
```
info={"name":"Ali", "age":21,"addr":"town"}

del info
print(info)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-14-a99933b232b5> in <module>
      2
      3 del info
----> 4 print(info)

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

In [16]:
```
info={"name":"Ali", "age":21,"addr":"town"}
del info["name"]
print(info)
```

```
{'age': 21, 'addr': 'town'}
```

In [19]:
```
info={"name":"Ali", "age":21,"addr":"town"}
info.pop("name")
print(info)
```

```
{'age': 21, 'addr': 'town'}
```

In [23]:
```
#pop last item....
info={"name":"Ali", "age":21,"addr":"town"}
```

```
info.popitem()
print(info)
```

```
{'name': 'Ali', 'age': 21}
```

In [25]:
```
info={"name":"Ali", "age":21,"addr":"town"}
info.clear()
print(info)
```

```
{}
```

2.6 Tuples

Tuples are used to store multiple items in a single variable.

A tuple is ordered and unchangeable.

(1)Accessing values in Tuples

You can access tuple items by referring to the index number.

In [28]:
```
info=("name", "age","addr")

print(info[1])
```

```
age
```

In [30]:
```
info=("name", "age","addr")

print(info[1:3])
```

```
('age', 'addr')
```

In [31]:
```
#negative indexing.....
info=("name", "age","addr")

print(info[-1])
```

```
addr
```

(2)Updating Tuples

Tuples are unchangeable that,s way we can convert into list and make changeable.

In [36]:
```
info=("name", "age","addr")
y=list(info)
y[1]="mahi"

info=tuple(y)
print(info)
```

```
('name', 'mahi', 'addr')
```

In [41]:
```
pencil=("red","green","blue")
pen=("black","white","gray")

pointer=pencil+pen
print(pointer)
ball=list(pointer)
ball[2]="orange"
pointer=tuple(ball)
print(pointer)
```

```
('red', 'green', 'blue', 'black', 'white', 'gray')
('red', 'green', 'orange', 'black', 'white', 'gray')
```

In [45]:
```
products=("orio","dairy","olpers")
apply=list(products)
apply.append("good milk")
products=tuple(apply)
print(products)
```

```
('orio', 'dairy', 'olpers', 'good milk')
```

(3)Delete Tuple elements

For deleting whole Tuple.

In [48]:
```
products=("orio","dairy","olpers")
del products
print(products)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-48-852fd29e520f> in <module>
```

```
      1 products=("orio","dairy","olpers")
      2 del products
----> 3 print(products)

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

In [52]:
```
products=("orio","dairy","olpers")
apple=list(products)
del apple[2]
products=tuple(apple)
print(products)
```

```
('orio', 'dairy')
```

In [55]:
```
products=("orio","dairy","olpers")
apple=list(products)
apple.clear()
products=tuple(apple)
print(products)
```

```
()
```

2.7 Sets

Sets are used to store multiple items in a single variable.

A set is unordered and unindexed.

Sets are written with curly brackets.

(1)Accessing set items

ou cannot access items in a set by referring to an index or a key.

In [57]:
```
item={"Ali","mujtaba","haroon"}
print(item)
```

```
{'Ali', 'mujtaba', 'haroon'}
```

In [59]:
```
item={"Ali","mujtaba","haroon"}
for i in item:
    print(i)
```

```
Ali
mujtaba
haroon
```

In [62]:
```
#finding type of list
item={"Ali","mujtaba","haroon"}
print(type(item))
```

```
<class 'set'>
```

(2)Updating Sets

modify sets randomly

In [65]:
```
item={"Ali","mujtaba","haroon"}
item.add("zaraq")
print(item)
```

```
{'zaraq', 'Ali', 'mujtaba', 'haroon'}
```

In [67]:
```
#unordered  update....
item={"Ali","mujtaba","haroon"}
item.update("mehtab")
print(item)
```

```
{'e', 'm', 'haroon', 'h', 't', 'a', 'mujtaba', 'Ali', 'b'}
```

In [69]:
```
#Duplication not allowed
item={"Ali","mujtaba","haroon"}
item1={"Ali","hamza","Moiz"}
item.update(item1)
print(item)
```

```
{'hamza', 'Ali', 'haroon', 'mujtaba', 'Moiz'}
```

(3)Delete Set elements

To remove an item in a set, use the remove(), or the discard() method.

In [71]:
```
# delete complete set
item={"Ali","mujtaba","haroon"}
del item[1]
print(item)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-71-51948c224c30> in <module>
      1 item={"Ali","mujtaba","haroon"}
----> 2 del item[1]
      3 print(item)

TypeError: 'set' object doesn't support item deletion
```

In [72]:
```
#remove element
item={"Ali","mujtaba","haroon"}
item.remove("Ali")
print(item)
```

```
{'mujtaba', 'haroon'}
```

In [78]:
```
#pop method
item={"Ali","mujtaba","haroon"}
item.pop()
print(item)
```

```
{'mujtaba', 'haroon'}
```

In [80]:
```
#dicard method
item={"Ali","mujtaba","haroon"}
item.discard("haroon")
print(item)
```

```
{'Ali', 'mujtaba'}
```

3. Comparison Operators

Different Operators are used to compare integer or floating type variables.

these are >,<,>=,<= , or , and

In [81]:
```
1>2
```

Out[81]:
```
False
```

In [82]:
```
2>1
```

Out[82]: True

In [85]:
```
3>=2
```

Out[85]: True

In [86]:
```
3>2 or 3>4
```

Out[86]: True

In [87]:
```
4<= 5 and 4>5
```

Out[87]: False

In [88]:
```
5 !=4
```

Out[88]: True

In [89]:
```
"mehtab"=="kazmi"
```

Out[89]: False

4. If-Else Statements

Python supports the usual logical conditions from mathematics.

Equal Operator: ==

Not equal Operator: !=

Less than: <

greater than: >

Greater and equal to: >=

Less and equal to <=

Simple If Condition

In [90]:
```
a=2
if a >0:
    print("a is greater")
```

```
a is greater
```

In [92]:
```
#else-if condition
a=0
if a >0:
    print("a is greater")
else:
    print("a is not greater")
```

```
a is not greater
```

In [93]:
```
a,b,c=7,3,5
if (a > b and a>c):
    print("a is greater")
elif(b > a and b>c):
    print("b is greater")
elif(c > b and c>a):
    print("a is greater")
```

```
a is greater
```

5.For and While Loop

Loops are use for iteration of any number, string , or any statement.

For loop

A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

It is works more like an iterator method as found in other object-orientated programming languages.

In for loop we can execute set of statements.

In [94]:
```
list1=[2,33,43,54]
for i in list1:
    print(i)
```

```
2
33
43
54
```

In [95]:
```
name="Mehtab"
for i in name:
    print(i)
```

```
M
e
h
t
a
b
```

In [96]:
```
for i in range(5):
    print(i)
```

```
0
1
2
3
4
```

In [98]:
```
for i in range(10):
    if i==3:
        break
    print(i)
```

```
0
1
2
```

In [99]:
```
for i in range(10):
    if i==4:
        continue
    print(i)
```

```
0
```

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

While Loop

A while loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string). With the while loop we can execute a set of statements as long as a condition is true.

In [103…
```
a=0
while a<6:
    print("index is",a)
    a +=1
```

```
index is 0
index is 1
index is 2
index is 3
index is 4
index is 5
```

In [104…
```
a=7
while a<6:
    print(a)
    a +=1
else:
    print("the value is greater than 6")
```

```
the value is greater than 6
```

In [110…
```
i=0
name="mehtab"
while i<=7:
    print(name[i])
    i+=1
```

```
m
e
```

```
h
t
a
b
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-110-6820973abb2c> in <module>
      2 name="mehtab"
      3 while i<=7:
----> 4     print(name[i])
      5     i+=1

IndexError: string index out of range
```

Functions:

A function is a block of code which only runs when it is called.

You can pass data, known as parameters, into a function.

A function can return data as a result.

In [113…
```
def func():
    print("this is my function")

func()
```

```
this is my function
```

In [116…
```
def func(name):
    print("my name is "+name)
func("mehtab")
```

```
my name is mehtab
```

In [117…
```
def func(fname,lname):
    print(fname+"  "+lname)
func("mehtab","kazmi")
```

```
mehtab  kazmi
```

In [120…
```
def func(*info):
```

```
    print("phone number is ",info[1])
func("mehtab","age",432)
```

```
phone number is  age
```

7. Lmabda Functions

A lambda function can take any number of arguments, but can only have one expression.

In [121…
```
value=lambda a:a+5
print(value(4))
```

```
9
```

In [129…
```
def func(n):

    return lambda a:a**n
myfunc=func(2)
print(myfunc(3))
```

```
9
```

In [11]:
```
def func(n):
    return lambda a:a**n

myfunc=func(3)
myfunc1=func(4)
print(myfunc(20))
print(myfunc1(10))
```

```
8000
10000
```

In [139…
```
def func(n):
    return lambda a,b:a**b+n

myfunc=func(3)
myfunc1=func(4)
print(myfunc(2,3))
print(myfunc1(10,3))
```

```
11
```

```
1004
```

7.1 Map

Calculate the lenght of each word in the Tuple.

The map() function executes a specified function for each item in an iterable. The item is sent to the function as a parameter.

In [147…
```
def func(list1):
    return len(list1)
x=map(func,("mehtab","kazmi","al"))

print(x)
print(list(x))
```

```
<map object at 0x000001B372DBD940>
[6, 5, 2]
```

In [150…
```
name="my name is mujtaba"
a=name.split()
print(a)
lenght=map(lambda b:len(b),a)
list(lenght)
```

```
['my', 'name', 'is', 'mujtaba']
```

Out[150…
```
[2, 4, 2, 7]
```

In [153…
```
def func(list1,list2):
    return list1+list2
x=map(func,("mehtab ","kazmi ","al "),("mehtab","Ahmed","Ali"))

print(x)
print(list(x))
```

```
<map object at 0x000001B3746AB760>
['mehtab mehtab', 'kazmi Ahmed', 'al Ali']
```

7.2 Filter()

Filter the array, and return a new array with only the values equal to or above

In [161…
```
list1 = [1,1,2,33,51,28,113,1,34,55]
result = filter(lambda x: x % 2, list1)
list(result)
```

Out[161…
```
[1, 1, 33, 51, 113, 1, 55]
```

In [162…
```
list1 = [1,1,2,33,51,28,113,1,34,55]
def func(x):
    if x <30:
        return True
    else:
        return False

x=filter(func,list1)

for i in x:
    print(i)
```

```
1
1
2
28
1
```

In [165…
```
#even values.......
list1 = [1,1,2,33,51,28,113,1,34,55]
result = filter(lambda x: x % 2==0, list1)
list(result)
```

Out[165…
```
[2, 28, 34]
```

8. File I/O

Python too supports file handling and allows users to handle files i.e., to read and write files, along with many other file handling options, to operate on files.

“ r “, for reading.

“ w “, for writing.

“ a “, for appending.

“ r+ “, for both reading and writing

Reading input from Keyboard

In [1]:
```
name=input("enter name")
print("name is "+name)
```

```
enter namemehtab
name is mehtab
```

In [37]:
```
file = open("JsonConversion.py", "r")

print(file.read())
file.close()
```

```
Now Now my name is mehtab kazmimy name is mehtab kazm
my name is mehtab kazm
```

In [35]:
```
#read specific code
file = open("JsonConversion.py", "r")

print(file.read(50))
file.close()
```

```
Now Now my name is mehtab kazmimy name is mehtab k
```

In [36]:
```
#for read line.....
file = open("JsonConversion.py", "r")

print(file.readline(50))
file.close()
```

```
Now Now my name is mehtab kazmimy name is mehtab k
```

In [13]:
```
#for loop you can read whole file line by line
file = open("JsonConversion.py", "r")
```

```
for x in file:
    print(x)
```

```
Now Now my name is mehtab kazmimy name is mehtab kazm

my name is mehtab kazm
```

In [14]:
```
file=open("JsonConversion.py","a")
file.write("my name is mehtab kazm\n")
file.close()

file=open("JsonConversion.py","r")
print(file.read())
```

```
Now Now my name is mehtab kazmimy name is mehtab kazm
my name is mehtab kazm
my name is mehtab kazm
```

In [90]:
```
#creating file....
#file=open("my.py","x")
#file.close()

# if file not exist, write function automatically create new file
file=open("my.py","a")
file.write("im doing MCS\n")
file.close()

file=open("my.py","r")
print(file.read())
file.close()
```

```
im doing MCSim doing MCSim doing MCSim doing MCSim doing MCS
im doing MCS
im doing MCS
```

In [81]:
```
import os
if os.path.exists("my.py"):
  os.remove("my.py")
else:
  print("The file does not exist")
```

File Position

Tell()

Tell() function tells us the position of file.

In [93]:
```
file=open("my.py","r")
print(file.readline())
print(file.tell())
```

```
im doing MCSim doing MCSim doing MCSim doing MCSim doing MCS

62
```

Seek()

Seek() function is use to change the current position of file.

In [102…
```
file=open("my.py","r")
file.seek(30)
print(file.readline())
```

```
ng MCSim doing MCSim doing MCS
```

Rename file

In [104…
```
#rename file ......
import os
os.rename("basic.py","basic2.py")
```

9. Pandas:

Indroduction:

Pandas deriveds from the word 'panel data', it is an econometric term for data set that includes observations multiple time period over same individuales.If you are thinking about data science as a career, then it is imperative that one of the first things you do is learn pandas.

Pandas is a open source high-level data manipulation tool built on the Numpy package providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language. Python with Pandas is used in a wide range of fields including academic and commercial domains including finance, economics, Statistics, analytics.

Python was majorly used for data mining and preparation. It has very little contribution towards data analysis. Pandas solved this problem. Using Pandas, we can accomplish five typical steps in the

- processing and analysis of data.
- Regardless of the origin of data — load.
- Prepare.
- Manipulate.
- Model.

10. Series

Series is a one-dimensional labeled array capable of holding data of any type (integer, string, float, python objects, etc.)

Labels need not be unique but must be a hashable type. The object supports both integer- and label-based indexing and provides a host of methods for performing operations involving the index.

The result index will be the sorted union of the two indexes.

10.1 From ndarray

In [21]:
```
# indexing through integer..
import pandas as pd
import numpy as np

lists=[1,21,3,4,5]

pd.Series(lists)
```

Out[21]:
```
0     1
1    21
2     3
3     4
```

```
4     5
dtype: int64
```

In [22]:
```
#indexing through label
import pandas as pd
lists=[1,21,3,4,5]
pd.Series(lists,index=['a','b','c','f','d'])
```

Out[22]:
```
a     1
b    21
c     3
f     4
d     5
dtype: int64
```

In [23]:
```
# get randon value
pd.Series(np.random.randn(5))
```

Out[23]:
```
0   -0.376751
1    2.499497
2   -2.154971
3   -0.807243
4    0.152303
dtype: float64
```

In [24]:
```
a=pd.Series(np.random.randn(5))
a.index
```

Out[24]:
```
RangeIndex(start=0, stop=5, step=1)
```

10.2 From Dict

If data is a dict, if index is passed the values in data corresponding to the labels in the index will be pulled out.

In [27]:
```
import pandas as pd
lists={"day1":1,"day2":2,"day3":3}
pd.Series(lists)
```

Out[27]:
```
day1    1
day2    2
day3    3
dtype: int64
```

In [28]:
```
import pandas as pd
lists={"day1":1,"day2":2,"day3":3}
pd.Series(lists,index=['day1','day2','day3'])
```

Out[28]:
```
day1    1
day2    2
day3    3
dtype: int64
```

10.3 From a scalar value

In [32]:
```
#scalar integer value
pd.Series(5, index=['a', 'b', 'c', 'd', 'e'])
```

Out[32]:
```
a    5
b    5
c    5
d    5
e    5
dtype: int64
```

In [34]:
```
#scalar floating value
pd.Series(5., index=['a', 'b', 'c', 'd', 'e'])
```

Out[34]:
```
a    5.0
b    5.0
c    5.0
d    5.0
e    5.0
dtype: float64
```

In [35]:
```
pd.Series("mehtab", index=['a', 'b', 'c', 'd', 'e'])
```

Out[35]:
```
a    mehtab
b    mehtab
c    mehtab
d    mehtab
e    mehtab
dtype: object
```

10.4 Series is ndarray-like

it is similar to ndarray and it is valid argument to most numPy functions.

Slicing is allow in ndarray like

In [39]:
```
import pandas as pd
lists=[1,21,3,4,5]
pd.Series(lists[2])
```

Out[39]:
```
0    3
dtype: int64
```

In [41]:
```
import pandas as pd
lists=["mehtab","Ali","zaraq","mujtaba"]
pd.Series(lists[:2])
```

Out[41]:
```
0    mehtab
1       Ali
dtype: object
```

In [42]:
```
import pandas as pd
lists=["mehtab","Ali","zaraq","mujtaba"]
pd.Series(lists[1:])
```

Out[42]:
```
0        Ali
1      zaraq
2    mujtaba
dtype: object
```

In [43]:
```
import pandas as pd
lists=["mehtab","Ali","zaraq","mujtaba"]
pd.Series(lists[1:2])
```

Out[43]:
```
0    Ali
dtype: object
```

In [55]:
```
pd.Series(list('mehtab')).values
```

Out[55]:
```
array(['m', 'e', 'h', 't', 'a', 'b'], dtype=object)
```

make categorioes from whole list

In [57]:
```
pd.Series(list('meeehtab')).astype('category').values
```

Out[57]:
```
['m', 'e', 'e', 'e', 'h', 't', 'a', 'b']
Categories (6, object): ['a', 'b', 'e', 'h', 'm', 't']
```

In [51]:
```
import pandas as pd

s[s > s.median()]
```

Out[51]:
```
2   -0.389495
3   -0.781896
6    0.599058
7    0.783040
9    0.744525
dtype: float64
```

In [53]:
```
# position of index
s[[3, 7, 1]]
```

Out[53]:
```
3   -0.781896
7    0.783040
1   -1.654205
dtype: float64
```

10.6 Vectorized operations and label alignment with Series

we can easily and quickly perform the same operation on many array elements.

In [68]:
```
import numpy as np
a=np.array([2,3,4,65])
a*2
```

Out[68]:
```
array([  4,   6,   8, 130])
```

To capitalize the strings

In [69]:
```
import numpy as np
a=np.array(["mehtab","Ali","zaraq","mujtaba"])
[s.capitalize() for s in a]
```

Out[69]:
```
['Mehtab', 'Ali', 'Zaraq', 'Mujtaba']
```

In [75]:
```
import numpy as np
```

```
a=np.array(["mehtab","Ali","zaraq","mujtaba"])
a.str.capitalize()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-75-669f8273ff24> in <module>
      1 import numpy as np
      2 a=np.array(["mehtab","Ali","zaraq","mujtaba"])
----> 3 a.str.capitalize()

AttributeError: 'numpy.ndarray' object has no attribute 'str'
```

In [70]:
```
a=np.array(["mehtab","Ali","zaraq","mujtaba"])
pd.Series(a)
```

Out[70]:
```
0     mehtab
1        Ali
2      zaraq
3    mujtaba
dtype: object
```

In [78]:
```
names = pd.Series(np.random.randn(5), name='something')
names
renames=names.rename("Mehtab")
renames
```

Out[78]:
```
0   -0.091328
1    0.685016
2   -0.927103
3    0.709362
4    0.478251
Name: Mehtab, dtype: float64
```

11. Data Frames

A Pandas DataFrame is a 2 dimensional data structure, like a 2 dimensional array, or a table with rows and columns.

Features of DataFrame:

Potentially columns are of different types

Size – Mutable

Labeled axes (rows and columns)

Can Perform Arithmetic operations on rows and columns

11.1 From dict of Series or dicts

make matrix through dictionary

In [80]:
```
import pandas as pd
information={'name':["mehtab","muji","zaraq","ali"],
             'age':[23,43,56,4],
             'gender':["male","male","male","male"]}
data=pd.DataFrame(information)
data
```

Out[80]:

	name	age	gender
0	mehtab	23	male
1	muji	43	male
2	zaraq	56	male
3	ali	4	male

In [89]:
```
import pandas as pd
information={'Name':pd.Series(["mehtab","muji","zaraq","ali"],index=["student1",'student2','student3','student4']),
             'Age':pd.Series([23,23,21],index=["student1",'student3','student4']),
             'Gender':pd.Series(["male","male"],index=['student2','student4'])}
data=pd.DataFrame(information)
data
```

Out[89]:

	Name	Age	Gender
student1	mehtab	23.0	NaN
student2	muji	NaN	male
student3	zaraq	23.0	NaN

	Name	Age	Gender
student4	ali	21.0	male

In [103…
```
lists=[{'a':4,'b':5,'c':76,'d':45}]
pd.DataFrame(lists, index=['1st','2nd','3rd','4th'])
```

Out[103…

	a	b	c	d
1st	4	5	76	45
2nd	4	5	76	45
3rd	4	5	76	45
4th	4	5	76	45

In [107…
```
lists=[{'a':4,'b':5,'c':76,'d':45},{'a':4,'b':5,'c':76},{'a':4,'c':6,'d':34}]
pd.DataFrame(lists, index=['1st','2nd','3rd'])
```

Out[107…

	a	b	c	d
1st	4	5.0	76	45.0
2nd	4	5.0	76	NaN
3rd	4	NaN	6	34.0

In [118…
```
import pandas as pd
information={'Name':pd.Series(["mehtab","muji","zaraq","ali"],index=["student1",'student2','student3','student4']),
             'Age':pd.Series([23,23,21],index=["student1",'student3','student4']),
             'Gender':pd.Series(["male","male"],index=['student2','student4'])}
data=pd.DataFrame(information)
print (data['Name'])
```

```
student1    mehtab
student2      muji
student3     zaraq
student4       ali
Name: Name, dtype: object
```

In [140…
```
info={"Name": pd.Series(["mehtab","muji","zaraq","ali"],index=["std1","std2","std3","std4"]),
```

```
     "Age": pd.Series([23,3,56,6],index=["std1","std2","std3","std4"])}
pf=pd.DataFrame(info)
pf["Gender"]=pd.Series(["male","male","female","male"],index=["std1","std2","std3","std4"])
print(pf,"\n\n")

pf["price"]=pd.Series([34000,56000,40000,120000],index=["std1","std2","std3","std4"])
print(pf)

pf["Bonus"]=pd.Series([5000,4500,4540,2300],index=["std1","std2","std3","std4"])
pf["total"]= pf["price"]+pf["Bonus"]
pf
```

```
        Name  Age  Gender
std1  mehtab   23    male
std2    muji    3    male
std3   zaraq   56  female
std4     ali    6    male


        Name  Age  Gender   price
std1  mehtab   23    male   34000
std2    muji    3    male   56000
std3   zaraq   56  female   40000
std4     ali    6    male  120000
```

Out[140…

	Name	Age	Gender	price	Bonus	total
std1	mehtab	23	male	34000	5000	39000
std2	muji	3	male	56000	4500	60500
std3	zaraq	56	female	40000	4540	44540
std4	ali	6	male	120000	2300	122300

In [142…

```
import pandas as pd

d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
   'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd']),
   'three' : pd.Series([10,20,30], index=['a','b','c'])}

df = pd.DataFrame(d)
print ("Our dataframe is:")
print(df)
```

```
# using del function
print ("Deleting the first column using DEL function:")
del df['one']
print(df)

# using pop function
print ("Deleting another column using POP function:")
df.pop('two')
print(df)
```

```
Our dataframe is:
   one  two  three
a  1.0    1   10.0
b  2.0    2   20.0
c  3.0    3   30.0
d  NaN    4    NaN
Deleting the first column using DEL function:
   two  three
a    1   10.0
b    2   20.0
c    3   30.0
d    4    NaN
Deleting another column using POP function:
   three
a   10.0
b   20.0
c   30.0
d    NaN
```

11.2 From dict of ndarrays / lists

the ndarray must have same lenght of columns.

if we pass index, we must clearly write same lenght as array. if no index passed, the result will be range(n)

In [145…

```
import pandas as pd
marks={
    "networking": [54,56,67,65],
    "Android":[34,54,3,4]
}
pd.DataFrame(marks)
```

Out[145…

	networking	Android
0	54	34
1	56	54
2	67	3
3	65	4

In [146…

```
import pandas as pd
marks={
    "networking": [54,56,67,65],
    "Android":[34,54,3,4]
}
pd.DataFrame(marks,index=["Ali",'Mehtab','haroon','moiz'])
```

Out[146…

	networking	Android
Ali	54	34
Mehtab	56	54
haroon	67	3
moiz	65	4

11.3 From a list of dicts

In [147…

```
marks=[{"math":34,"Phy":45,"Bio":67},{"math":68,"Phy":54,"Bio":63}]
pd.DataFrame(marks)
```

Out[147…

	math	Phy	Bio
0	34	45	67
1	68	54	63

In [149…

```
marks=[{"math":34,"Phy":45,"Bio":67},{"math":68,"Phy":54,"Bio":63},{"math":68}]
pd.DataFrame(marks)
```

Out[149…

	math	Phy	Bio
0	34	45.0	67.0
1	68	54.0	63.0
2	68	NaN	NaN

In [151…

```
import pandas as pd
lists=[{34,454,45}]
pd.DataFrame(lists,index=["math",'Phy','Chem'])
```

Out[151…

	0	1	2
math	34	45	454
Phy	34	45	454
Chem	34	45	454

In [152…

```
import pandas as pd
lists=[{34,454,45},{34,44,4},{44,54,5}]
pd.DataFrame(lists,index=["math",'Phy','Chem'])
```

Out[152…

	0	1	2
math	34	45	454
Phy	34	44	4
Chem	44	5	54

11.4 From a dict of tuples

You can automatically create a multi-indexed frame by passing a tuples dictionary

In [28]:

```
import pandas as pd
pd.DataFrame({('SPA', 'networking'): {('A Section', 'ALi'): 60, ('A Section', 'Hamza'): 56},
('SPA', 'Python'): {('A Section', 'Hamza'): 87, ('A Section', 'ALi'): 88},
('SPA', 'Android'): {('A Section', 'ALi'): 76, ('A Section', 'Hamza'): 60},
```

```
('SST', 'Python'): {('A Section', 'Rehan'): 87, ('A Section', 'Haroon'): 90},
('SST', 'networking'): {('A Section', 'Haroon'): 78, ('A Section', 'Haroon'): 80}})
```

Out[28]:

			SPA		SST	
		networking	Python	Android	Python	networking
A Section	ALi	60.0	88.0	76.0	NaN	NaN
	Hamza	56.0	87.0	60.0	NaN	NaN
	Rehan	NaN	NaN	NaN	87.0	NaN
	Haroon	NaN	NaN	NaN	90.0	80.0

In [3]:

```
import pandas as pd
pd.DataFrame({('SPA','MCS','Semester 1', 'Intro to Programming'): {('A Section', 'ALi','F2019027015'): 65, ('A Sectio
('SPA','MCS','Semester 1', 'networking'): {('A Section', 'ALi','F2019027015'): 60, ('A Section', 'Hamza','F201902701
('SPA','MCS','Semester 1', 'Python'): {('A Section', 'Hamza','F2019027017'): 87, ('A Section', 'ALi','F2019027015'):
('SPA','MCS','Semester 1', 'Android'): {('A Section', 'ALi','F2019027015'): 76, ('A Section', 'Hamza','F2019027017')
              ('SPA','MCS','Semester 2', 'networking'): {('A Section', 'ALi','F2019027015'): 60, ('A Section', 'Hamza
('SPA','MCS','Semester 2', 'Python'): {('A Section', 'Hamza','F2019027017'): 87, ('A Section', 'ALi','F2019027015'):
('SPA','MCS','Semester 2', 'Android'): {('A Section', 'ALi','F2019027015'): 76, ('A Section', 'Hamza','F2019027017')
              ('SPA','MCS','Semester 3', 'networking'): {('A Section', 'ALi','F2019027015'): 60, ('A Section', 'Hamza
('SPA','MCS','Semester 3', 'Python'): {('A Section', 'Hamza','F2019027017'): 87, ('A Section', 'ALi','F2019027015'):
('SPA','MCS','Semester 3', 'Android'): {('A Section', 'ALi','F2019027015'): 76, ('A Section', 'Hamza','F2019027017')
('SST','MIT','Semester 4', 'Python'): {('A Section', 'Rehan','F2019027014'): 87, ('A Section', 'Haroon','F2019027012
('SST','MIT','Semester 4', 'networking'): {('A Section', 'Haroon','F2019027012'): 78, ('A Section', 'Haroon','F201902
```

Out[3]:

			SPA										
			MCS										
			Semester 1			Semester 2			Semester 3			Se	
			Intro to Programming	networking	Python	Android	networking	Python	Android	networking	Python	Android	Python	ne
A Section	ALi	F2019027015	65.0	60.0	88.0	76.0	60.0	88.0	76.0	60.0	88.0	76.0	NaN	
	Hamza	F2019027017	56.0	56.0	87.0	60.0	56.0	87.0	60.0	56.0	87.0	60.0	NaN	
	Rehan	F2019027014	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	87.0	
	Haroon	F2019027012	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	90.0	

11.5 Alternate Constructors

DataFrame.from_dict

DataFrame.from_dict takes a dict of dicts or a dict of array-like sequences and returns a DataFrame.

It operates like the DataFrame constructor except for the orient parameter which is 'columns' by default, but which can be set to 'index' in order to use the dict keys as row labels.

In [14]:
```
import pandas as pd
pd.DataFrame.from_dict(dict([('A',[2,3,8]),('B',[4,5,6])]))
```

Out[14]:

	A	B
0	2	4
1	3	5
2	8	6

In [22]:
```
import pandas as pd
pd.DataFrame.from_dict(dict([('A',[3,5,6,7]),('B',[4,56,67,8]),('C',[4,6,6,8])]),orient='index'
                       ,columns=['one','two','three','four'])
```

Out[22]:

	one	two	three	four
A	3	5	6	7
B	4	56	67	8
C	4	6	6	8

DataFrame.from_records

In [55]:
```
data = np.zeros((4,), dtype=[('A', 'i4'),('B', 'f4'),('C', 'a10'),('D', 'a10')])
pd.DataFrame.from_records(data, index='D')
```

Out[55]:

	A	B	C

D	A	B	C
D			
b"	0	0.0	b"
b"	0	0.0	b"
b"	0	0.0	b"
b"	0	0.0	b"

DataFrame.from_items

DataFrame.from_items works analogously to the form of the dict constructor that takes a sequence of (key, value) pairs, where the keys are column names, and the value are the column values. It can be useful for constructing a DataFrame with the columns in a particular order without having to pass an explicit list of columns.

But now python deplicated this library, so it is not working.

In [63]:
```
import pandas as pd
pd.DataFrame.from_items([('A',[35,65,7]),('B',[35,65,7])])
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-63-a5acfad684bc> in <module>
      1 import pandas as pd
----> 2 pd.DataFrame.from_items([('A',[35,65,7]),('B',[35,65,7])])

AttributeError: type object 'DataFrame' has no attribute 'from_items'
```

11.6 Column selection, addition, deletion

Column selection

this method is use to select specific columns from whole table.

In [72]:
```
import pandas as pd
data={ 'Student': pd.Series([3,5,"m"],index=['a','b','c']),
     'Teacher': pd.Series([3,53,"s"],index=['a','b','c'])}
```

```
fdata=pd.DataFrame(data)
fdata['Teacher']
```

Out[72]:
```
a     3
b    53
c     s
Name: Teacher, dtype: object
```

Column addition

this function is used to add new columns in the table

In [78]:
```
import pandas as pd
data ={'id':pd.Series([1,2,3,4]),
       'name':pd.Series(['muji','zaraq','haroon'])}

data1=pd.DataFrame(data)
data1

data1['gender']=pd.Series(['male','male','male','male'])
data1
```

Out[78]:

	id	name	gender
0	1	muji	male
1	2	zaraq	male
2	3	haroon	male
3	4	NaN	male

In [91]:
```
import pandas as pd
data ={'id':pd.Series([1,2,3,4]),
       'name':pd.Series(['muji','zaraq','haroon'])}

data1=pd.DataFrame(data)
data1

data1['price']=pd.Series([300,400,2300,500])
data1['bonus']=pd.Series([30,40,30,20])
```

```
data1['total']=data1['price']+data1['bonus']
data1
```

Out[91]:

	id	name	price	bonus	total
0	1	muji	300	30	330
1	2	zaraq	400	40	440
2	3	haroon	2300	30	2330
3	4	NaN	500	20	520

Column Deletion

Deletion in column is used to delete columns from table

In [84]:

```
import pandas as pd
data ={'id':pd.Series([1,2,3,4]),
       'name':pd.Series(['muji','zaraq','haroon']),
       'gender':pd.Series(['male','male','male','male'])}

data1=pd.DataFrame(data)

del data1['gender']
data1
```

Out[84]:

	id	name
0	1	muji
1	2	zaraq
2	3	haroon
3	4	NaN

In [87]:

```
import pandas as pd
data ={'id':pd.Series([1,2,3,4]),
       'name':pd.Series(['muji','zaraq','haroon']),
       'gender':pd.Series(['male','male','male','male'])}
```

```
data1=pd.DataFrame(data)

data1.pop('gender')
data1
```

Out[87]:

	id	name
0	1	muji
1	2	zaraq
2	3	haroon
3	4	NaN

11.7 Indexing / Selection

The Python and NumPy indexing operators [] and attribute operator '.' (dot) provide quick and easy access to pandas data structures across a wide range of use cases. The index is like an address, that's how any data point across the data frame or series can be accessed. Rows and columns both have indexes.

.loc()

Pandas provide various methods to have purely label based indexing.

Integers are valid labels, but they refer to the label and not the position.

It has multiple access methods-

- A single scalar label
- A list of labels
- A slice object
- A Boolean array

In [111…

```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.randn(8, 4),
```

```
index = ['a','b','c','d','e','f','g','h'], columns = ['A', 'B', 'C', 'D'])
df.loc['a']
```

Out[111…

```
A    0.360374
B   -0.933167
C    1.114144
D   -0.891083
Name: a, dtype: float64
```

In [103…

```
import pandas as pd
data ={'id':pd.Series([1,2,3,4]),
       'name':pd.Series(['muji','zaraq','haroon']),
       'gender':pd.Series(['male','male','male','male'])}

data1=pd.DataFrame(data)
data1.loc[0]
```

Out[103…

```
id           1
name      muji
gender    male
Name: 0, dtype: object
```

In [107…

```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.randn(8, 4),
index = ['a','b','c','d','e','f','g','h'], columns = ['A', 'B', 'C', 'D'])
df.loc[:,['A','C']]
```

Out[107…

	A	C
a	-0.805718	-0.004676
b	-1.201062	2.178751
c	-1.356213	-0.262519
d	-0.253268	-2.142808
e	-0.495735	-1.706757
f	-1.152883	-0.741921
g	-0.196683	1.171625

	A	C
h	-1.611295	-1.263552

In [109…

```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.randn(8, 4),
index = ['a','b','c','d','e','f','g','h'], columns = ['A', 'B', 'C', 'D'])
df.loc[['a','b','f','h'],['A','C']]
```

Out[109…

	A	C
a	-2.425232	-0.492693
b	1.629423	0.871836
f	-0.723004	-0.061604
h	-0.112835	-0.906768

In [118…

```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.randn(8, 4),
index = ['a','b','c','d','e','f','g','h'], columns = ['A', 'B', 'C', 'D'])
df.loc['a':'d']
```

Out[118…

	A	B	C	D
a	0.168226	-2.926654	-0.236836	-0.253244
b	-0.481968	-1.047359	-0.594904	0.977006
c	-0.080683	-1.941215	0.139479	0.463931
d	-0.403353	-0.681330	-0.144203	1.004590

.iloc()

Pandas provide various methods in order to get purely integer based indexing. Like python and numpy, these are 0-based indexing.

The various access methods are as follows −

- An Integer
- A list of integers
- A range of values

In [128…
```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.randn(6, 3), columns = ['A', 'B', 'C'])
df.iloc[:4]
```

Out[128…

	A	B	C
0	0.292067	0.934936	1.134429
1	-1.590198	0.532974	-1.146146
2	0.674859	2.831381	1.474639
3	0.769699	0.758830	-0.236846

In [130…
```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.randn(8, 4), columns = ['A', 'B', 'C', 'D'])
df.iloc[1:5, 2:4]
```

Out[130…

	C	D
1	0.074420	0.282037
2	0.216386	0.839305
3	-1.828433	-0.403290
4	-1.295635	-0.621461

11.8 Data alignment and arithmetic

In [134…
```
import pandas as pd
```

```
data ={'id':pd.Series([1,2,3,4]),
       'name':pd.Series(['muji','zaraq','haroon'])}

data1=pd.DataFrame(data)
data1['price']=pd.Series([300,400,2300,500])
data1['bonus']=pd.Series([30,40,30,20])

data1['total']=data1['price']+data1['bonus']
data1
```

Out[134…

	id	name	price	bonus	total
0	1	muji	300	30	330
1	2	zaraq	400	40	440
2	3	haroon	2300	30	2330
3	4	NaN	500	20	520

In [137…

```
data ={'id':pd.Series([1,2,3,4]),
       'name':pd.Series(['muji','zaraq','haroon','Ali'])}

data1 ={'id':pd.Series([1,2,3]),
       'name':pd.Series(['muji','zaraq','haroon'])}
pf1=pd.DataFrame(data)
pf2=pd.DataFrame(data1)
pf1+pf2
```

Out[137…

	id	name
0	2.0	mujimuji
1	4.0	zaraqzaraq
2	6.0	haroonharoon
3	NaN	NaN

In [185…

```
import pandas as pd
import numpy as np

data={
```

```
    'price': ([34,565,443,345,45,5,3,23]),
    'bonus': ([4,55,43,33,56,67,7,32]),
    'Deduction': ([32,50,41,33,5,67,12,21]),
    'Total': ([3,5,4,3,5,71,7,8])
}
df=pd.DataFrame(data)
df-df.loc[0]
```

Out[185…

	price	bonus	Deduction	Total
0	0	0	0	0
1	531	51	18	2
2	409	39	9	1
3	311	29	1	0
4	11	52	-27	2
5	-29	63	35	68
6	-31	3	-20	4
7	-11	28	-11	5

In [184…

```
import pandas as pd
import numpy as np

data={
    'price': ([34,565,443,345,45,5,3,23]),
    'bonus': ([4,55,43,33,56,67,7,32]),
    'Deduction': ([32,50,41,33,5,67,12,21]),
    'Total': ([3,5,4,3,5,71,7,8])
}
df=pd.DataFrame(data)
df *2
```

Out[184…

	price	bonus	Deduction	Total
0	68	8	64	6
1	1130	110	100	10
2	886	86	82	8

	price	bonus	Deduction	Total
3	690	66	66	6
4	90	112	10	10
5	10	134	134	142
6	6	14	24	14
7	46	64	42	16

In [183…

```
import pandas as pd
import numpy as np

data={
    'price': ([34,565,443,345,45,5,3,23]),
    'bonus': ([4,55,43,33,56,67,7,32]),
    'Deduction': ([32,50,41,33,5,67,12,21]),
    'Total': ([3,5,4,3,5,71,7,8])
}
df=pd.DataFrame(data)
df *10-20
```

Out[183…

	price	bonus	Deduction	Total
0	320	20	300	10
1	5630	530	480	30
2	4410	410	390	20
3	3430	310	310	10
4	430	540	30	30
5	30	650	650	690
6	10	50	100	50
7	210	300	190	60

In [182…

```
import pandas as pd
import numpy as np
```

```
data={
    'price': ([34,565,443,345,45,5,3,23]),
    'bonus': ([4,55,43,33,56,67,7,32]),
    'Deduction': ([32,50,41,33,5,67,12,21]),
    'Total': ([3,5,4,3,5,71,7,8])
}
df=pd.DataFrame(data)
df /2
```

Out[182…

	price	bonus	Deduction	Total
0	17.0	2.0	16.0	1.5
1	282.5	27.5	25.0	2.5
2	221.5	21.5	20.5	2.0
3	172.5	16.5	16.5	1.5
4	22.5	28.0	2.5	2.5
5	2.5	33.5	33.5	35.5
6	1.5	3.5	6.0	3.5
7	11.5	16.0	10.5	4.0

In [181…

```
import pandas as pd
import numpy as np

data={
    'price': ([34,565,443,345,45,5,3,23]),
    'bonus': ([4,55,43,33,56,67,7,32]),
    'Deduction': ([32,50,41,33,5,67,12,21]),
    'Total': ([3,5,4,3,5,71,7,8])
}
df=pd.DataFrame(data)
df **2
```

Out[181…

	price	bonus	Deduction	Total
0	1156	16	1024	9

	price	bonus	Deduction	Total
1	319225	3025	2500	25
2	196249	1849	1681	16
3	119025	1089	1089	9
4	2025	3136	25	25
5	25	4489	4489	5041
6	9	49	144	49
7	529	1024	441	64

In [190…
```
df1 = pd.DataFrame({'a' : [1, 0, 1,1], 'b' : [0, 1, 1,1] }, dtype=bool)
df1
```

Out[190…

	a	b
0	True	False
1	False	True
2	True	True
3	True	True

In [191…
```
df1 = pd.DataFrame({'a' : [1, 0, 1,1], 'b' : [0, 1, 1,0] }, dtype=bool)
df1['a'] & df1['b']
```

Out[191…
```
0    False
1    False
2     True
3    False
dtype: bool
```

In [192…
```
df1 = pd.DataFrame({'a' : [1, 0, 1,1], 'b' : [0, 1, 1,0] }, dtype=bool)
df1['a'] | df1['b']
```

Out[192…
```
0    True
1    True
2    True
```

```
3    True
dtype: bool
```

In [193…
```
df1 = pd.DataFrame({'a' : [1, 0, 1,1], 'b' : [0, 1, 1,0] }, dtype=bool)
-df1
```

Out[193…

	a	b
0	False	True
1	True	False
2	False	False
3	False	True

11.9 Transposing

To convert row into column and column into row is called transposing.

Tramspose of whole table.

In [194…
```
import pandas as pd
import numpy as np

data={
    'price': ([34,565,443,345,45,5,3,23]),
    'bonus': ([4,55,43,33,56,67,7,32]),
    'Deduction': ([32,50,41,33,5,67,12,21]),
    'Total': ([3,5,4,3,5,71,7,8])
}
df=pd.DataFrame(data)
df.T
```

Out[194…

	0	1	2	3	4	5	6	7
price	34	565	443	345	45	5	3	23
bonus	4	55	43	33	56	67	7	32
Deduction	32	50	41	33	5	67	12	21
Total	3	5	4	3	5	71	7	8

Transpose of slicing table.

In [196…
```
import pandas as pd
import numpy as np

data={
    'price': ([34,565,443,345,45,5,3,23]),
    'bonus': ([4,55,43,33,56,67,7,32]),
    'Deduction': ([32,50,41,33,5,67,12,21]),
    'Total': ([3,5,4,3,5,71,7,8])
}
df=pd.DataFrame(data)
df[:4].T
```

Out[196…

	0	1	2	3
price	34	565	443	345
bonus	4	55	43	33
Deduction	32	50	41	33
Total	3	5	4	3

In [197…
```
import pandas as pd
import numpy as np

data={
    'price': ([34,565,443,345,45,5,3,23]),
    'bonus': ([4,55,43,33,56,67,7,32]),
    'Deduction': ([32,50,41,33,5,67,12,21]),
    'Total': ([3,5,4,3,5,71,7,8])
}
df=pd.DataFrame(data)
df[2:6].T
```

Out[197…

	2	3	4	5
price	443	345	45	5
bonus	43	33	56	67
Deduction	41	33	5	67

	2	3	4	5
Total	4	3	5	71

In [205…

```
dates = pd.date_range('20210101', periods=6)
dates
data={
    'price': ([34,565,443,345,45,5]),
    'bonus': ([4,55,43,33,56,32]),
    'Deduction': ([32,33,5,67,12,21]),
    'Total': ([3,5,4,3,7,8])
}
df = pd.DataFrame(data, index=dates)
df
```

Out[205…

	price	bonus	Deduction	Total
2021-01-01	34	4	32	3
2021-01-02	565	55	33	5
2021-01-03	443	43	5	4
2021-01-04	345	33	67	3
2021-01-05	45	56	12	7
2021-01-06	5	32	21	8

12. Viewing Data

We can view data / display data in different ways: See the top & bottom rows of the frame Selecting a single column Selecting via [], which slices the rows For getting a cross section using a label Selecting on a multi-axis by labe Showing label slicing, both endpoints are included Reduction in the dimensions of the returned object For getting a scalar value For getting fast access to a scalar Select via the position of the passed integers By integer slices, acting similar to numpy/python By lists of integer position locations, similar to the numpy/python style For slicing rows explicitly For slicing columns explicitly For getting a value explicitly For getting fast access to a scalar Using a single column’s values to select data. Selecting values from a DataFrame where a boolean condition is met. Using the isin() method for filtering

head() function get first 5 rows from table

In [207…

```
dates = pd.date_range('20210101', periods=6)
dates
data={
    'price': ([34,565,443,345,45,5]),
    'bonus': ([4,55,43,33,56,32]),
    'Deduction': ([32,33,5,67,12,21]),
    'Total': ([3,5,4,3,7,8])
}
df = pd.DataFrame(data, index=dates)
df.head()
```

Out[207…

	price	bonus	Deduction	Total
2021-01-01	34	4	32	3
2021-01-02	565	55	33	5
2021-01-03	443	43	5	4
2021-01-04	345	33	67	3
2021-01-05	45	56	12	7

tail() function get last rows from table

In [210…

```
dates = pd.date_range('20210101', periods=6)
dates
data={
    'price': ([34,565,443,345,45,5]),
    'bonus': ([4,55,43,33,56,32]),
    'Deduction': ([32,33,5,67,12,21]),
    'Total': ([3,5,4,3,7,8])
}
df = pd.DataFrame(data, index=dates)
df.tail(4)
```

Out[210…

	price	bonus	Deduction	Total
2021-01-03	443	43	5	4
2021-01-04	345	33	67	3
2021-01-05	45	56	12	7

	price	bonus	Deduction	Total
2021-01-06	5	32	21	8

Index

get indexes from table record

In [213…
```
dates = pd.date_range('20210101', periods=6)
dates
data={
    'price': ([34,565,443,345,45,5]),
    'bonus': ([4,55,43,33,56,32]),
    'Deduction': ([32,33,5,67,12,21]),
    'Total': ([3,5,4,3,7,8])
}
df = pd.DataFrame(data, index=dates)
df.index
```

Out[213…
```
DatetimeIndex(['2021-01-01', '2021-01-02', '2021-01-03', '2021-01-04',
               '2021-01-05', '2021-01-06'],
              dtype='datetime64[ns]', freq='D')
```

Columns

get column name of DataFrame

In [215…
```
dates = pd.date_range('20210101', periods=6)
dates
data={
    'price': ([34,565,443,345,45,5]),
    'bonus': ([4,55,43,33,56,32]),
    'Deduction': ([32,33,5,67,12,21]),
    'Total': ([3,5,4,3,7,8])
}
df = pd.DataFrame(data, index=dates)
df.columns
```

Out[215…
```
Index(['price', 'bonus', 'Deduction', 'Total'], dtype='object')
```

values

get values of cells

In [216…
```
dates = pd.date_range('20210101', periods=6)
dates
data={
    'price': ([34,565,443,345,45,5]),
    'bonus': ([4,55,43,33,56,32]),
    'Deduction': ([32,33,5,67,12,21]),
    'Total': ([3,5,4,3,7,8])
}
df = pd.DataFrame(data, index=dates)
df.values
```

Out[216…
```
array([[ 34,   4,  32,   3],
       [565,  55,  33,   5],
       [443,  43,   5,   4],
       [345,  33,  67,   3],
       [ 45,  56,  12,   7],
       [  5,  32,  21,   8]], dtype=int64)
```

Sorting

In [217…
```
dates = pd.date_range('20210101', periods=6)
dates
data={
    'price': ([34,565,443,345,45,5]),
    'bonus': ([4,55,43,33,56,32]),
    'Deduction': ([32,33,5,67,12,21]),
    'Total': ([3,5,4,3,7,8])
}
df = pd.DataFrame(data, index=dates)
df.sort_index(axis=1,ascending=False)
```

Out[217…

	price	bonus	Total	Deduction
2021-01-01	34	4	3	32
2021-01-02	565	55	5	33
2021-01-03	443	43	4	5
2021-01-04	345	33	3	67
2021-01-05	45	56	7	12

	price	bonus	Total	Deduction
2021-01-06	5	32	8	21

In [232…

```
dates = pd.date_range('20210101', periods=6)
dates
data={
    'MCS': (['muji','zaraq','haroon','Ali','aiman','aiza']),
    'MIT': (['danish','rashid','asif','ramish','ariba','aima']),
    'BCS': (['rehan','sultan','moiz','kaleem','aliza','rahool']),
    'BS': (['muji','zaraq','haroon','Ali','mannan','raheel'])
}
df = pd.DataFrame(data, index=dates)
df.sort_values(by='MCS')
```

Out[232…

	MCS	MIT	BCS	BS
2021-01-04	Ali	ramish	kaleem	Ali
2021-01-05	aiman	ariba	aliza	mannan
2021-01-06	aiza	aima	rahool	raheel
2021-01-03	haroon	asif	moiz	haroon
2021-01-01	muji	danish	rehan	muji
2021-01-02	zaraq	rashid	sultan	zaraq

In [234…

```
dates = pd.date_range('20210101', periods=6)
dates
data={
    'MCS': (['muji','zaraq','haroon','Ali','aiman','aiza']),
    'MIT': (['danish','rashid','asif','ramish','ariba','aima']),
    'BCS': (['rehan','sultan','moiz','kaleem','aliza','rahool']),
    'BS': (['muji','zaraq','haroon','Ali','mannan','raheel'])
}
df = pd.DataFrame(data, index=dates)
df.describe()
```

Out[234…

	MCS	MIT	BCS	BS

	MCS	MIT	BCS	BS
count	6	6	6	6
unique	6	6	6	6
top	aiman	ramish	sultan	raheel
freq	1	1	1	1

.loc()

Pandas provide various methods to have purely label based indexing.

Integers are valid labels, but they refer to the label and not the position.

It has multiple access methods

In [236…
```
import pandas as pd
data ={'id':pd.Series([1,2,3,4]),
       'name':pd.Series(['muji','zaraq','haroon']),
       'gender':pd.Series(['male','male','male','male'])}

data1=pd.DataFrame(data)
data1.loc[0]
```

Out[236…
```
id           1
name      muji
gender    male
Name: 0, dtype: object
```

In [237…
```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.randn(8, 4),
index = ['a','b','c','d','e','f','g','h'], columns = ['A', 'B', 'C', 'D'])
df.loc[:,['A','C']]
```

Out[237…

	A	C
a	0.271017	-1.080753

	A	C
b	-1.136609	-0.243522
c	0.321763	0.339953
d	-0.459563	-0.582438
e	0.696101	0.418218
f	0.430632	0.724241
g	-0.728226	-0.140671
h	-0.457201	1.830285

In [239…

```
dates = pd.date_range('20210101', periods=6)
dates
data={
    'MCS': (['muji','zaraq','haroon','Ali','aiman','aiza']),
    'MIT': (['danish','rashid','asif','ramish','ariba','aima']),
    'BCS': (['rehan','sultan','moiz','kaleem','aliza','rahool']),
    'BS': (['muji','zaraq','haroon','Ali','mannan','raheel'])
}
df = pd.DataFrame(data, index=dates)
df.loc['20210102':'20210104',['MCS','MIT']]
```

Out[239…

	MCS	MIT
2021-01-02	zaraq	rashid
2021-01-03	haroon	asif
2021-01-04	Ali	ramish

In [240…

```
dates = pd.date_range('20210101', periods=6)
dates
data={
    'MCS': (['muji','zaraq','haroon','Ali','aiman','aiza']),
    'MIT': (['danish','rashid','asif','ramish','ariba','aima']),
    'BCS': (['rehan','sultan','moiz','kaleem','aliza','rahool']),
    'BS': (['muji','zaraq','haroon','Ali','mannan','raheel'])
}
```

```
df = pd.DataFrame(data, index=dates)
df.loc['20210104',['MCS','MIT']]
```

Out[240…
```
MCS       Ali
MIT    ramish
Name: 2021-01-04 00:00:00, dtype: object
```

In [245…
```
dates = pd.date_range('20210101', periods=6)
dates
data={
    'MCS': (['muji','zaraq','haroon','Ali','aiman','aiza']),
    'MIT': (['danish','rashid','asif','ramish','ariba','aima']),
    'BCS': (['rehan','sultan','moiz','kaleem','aliza','rahool']),
    'BS': (['muji','zaraq','haroon','Ali','mannan','raheel'])
}
df = pd.DataFrame(data, index=dates)
df.at[dates[0],'MIT']
```

Out[245…
```
'danish'
```

In [246…
```
data={
    'MCS': (['muji','zaraq','haroon','Ali','aiman','aiza']),
    'MIT': (['danish','rashid','asif','ramish','ariba','aima']),
    'BCS': (['rehan','sultan','moiz','kaleem','aliza','rahool']),
    'BS': (['muji','zaraq','haroon','Ali','mannan','raheel'])
}
df = pd.DataFrame(data)
df.loc[1:3]
```

Out[246…

	MCS	MIT	BCS	BS
1	zaraq	rashid	sultan	zaraq
2	haroon	asif	moiz	haroon
3	Ali	ramish	kaleem	Ali

In [256…
```
data={
    'MCS': (['muji','zaraq','haroon','Ali','aiman','aiza']),
    'MIT': (['danish','rashid','asif','ramish','ariba','aima']),
    'BCS': (['rehan','sultan','moiz','kaleem','aliza','rahool']),
    'BS': (['muji','zaraq','haroon','Ali','mannan','raheel'])
```

```
}
df = pd.DataFrame(data)
df.iloc[:,1:2]
```

Out[256…

	MIT
0	danish
1	rashid
2	asif
3	ramish
4	ariba
5	aima

In [257…

```
data={
    'MCS': (['muji','zaraq','haroon','Ali','aiman','aiza']),
    'MIT': (['danish','rashid','asif','ramish','ariba','aima']),
    'BCS': (['rehan','sultan','moiz','kaleem','aliza','rahool']),
    'BS': (['muji','zaraq','haroon','Ali','mannan','raheel'])
}
df = pd.DataFrame(data)
df.iloc[1:4,1:2]
```

Out[257…

	MIT
1	rashid
2	asif
3	ramish

In [261…

```
import pandas as pd
data={
    'MCS': (['muji','zaraq','haroon','Ali','aiman','aiza']),
    'MIT': (['danish','rashid','asif','ramish','ariba','aima']),
    'BCS': (['rehan','sultan','moiz','kaleem','aliza','rahool']),
    'BS': (['muji','zaraq','haroon','Ali','mannan','raheel'])
}
df = pd.DataFrame(data)
```

```
df2=df.copy()
df2
```

Out[261…

	MCS	MIT	BCS	BS
0	muji	danish	rehan	muji
1	zaraq	rashid	sultan	zaraq
2	haroon	asif	moiz	haroon
3	Ali	ramish	kaleem	Ali
4	aiman	ariba	aliza	mannan
5	aiza	aima	rahool	raheel

In [ ]: