Day_01 Datatypes(Variables & Numbers)

Variables

Definition

A characteristic, number, or quantity that increases or decreases over time, or takes different values in different situations.

Two basic types

Independent variable: that can take different values and can cause corresponding changes in other variables.

Dependent variable: that can take different values only in response to an independent variable.

Purpose

Variables are used to store information to be referenced and manipulated in a computer program. They also provide a way of labeling data with a descriptive name, so our programs can be understood more clearly by the reader and ourselves. It is helpful to think of variables as containers that hold information.

Importance

Variables play an important role in computer programming because they enable programmers to write flexible programs. Rather than entering data directly into a program, a programmer can use variables to represent the data. The opposite of a variable is a constant. Constants are values that never change.

Application

Variables are used in any computer program they are very usefull as they help to store many of the information such as names,list even the complete functions.

They are used in all of the Systems such as "Library Management Sytem","Database Management Sytem

Strengths

The greatest advantage of the variables is that they enable one and the same program to execute various sets of data. In the light of the afore-stated, a variable refers to a symbol for a varying value, which is stored in the system's memory.

They helps us in stoping the code repeation

They helps us better manage our code

If you use variables, you can make your code more clean and simple to you and others understand. Also, it's good practice.

I think you can add the variables you declare inside multiple functions. Whereas yours is a one time use scenario. I could be wrong though... Variables are used for storing data though, and then you can carry that variable over from one function to another.

Weakness

The Biggest weakness of using hte variables in the python is that if a variable of string type is defined in a program and and you mistankkenly or forget taht you have already initalize that variable and you save the integer datatype in the variable the variable property will change and it will become the integer variable instead of string type.

Example 01 :

Task:

Create a Variable of String Type and save your Name inside it and print the Name and show that The datatype of the variables in the Python can be changed just by assigning value of different datatype to the variable

In [37]:
```
# Code
Name="Muhammad Umair"
print(Name)
print(type(Name))
Name=5
print(Name)
print(type(Name))
#Output
```
```
Muhammad Umair
<class 'str'>
5
<class 'int'>
```

Example 02 :

Task:

Save your personal information into the four variables and print there value

In [38]:
```
# Code
age=20
weight=65.2
name="Muhammad Umair"
fatherName="Muhammad Akram"
```
In [39]:
```
# OutPut
print(name)
print(age)
print(fatherName)
print(weight)
```
```
Muhammad Umair
20
Muhammad Akram
65.2
```

Operator

Definition

An operator in a programming language is a symbol that tells the compiler or interpreter to perform specific mathematical, relational or logical operation and produce final result.

Purpose

They are used to do many of the basic mathematical Computaion

Importance

Operators are very usefull in the programming language as they help us perform so many mathematical computaion and also the logival operators such as AND OR NOT they help us make the decisions onthe basis of certian condition

Application

There are numbers of applications of the opertors such as calculator program,computer program used in the banks to do computing on the money to calculate all the ups and downs in the money

Strengths

They helps us by reducing so much of our work as we cannot image how much of asembly language code we will have to write to simply add two numbers and it would be so difficult to do a AND,OR,NOT function

Weaknesses

As they are simple math functions there are not any waekness of them. The only weakness of the them is how we use them if our own math or computation is wrong then they will perform wrong

Suitable to Use

They are suitable to use in any senario where you want to use the mathematical computaion or logical computation

Example 03 :

Task:

Take four variables of integer type and apply math operations on them

In [40]:
```
# Code
numberOne=10
numberTwo=20
numberThree=30.5
numberFour=6.7
```
In [41]:
```
numberOne+numberTwo #Adding or subtracting two integer variables will return Integer
```
Out[41]: `30`

In [42]:
```
numberOne-numberTwo
```
Out[42]: `-10`

In [43]:
```
numberOne+numberThree #Adding One Integer and one float type variables will return the float
```
Out[43]: `40.5`

In [44]:
```
numberThree*numberFour #Multipling two variables of same datatype will return same Datatype
```
Out[44]: `204.35`

In [45]:
```
numberOne**2 #for exponent use baseNumber**exponent will return the exponent of base number to the power
#for example 10^2=100
```
Out[45]: `100`

Day_02 Datatypes(String & List & Dictionaries & Tuples)

Strings

Definition

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. Many Python methods, such as replace(), join(), or split() modify strings. However, they do not modify the original string. They create a copy of a string which they modify and return to the caller.

Purpose

string is a data type used in programming, such as an integer and floating point unit, but is used to represent text rather than numbers. It is comprised of a set of characters that can also contain spaces and numbers. For example, the word "hamburger" and the phrase "I ate 3 hamburgers" are both strings

Importance

As with any programming language, strings are one of the most important things to know about Python. Also as we have experienced in the other languages so far, strings contain characters. Strings are not picky by any means. They can contain almost anything if used properly. The are also not picky by the amount of characters you put in them.

Application

Strings are so widely used component in the any programming language that thaere are so many application of it in the programmings.Like in any program with text data strings are used like in database to store names,addresses and so much.

Strengths

Compilation creates unique strings. At compile time, strings are resolved as far as possible. This includes applying the concatenation operator and converting other literals to strings. So “hi7” and (“hi”+7) both get resolved at compile time to the same string and are identical objects in the class string pool. Compilers differ in their ability to achieve this resolution. You can always check your compiler (e.g., by decompiling some statements involving concatenation) and change it if needed.

Because String objects are immutable, a substring operation doesn’t need to copy the entire underlying sequence of characters. Instead, a substring can use the same char array as the original string and simply refer to a different start point and endpoint in the char array. This means that substring operations are efficient, being both fast and conserving of memory; the extra object is just a wrapper on the same underlying char array with different pointers into that array.

Strings are implemented in the JDK as an internal char array with index offsets (actually a start offset and a character count). This basic structure is extremely unlikely to be changed in any version of Java.

Strings have strong support for internationalization. It would take a large effort to reproduce the internationalization support for an alternative class.

The close relationship with StringBuffers allows Strings to reference the same char array used by the StringBuffer. This is a double-edged sword. For typical practice, when you use a StringBuffer to manipulate and append characters and data types and then convert the final result to a String, this works just fine.

The StringBuffer provides efficient mechanisms for growing, inserting, appending, altering, and other types of String manipulation. The resulting String then efficiently references the same char array with no extra character copying. This is very fast and reduces the number of objects being used to a minimum by avoiding intermediate objects. However, if the StringBuffer object is subsequently altered, the char array in that StringBuffer is copied into a new char array that is now referenced by the StringBuffer.

The String object retains the reference to the previously shared char array. This means that copying overhead can occur at unexpected points in the application. Instead of the copying occurring at the toString( ) method call, as might be expected, any subsequent alteration of the StringBuffer causes a new char array to be created and an array copy to be performed. To make the copying overhead occur at predictable times, you could explicitly execute some method that makes the copying occur, such as StringBuffer.setLength( ). This allows StringBuffers to be reused with more predictable performance.

Weakness

Not being able to subclass String means that it is not possible to add behavior to String for your own needs.

The previous point means that all access must be through the restricted set of currently available String methods, imposing extra overhead.

The only way to increase the number of methods allowing efficient manipulation of String characters is to copy the characters into your own array and manipulate them directly, in which case String is imposing an extra step and extra objects you may not need.

char arrays are faster to process directly.

The tight coupling with StringBuffer can lead to unexpectedly high memory usage. When StringBuffer.toString( ) creates a String, the current underlying array holds the string, regardless of the size of the array (i.e., the capacity of the StringBuffer).

For example, a StringBuffer with a capacity of 10,000 characters can build a string of 10 characters. However, 10-character String continues to use a 10,000-char array to store the 10 characters. If the StringBuffer is now reused to create another 10-character string, the StringBuffer first creates a new internal 10,000-char array to build the string with; then the new String also uses that 10,000-char array to store the 10 characters. Obviously, this process can continue indefinitely, using vast amounts of memory were not expected.

Example 04 :

Task:

Take three variables of String type and store information in them and print them

In [46]:
```
# Code
firstName="Muhammad"
lastName="Umair"
university="Comsats Lahore"
```
In [47]:
```
# Output
print(firstName,lastName,university)
```
```
Muhammad Umair Comsats Lahore
```

Example 05 :

Task:

Use Slicing to print a substring from the string and print it

In [48]:
```
# Code
fullName="Muhammad Umair Akram"
firstName=fullName[0:8]
middleName=fullName[9:14]
lastName=fullName[15:20]
# Output
print(firstName,middleName,lastName)
```
```
Muhammad Umair Akram
```

Example 06 :

Task:

Use + Operator to add(join) two strings togeather and print the output

In [49]:
```
greetingsStr="Hy I am "+middleName+".Nice to meet You"
print(greetingsStr)
```
```
Hy I am Umair.Nice to meet You
```
In [50]:
```
brotherName=fullName[:9]+"Usama Akram"
```
In [51]:
```
print(brotherName)
```
```
Muhammad Usama Akram
```

Example 07 :

Task:

Use the del operator to delete the string and check if the string is realy deleted

In [52]:
```
del greetingsStr
```
In [53]:
```
print(greetingsStr)
```
```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-53-e9c047401d1c> in <module>
----> 1 print(greetingsStr)

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

Example 08 :

Task:

use the formated string(use + to join the two strings in print function and print them)

In [54]:
```
print("Hello "+middleName+" How are You?")
```
```
Hello Umair How are You?
```
In [55]:
```
newStr="I Love Pakistan"
print(((newStr+" "+"\n")*10))
```
```
I Love Pakistan
I Love Pakistan
I Love Pakistan
I Love Pakistan
I Love Pakistan
I Love Pakistan
I Love Pakistan
I Love Pakistan
I Love Pakistan
I Love Pakistan
```

Example 09 :

Task:

Use slicing to print chunks from the string

In [56]:
```
print(newStr[7:])
```
```
Pakistan
```
In [57]:
```
print(newStr[7])
```
```
P
```
In [58]:
```
print((newStr[2:6]+"\t")*5)
```
```
Love    Love    Love    Love    Love
```

Example 10 :

Task:

Use use in operator to serach that specified string in the complete string

In [59]:
```
print("Umair" in fullName)
```
```
True
```
In [60]:
```
print("Usama" in fullName)
```
```
False
```
In [61]:
```
print("Usama" not in fullName)
```
```
True
```

Example 11 :

Task:

Use use String Formator to print the formated string using print function

In [62]:
```
print("I am %s. I am %d years old. I love to eat Meat"%(name,age))
```
```
I am Muhammad Umair. I am 20 years old. I love to eat Meat
```
In [63]:
```
print("I am {name}. I am {age} years old. I love to eat Biryani".format(name=name,age=age))
```
```
I am Muhammad Umair. I am 20 years old. I love to eat Biryani
```

Lists

Difinition

A list is a data structure in Python that is a mutable, or changeable, ordered sequence of elements. Each element or value that is inside of a list is called an item. Just as strings are defined as characters between quotes, lists are defined by having values between square brackets [ ]

Purpose

They are used to store an ordered collection of items, which might be of different types but usually they aren’t. Commas separate the elements that are contained within a list and enclosed in square brackets

Importance

Lists are just like the arrays, declared in other languages. Lists need not be homogeneous always which makes it a most powerful tool in Python. In Python, list is a type of container in Data Structures, which is used to store multiple data at the same time. Unlike Sets, the list in Python are ordered and have a definite count. The elements in a list are indexed according to a definite sequence and the indexing of a list is done with 0 being the first index. Each element in the list has its definite place in the list, which allows duplicating of elements in the list, with each element having its own distinct place and credibility.

The list is a most versatile datatype available in Python which can be written as a list of comma-separated values (items) between square brackets.Important thing about a list is that items in a list need not be of the same type.List is majorly used with dictionaries when there is large number of data.

Applications

Due to the verstality of hte lists in python they are widely used in puthon to hold huge ammount of data of different datatypes.

Such as there can be the list of lists,tuple,dictionaries.They are used to hold image type data after converting them.

Strengths

Add/remove any objects/items to/from it.Mutable i.e. you can change the items anytime you like

Lists are numerically keyed and can be sorted and have values removed or added.

Lists are mutable. Tuples and Dictionaries are immutable so their values are fixed. There can be no precise explanation to your question since the real question is what do you want your program to do? If you want to collect values and store them somewhere then Lists is the solution you are searching for, since they can be mutated. If you want to get values from something like a database then Tuples or Dictionaries might be a good solution. Dictionaries give you the perk of a key usage also. So it all depends on what you want to achieve.

Weakness

You cannot use a list as a key to a dictoinary because it is mutable which refers to the fact that we want a key to be a constant (non-changing entity).

Example 12 :

Task:

make the list of names and numbers and print them

In [64]:
```
namesList=["Umair","Usama","Akram","Bilal","Abubakar","Umar","Ali"]
rollNoList=[1,2,3,4,5,6,7]
print(namesList[0],rollNoList[0])
print(namesList[0:7],rollNoList[0:7])
print(namesList[5:7],rollNoList[5:7])
print(namesList)
```
```
Umair 1
['Umair', 'Usama', 'Akram', 'Bilal', 'Abubakar', 'Umar', 'Ali'] [1, 2, 3, 4, 5, 6, 7]
['Umar', 'Ali'] [6, 7]
['Umair', 'Usama', 'Akram', 'Bilal', 'Abubakar', 'Umar', 'Ali']
```
In [65]:
```
print(rollNoList)
```
```
[1, 2, 3, 4, 5, 6, 7]
```

Example 13 :

Task:

Append the list or add new string into the list using append function print the output

In [66]:
```
namesList.append("Usman")
```
In [67]:
```
print(namesList)
```
```
['Umair', 'Usama', 'Akram', 'Bilal', 'Abubakar', 'Umar', 'Ali', 'Usman']
```

Example 14 :

Task:

Delete an Element from the list using del keyword print and check the output

In [68]:
```
del namesList[3]
```
In [69]:
```
print(namesList)
```
```
['Umair', 'Usama', 'Akram', 'Abubakar', 'Umar', 'Ali', 'Usman']
```

Example 15 :

Task:

Remove the Element from the list using remove function of list

In [70]:
```
rollNoList.remove(4)
```
In [71]:
```
print(namesList+rollNoList)
```
```
['Umair', 'Usama', 'Akram', 'Abubakar', 'Umar', 'Ali', 'Usman', 1, 2, 3, 5, 6, 7]
```

Dictionaries

Definition

Dictionaries are Python's implementation of a data structure that is more generally known as an associative array. A dictionary consists of a collection of key-value pairs. Each key-value pair maps the key to its associated value.

Purpose

Dictionary in Python is an unordered collection of data values, used to store data values like a map, which unlike other Data Types that hold only single value as an element, Dictionary holds key:value pair. Key value is provided in the dictionary to make it more optimized

Importance

The Python dictionary makes it easier to read and change data, thereby rendering it more actionable for predictive modeling. A Python dictionary is an unordered collection of data values. Unlike other data types that hold only one value as an element, a Python dictionary holds a key: value pair

Strenghts of Dictionaries

The Python dictionary makes it easier to read and change data, thereby rendering it more actionable for predictive modeling. A Python dictionary is an unordered collection of data values. Unlike other data types that hold only one value as an element, a Python dictionary holds a key: value pair.

It improves the readability of your code. Writing out Python dictionary keys along with values adds a layer of documentation to the code. If the code is more streamlined, it is a lot easier to debug. Ultimately, analyses get done a lot quicker and models can be fitted more efficiently.

Apart from readability, there’s also the question of sheer speed. You can look up a key in a Python dictionary very fast. The speed of a task like looking up keys is measured by looking at how many operations it takes to finish. Looking up a key is done in constant time vis-a-vis looking up an item in a large list which is done in linear time.

To look up an item in a huge list, the computer will look through every item in the list. If every item is assigned a key-value pair then you only need to look for the key which makes the entire process much faster. A Python dictionary is basically an implementation of a hash table. Therefore, it hs all the benefits of the hashtable which include membership checks and speedy tasks like looking up keys.

Weakness

Dictionaries are unordered. In cases where the order of the data is important, the Python dictionary is not appropriate.

Python dictionaries take up a lot more space than other data structures. The amount of space occupied increases drastically when there are many Python Dictionary keys. Of course, this isn’t too much of a disadvantage because memory isn’t very expensive.

Example 16 :

Task:

Create an Dictionary and print different values from it delete an element from dictionary using del keyword and add new element in the dictionary

In [72]:
```
studentsDict={'Student1':{'name':"Muhammad Umair",'age':20,'Department':"BSE",'Semester':6},
              'Student2':{'name':"Hashim Shakoor",'age':22,'Department':"BSE",'Semester':6},
             'Student3':{'name':"Muhammad Abdullah Tahir",'age':20,'Department':"BSE",'Semester':6}
              }
```
In [73]:
```
studentsDict['Student1']
```
Out[73]: `{'name': 'Muhammad Umair', 'age': 20, 'Department': 'BSE', 'Semester': 6}`

In [74]:
```
studentsDict['Student1']['name']
```
Out[74]: `'Muhammad Umair'`

In [75]:
```
studentsDict['Student1']['age']
```
Out[75]:
```
20
```

In [76]:
```
studentsDict['Student1']['Department']
```
Out[76]:
```
'BSE'
```

In [77]:
```
studentsDict['Student3']['Department']='BCS'
```

In [78]:
```
studentsDict['Student3']['Department']
```
Out[78]:
```
'BCS'
```

In [79]:
```
del studentsDict['Student2']
```

In [80]:
```
print(studentsDict)
```
```
{'Student1': {'name': 'Muhammad Umair', 'age': 20, 'Department': 'BSE', 'Semester': 6}, 'Student3':
{'name': 'Muhammad Abdullah Tahir', 'age': 20, 'Department': 'BCS', 'Semester': 6}}
```

In [81]:
```
studentsDict['Student2']={'name':"Hamdan Ijaz",'age':22,'Department':"BSE",'Semester':6}
```

In [82]:
```
studentsDict
```
Out[82]:
```
{'Student1': {'name': 'Muhammad Umair',
  'age': 20,
  'Department': 'BSE',
  'Semester': 6},
 'Student3': {'name': 'Muhammad Abdullah Tahir',
  'age': 20,
  'Department': 'BCS',
  'Semester': 6},
 'Student2': {'name': 'Hamdan Ijaz',
  'age': 22,
  'Department': 'BSE',
  'Semester': 6}}
```

Tuples

Difinition

A tuple is a collection of objects which ordered and immutable. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.

Purpose

A tuple lets us “chunk” together related information and use it as a single thing. Tuples support the same sequence operations as strings. ... So like strings, tuples are immutable. Once Python has created a tuple in memory, it cannot be changed.

Importance of Tuples

tuples are immutable. The reasons for having immutable types apply to tuples:

copy efficiency: rather than copying an immutable object, you can alias it (bind a variable to a reference)

comparison efficiency: when you're using copy-by-reference, you can compare two variables by comparing location, rather than content

interning: you need to store at most one copy of any immutable value

there's no need to synchronize access to immutable objects in concurrent code

const correctness: some values shouldn't be allowed to change. This (to me) is the main reason for immutable types.

immutable objects can allow substantial optimization; this is presumably why strings are also immutable in Java, developed quite separately but about the same time as Python, and just about everything is immutable in truly-functional languages.

in Python in particular, only immutables can be hashable (and, therefore, members of sets, or keys in dictionaries). Again, this afford optimization, but far more than just "substantial" (designing decent hash tables storing completely mutable objects is a nightmare -- either you take copies of everything as soon as you hash it, or the nightmare of checking whether the object's hash has changed since you last took a reference to it rears its ugly head).

Strengths

Allows you to output the whole tuple

Allows you to output a specific element

Allows you to combine

Allows you find an item using the index function

Allows you to calculat the length of your tuple

Weakness

You can't add an element but in a list you can

You can't sort a tuple but in a list you can

You can't delete an element but you can in a list

You can't replace an element but you can in a list

Example 17 :

Task:

Create a tuple store some values in it print them and add new value in the tuple using + operator delete and element from the tuple using del keyword use the type

In [83]:
```
studentsResult=(("Muhammad Umair",3.06),("Hamdad Ijaz",2.8),("Muhammad Abdullah Tahir",2.7))
```

In [84]:
```
print(studentsResult)
```
```
(('Muhammad Umair', 3.06), ('Hamdad Ijaz', 2.8), ('Muhammad Abdullah Tahir', 2.7))
```

In [85]:
```
print(studentsResult[0],studentsResult[1],studentsResult[2])
```
```
('Muhammad Umair', 3.06) ('Hamdad Ijaz', 2.8) ('Muhammad Abdullah Tahir', 2.7)
```

In [86]:
```
print(studentsResult[1:2])
```
```
(('Hamdad Ijaz', 2.8),)
```

In [87]:
```
newStudent=(("Muhammad Aaqib Munir",2.5))
```

In [88]:
```
# studentsResult+=newStudent
```

In [89]:
```
print(studentsResult)
```
```
(('Muhammad Umair', 3.06), ('Hamdad Ijaz', 2.8), ('Muhammad Abdullah Tahir', 2.7))
```

In [90]:
```
del studentsResult[2]
```
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-90-edf27cddc713> in <module>
----> 1 del studentsResult[2]

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

In [91]:
```
studentsResult=studentsResult-newStudent
```
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-91-ee9a9fe6cd50> in <module>
----> 1 studentsResult=studentsResult-newStudent

TypeError: unsupported operand type(s) for -: 'tuple' and 'tuple'
```

In [92]:
```
# del studentsResult
```

In [93]:
```
name="Muhammad Umair"
```

In [94]:
```
type(name)
```
Out[94]:
```
str
```

In [95]:
```
name=5
```

In [96]:
```
type(name)
```
Out[96]:
```
int
```

Day_03 Datatypes(Sets & Frozen Sets & Comparison Operators & If-Else Statement & Functions & Lambda Functions)

Sets

Definition

A set is an unordered collection of items. Every set element is unique (no duplicates) and must be immutable (cannot be changed). However, a set itself is mutable. We can add or remove items from it. Sets can also be used to perform mathematical set operations like union, intersection, symmetric difference, etc.

Purpose

A Set is an unordered collection data type that is iterable, mutable and has no duplicate elements. The major advantage of using a set, as opposed to a list, is that it has a highly optimized method for checking whether a specific element is contained in the set.

Importance

Because sets cannot have multiple occurrences of the same element, it makes sets highly useful to efficiently remove duplicate values from a list or tuple and to perform common math operations like unions and intersections.

Strengths

The major advantage of using a set, as opposed to a list, is that it has a highly optimized method for checking whether a specific element is contained in the set. This is based on a data structure known as a hash table. Since sets are unordered, we cannot access items using indexes like we do in lists.

Because sets cannot have multiple occurrences of the same element, it makes sets highly useful to efficiently remove duplicate values from a list or tuple and to perform common math operations like unions and intersections

They help a lot when there is a need for this uniqueness of the elements that need to be processed.

the property that defines sets is the possibility to apply the methods of intersection, union, difference and symmetric difference between them (like sets in mathematics I guess). Mind you cannot do these operations with lists or dictionaries (you have to convert them in sets beforehand).

Weakness

Since set items are not indexed, sets don't support any slicing or indexing operations.Sets do not support indexing, slicing, or other sequence-like behavior. There are currently two built-in set types, set, and frozenset. The set type is mutable - the contents can be changed using methods like add() and remove()

Example 18 :

Task:

create a simple set from {} and using set function add value to the set also use update,discard and remove function on it.

In [98]:
```
numbersSet={3.2,3.06,0.29,0.36,1}
rollNoList
newNumbersSet=set(rollNoList)
```

In [99]:
```
print(newNumbersSet)
```
```
{1, 2, 3, 5, 6, 7}
```

In [100]:
```
type(newNumbersSet)
```
Out[100]:
```
set
```

In [101]:
```
numbersSet.add(3)
```

In [102]:
```
print(numbersSet)
```
```
{0.36, 0.29, 1, 3.06, 3.2, 3}
```

In [103]:
```
numbersSet.update({3,3,4})
```

In [104]:
```
numbersSet
```
Out[104]:
```
{0.29, 0.36, 1, 3, 3.06, 3.2, 4}
```

In [105]:
```
numbersSet.update({2.2,2.3,3.7})
```

In [106]:
```
numbersSet
```
Out[106]:
```
{0.29, 0.36, 1, 2.2, 2.3, 3, 3.06, 3.2, 3.7, 4}
```

In [107]:
```
numbersSet.discard(1)
```

In [108]:
```
print(numbersSet)
```
```
{0.36, 0.29, 3.06, 3.2, 3, 4, 3.7, 2.2, 2.3}
```

In [109]:
```
numbersSet.remove(0.36)
```

In [110]:
```
print(numbersSet)
```
```
{0.29, 3.06, 3.2, 3, 4, 3.7, 2.2, 2.3}
```

In [111]:
```
numbersSet.discard(1)
```

In [112]:
```
print(numbersSet)
```
```
{0.29, 3.06, 3.2, 3, 4, 3.7, 2.2, 2.3}
```

In [113]:
```
numbersSet.remove(1)
```
```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-113-f1792ffe8773> in <module>
----> 1 numbersSet.remove(1)

KeyError: 1
```

Frozen Sets

Definition

Frozen set is just an immutable version of a Python set object. While elements of a set can be modified at any time, elements of the frozen set remain the same after creation. Due to this, frozen sets can be used as keys in Dictionary or as elements of another set.

Purpose

The frozenset() is an inbuilt function is Python which takes an iterable object as input and makes them immutable. Simply it freezes the iterable objects and makes them unchangeable.

In Python, frozenset is same as set except its elements are immutable. This function takes input as any iterable object and converts them into immutable object. The order of element is not guaranteed to be preserved.

Importaance

there main ability that they are immutable makes them very important as you want to have a data set that if once is is declayred no one can add something onto it.

Strengths

set cannot have mutable elements but frozen set have

Set is mutable in nature can be modified but frozen set cannot

set cannot be used as key in dictionary but frozen sets can be used

set cannot have set as element frozen set can have

frozenset can be used as dictionary key

Example 19 :

Task:

Use Frozen set as an element of set

In [114]:
```
#frozenset as an element of set
S1 = frozenset([1,2,3,4])
S2 = {S1}
print(S2)
```
```
{frozenset({1, 2, 3, 4})}
```

Example 20 :

Task:

Use Frozen sets as a Dictionary

In [115]:
```
#frozenset can be used as dictionary key
S1=frozenset([1,2,3,4])
D1={S1:"1234"}
print(D1)
```
```
{frozenset({1, 2, 3, 4}): '1234'}
```

Weakness

major disadvantage of a frozenset is that since they are immutable, it means that you cannot add or remove values.

Comparison Operator & Logical AND OR NOT operator

Example 21 :

Task:

use different opertors to check different conditions and there output

In [116]:
```
5>6
```
Out[116]:
```
False
```

In [117]:
```
6<8
```
Out[117]:
```
True
```

In [118]:
```
15>=15
```
Out[118]:
```
True
```

In [119]:
```
15>=14
```
Out[119]:
```
True
```

In [120]:
```
15<=16
```
Out[120]:
```
True
```

In [121]:
```
"Umair" == "Usama"
```
Out[121]:
```
False
```

In [122]:
```
"Umair" == "Umair"
```
Out[122]:
```
True
```

In [123]:
```
"Umair" != "Usama"
```
Out[123]:
```
True
```

In [124]:
```
(3.06<4) or ("Umair" == "Usama")
```
Out[124]:
```
True
```

In [125]:
```
("Umair"=="Umair") and (5<6)
```
Out[125]:
```
True
```

If Statements

Definition

If statements are control flow statements which helps us to run a particular code only when a certain condition is satisfied. For example, you want to print a message on the screen only when a condition is true then you can use if statement to accomplish this in programming.

Purpose

Python if Statement is used for decision-making operations. It contains a body of code which runs only when the condition given in the if statement is true. ... When you want to justify one condition while the other condition is not true, then you use "if statement

Importance

If statement in Python tells the program what to do if the condition is true. In case the condition is false, the program just goes on to execute what comes after if statements

Strengths

The major advantage of if else statements is that they help us decide what to do if one of our expected output/decisoin gets changed for example if the user is logged in show him logout button else show him log in button.

They help our code by stoping the chrashing of our system if we did not get the desired result we will simply print the message on the screen instead of throwing the same wrong data and getting error.

It helps us bettor our code by applying certain conditions

Weakness

Weakness of the if else statements is that if we have so many conditions to check or so many answeres to check agains the same conditions than it will get really complicated for ourself to read the code and for the compiler to compile it.

if the number of if-else statements increase it will load the system and delayed the execution time.

Example 22 :

Task:

Use if-else statement to check if the number is even or odd

In [126]:
```
number=39
if(number%2==0):
    print("{} is even.".format(number))
else:
    print("{} is Odd.".format(number))
```
```
39 is Odd.
```

Example 23 :

Task:

Use if statement to check if the student's age from the studentDictionary initialize above is greater than 18 or not.

In [127]:
```
if((studentsDict['Student1']['age'])>18):
    print("{name} you can watch Horror Movie".format(name=studentsDict['Student1']['name']))
```
```
Muhammad Umair you can watch Horror Movie
```

Loops in Python

Example 24 :

Task:

Use for loop to print Fibonachi series untill the limit.

In [128]:
```
firstNumber=0
secondNumber=1
numberThree=0
limit=10
fibList=[0,1]
for i in range(1,limit):
    numberThree=firstNumber+secondNumber
    firstNumber=secondNumber
    secondNumber=numberThree
    fibList.append(numberThree)
```

In [129]:
```
fibList
```
Out[129]:
```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
```

Example 25 :

Task:

Use While Loop to find out that if the string is palindrom or not

In [130]:
```
givenString="radar"
i=0
check=False
while(i<(len(givenString)/2)):
    check=((givenString[i]==givenString[(len(givenString)-1)-i]))
    if(i!=0 and check==False):
        print("{string} is a not palindrom".format(string=givenString))
        break
    i=i+1
if(check==True):
    print("{string} is a palindrom".format(string=givenString))
```
```
radar is a palindrom
```

Functions

Definition

function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

Purpose

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.

importance

You use functions in programming to bundle a set of instructions that you want to use repeatedly or that, because of their complexity, are better self-contained in a sub-program and called when needed. That means that a function is a piece of code written to carry out a specified task.

Strengths

Reducing duplication of code

Decomposing complex problems into simpler pieces

Improving clarity of the code

Reuse of code

Information hiding

Weakness

the function is the wraped piece of code that is used to perform sppecific function when needed the only weakness of it is the wrong way of using them.

Example 26 :

Task:

use functions to create a function that print the values of tuple

In [131]:
```
def printResult():
    i=0
    while(i <(len(studentsResult))):
        print("Name:{} CGPA:{}".format(studentsResult[i][0],studentsResult[i][1]))
        i=i+1
```

In [132]:
```
printResult()
```
```
Name:Muhammad Umair CGPA:3.06
Name:Hamdad Ijaz CGPA:2.8
Name:Muhammad Abdullah Tahir CGPA:2.7
```

Example 27 :

Task:

use functions to create a function that check if the string is palindrom or not.

In [133]:
```
def checkPalindrom(givenString):
    i=0
    check=False
    while(i<(len(givenString)/2)):
        check=((givenString[i]==givenString[(len(givenString)-1)-i]))
        if(i!=0 and check==False):
            print("{string} is a not palindrom".format(string=givenString))
            return False
            break
        i=i+1
    if(check==True):
        print("{string} is a palindrom".format(string=givenString))
        return True
```

In [134]:
```
checkPalindrom("radar")
```
```
radar is a palindrom
```
Out[134]:
```
True
```

Example 28 :

Task:

use functions to create a function to print the fibonachi series taking limit as argument

In [135]:
```
def printFibonachi(limit):
    firstNumber=0
    secondNumber=1
    numberThree=0
    fibList=[0,1]
    for i in range(1,limit):
        numberThree=firstNumber+secondNumber
        firstNumber=secondNumber
        secondNumber=numberThree
        fibList.append(numberThree)
    print(fibList)
    return fibList
```

In [136]:
```
printFibonachi(15)
```
```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610]
```
Out[136]:
```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610]
```

Lambda Functions

Definition

a lambda function is a single-line function declared with no name, which can have any number of arguments, but it can only have one expression. Such a function is capable of behaving similarly to a regular function declared using the Python's def keyword. Often times a lambda function is passed as an argument to another function.

Purpose

We use lambda functions when we require a nameless function for a short period of time. In Python, we generally use it as an argument to a higher-order function (a function that takes in other functions as arguments). Lambda functions are used along with built-in functions like filter() , map() etc.

Importance

A lambda is part of a very important abstraction mechanism which deals with higher order functions. To get proper understanding of its value, please watch high quality lessons from Abelson and Sussman, and read the book SICP

Strengths

The code is simple and clear.

No additional variables are added.

Save Memory by declaring and initializing less variables

Weakness

Lambda expressions are a strange and unfamiliar syntax to many Python programmers.

Lambda functions themselves lack names and documentation, meaning that the only way to know what they do is to read the code.

Lambda expressions can only contain one statement, so some readable language features, such as tuple unpacking, cannot be used with them.

Lambda functions can often be replaced with existing functions in the standard library or Python built-in functions.

Example 29 :

Task:

use lambda function to create a square function and create a list of 10 numbers adn apply the lambda function of square and cube on it.

In [137]:
```
square=lambda number:number**2
```

In [138]:
```
squareList=[]
```

In [139]:
```
numberList=list(range(1,10))
```

In [140]:
```
for number in numberList:
    squareList.append(square(number))
```

In [141]:
```
squareList
```

Out[141]:
```
[1, 4, 9, 16, 25, 36, 49, 64, 81]
```

In [142]:
```
[(lambda x: x**3)(x) for x in range(10)]
```

Out[142]:
```
[0, 1, 8, 27, 64, 125, 216, 343, 512, 729]
```

Day 04

Map

Definition

The Python map() method is used to perform a function on every item in a list, dictionary, tuple, or set. The map() method accepts a function and an object to apply that the function will operate on. ... Python includes a built-in method of applying a specific function to all elements within an iterable object: map()

Purpose

map() function returns a map object(which is an iterator) of the results after applying the given function to each item of a given iterable (list, tuple etc.) ... fun : It is a function to which map passes each element of given iterable.

Importance

Python map() function is used to apply a function on all the elements of specified iterable and return map object. Python map object is an iterator, so we can iterate over its elements. We can also convert map object to sequence objects such as list, tuple etc.

Strengths

advantages of a Map include the size property, an easy way to get the number of items in the Map. With an Object, you would be on your own to figure out its size.

Weakness

Map is pretty awesome function that helps us alot in applying the single function to the whole list.The weakness is the wrong way using it like if you mistakenly give it the wrong function to apply on the list.

Example 30 :

Task:

use map function to map the lambda function that find the length of the string onto the list of strings to find the length of each element string in the list

In [144]:
```
checkString='The quick brown fox jumps over the lazy dog'
words=checkString.split()
print(words)
lengths=map(lambda word:len(word),words)
list(lengths)
```

```
['The', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog']
```

Out[144]:
```
[3, 5, 5, 3, 5, 4, 3, 4, 3]
```

Example 31 :

Task:

map the lambda function that find if the string is palindrom and apply it onto the list

In [145]:
```
resultOfPalindrom=map(lambda word:checkPalindrom(word),words)
list(resultOfPalindrom)
```

```
The is a palindrom
quick is a not palindrom
brown is a not palindrom
fox is a palindrom
jumps is a not palindrom
over is a not palindrom
the is a palindrom
lazy is a not palindrom
dog is a palindrom
```

Out[145]:
```
[True, False, False, True, False, False, True, False, True]
```

Example 32 :

Task:

map the greetings function onto the list and print it

In [146]:
```
def greetings(student):
    print("Hy {studentName} you has got {studentCGPA} CGPA".format(studentName=student[0],studentCGPA=student[1]))
    return "Hy {studentName} you has got {studentCGPA} CGPA\n".format(studentName=student[0],studentCGPA=student[1])
greetingsStudent=map(greetings,studentsResult)
greetingsStudent=list(greetingsStudent)
```

```
Hy Muhammad Umair you has got 3.06 CGPA
Hy Hamdad Ijaz you has got 2.8 CGPA
Hy Muhammad Abdullah Tahir you has got 2.7 CGPA
```

Filter

Definition

The filter() method constructs an iterator from elements of an iterable for which a function returns true. In simple words, filter() method filters the given iterable with the help of a function that tests each element in the iterable to be true or not.

Purpose

The filter() method filters the given sequence with the help of a function that tests each element in the sequence to be true or not.

Importance

The filter() function in Python is a built-in function that filters a given input sequence(an iterable) of data based on a function which generally returns either true or false for data elements of the input sequence

Strengths

Filters methods belong to the category of feature selection methods that select features independently of the machine learning algorithm model.

filter methods is that they are very fast.

Weakness

the weakness of filter function is that if we will give it wrong condition to apply

It will take large time for large ammount of Data

Example 33 :

Task:

use filter function to filter out the even numbers from the list of fibonachi series

In [147]:
```
evenFib=filter(lambda x:x%2,fibList)
list(evenFib)
```

Out[147]:
```
[1, 1, 3, 5, 13, 21, 55]
```

In [148]:
```
studentsResult
```

Out[148]:
```
(('Muhammad Umair', 3.06),
 ('Hamdad Ijaz', 2.8),
 ('Muhammad Abdullah Tahir', 2.7))
```

Example 34 :

Task:

Use the filter function to filter out the students that have cgpa greater than 2 from the tuple using lambda functions

In [149]:
```
passStudetsList=filter(lambda student:student[1]>2,studentsResult)
```

In [150]:
```
list(passStudetsList)
```

Out[150]:
```
[('Muhammad Umair', 3.06),
 ('Hamdad Ijaz', 2.8),
 ('Muhammad Abdullah Tahir', 2.7)]
```

File I/O

Definition

A file is some information or data which stays in the computer storage devices. ... Python gives you easy ways to manipulate these files. Generally we divide files in two categories, text file and binary file. Text files are simple text where as the binary files contain binary data which is only readable by computer.

Purpose

Python file handling (a.k.a File I/O) is one of the essential topics for programmers and automation testers. It is required to work with files for either writing to a file or read data from it.

Also, if you are not already aware, I/O operations are the costliest operations where a program can stumble. Hence, you should be quite careful while implementing file handling for reporting or any other purpose. Optimizing a single file operation can help you produce a high-performing application or a robust solution for automated software testing.

Let's take an example, say, you are going to create a big project in Python that contains a no. of workflows. Then, it's inevitable for you not to create a log file. And you'll also be doing both the read/write operations on the log file. Log files are a great tool to debug large programs. It's always better to think about a scalable design from the beginning, as you won't regret it later that you didn't do it.

Importance

File handling is one of the most important parts of any language. Python language supports two types of files. The first one is a text file that store data in the form of text and readable by humans and computers. The second one is binary file that store binary data and readable by computer only.

Strengths

File are used to save the important data produced during the execution of the program like you want to save the username after providing user a input feild to take his name and than to save it into the file to use t later

Weakness

The weakness of the files is that they contain the impotant information if they get deleted all the information will loss and we can mistakenly overwrite the data that also create the data loss

Example 35 :

Task:

Use input function to take input from user and store it into the variable and print it.

In [151]:
```
from six.moves import input
string = input("Enter your name: ");
print(string)
```

```
Enter your name:
```

Example 36 :

Task:

Create a text file and write something into it

In [152]:
```
fileOpen = open("Bio.txt", "w")
for greetings in greetingsStudent:
    fileOpen.write(greetings);
fileOpen.close()
```

Example 37 :

Task:

Create a text file and read something from it

In [153]:
```
fileOpen = open("Bio.txt", "r+")
str = fileOpen.read();
print (str)
fileOpen.close()
```

```
Hy Muhammad Umair you has got 3.06 CGPA
Hy Hamdad Ijaz you has got 2.8 CGPA
Hy Muhammad Abdullah Tahir you has got 2.7 CGPA
```

In [154]:
```
fo = open("Bio.txt", "r+")
str = fo.read(10);
print ("Read String is : \n", str)
position = fo.tell();
print ("Current file position : \n", position)
position = fo.seek(11, 0);
str = fo.read(10);
print ("Again read String is : \n", str)
# Close opend file
fo.close()
```

```
Read String is :
 Hy Muhamma
Current file position :
 10
Again read String is :
  Umair you
```

In [155]:
```
import os
os.rename("Bio.txt", "StudentsGreetings.txt")
```

Day 5

Pandas

Series

Definition

A series in Python is a kind of one-dimensional array of any data type that we specified in the Numpy module. The only difference you can find was, each value in a Python series is associated with the index. The default index value of Python Series is from 0 to n-1, or you can specify your own indexes.

Pandas Series is nothing but a column in an excel sheet. As depicted in the picture below, columns with Name, Age and Designation representing a Series

Purpose

Series is a one-dimensional labeled array capable of holding data of any type (integer, string, float, python objects, etc.). The axis labels are collectively called index.

Importance

Series is the very important data structure in the python it is like a excel datasheet or a database that is used to saved our data in the form of tables so that data will remain categorized and readable.

They are used in many ML Algorithms to hold different type of data and to perform different functions.

Strengths

It helps us in many functions to save the data in tabular form to use it and it makes the data more readable and the series datatype is used in many algorithms.

There are some algorithms nuild in the libraries fo python that require the data in the series format as it is easy to manipulate the data in series.

It is very easy to index and geet the relevent subset of information from the series containg large amount of data.

It is easy to replicate the series and to make changes in them and to update it

Weakness

It does not has a weakness as i say but there are some functions that allow th manipulation on the series data and other functions they

Example 38 :

Task:

create a pandas series use index,values and use slicing to get some data out of it.

In [156]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [157]:
```
np.random.randn(5)
```

Out[157]:
```
array([ 0.58100127, -0.08284663, -1.10031443, -1.3100632 ,  1.690669  ])
```

In [158]:
```
s = pd.Series(np.random.randn(5), index=['a', 'b', 'c', 'd', 'e'])
```

In [159]:
```
s
```

Out[159]:
```
a    1.363077
b   -0.957584
c    0.192168
d   -0.441061
e   -0.904848
dtype: float64
```

In [160]:
```
s.index
```

Out[160]:
```
Index(['a', 'b', 'c', 'd', 'e'], dtype='object')
```

In [161]:
```
s.values
```

Out[161]:
```
array([ 1.36307667, -0.95758415,  0.19216809, -0.44106068, -0.90484761])
```

In [162]:
```
pd.Series(np.random.randn(5))
```

Out[162]:
```
0    0.363901
1   -0.013482
2    1.105250
3    0.300175
4    0.867615
dtype: float64
```

In [163]:
```
studentsSeries=pd.Series(studentsDict)
```

In [164]:
```
studentsSeries
```

Out[164]:
```
Student1    {'name': 'Muhammad Umair', 'age': 20, 'Departm...
Student3    {'name': 'Muhammad Abdullah Tahir', 'age': 20,...
Student2    {'name': 'Hamdan Ijaz', 'age': 22, 'Department...
dtype: object
```

In [165]:
```
studensMarks=pd.Series(studentsResult)
```

In [166]:
```
studensMarks
```

Out[166]:
```
0             (Muhammad Umair, 3.06)
1                 (Hamdad Ijaz, 2.8)
2    (Muhammad Abdullah Tahir, 2.7)
dtype: object
```

In [168]:
```
pd.Series({3.2,3.06,0.29,0.36,1},index=['s1','s2','s3','s4','s5'])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-168-57b600e94cf5> in <module>
----> 1 pd.Series({3.2,3.06,0.29,0.36,1},index=['s1','s2','s3','s4','s5'])

~/anaconda3/lib/python3.8/site-packages/pandas/core/series.py in __init__(self, data, index, dtype, name, copy, fastpath)
    272                 pass
    273             elif isinstance(data, (set, frozenset)):
--> 274                 raise TypeError(f"'{type(data).__name__}' type is unordered")
    275             elif isinstance(data, ABCSparseArray):
    276                 # handle sparse passed here (and force conversion)

TypeError: 'set' type is unordered
```

In [169]:
```
studensMarks[0][0]
```

Out[169]:
```
'Muhammad Umair'
```

In [170]:
```
studensMarks[0:3]
```

Out[170]:
```
0             (Muhammad Umair, 3.06)
1                 (Hamdad Ijaz, 2.8)
2    (Muhammad Abdullah Tahir, 2.7)
dtype: object
```

Example 39 :

Task:

Get conditional Data out of series

In [171]:
```
s[s > s.median()]
```

Out[171]:
```
a    1.363077
c    0.192168
dtype: float64
```

In [172]:
```
s[[4, 3, 1]]
```

Out[172]:
```
e   -0.904848
d   -0.441061
b   -0.957584
dtype: float64
```

Example 40 :

Task:

Get exponent of the data using exp function and get the data out of it.

In [173]:
```
np.exp(s)
```

Out[173]:
```
a    3.908199
b    0.383819
c    1.211874
d    0.643354
e    0.404604
dtype: float64
```

In [174]:
```
s['a']
```

Out[174]:
```
1.3630766691037222
```

In [175]:
```
s['e'] = 12.
```

In [176]:
```
s
```

Out[176]:
```
a     1.363077
b    -0.957584
c     0.192168
d    -0.441061
e    12.000000
dtype: float64
```

In [177]:
```
'e' in s
```

Out[177]:
```
True
```

In [178]:
```
s['f']
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~/anaconda3/lib/python3.8/site-packages/pandas/core/indexes/base.py in get_value(self, series, key)
   4410             try:
-> 4411                 return libindex.get_value_at(s, key)
   4412             except IndexError:

pandas/_libs/index.pyx in pandas._libs.index.get_value_at()

pandas/_libs/index.pyx in pandas._libs.index.get_value_at()

pandas/_libs/util.pxd in pandas._libs.util.get_value_at()

pandas/_libs/util.pxd in pandas._libs.util.validate_indexer()

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

During handling of the above exception, another exception occurred:

KeyError                                  Traceback (most recent call last)
<ipython-input-178-c23937ec966b> in <module>
----> 1 s['f']

~/anaconda3/lib/python3.8/site-packages/pandas/core/series.py in __getitem__(self, key)
    869         key = com.apply_if_callable(key, self)
    870         try:
--> 871             result = self.index.get_value(self, key)
    872 
    873             if not is_scalar(result):

~/anaconda3/lib/python3.8/site-packages/pandas/core/indexes/base.py in get_value(self, series, key)
   4417                     raise InvalidIndexError(key)
   4418                 else:
-> 4419                     raise e1
   4420             except Exception:
   4421                 raise e1

~/anaconda3/lib/python3.8/site-packages/pandas/core/indexes/base.py in get_value(self, series, key)
   4403         k = self._convert_scalar_indexer(k, kind="getitem")
   4404         try:
-> 4405             return self._engine.get_value(s, k, tz=getattr(series.dtype, "tz", None))
   4406         except KeyError as e1:
   4407             if len(self) > 0 and (self.holds_integer() or self.is_boolean()):

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_value()

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_value()

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'f'
```

Example 41 :

Task:

Get Data out of series using get function and apply +, * ,'**'

In [179]:
```
s.get('f')
```

In [180]:
```
s.get('f', np.nan)
```

Out[180]:
```
nan
```

In [181]:
```
s+s
```

Out[181]:
```
a     2.726153
b    -1.915168
c     0.384336
d    -0.882121
e    24.000000
dtype: float64
```

In [182]:
```
s*2
```

Out[182]:
```
a     2.726153
b    -1.915168
c     0.384336
d    -0.882121
e    24.000000
dtype: float64
```

In [183]:
```
s**2
```

Out[183]:
```
a      1.857978
b      0.916967
c      0.036929
d      0.194535
e    144.000000
dtype: float64
```

In [184]:
```
s = pd.Series(np.random.randn(5), name='something')
s
```

Out[184]:
```
0    1.278812
1   -0.416320
2    1.495156
3    0.313534
4   -1.240909
Name: something, dtype: float64
```

Example 42 :

Task:

use name function on the series and use rename funchange its name

In [185]:
```
s.name
```

Out[185]:
```
'something'
```

In [186]:
```
s2 = s.rename("different")
```

In [187]:
```
s2.name
```

Out[187]:
```
'different'
```

Day 6

Data Frames

Definition

Pandas DataFrame is two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns.

Purpose

DataFrames make manipulating your data easy, from selecting or replacing columns and indices to reshaping your data.

Important

Pandas DataFrame is a 2-D labeled data structure with columns of potentially different type. Just like excel, Pandas DataFrame provides various functionalities to analyze, change, and extract valuable information from the given dataset

Strengths

Python Data Analysis Library

pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

While pandas can certainly access data via SQL, or from several other data storage methods, its primary purpose is to make it easier when using Python to do data analysis.

To that end pandas has various methods available that allow some relational algebra operations that can be compared to SQL.

Also Pandas provides easy access to NumPy, which is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities

Weakness

Disadvantages:

Pandas does not persist data. It even has a (slow) function called TO_SQL that will persist your pandas data frame to an RDBMS table.

Pandas will only handle results that fit in memory, which is easy to fill. You can either use dask to work around that, or you can work on the data in the RDBMS (which uses all sorts of tricks like temp space) to operate on data that exceeds RAM.

Example 43 :

Task:

create a Dataframe from the dictionary

In [189]:
```
d = {
'one' : pd.Series([1., 2., 3.], index=['a', 'b', 'c']),
'two' : pd.Series([1., 2., 3., 4.], index=['a', 'b', 'c', 'd'])
}
```

In [190]:
```
df=pd.DataFrame(d)
```

In [191]:
```
d
```

Out[191]:
```
{'one': a    1.0
 b    2.0
 c    3.0
 dtype: float64,
 'two': a    1.0
 b    2.0
 c    3.0
 d    4.0
 dtype: float64}
```

In [192]:
```
df
```

Out[192]:

	one	two
a	1.0	1.0
b	2.0	2.0
c	3.0	3.0
d	NaN	4.0

Example 44 :

Task:

Create a Dataframe from dictionary and assign different indexes and column to the dataframe

In [193]:
```
pd.DataFrame(d, index=['d', 'b', 'a'])
```

Out[193]:

	one	two
d	NaN	4.0
b	2.0	2.0
a	1.0	1.0

In [194]:
```
pd.DataFrame(d, index=['d', 'b', 'a'], columns=['two', 'three'])
```

Out[194]:

	two	three
d	4.0	NaN
b	2.0	NaN
a	1.0	NaN

In [195]:
```
df.columns
```

Out[195]:
```
Index(['one', 'two'], dtype='object')
```

In [196]:
```
df.index
```

Out[196]:
```
Index(['a', 'b', 'c', 'd'], dtype='object')
```

Example 45 :

Task:

create dictionary from the two series then make a dataframe using that dictionary practicec and play with the series adn dictionaries inside Datafram

In [197]:
```
boys=pd.Series([22,24,55],index=["Muhammad Umair","Muhammad Usama","Muhammad Akram"])
```

In [198]:
```
girls=pd.Series([18,25,30,25,45],["Alina","Aneeza","Shahnaz","Sajida","Frazeen"])
```

In [199]:
```
new_Dict={
    'Boys':boys,
    'Girls':girls
}
```

In [200]:
```
pd.DataFrame(new_Dict)
```

Out[200]:

	Boys	Girls
Alina	NaN	18.0
Aneeza	NaN	25.0
Frazeen	NaN	45.0
Muhammad Akram	55.0	NaN
Muhammad Umair	22.0	NaN
Muhammad Usama	24.0	NaN
Sajida	NaN	25.0
Shahnaz	NaN	30.0

In [201]:
```
a_Dict={
    'Age':boys+girls,
}
```

In [202]:
```
pd.DataFrame(a_Dict)
```

Out[202]:

	Age
Alina	NaN
Aneeza	NaN
Frazeen	NaN
Muhammad Akram	NaN
Muhammad Umair	NaN
Muhammad Usama	NaN
Sajida	NaN
Shahnaz	NaN

In [203]:
```
boys+girls
```

Out[203]:
```
Alina             NaN
Aneeza            NaN
Frazeen           NaN
Muhammad Akram    NaN
Muhammad Umair    NaN
Muhammad Usama    NaN
Sajida            NaN
Shahnaz           NaN
dtype: float64
```

In [204]:
```
students_Dict=pd.Series([22,24,55,18,25,30,25,45],index=["Muhammad Umair","Muhammad Usama","Muhammad Akram","Alina","Aneeza","Shahnaz","Sajida","Frazeen"])
```

In [205]:
```
students_Dict
```

Out[205]:
```
Muhammad Umair    22
Muhammad Usama    24
Muhammad Akram    55
Alina             18
Aneeza            25
Shahnaz           30
Sajida            25
Frazeen           45
dtype: int64
```

In [206]:
```
new_Students_dictionary={
 'Age':students_Dict
}
```

In [207]:
```
new_Students_dictionary
```

Out[207]:
```
{'Age': Muhammad Umair    22
 Muhammad Usama    24
 Muhammad Akram    55
 Alina             18
 Aneeza            25
 Shahnaz           30
 Sajida            25
 Frazeen           45
 dtype: int64}
```

In [208]:
```
pd.DataFrame(new_Students_dictionary)
```

Out[208]:

	Age
Muhammad Umair	22
Muhammad Usama	24
Muhammad Akram	55
Alina	18
Aneeza	25
Shahnaz	30
Sajida	25
Frazeen	45

In [209]:
```
d = {
'one' : [1., 2., 3., 4.],
'two' : [4., 3., 2., 1.]
}
```

In [210]:
```
d
```

Out[210]:
```
{'one': [1.0, 2.0, 3.0, 4.0], 'two': [4.0, 3.0, 2.0, 1.0]}
```

In [211]:
```
pd.DataFrame(d)
```

Out[211]:

	one	two
0	1.0	4.0
1	2.0	3.0
2	3.0	2.0
3	4.0	1.0

In [212]:
```
pd.DataFrame(d, index=['a', 'b', 'c', 'd'])
```

Out[212]:

	one	two
a	1.0	4.0
b	2.0	3.0
c	3.0	2.0
d	4.0	1.0

In [213]:
```
studentsDict
```

Out[213]:
```
{'Student1': {'name': 'Muhammad Umair',
  'age': 20,
  'Department': 'BSE',
  'Semester': 6},
 'Student3': {'name': 'Muhammad Abdullah Tahir',
  'age': 20,
  'Department': 'BCS',
  'Semester': 6},
 'Student2': {'name': 'Hamdan Ijaz',
  'age': 22,
  'Department': 'BSE',
  'Semester': 6}}
```

In [214]:
```
students_series=pd.Series(studentsDict)
```

In [215]:
```
dict_students={
    'students':students_series
}
```

In [216]:
```
dict_students
```

Out[216]:
```
{'students': Student1    {'name': 'Muhammad Umair', 'age': 20, 'Departm...
 Student3    {'name': 'Muhammad Abdullah Tahir', 'age': 20,...
 Student2    {'name': 'Hamdan Ijaz', 'age': 22, 'Department...
 dtype: object}
```

In [217]:
```
pd.DataFrame(dict_students)
```

Out[217]:

	students
Student1	{'name': 'Muhammad Umair', 'age': 20, 'Departm...
Student3	{'name': 'Muhammad Abdullah Tahir', 'age': 20,...
Student2	{'name': 'Hamdan Ijaz', 'age': 22, 'Department...

In [218]:
```
pd.DataFrame(students_Dict)
```

Out[218]:

	0
Muhammad Umair	22
Muhammad Usama	24
Muhammad Akram	55
Alina	18
Aneeza	25
Shahnaz	30
Sajida	25
Frazeen	45

In [219]:
```
studentsDict
```

Out[219]:
```
{'Student1': {'name': 'Muhammad Umair',
  'age': 20,
  'Department': 'BSE',
  'Semester': 6},
 'Student3': {'name': 'Muhammad Abdullah Tahir',
  'age': 20,
  'Department': 'BCS',
  'Semester': 6},
 'Student2': {'name': 'Hamdan Ijaz',
  'age': 22,
  'Department': 'BSE',
  'Semester': 6}}
```

In [220]:
```
studentsDataFrame1=pd.DataFrame(studentsDict)
studentsDataFrame1
```

Out[220]:

	Student1	Student3	Student2
name	Muhammad Umair	Muhammad Abdullah Tahir	Hamdan Ijaz
age	20	20	22
Department	BSE	BCS	BSE
Semester	6	6	6

In [221]:
```
studentsDict2={'Student4':{'name':"Hamdan Ijaz",'age':20,'Department':"BSE",'Semester':6},
             'Student5':{'name':"Aaqib Munir",'age':22,'Department':"BSE",'Semester':6},
            'Student6':{'name':"Ammar Naveed",'age':20,'Department':"BSE",'Semester':6}
             }
```

In [222]:
```
studentsDataFrame2=pd.DataFrame(studentsDict2)
studentsDataFrame2
```

Out[222]:

	Student4	Student5	Student6
name	Hamdan Ijaz	Aaqib Munir	Ammar Naveed
age	20	22	20
Department	BSE	BSE	BSE
Semester	6	6	6

In [223]:
```
studentsDataFrame1.append(studentsDataFrame2)
```

Out[223]:

	Student1	Student3	Student2	Student4	Student5	Student6
name	Muhammad Umair	Muhammad Abdullah Tahir	Hamdan Ijaz	NaN	NaN	NaN
age	20	20	22	NaN	NaN	NaN
Department	BSE	BCS	BSE	NaN	NaN	NaN
Semester	6	6	6	NaN	NaN	NaN
name	NaN	NaN	NaN	Hamdan Ijaz	Aaqib Munir	Ammar Naveed
age	NaN	NaN	NaN	20	22	20
Department	NaN	NaN	NaN	BSE	BSE	BSE
Semester	NaN	NaN	NaN	6	6	6

In [224]:
```
d = {
'one' : [1., 2., 3., 4.],
'two' : [4., 3., 2., 1.]
}
```

In [225]:
```
pd.DataFrame(d)
```

Out[225]:

	one	two
0	1.0	4.0
1	2.0	3.0
2	3.0	2.0
3	4.0	1.0

In [226]:
```
pd.DataFrame(d, index=['a', 'b', 'c', 'd'])
```

Out[226]:

	one	two
a	1.0	4.0
b	2.0	3.0
c	3.0	2.0
d	4.0	1.0

Example 46 :

Task:

create a dataframe of list of students dictionaries

In [227]:
```
List_Students_Dict=[
    {'name':"Muhammad Umair",'age':20,'Department':"BSE",'Semester':6},
    {'name':"Hashim Shakoor",'age':22,'Department':"BSE",'Semester':6},
    {'name':"Muhammad Abdullah Tahir",'age':20,'Department':"BSE",'Semester':6},
    {'name':"Hamdan Ijaz",'age':20,'Department':"BSE",'Semester':6},
     {'name':"Aaqib Munir",'age':22,'Department':"BSE",'Semester':6},
    {'name':"Ammar Naveed",'age':20,'Department':"BSE",'Semester':6}
]
```

In [228]:
```
pd.DataFrame(List_Students_Dict)
```

Out[228]:

	name	age	Department	Semester
0	Muhammad Umair	20	BSE	6
1	Hashim Shakoor	22	BSE	6
2	Muhammad Abdullah Tahir	20	BSE	6
3	Hamdan Ijaz	20	BSE	6
4	Aaqib Munir	22	BSE	6
5	Ammar Naveed	20	BSE	6

In [229]:
```
pd.DataFrame(List_Students_Dict,index=['005','102','106','054','056','108'])
```

Out[229]:

	name	age	Department	Semester
005	Muhammad Umair	20	BSE	6
102	Hashim Shakoor	22	BSE	6
106	Muhammad Abdullah Tahir	20	BSE	6
054	Hamdan Ijaz	20	BSE	6
056	Aaqib Munir	22	BSE	6
108	Ammar Naveed	20	BSE	6

Example 47 :

Task:

Create a pandas Dataframe from list of tuples

In [230]:
```
pd.DataFrame({('a', 'b'): {('A', 'B'): 1, ('A', 'C'): 2},
('a', 'a'): {('A', 'C'): 3, ('A', 'B'): 4},
('a', 'c'): {('A', 'B'): 5, ('A', 'C'): 6},
('b', 'a'): {('A', 'C'): 7, ('A', 'B'): 8},
('b', 'b'): {('A', 'D'): 9, ('A', 'B'): 10}})
```

Out[230]:

		a			b	
		b	a	c	a	b
A	B	1.0	4.0	5.0	8.0	10.0
	C	2.0	3.0	6.0	7.0	NaN
	D	NaN	NaN	NaN	NaN	9.0

Day 7

Example 48 :

Task:

Create a pandas Dataframe from numpy array of random numbers using np.random.rand

In [231]:
```
import numpy as np
```

In [232]:
```
pd.DataFrame(np.random.rand(5, 5),
             columns=['A', 'B','C','D','E'],
             index=[1,2,3,4,5])
```

Out[232]:

	A	B	C	D	E
1	0.408195	0.326013	0.630184	0.193325	0.669882
2	0.307215	0.418782	0.872658	0.385257	0.638993
3	0.008182	0.142798	0.948874	0.482652	0.597979
4	0.985341	0.131667	0.385328	0.983306	0.277935
5	0.499608	0.025407	0.113404	0.073284	0.665583

Example 49 :

Task:

Create a dataframe of numpy array containing all zeros hint:use np.zeroes function

In [235]:
```
A = np.zeros(3, dtype=[('A', 'i8'), ('B', 'f8')])
A
```

Out[235]:
```
array([(0, 0.), (0, 0.), (0, 0.)], dtype=[('A', '<i8'), ('B', '<f8')])
```

In [236]:
```
pd.DataFrame(A)
```

Out[236]:

	A	B
0	0	0.0
1	0	0.0
2	0	0.0

Example 50 :

Task:

create a dataframe of series and get the values out of it explicitly and implicitly

In [237]:
```
newData=pd.Series(['Muhammad Umair','Muhammad Usama','Muhammad Akram','Abdullah Tahir','Hamdan Ijaz'
],
                 index=[5,10,15,106,54])
```

In [238]:
```
newData
```

Out[238]:
```
5      Muhammad Umair
10     Muhammad Usama
15     Muhammad Akram
106    Abdullah Tahir
54        Hamdan Ijaz
dtype: object
```

In [239]:
```
# Explicit Indexing
newData[10]
```

Out[239]:
```
'Muhammad Usama'
```

In [240]:
```
# Implicit Indexing
newData[1:3]
```

Out[240]:
```
10    Muhammad Usama
15    Muhammad Akram
dtype: object
```

Example 51 :

Task:

use loc,iloc for explicit and implicit indexing

In [241]:
```
# Explicit Indexing using loc
newData.loc[10]
```

Out[241]:
```
'Muhammad Usama'
```

In [242]:
```
# Explicit Indexing using loc
newData.loc[10:106]
```

Out[242]:
```
10     Muhammad Usama
15     Muhammad Akram
106    Abdullah Tahir
dtype: object
```

In [243]:
```
# implicit Indexing using loc
newData.iloc[3]
```

Out[243]:
```
'Abdullah Tahir'
```

In [244]:
```
# Explicit Indexing using loc
newData.iloc[2:4]
```

Out[244]:
```
15     Muhammad Akram
106    Abdullah Tahir
dtype: object
```

In [245]:
```
# Explicit Indexing using loc
newData.iloc[3]
```

Out[245]:
```
'Abdullah Tahir'
```

Example 52 :

Task:

Access values of Dataframe like dictionary style indexing and like database column '.' method

In [246]:
```
StudentsData=pd.DataFrame(['Muhammad Umair','Muhammad Usama','Muhammad Akram','Abdullah Tahir','Hamd
an Ijaz'],
                index=[5,10,15,106,54],columns=["names"])
```

In [247]:
```
StudentsData
```

Out[247]:

	names
5	Muhammad Umair
10	Muhammad Usama
15	Muhammad Akram
106	Abdullah Tahir
54	Hamdan Ijaz

In [248]:
```
StudentsData["names"]
```

Out[248]:
```
5      Muhammad Umair
10     Muhammad Usama
15     Muhammad Akram
106    Abdullah Tahir
54        Hamdan Ijaz
Name: names, dtype: object
```

In [249]:
```
StudentsData.names
```

Out[249]:
```
5      Muhammad Umair
10     Muhammad Usama
15     Muhammad Akram
106    Abdullah Tahir
54        Hamdan Ijaz
Name: names, dtype: object
```

Example 53 :

Task:

create a Dataframe than convert columns to rows and print all the values from the Database

In [250]:
```
stuDataframe=pd.DataFrame(students_Dict)
```

In [251]:
```
stuDataframe.values
```

Out[251]:
```
array([[22],
       [24],
       [55],
       [18],
       [25],
       [30],
       [25],
       [45]])
```

In [252]:
```
stuDataframe.T
```

Out[252]:

	Muhammad Umair	Muhammad Usama	Muhammad Akram	Alina	Aneeza	Shahnaz	Sajida	Frazeen
0	22	24	55	18	25	30	25	45

In [ ]:

In [ ]: