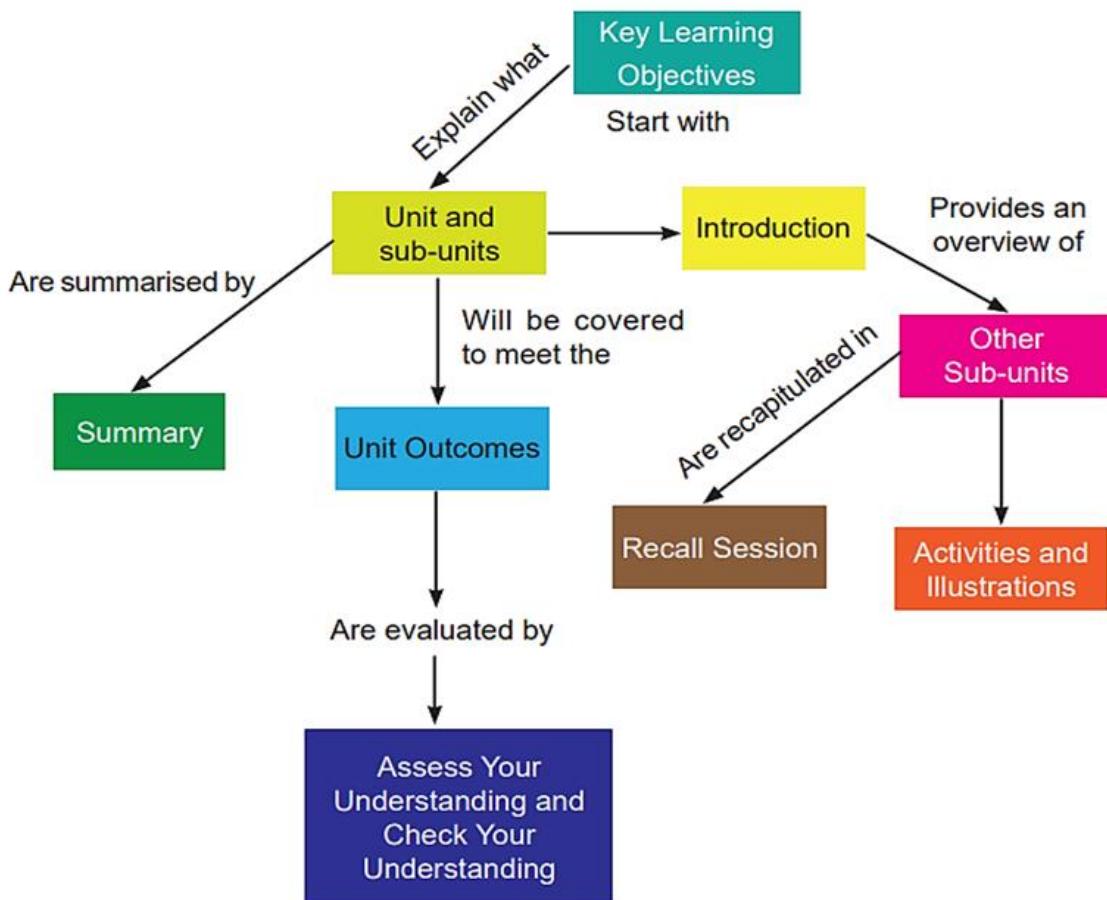


PYTHON PROGRAMMING

Table of Contents

Module 1: Introduction to Python and Basics	4
Unit 1- Introduction to Python	5
Unit 2- Python Sequence	29
Unit 3- Experiment no. 1	63
Unit 4- Conditional Statements	67
Unit 5- Experiment no. 2	82
Unit 6- Function	84
Unit 7- Experiment no. 3	114
Unit 8- Class	115
Unit 9 -Experiment no. 4	132
Module 2: Data Analytics	140
Unit 10- Introduction to Data Analytics	141
Unit 11- Experiment no. 5	167
Unit 12- Introduction to Matplotlib	170
Unit 13- Experiment no. 6	204
Unit 14- Experiment no. 7	208
Unit 15- Experiment no. 8	211
Module 3: Statistics	214
Unit 16- Statistics	215
Unit 17- Experiment no. 9	265
Unit 18- Data visualization using seaborn and plotly	269
Unit 19- Experiment no. 10	320

Learning Map:



Module 1: Introduction to Python and Basics

Key Learning Outcome:

At the end of the module, you will be able to:

- To get the basis of Python, such as features, advantages, usability, variable, identifier, keyword and different data types
- To evaluate the comprehensive analysis for the different types of Python sequence such as Lists-Accessing elements, Using individual values from a list, Changing/Adding and removing Elements, Avoiding Indentation Error, List comprehension, Tuple, Set, Dictionary and Looping through dictionaries
- To analyze the different types of conditional statements such as if, else, switch, and various loop such as For Loop, While Loop, and Nested Loop that are useful in Python
- To evaluate the different types of functions such as Map () function, Filter function, Reduce () function and Python inbuilt functions that developer use for their day-to-day usage
- To get a comprehensive idea about the different types Object Oriented Features such as class, object, access modifier, inheritance, Polymorphism, and Importing Classes

Unit 1- Introduction to Python

Unit 1.1- Python Basics

Python has a very simple syntax like any other programming language. Python has a syntax that allows different developers to write the program within fewer lines than the other programming languages. Python runs on the interpreter system so that the code will be executed as soon as it is written in Python so that both the processing speed and code usability are greater in Python. Python is an interpreted and high-level based programming language incorporating dynamic semantics. Python has a built-in data structure that combines with dynamic binding, making it attractive for rapid application development. It has been used as a script for connecting the different components. Python supports the other modules and respective packages that encounter the reusability of the code. In this section, a comprehensive analysis has been made of some of the most important facts about the Python

1. Python is one of the best examples of multipurpose and high-level based programming language
2. Python allows programming both in the form of object-oriented as well as procedural paradigms
3. Python programming is relatively smaller than the other types of programming languages, for example, Java, by taking rather less time for the computation
4. One of the biggest advantages of Python is the large collection of different types of libraries that are used in the
 - Machine Learning
 - Web Framework, for example, Django
 - Different GUI Application
 - Image processing, for example, Open CV
 - Different types of web scraping techniques, for example, Scrapy and Selenium
 - Scientific Computing

Unit 1.2- Features of Python

- **Python is a free and open source-based programming language** developed under the OSI-approved available source-based language. So the usage of Python is completely free, even for commercial purposes. It does not cost anything to download, and it can be easily modified
- **It is very easy to code** – Anyone can master the Python domain within a few days. Mastering Python and other concepts, for example, module, package, and advanced concepts, take some time.
- **Python is also very easy to read** – The code of Python looks like simple English words. So there are no proper usages of the semicolon or brackets, by the developer evaluated the code simply by looking at the syntax
- Python is a combination of both **object-oriented languages as well as procedure-oriented languages**. The programming language is called object-oriented if it focuses on the data and respective objects rather than the different functions. On the other hand, the programming language is called procedure-oriented if it focuses more on the various parts., Python support both object-oriented language as well as procedure-oriented language in an extensible manner
- **Provides multiple supports of the GUI programming** – One of the major key aspects of any programming language is that it supports different types of GUI. The user is easily interacting with various software by utilizing the capability of GUI. In addition, Python provides other toolkits, for example, Tkinter and wxPython, which provides the fast execution of GUI.
- Python is very easy to debug because it is the best portable language

Unit 1.3- Setting up Programming Environment

In general, a programming environment integrates the internal capability of both software and hardware, allowing the software developer to implement different types of applications. The programming environment in Python, also known as the IDEs, is the specific software platform that provides both the programmer and a software developer with different updated tools for software development for single products.

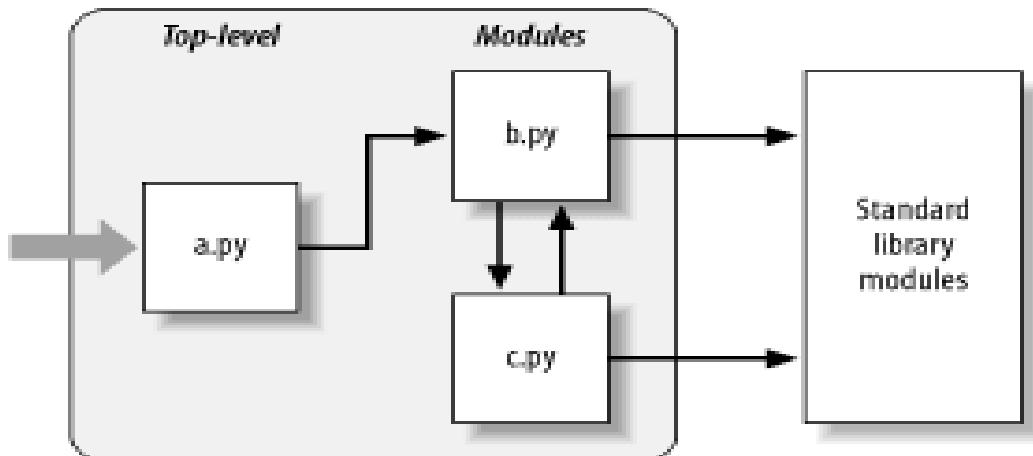
The following steps the developer needs to implement for setting up the programming environment of Python

1. Installation of the supported version of Python that is compatible with the Google Cloud
2. Usage of the venv for isolating different dependencies
3. Installation of the editors, for example, Sublime Text or PyCharm
4. Installation of the Google Cloud CLI
5. Installation of the Cloud Client-based libraries for the Python
6. Installation of the different useful tools

Unit 1.4- Python Program Structure

The programming structure is the specific type of programming paradigm that facilitates the development of different programs with readable code and various reusable components. The program structure is very important because it gives many software developers the internal capability to work toward web development without following their respective methods.

Python's program structure contains three different file types such as a.py, b.py, and c.py. The a.py is the specific high-level-based files executed from the bottom to the top. Both the b.py and c.py are referred to as the module that has been calculated as the better way of program solving. For example, b.py is defined by the function spam, so the def statement of Python starts the function. Later operated by passing the different types of values to the Python program



Unit 1.5- Keywords in Python

The keyword is the specific predefined and reserved word in Python with some special meaning. The keywords are mainly used for defining the coding syntax. The entire keyword is written lowercase except for the True and the False. Python has different sets of keywords called reserve words that cannot be used as variable and function names. Assigning the keyword in Python means that it will be used for only special purposes; it will not be used in the multipurpose. The user would get the Syntax Error if the user assigned the word rather than the reserve keywords. There are in-total 35 keywords present in the latest version of Python, which is 3.8

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

Description of Keywords in Python

True and false are the most important truth values in Python, which are the results of the different comparison operations or the Boolean Operation

```
>>> 1 == 1
True
>>> 5 > 3
True
>>> True or False
True
>>> 10 <= 1
False
>>> 3 > 7
False
>>> True and False
False
```

The figure states that the first three statements are true so that the particular interpreter will return true and false for the next three statements. The True and False are the same as the Boolean 1 and 0 in the next example

```
>>> True == 1
True
>>> False == 0
False
>>> True + True
2
```

None Keyword

None is the specific type of constant representing the Null value or the absence of the matter, and it represents the object of the own data type called None Type. The user cannot develop different multiple None objects, but the user can assign them to the variable.

The Void function, which does not return anything, will automatically replace the None object. None is produced by the specific process by which the flow of the program cannot encounter a return statement

Input

```
def a_void_function():
    a = 1
    b = 2
    c = a + b

x = a_void_function()
print(x)
```

Output

```
None
```

Justification

The program has a function that does not return any value; instead, it does some of the operations inside the program block. As per the above code, when the user prints X, the user only gets None, which has returned automatically.

Unit 1.6- Identifier in Python

The identifier in Python is mainly used to identify the variable, class, and function. It is the combination of both character digits as well as the underscore. So the name of the identifier starts with a specific character, then uses the number. The identifier is mainly used in the python program to define the program code's full syntax and respective structure. There are in-total 33 keywords present in the latest version of Python, which is 3.7.

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

Rules for Writing Python Identifier

1. Identifier could be the combination of both lowercase that is a-z as well as uppercase that is A-Z, or the digits that are 0 to 9
2. Keyword cannot be used as the identifier

Input

```
global = 1
```

Output

```
File "<interactive input>", line 1
    global = 1
          ^
SyntaxError: invalid syntax
```

3. The user cannot use any special symbol, for example, @, \$, and #, in Python Identifier

Input

```
a@ = 0
```

Output

```
File "<interactive input>", line 1
    a@ = 0
          ^
SyntaxError: invalid syntax
```

4. The identifier in Python could be any of the lengths

Example of Identifier

```
int amount;  
  
double totalbalance;
```

Justification

In the above example, both the amount and the Total balance are identifiers, whereas int and double are the keywords.

Unit 1.7- Variables in Python

The variable in the context of Python is the specific symbolic name that is a reference or pointer to particular objects. Once the exact thing has been assigned to the specific variable, the developer refers to that particular object by its name, but the data is contained within each object. In Python, there is no command for declaring a specific variable.

Rule of Python Variable

- The name of the variable starts with the underscore or the letter
- Any number cannot be able to create the variable name
- The variable name needs to be case sensitive (PERSON is not the same as a person)
- The special reserve keyword could not able to use for variable name

Example

Input

```
Var = "Geeksforgeeks"  
print(Var)
```

Output

```
Geeksforgeeks
```

Creating Variable

To define the variable, the user needs to use the following syntax

```
variable_name = value
```

In this syntax, = is stated as the assignment operator, so in this syntax, the user gives a particular value to the variable name (number or string)

The following example states the name of the variable as counter and assigns the number 1 to the variable

```
counter = 1
```

Unit 1.8- Data Types in Python

Data types in Python are associated with classifying the different data items, representing the specific types of value that tell the users what operation has been performed in the detailed data. Generally, there are five build-in data types in the Python

1. Numeric

In Python, the numeric data type represents the specific data with particular numeric values. There are a total of three numerical data types presents in Python

- Integer
- Float
- Complex Number

In addition, Python also provides the type () function that knows the respective variable's data types. The instance () procedure is utilized for checking the particular object belongs to the specific class

Python implements the specific number objects when particular numbers are assigned to the variable.

```
a = 5
print("The type of a", type(a))

b = 40.5
print("The type of b", type(b))

c = 1+3j
print("The type of c", type(c))
print(" c is a complex number", isinstance(1+3j,complex))
```

Output

```
The type of a <class 'int'>
The type of b <class 'float'>
The type of c <class 'complex'>
c is complex number: True
```

The type of value in “c” is the Complex Number (true)

2. Boolean

Boolean data type provides two of the most important built-in function

- True
- False

This value is mainly used to analyze whether the statement is true or false.

```
# Python program to check the boolean type
print(type(True))
print(type(False))
print(false)
```

Output

```
<class 'bool'>
<class 'bool'>
NameError: name 'false' is not defined
```

The resulting output in this program is false

3. Dictionary

A dictionary is the specific un-ordered based particular key pair's values of items. So it is very similar to the associative array or the hash table in which different key stores one specific matter.

```
d = {1:'Jimmy', 2:'Alex', 3:'john', 4:'mike'}
```

```
# Printing dictionary
```

```
print (d)
```

```
# Accesing value using keys
```

```
print("1st name is "+d[1])
```

```
print("2nd name is "+ d[4])
```

```
print (d.keys())
```

```
print (d.values())
```

Output

```
1st name is Jimmy
2nd name is mike
{1: 'Jimmy', 2: 'Alex', 3: 'john', 4: 'mike'}
dict_keys([1, 2, 3, 4])
dict_values(['Jimmy', 'Alex', 'john', 'mike'])
```

The resulting output in this program is they pair of key-value as specified in the form of a dictionary

4. Sequence Type

There are a total of three sequence types associated with Python; are

- String
- List
- Tuple

The following example shows the line in Python

Input

```
str = "string using double quotes"  
print(str)  
  
s = """A multiline  
string"""  
print(s)
```

Output

```
string using double quotes  
A multiline  
string
```

In this figure, String is the output value defined by the user

5. Set

The set in Python is the un-order collection of different data types. The group is implemented in Python with the help of the set () function or sequence of the specific elements passed to the curly braces

Example

Input

```
# Creating Empty set
set1 = set()

set2 = {'James', 2, 3,'Python'}

#printing Set value
print(set2)

# Adding element to the set

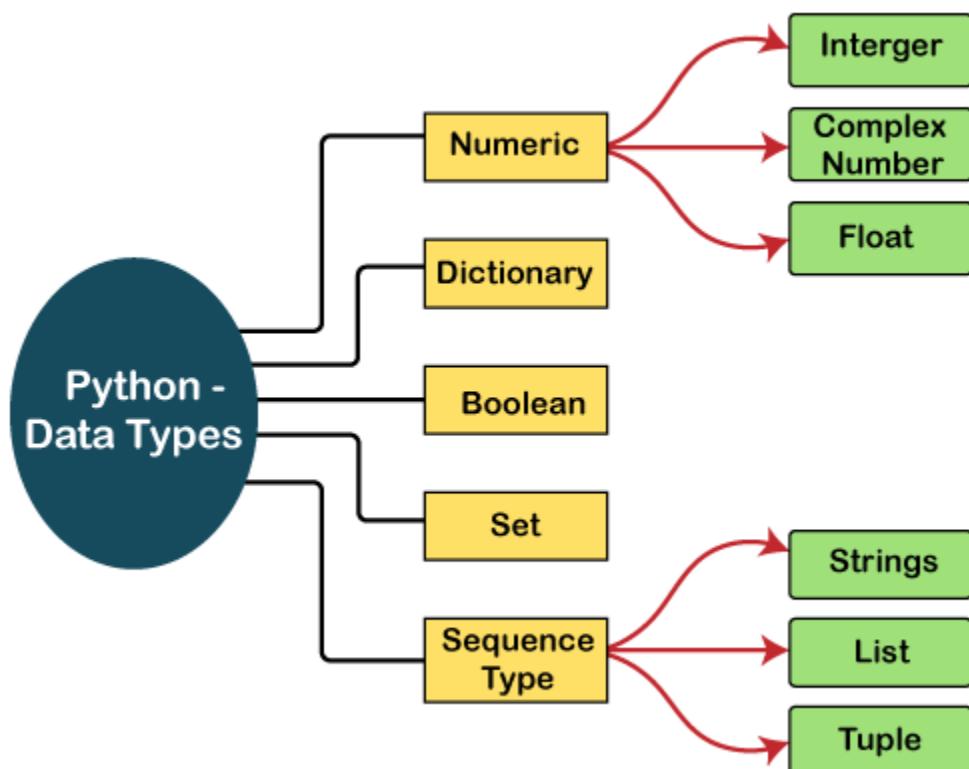
set2.add(10)
print(set2)

#Removing element from the set
set2.remove(2)
print(set2)
```

Output

```
{3, 'Python', 'James', 2}  
{'Python', 'James', 3, 2, 10}  
{'Python', 'James', 3, 10}
```

The above output demonstrates the example of the set () function that is passed to the curly braces (User Defined Values)



Unit 1.9- Operators in Python

The operator in Python performs any operation on the variable and respective values. Different types of standard symbols are used for both arithmetic operations as well as the logical operation

- OPERATOR defines as a specific symbol such as + and –
- OPERAND defines as the particular value in which the operator is applied

Different Types of Operators in Python

There are a total of seven different types of operator present, which we are going to discuss in this book

Arithmetic operators

This operator is used to perform the specific mathematical operation

Operator	Name	Example
+	Addition	$x + y$
-	Subtraction	$x - y$
*	Multiplication	$x * y$
/	Division	x / y
%	Modulus	$x \% y$
**	Exponentiation	$x ** y$
//	Floor division	$x // y$

Assignment operators

The assignment operator is used for assigning particular values to the variable

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
=	x = 3	x = x 3
^=	x ^= 3	x = x ^ 3

Comparison operators

The usage of the comparison operator in Python is to compare the two respective values

Operator	Name	Example
<code>==</code>	Equal	<code>x == y</code>
<code>!=</code>	Not equal	<code>x != y</code>
<code>></code>	Greater than	<code>x > y</code>
<code><</code>	Less than	<code>x < y</code>
<code>>=</code>	Greater than or equal to	<code>x >= y</code>
<code><=</code>	Less than or equal to	<code>x <= y</code>

Logical operators

This operator is mainly used for combining the different conditional statement

Operator	Description	Example
<code>and</code>	Returns True if both statements are true	<code>x < 5 and x < 10</code>
<code>or</code>	Returns True if one of the statements is true	<code>x < 5 or x < 4</code>
<code>not</code>	Reverse the result, returns False if the result is true	<code>not(x < 5 and x < 10)</code>

Identity operators

This operator is used to make the comparison between the different objects; not the things are equal, but if they are the same object, then it will allocate the same memory location

Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

Membership operators

The membership operator is used for testing if the particular sequence is present within an object or not

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

Bitwise operators

This operator is mainly used for comparing the binary numbers

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

MCQ

Question 1- Which types of Programming that Python Support?

- a) Functional Programming
- b) Object Oriented based Programming
- c) Structured Programming
- d) All of the above

Question 2- When Python is associated with the identifier, is it case-sensitive?

- a) Yes
- b) No
- c) Machine Dependent
- d) None of the Above

Question 3- Which of the following is the correct statement about the Python File?

- a) pa
- b) pv
- c) pf
- d) py

Question 4- Is the Python Code either Compiled or Interpreted?

- a) Only compile
- b) Only Interpret
- c) Either or
- d) Neither Nor

Question 5- What is the value of the following expression of Python?

4+3%5

- a) 1
- b) 2
- c) 5
- d) 7

Question 6- Which is the following mainly used for defining the Code Block in Python?

- a) Key
- b) Block
- c) Indentation
- d) None

Question 7- What is the Data type of print (type (10))

- a) Int
- b) Float
- c) Bool
- d) Integer

Question 8- Which one is not the core data type in Python?

- a) List
- b) Bool
- c) Tuple
- d) Class

Case Studies

[http://103.110.113.54/pen/video/Programming Languages Python/Ucoluk, Kalkan - Introduction to Programming Concepts with Case Studies in Python.pdf](http://103.110.113.54/pen/video/Programming_Languages_Python/Ucoluk,_Kalkan_-_Introduction_to_Programming_Concepts_with_Case_Studies_in_Python.pdf)

[https://www.researchgate.net/publication/343442500 Python Programming with Case Studies](https://www.researchgate.net/publication/343442500_Python_Programming_with_Case_Studies)

Reference

<https://developers.google.com/edu/python/introduction#:~:text=Python%20is%20a%20dynamic%2C%20interpreted,checking%20of%20the%20source%20code.>

https://www.w3schools.com/python/python_intro.asp

<https://www.geeksforgeeks.org/introduction-to-python/>

<https://www.tutorialspoint.com/python/index.htm>

Unit 2- Python Sequence

Unit 2.1- Lists-Accessing Elements

The list is one of the most important data types available in Python, written as the list of the items between the square brackets. The index of the list starts at 0, and then the list is sliced and lastly concatenated

The user can access any list item from the group of objects by referring to its index number only. To access the respective values in the list, the developer needs to use the square brackets to slice it alongside either the index or specific indices to obtain the particular value available in the respective index position.

Example

Input



```
thislist = ["apple", "banana",
"cherry"]
print(thislist[1])
```

Output

```
banana
```

Explanation

This example prints the second item of the specified list

Most developers use () for accessing the particular values in the respective lists and, simultaneously, get the index value of those lists. This method enumerates for index position alongside the values.

Example

Input

```
# Python3 code to demonstrate  
# to get index and value  
# using enumerate  
  
# initializing list  
test_list = [1, 4, 5, 6, 7]  
  
# Printing list  
print ("Original list is : " + str(test_list))  
  
# using enumerate to  
# get index and value  
print ("List index-value are : ")  
for index, value in enumerate(test_list):  
    print(index, value)
```

Output

```
Original list is : [1, 4, 5, 6, 7]
List index-value are :
0 1
1 4
2 5
3 6
4 7
```

Unit 2.2- Index Position

The index () function is capable of searching the given elements from the very start of the lots, and then it will return for the index of the first occurrence

Syntax

Syntax: list_name.index(element, start, end)

Example

Input

```
# list of items
list2 = ['cat', 'bat', 'mat', 'cat', 'pet']

# Will print the index of 'cat' in list2
print(list2.index('bat'))
```

Output

1

Explanation

In this example, the user finds the index of the “bat” with the help of the index ()

Unit 2.3- Using individual values from a list

With the help of the list traversal, the user traverses each element in the respected list and then checks if the respective components are in the impressive list already. If it is not over, then the user can append the list to the form of unique_list

Example

Input

```
def unique(list1):

    # initialize a null list
    unique_list = []

    # traverse for all elements
    for x in list1:
        # check if exists in unique_list or not
        if x not in unique_list:
            unique_list.append(x)
    # print list
    for x in unique_list:
        print x,

# driver code
list1 = [10, 20, 10, 30, 40, 40]
print("the unique values from 1st list is")
unique(list1)

list2 = [1, 2, 1, 1, 3, 4, 3, 3, 5]
print("\nthe unique values from 2nd list is")
unique(list2)
```

Output

```
the unique values from 1st list is  
10 20 30 40  
the unique values from 2nd list is  
1 2 3 4 5
```

Unit 2.4- Changing/Adding and Removing Elements

Changes of Elements

Changing the element of the lists is possible with the help of list indexing. This is the simplest method for changing or replacing the list's first item, so you can the user change it using the index position 0.

Example

Input

```
# Replace Values in a List using indexing  
  
# define list  
l = [ 'Hardik', 'Rohit', 'Rahul', 'Virat', 'Pant']  
  
# replace first value  
l[0] = 'Shardul'  
  
# print list  
print(l)
```

Output

```
['Shardul', 'Rohit', 'Rahul', 'Virat', 'Pant']
```

Explanation

In the above example, the index is a specific index of the item that the user wants to change or replace, and the respective new value is the particular value that the user change or replace with the old value in the intended list

Addition of Elements

Insert of the item in the Python list is possible with the help of the append () method by inserting the data into the list end

Example

Input

```
# Define the list
listdata = ["Dell", "HP", "Leveno", "Asus"]

# Insert data using append method
listdata.append("Toshiba")

# Display the list after insert
print("The List elements are")

for i in range(0, len(listdata)):
    print(listdata[i])
```

Output

```
fahmida@fahmida-VirtualBox:~/python$ python3 list2.py
The list elements are
Dell
HP
Leveno
Asus
Toshiba
fahmida@fahmida-VirtualBox:~/python$ █
```

Explanation

In this example, ‘Toshiba’ will be inserted into the end of the list of data

Deletion of Elements

Deletion of the elements in the Python list is possible with the help of the remove () method. If the specific item value used as the argument value of the remove () exists on the respected list, then the item will be removed

Example

Input

```
# Define the list
list = ['Cake', 'Pizza', 'Juice', 'Pasta', 'Burger']

# Print the list before delete
print("List before delete")
print(list)

# Remove an item
list.remove('Juice')

# Print the list after delete
print("List after delete")
print(list)
```

Output

```
fahmida@fahmida-VirtualBox:~/python$ python3 list4.py
List before delete
['Cake', 'Pizza', 'Juice', 'Pasta', 'Burger']
List after delete
['Cake', 'Pizza', 'Pasta', 'Burger']
fahmida@fahmida-VirtualBox:~/python$
```

Explanation

In this example, the value, namely ‘Juice’ in the respective list will be removed

Unit 2.5- Organizing a list

The list sort () function in Python is mainly used for organizing a list in the form of ascending, descending as well as user-defined based order

Syntax

Syntax: `List_name.sort(reverse=True/False, key=myFunc)`

Example

Input

```
# Python program to demonstrate to
# sorting numbers in Ascending Order

numbers = [1, 3, 4, 2]
# Sorting list of Integers in ascending
numbers.sort()
print(numbers)
```

Output

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

Explanation

This example illustrates the example of the sort function for sorting the elements in the form of ascending order

Unit 2.6- Loop through an entire list

There are different ways in which the user iterates over the list in Python, but in this book, we use the concept of For Loop

Example

Input

```
# Python3 code to iterate over a list
list = [1, 3, 5, 7, 9]

# Using for loop
for i in list:
    print(i)
```

Output

```
1  
3  
5  
7  
9
```

Explanation

As can see from the above example, when the loop is iterated with the help of 5 list elements then, it will print the same values

Unit 2.7- Avoiding Indentation Error

To avoid the indentation error in the Python list, the developer needs to go through each line of code separately. Thus it will help the developer for which line of the respective code contains the error. Then the user needs to solve both the compile type error as well as runtime error to solve the indentation errors

Unit 2.8- Numerical Lists

The numerical list in Python is implemented with the help of the Naïve approach. In this approach, the developer needs to develop lists within the given range by first creating the empty list and then appending the successor for each integer through loop iteration

Example

Input

```
# Python3 Program to Create list
# with integers within given range

def createList(r1, r2):

    # Testing if range r1 and r2
    # are equal
    if (r1 == r2):
        return r1

    else:

        # Create empty list
        res = []

        # loop to append successors to
        # list until r2 is reached.
        while(r1 < r2+1 ):

            res.append(r1)
            r1 += 1
        return res

# Driver Code
r1, r2 = -1, 1
print(createList(r1, r2))
```

Output

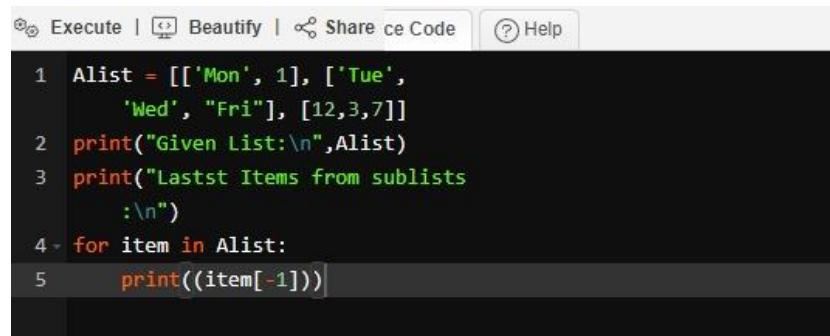
```
[-1, 0, 1]
```

Unit 2.9- Sublist

The list in Python consists of the list inside each element called the sub-list. The retrieval of the last pieces in every sub-list in the given index is possible with the help of the For Loop

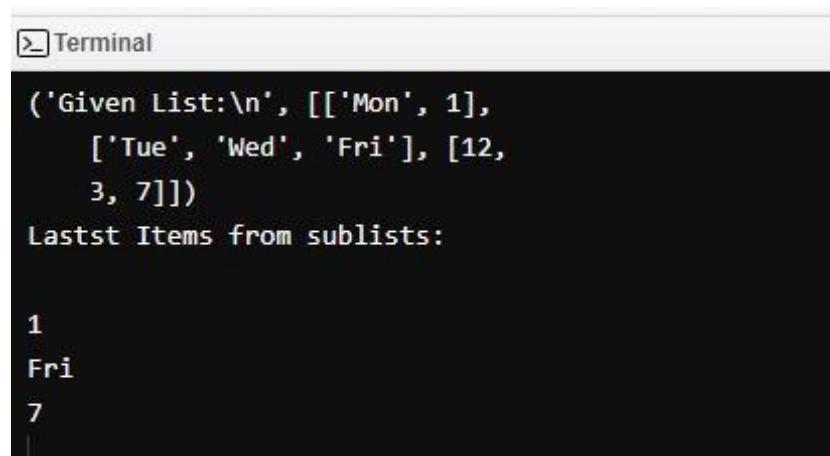
Example

Input



```
Execute | Beautify | Share ce Code | Help  
1 Alist = [['Mon', 1], ['Tue',  
     'Wed', 'Fri'], [12,3,7]]  
2 print("Given List:\n",Alist)  
3 print("Lastst Items from sublists  
      :\n")  
4 for item in Alist:  
5     print((item[-1]))
```

Output



```
Terminal  
('Given List:\n', [['Mon', 1],  
     ['Tue', 'Wed', 'Fri'], [12,  
      3, 7]])  
Lastst Items from sublists:  
  
1  
Fri  
7
```

Explanation

As it can be seen from the above example is that the developer uses the loop through the sub-lists that is capable of fetching the item in the position of the index at -1

Unit 2.10- List Comprehension

The list comprehension in Python contains the brackets and expressions executed for every element and for loop for iterating every element present within the Python list. The list comprehension in Python provides the shorter syntax to create a new list item from the existing list

Advantages of List Comprehension

It provides more time efficiency as well as space efficiency than the loops

List comprehension requires very fewer lines to code

It is capable of transforming the entire iterative statement into the form of a formula

Syntax of List Comprehension

```
newList = [ expression(element) for element in oldList if  
           condition ]
```

Example [Iteration using Advantage of List Comprehension]

Input

```
# Using list comprehension to iterate through loop  
List = [character for character in [1, 2, 3]]  
  
# Displaying list  
print(List)
```

Output

```
[1, 2, 3]
```

Unit 2.11- Tuple

The tuple in Python is the collection of different types of objects that commas have separated. The tuple is very similar to the lists based on the indexing and nested objects, but the major difference is that the tuple in Python are immutable

Creating a Tuple

A tuple is implemented by placing the elements inside the parenthesis, separated by the comma (Parenthesis are optional). The Tuple consists of any number of items, and it can be (an integer, string, or float)

Input

```
# Different types of tuples

# Empty tuple
my_tuple = ()
print(my_tuple)

# Tuple having integers
my_tuple = (1, 2, 3)
print(my_tuple)

# tuple with mixed datatypes
my_tuple = (1, "Hello", 3.4)
print(my_tuple)

# nested tuple
my_tuple = ("mouse", [8, 4, 6], (1, 2, 3))
print(my_tuple)
```

Output

```
()  
(1, 2, 3)  
(1, 'Hello', 3.4)  
('mouse', [8, 4, 6], (1, 2, 3))
```

The tuple is created without the proper usage of the parenthesis; it is known as tuple packing.

Input

```
my_tuple = 3, 4.6, "dog"  
print(my_tuple)  
  
# tuple unpacking is also possible  
a, b, c = my_tuple  
  
print(a)      # 3  
print(b)      # 4.6  
print(c)      # dog
```

Output

```
(3, 4.6, 'dog')  
3  
4.6  
dog
```

Implementing a tuple with the help of only one element is not simple. To do so, the user must trail the comma to indicate the tuple.

Input

```
my_tuple = ("hello")
print(type(my_tuple)) # <class 'str'>

# Creating a tuple having one element
my_tuple = ("hello",)
print(type(my_tuple)) # <class 'tuple'>

# Parentheses is optional
my_tuple = "hello",
print(type(my_tuple)) # <class 'tuple'>
```

Output

```
<class 'str'>
<class 'tuple'>
<class 'tuple'>
```

Unit 2.12- Set

The set is the unordered collection of the different types of data types which is inerrable and has no duplicate elements. The Set () method is mainly used for the typical conversation in Python. The set is a particular collection of similar un-order items in which every element should be unique, and the location can remove the duplicate items. A proper index is associated with every set element, so the user can not access any aspects of the respective set with the help of the index number.

Example

Input

```
# Python program to
# demonstrate sets

# Same as {"a", "b", "c"}
myset = set(["a", "b", "c"])
print(myset)

# Adding element to the set
myset.add("d")
print(myset)
```

Output

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

Set Operation

Creating a Set

A set is implemented with the help of either the set () function or placing the entire elements within the curly braces

Example

```
Days=set(["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"])
Months={"Jan", "Feb", "Mar"}
Dates={21, 22, 17}
print(Days)
print(Months)
print(Dates)
```

Output

When the above mention code is executed, then it will give the following output (Changes in the order of elements is visible in this results)

```
set(['Wed', 'Sun', 'Fri', 'Tue', 'Mon', 'Thu', 'Sat'])
set(['Jan', 'Mar', 'Feb'])
set([17, 21, 22])
```

Accessing the Value in Set

The user could not access the individual value in the respective set; the user can access each element together depicted in the previous example. The user can get the particular list of every component by looping it through the set

Example

```
Days=set(["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"])

for d in Days:
    print(d)
```

Output

```
Wed  
Sun  
Fri  
Tue  
Mon  
Thu  
Sat
```

Adding Item to Set

The user can add any elements to the respective set with the help of the add () method

Example

```
Days=set(["Mon","Tue","Wed","Thu","Fri","Sat"])

Days.add("Sun")
print(Days)
```

Output

```
set(['Wed', 'Sun', 'Fri', 'Tue', 'Mon', 'Thu', 'Sat'])
```

Removing Item to Set

The user can delete any elements to the respective set with the help of the discard () method

Example

```
Days=set(["Mon","Tue","Wed","Thu","Fri","Sat"])

Days.discard("Sun")
print(Days)
```

Output

```
set(['Wed', 'Fri', 'Tue', 'Mon', 'Thu', 'Sat'])
```

Unit 2.13- Dictionary- Working with dictionaries

The dictionary in the context of Python is the updated collection of different types of key values that are mainly used for storing the different kinds of data values, such as a map that holds only a single value as the elements

Python dictionary is mainly used for storing data in the form of key–value pairs. The dictionary is the specific Python data type that stimulates real-life-based data arrangement in which some particular value exists for some specific key. Thus the dictionary is referred to the element segmentation of keys and values

- The key needs to be single elements
- The value needs to be any of the data types, such as list and tuple

Importance of Dictionary in Python

A dictionary is the specific data structure in Python that uses the indexing key. Dictionary is the un-order sequence of different items which means the specific order is not preserved because the key is immutable. With the usage of the dictionary, the developer is able to solve the three of the most severe problem

- The index could be the any items in the respective data that the user wants (Name and pair of Integer)
- Finding the data based on the respective key is really quick regardless the key type
- The dictionary in Python does not waste any kinds of space for key that does not exist rather than they use the different types of key in the same dictionary

Example

Input

```
Dict = {1: 'Geeks', 2: 'For', 3: 'Geeks'}
print(Dict)
```

Output

```
{1: 'Geeks', 2: 'For', 3: 'Geeks'}
```

Explanation

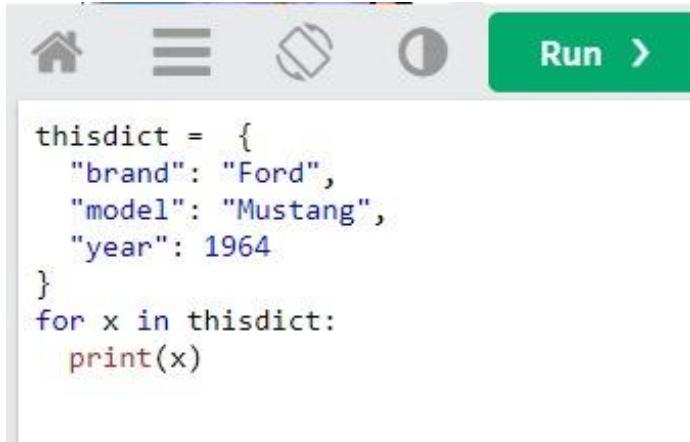
In this above example, the dictionary holds the key: value pair. The pairing of key value makes the Python Dictionary more optimized

Unit 2.14- Looping Through Dictionaries

Looping through the dictionary is possible with the help of for loop. When the developer loop thought the dictionary, then it will return the value as the key of the dictionary but at the same time there are different types of methods for returning the value

Example

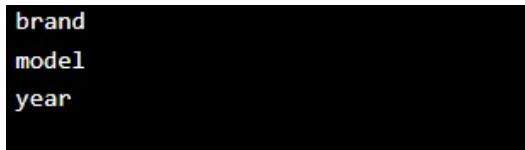
Input



A screenshot of a Python code editor interface. At the top, there are several icons: a house, three horizontal lines, a square with diagonal lines, and a circle. To the right of these is a green 'Run' button with a white arrow. Below the icons is a code block containing the following Python code:

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
for x in thisdict:  
    print(x)
```

Output



A screenshot of a terminal window. The output consists of three lines of text: 'brand', 'model', and 'year', each on a new line.

Explanation

In this example, the developer prints the entire key name in the respective dictionary in one by one

Different Operation on Python Dictionary

Accessing Elements from Dictionary

While indexing is mainly used with the other data types for accessing value, the dictionary uses the keys. Thus the key has been used either inside the square brackets or with the help of the get() method

Input

```
# get vs [] for retrieving elements
my_dict = {'name': 'Jack', 'age': 26}

# Output: Jack
print(my_dict['name'])

# Output: 26
print(my_dict.get('age'))

# Trying to access keys which doesn't exist throws error
# Output None
print(my_dict.get('address'))

# KeyError
print(my_dict['address'])
```

Output

```
Jack
26
None
Traceback (most recent call last):
  File "<string>", line 15, in <module>
    print(my_dict['address'])
KeyError: 'address'
```

Changing and Adding Elements

Dictionary is mutable so the user can add the new items or change the value of the existing item with the help of the assignment operator. If the respective key is already present, then the respective existing value then updated. If the respective key is not present then key : value pair is added in the dictionary

Input

```
# Changing and adding Dictionary Elements
my_dict = {'name': 'Jack', 'age': 26}

# update value
my_dict['age'] = 27

#Output: {'age': 27, 'name': 'Jack'}
print(my_dict)

# add item
my_dict['address'] = 'Downtown'

# Output: {'address': 'Downtown', 'age': 27, 'name': 'Jack'}
print(my_dict)
```

Output

```
{'name': 'Jack', 'age': 27}
{'name': 'Jack', 'age': 27, 'address': 'Downtown'}
```

Removing Elements from Dictionary

The user can remove a particular item in the respective dictionary with the help of the `pop()` method. This particular removes a specific item with the help of the key value and then return the respective value.

Input

```
# Removing elements from a dictionary

# create a dictionary
squares = {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

# remove a particular item, returns its value
# Output: 16
print(squares.pop(4))

# Output: {1: 1, 2: 4, 3: 9, 5: 25}
print(squares)

# remove an arbitrary item, return (key,value)
# Output: (5, 25)
print(squares.popitem())

# Output: {1: 1, 2: 4, 3: 9}
print(squares)

# remove all items
squares.clear()

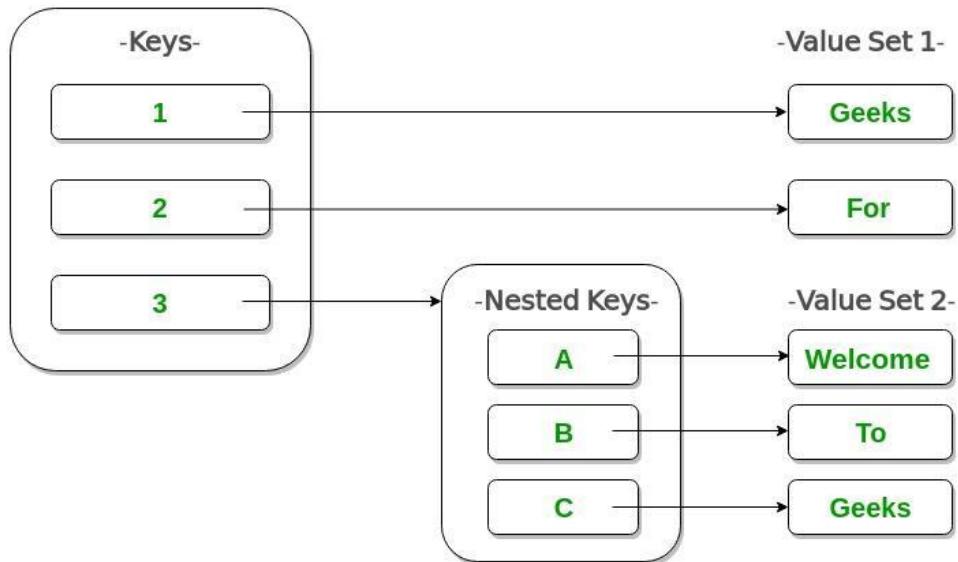
# Output: {}
print(squares)
```

Output

```
16
{1: 1, 2: 4, 3: 9, 5: 25}
(5, 25)
{1: 1, 2: 4, 3: 9}
{}
Traceback (most recent call last):
  File "<string>", line 30, in <module>
    print(squares)
NameError: name 'squares' is not defined
```

Unit 2.15- Nesting

The nesting dictionary in the Python has associated with the one dictionary inside the another dictionary so with the help of the nested pattern the developer can model any program



Example (Creation in Nesting Dictionary)

Input

A screenshot of a Python code editor showing a file named "main.py" with the following content:

```
main.py
1 people = {1: {'name': 'John', 'age': '27', 'sex': 'Male'},
2     |   2: {'name': 'Marie', 'age': '22', 'sex': 'Female'}}
3
4 print(people)
```

The code defines a dictionary "people" where the keys are integers 1 and 2. Each key maps to another dictionary containing "name", "age", and "sex" fields. The "print" statement at the bottom is intended to output the entire nested structure.

Output

```
Shell  
{1: {'name': 'John', 'age': '27', 'sex': 'Male'}, 2: {'name': 'Marie', 'age':  
: '22', 'sex': 'Female'}}  
> |
```

Clear

Explanation

In this program, People are nested dictionary. So the internal dictionary 1 and 2 has been assigned to the people. So both the dictionary in the above example has the key value of name, sex and age with different types of values. So now the user will print the result of people

Indexing

Indexing is referred to the particular elements of the inerrable by its respective position within the every iteration. Each of the characters of the string that is correspondence to the specific index number access with its unique index number

Example (Positive Index Number)

Input

```
# declaring the string
str = "Geeks for Geeks !"

# accessing the character of str at 0th index
print(str[0])

# accessing the character of str at 6th index
print(str[6])

# accessing the character of str at 10th index
print(str[10])
```

Output

```
G
f
G
```

Explanation

In the above example, the user passes the Positive Index (that the user wants to access) inside the square brackets. The index number starts from the index position 0 that denotes first character of the respective string

Slicing

In Python, slicing is the most updated method which enables accessing part of the sequence. By slicing a string, the user needs to first create a sub string which is a string which exists within another string.

Example (Original and Reverse a string)

Input

```
# declaring the string
str ="Geeks for Geeks !"

print("Original String :-")
print(str)

# reversing the string using slicing
print("Reverse String :-")
print(str[::-1])
```

Output

```
Original String :-
Geeks for Geeks !
Reverse String :-
! skeeG rof skeeG
```

MCQ

Question 1- Suppose the ListExample is [‘h’,’e’,’l’,’l’,’o’], then what is the length?

- a) 1
- b) 2
- c) 4
- d) 5

Question 2- Suppose there is a List= [1, 5, 9] then what is the sum (list)?

- a) 1
- b) 11
- c) 10
- d) 15

Question 3- Suppose there is a List= [2, 33, 222, 14, 25], then the value of list [-1]?

- a) 5
- b) 10
- c) 15
- d) 25

Question 4- Output ?

```
1. names1 = ['Amir', 'Bean', 'Charlton', 'Daman']
2. names2 = names1
3. names3 = names1[:]
4.
5. names2[0] = 'Alice'
6. names3[1] = 'Bob'
7.
8. sum = 0
9. for ls in (names1, names2, names3):
10.     if ls[0] == 'Alice':
11.         sum += 1
12.     if ls[1] == 'Bob':
13.         sum += 10
14.
15. print sum
```

- a) 10
- b) 11
- c) 12
- d) 13

Question 5- Output

```
L=[1,2,3,4,5]
for i in L:
    print(i,end=" ")
    i=i+1
```

- a) 1, 2, 3, 4, 5
- b) 5
- c) 6
- d) 7

Question 6- Output

```
1. >>>names = ['Amir', 'Bear', 'Charlton', 'Daman']
2. >>>print(names[-1][-1])
```

- a) n
- b) a
- c) n
- d) 0

Question 7- Square brackets in the specific assignment statement will create which of the following types of data structure

- a) Queue
- b) Stack
- c) Tuple
- d) List

Question 8- Which of the following is the correct code to create the list of names?

```
nameList = John, Harry, Jesse, John, Harry, Harry
nameList = ("John", "Harry", "Jesse", "John", "Harry", "Harry")
nameList = ["John", "Harry", "Jesse", "John", "Harry", "Harry"]
nameList = [John, Harry, Jesse, John, Harry, Harry]
```

Question 9- What will be the output for the following code?

```
colours = []
colours.append("blue")
colours.append("green")
colours.append("red")
print(colours[2])
```

- a) Red
- b) Blue
- c) Green
- d) Compilation Error

Case Studies

<https://python.plainenglish.io/python-beginner-tutorial-case-studies-for-data-scientists-to-start-right-away-b426f3ca950d>

Reference

https://www.w3schools.com/python/python_lists.asp

<https://www.programiz.com/python-programming/list>

<https://www.geeksforgeeks.org/python-lists/>

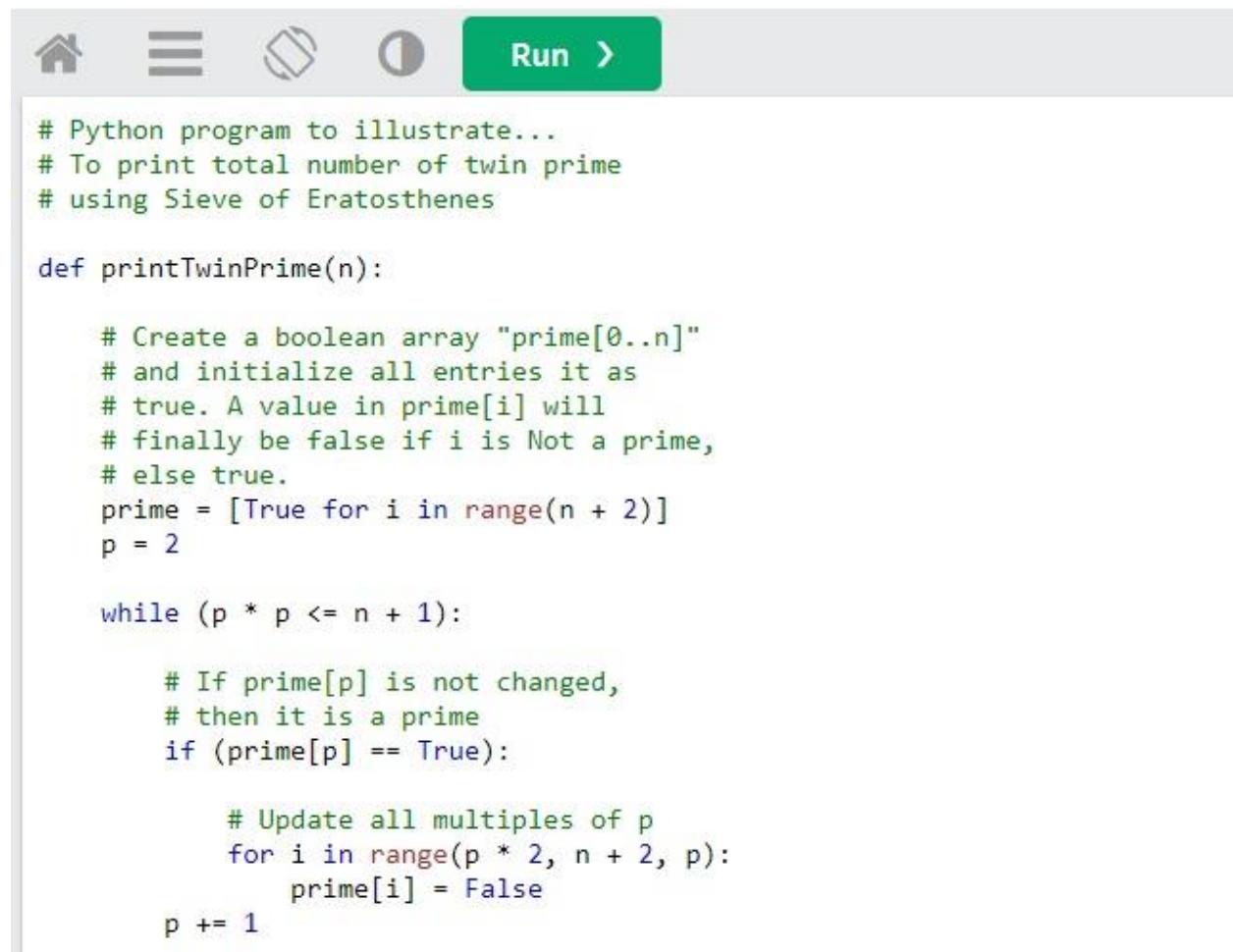
<https://developers.google.com/edu/python/lists>

Unit 3- Experiment no. 1

Unit 3.1- Program 1

“Write a program to print twin primes less than 1000. If two consecutive odd numbers are both prime then they are known as twin primes”.

Program Code



The screenshot shows a Python code editor interface with a toolbar at the top featuring icons for home, file, copy, paste, and run. The main area contains the following Python code:

```
# Python program to illustrate...
# To print total number of twin prime
# using Sieve of Eratosthenes

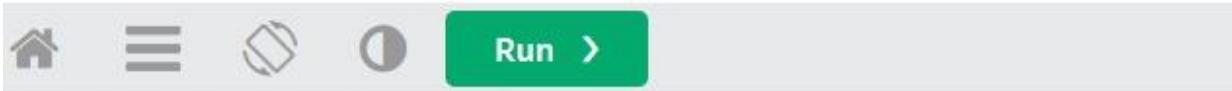
def printTwinPrime(n):

    # Create a boolean array "prime[0..n]"
    # and initialize all entries it as
    # true. A value in prime[i] will
    # finally be false if i is Not a prime,
    # else true.
    prime = [True for i in range(n + 2)]
    p = 2

    while (p * p <= n + 1):

        # If prime[p] is not changed,
        # then it is a prime
        if (prime[p] == True):

            # Update all multiples of p
            for i in range(p * 2, n + 2, p):
                prime[i] = False
            p += 1
```



```
# Python program to illustrate...
# To print total number of twin prime
# using Sieve of Eratosthenes

def printTwinPrime(n):

    # Create a boolean array "prime[0..n]"
    # and initialize all entries it as
    # true. A value in prime[i] will
    # finally be false if i is Not a prime,
    # else true.
    prime = [True for i in range(n + 2)]
    p = 2

    while (p * p <= n + 1):

        # If prime[p] is not changed,
        # then it is a prime
        if (prime[p] == True):

            # Update all multiples of p
            for i in range(p * 2, n + 2, p):
                prime[i] = False
            p += 1

    # check twin prime numbers
    # display the twin prime numbers
    for p in range(2, n-1):
        if prime[p] and prime[p + 2]:
            print(",p,", (p + 2), ")", ,end='')

# driver program
if __name__=='__main__':
    # static input
    n = 25

    # Calling the function
    printTwinPrime(n)
```

Output

```
(3, 5)(5, 7)(11, 13)(17, 19)
```

Unit 3.2-Program 2

“Write a program to implement these formulae of permutations and combinations. Number of permutations of n objects taken r at a time: $p(n, r) = n! / (n-r)!$. Number of combinations of n objects taken r at a time is: $c(n, r) = n! / (r!(n-r)!) = p(n,r) / r!$ Number of combinations of n objects taken r at a time is: $c(n, r) = n! / (r!(n-r)!) = p(n,r) / r!$ ”

Program Code



```
# Python3 program to find the number of permutations of
# n different things taken r at a time
# with k things grouped together

# def to find factorial
# of a number
def factorial(n):

    fact = 1;

    for i in range(2,n+1):
        fact = fact * i;

    return fact;

# def to calculate p(n, r)
def npr(n, r):

    pnr = factorial(n) / factorial(n - r);

    return pnr;
```

```
# def to find the number of permutations of
# n different things taken r at a time
# with k things grouped together
def countPermutations(n, r, k):

    return int(factorial(k) * (r - k + 1) * npr(n - k, r - k));

# Driver code
n = 8;
r = 5;
k = 2;

print(countPermutations(n, r, k));

# this code is contributed by mits
```

Output

```
960
```

Unit 4- Conditional Statements

Unit 4.1- Concept of Indentation

The internal concepts of indentation is referred to the particular spaces at the beginning of the given code line. Where in the other programming language, the indentation in the line code is used for the code readability only but the indentation in the context of Python is most powerful one. In Python, the programmer used the indentation for indicating the particular code of block.

Example

In this above program, the Python code will give the error if the user skips the indentation

```
if 5 > 2:  
    print("Five is greater than two!")
```

Example (Syntax Error)

The user need to use the same number of the spaces in the respective same code block if not then the Python programming will give error to the users

Syntax Error:

```
if 5 > 2:  
    print("Five is greater than two!")  
        print("Five is greater than two!")
```

4.1.1- If Condition

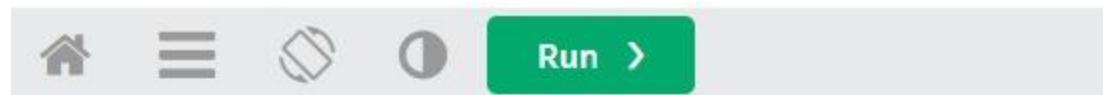
The IF condition is the most basic types of the decision making statement that is mainly used in the program for evaluating whether the particular statement will be executed or not

Syntax

```
if condition:  
    # Statements to execute if  
    # condition is true
```

Example

Input



```
# python program to illustrate If statement  
  
i = 10  
  
if (i > 15):  
    print("10 is less than 15")  
print("I am Not in if")
```

Output

```
I am Not in if
```

Explanation

As in this example, the condition that are present in the if statement is false so the above block of the if statement is executed

Unit 4.2- If Else Condition

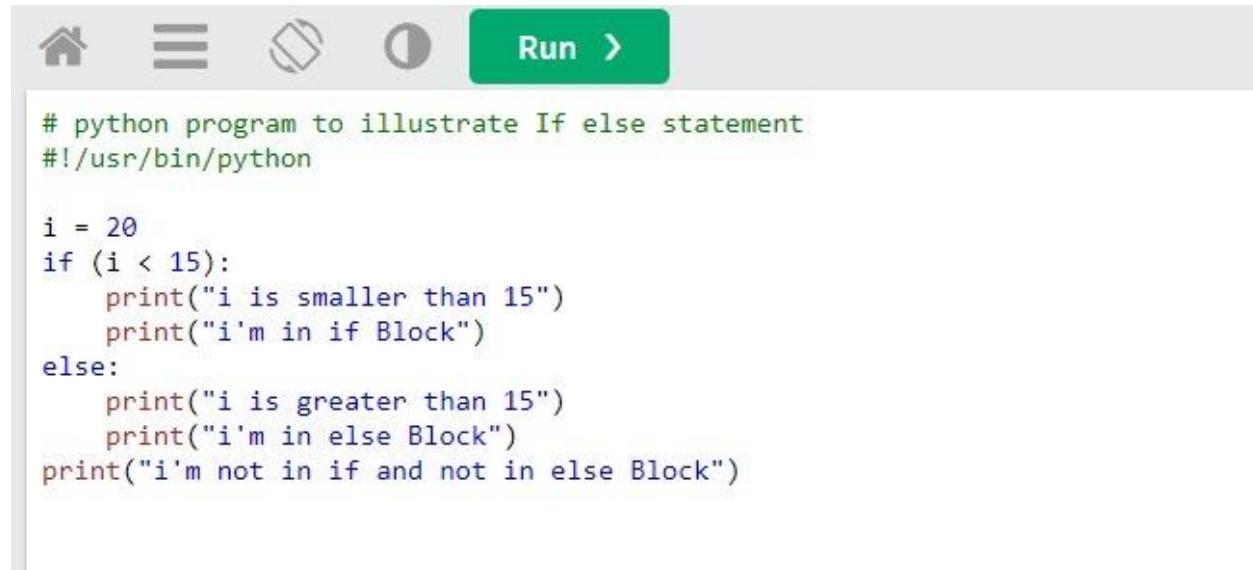
The developer use the else statement with the active usage of the if statement for executing the particular code of the block when the particular condition is false

Syntax

```
if (condition):
    # Executes this block if
    # condition is true
else:
    # Executes this block if
    # condition is false
```

Example

Input



A screenshot of a Python code editor interface. At the top, there are several icons: a house, three horizontal lines, a square with a circle, and a circular arrow. To the right of these is a green 'Run' button with a white arrow. Below the toolbar, the code is displayed in a light gray text area:

```
# python program to illustrate If else statement
#!/usr/bin/python

i = 20
if (i < 15):
    print("i is smaller than 15")
    print("i'm in if Block")
else:
    print("i is greater than 15")
    print("i'm in else Block")
print("i'm not in if and not in else Block")
```

Output

```
i is greater than 15  
i'm in else Block  
i'm not in if and not in else Block
```

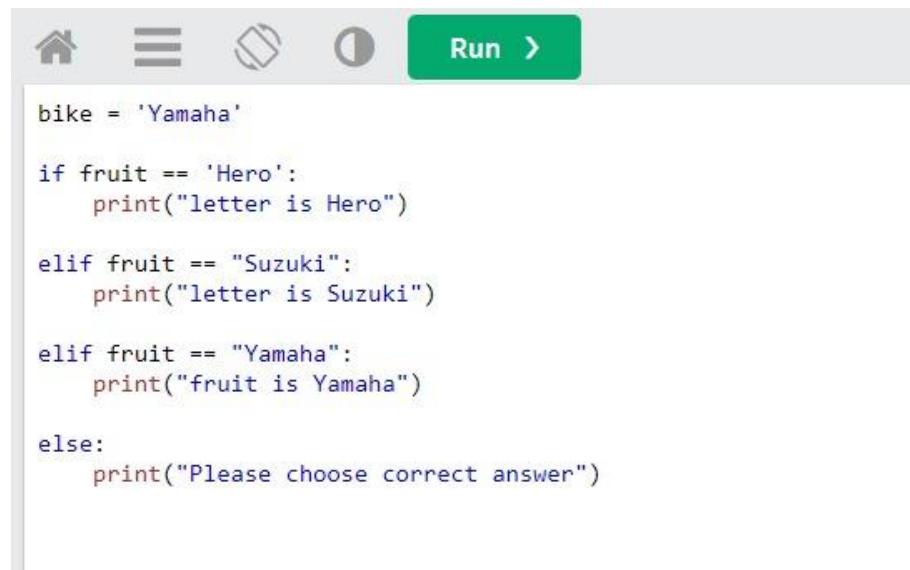
Explanation

The particular block of the code that are followed by the else statement is executed as the specific condition in if statement is dales after calling the statement that is not in the block

Unit 4.3- Switch Condition

If else is another type of method for implementing the switch vase. So it is mainly used for determining whether the particular statement will be performed or not performed

Example



```
home icon  three-line icon  refresh icon  Run >  
  
bike = 'Yamaha'  
  
if fruit == 'Hero':  
    print("letter is Hero")  
  
elif fruit == "Suzuki":  
    print("letter is Suzuki")  
  
elif fruit == "Yamaha":  
    print("fruit is Yamaha")  
  
else:  
    print("Please choose correct answer")
```

Unit 4.4- For Loop

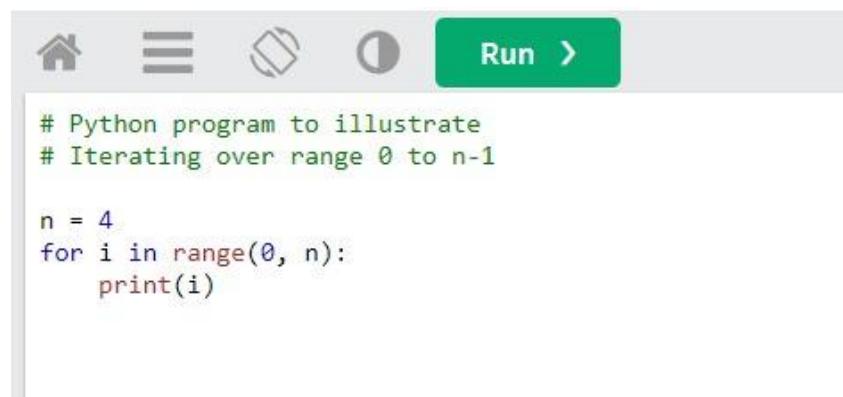
These types of loops are utilized when the client knows ahead of time, how frequently the block needs to execute. That is, the quantity of cycles is known in advance.

Syntax

```
for iterator_var in sequence:  
    statements(s)
```

Example

Input



A screenshot of a Python code editor interface. At the top, there are several icons: a house, three horizontal lines, a circular arrow, and a refresh symbol. To the right of these is a green 'Run' button with a white arrow. Below the toolbar, the code is displayed in a light gray text area:

```
# Python program to illustrate  
# Iterating over range 0 to n-1  
  
n = 4  
for i in range(0, n):  
    print(i)
```

Output

```
0  
1  
2  
3
```

Unit 4.5- While Loop

The while loop is additionally a passage control loop like for loops i.e., it first really takes a look at the condition toward the beginning of the loop

Syntax

```
while expression:  
    statement(s)
```

Example

Input



A screenshot of a Python code editor interface. At the top, there are several icons: a house (Home), three horizontal lines (File/Menu), a document (New/Save), and a circular arrow (Run). To the right of these is a green 'Run' button with a white arrow. Below the toolbar, the code is displayed in a light gray text area:

```
# Python program to illustrate  
# while loop  
count = 0  
while (count < 3):  
    count = count + 1  
    print("Hello Geek")
```

Output

```
Hello Geek  
Hello Geek  
Hello Geek
```

Unit 4.6- Nested loops

The python programming is capable for allowing the one loop under the another loop with the help of the nested loop

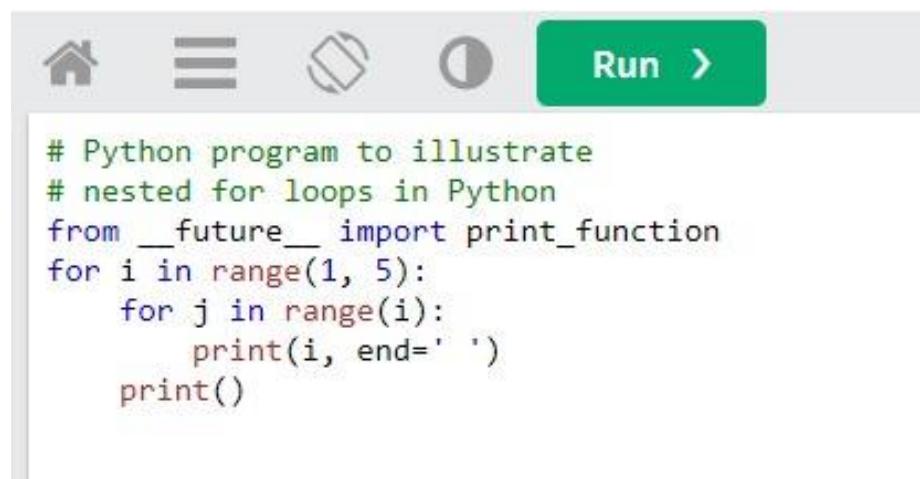
Syntax



```
while expression:  
    while expression:  
        statement(s)  
        statement(s)
```

Example

Input



```
# Python program to illustrate  
# nested for loops in Python  
from __future__ import print_function  
for i in range(1, 5):  
    for j in range(i):  
        print(i, end=' ')  
    print()
```

Output

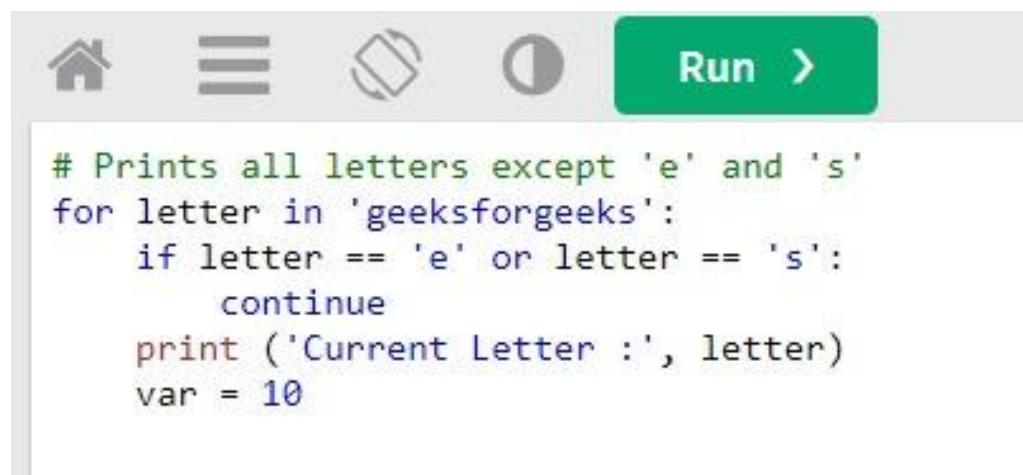
```
1  
2 2  
3 3 3  
4 4 4 4
```

Unit 4.7- Jumping Statements

Continue Statement- This statement return the control to the loop beginning

Example

Input



```
# Prints all letters except 'e' and 's'  
for letter in 'geeksforgeeks':  
    if letter == 'e' or letter == 's':  
        continue  
    print ('Current Letter :', letter)  
var = 10
```

Output

```
Current Letter : g
Current Letter : k
Current Letter : f
Current Letter : o
Current Letter : r
Current Letter : g
Current Letter : k
```

Break Statement- This statement control out of the loop

Example

Input



```
for letter in 'geeksforgeeks':  
  
    # break the loop as soon it sees 'e'  
    # or 's'  
    if letter == 'e' or letter == 's':  
        break  
  
    print 'Current Letter ::', letter
```

Output

```
Current Letter : e
```

Pass Statement- The user mainly passes statement for writing the empty loop

Example

Input



A screenshot of a Python code editor interface. At the top, there are several icons: a house (Home), three horizontal lines (File), a circular arrow (Edit), and a circle with a dot (Run). To the right of these is a green button labeled "Run >". Below the toolbar, the code area contains the following Python script:

```
# An empty loop
for letter in 'geeksforgeeks':
    pass
print 'Last Letter :', letter
```

Output

```
Last Letter : s
```

MCQ

Question 1- Which of the following is the valid Python Statement?

- a) if a>=2
- b) if a>=+2
- c) if a>=-2
- d) None

Question 2- Which of the following is not the decision making statement?

- a) if
- b) for
- c) if else
- d) none

Question 3- Output

```
list1 = [3 , 2 , 5 , 6 , 0 , 7, 9]
```

```
sum = 0
```

```
sum1 = 0
```

```
for elem in list1:
```

```
    if (elem % 2 == 0):
```

```
        sum = sum + elem
```

```
    continue
```

```
    if (elem % 3 == 0):
```

```
        sum1 = sum1 + elem
```

```
print(sum , end=" ")
```

```
print(sum1)
```

a) 812

b) 365

c) 654

d) 565

Question 4- Output

X=3

If x>2 or x<5 and x==6:

Print("ok")

else:

print("no output")

- a) ok
- b) ok ok
- c) ok ok ok
- d) No Output

Question 5- Output

x,y=2,4

if(x+y= =10):

print("true")

else:

print("false")

- a) True
- b) False
- c) Either or
- d) Neither Nor

Question 6- Output

```
x = float(input())
```

```
if(x==1):
```

```
    print("Yes")
```

```
elif (x >= 2):
```

```
    print("Maybe")
```

```
else:
```

```
    print ("No")
```

a) Yes

b) No

c) Maybe

d) Can't Determine

Case Studies

<https://www.studocu.com/in/document/gems-business-school/python/python-conditional-statements-ifelse-elif-switch-case/14808608>

<https://www.toolsqa.com/python/conditional-statements-in-python/>

Reference

<https://realpython.com/python-conditional-statements/>

<https://www.softwaretestinghelp.com/python/python-conditional-statements/>

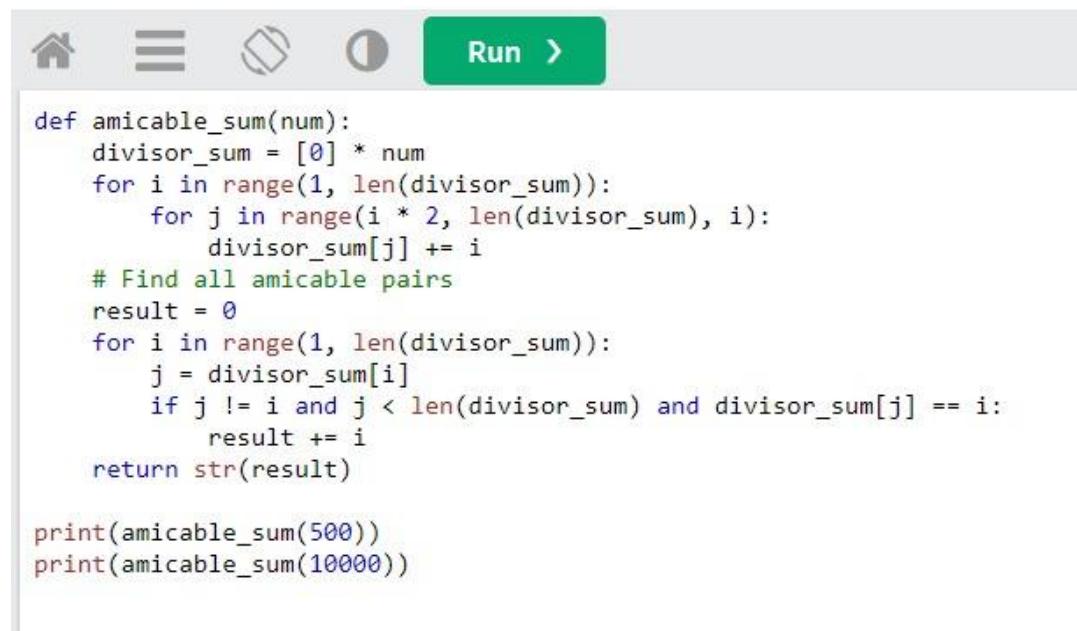
<https://www.w3resource.com/python-exercises/python-conditional-statements-and-loop-exercises.php>

Unit 5- Experiment no. 2

Unit 5.1- Program 3

“Two different numbers are called amicable numbers if the sum of the proper divisors of each is equal to the other number for example 220 and 284 are amicable numbers Sum of proper divisors of $220 = 1+2+4+5+10+11+20+22+44+55+110 = 284$ Sum of proper divisors of $284 = 1+2+4+71+142 = 220$ Write a function to print pairs of amicable numbers in a range”

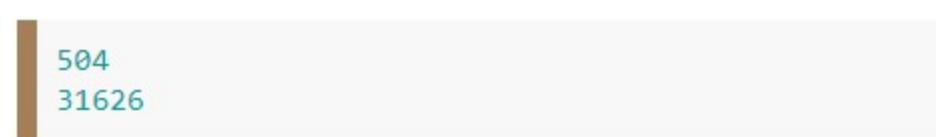
Program Code



```
def amicable_sum(num):
    divisor_sum = [0] * num
    for i in range(1, len(divisor_sum)):
        for j in range(i * 2, len(divisor_sum), i):
            divisor_sum[j] += i
    # Find all amicable pairs
    result = 0
    for i in range(1, len(divisor_sum)):
        j = divisor_sum[i]
        if j != i and j < len(divisor_sum) and divisor_sum[j] == i:
            result += i
    return str(result)

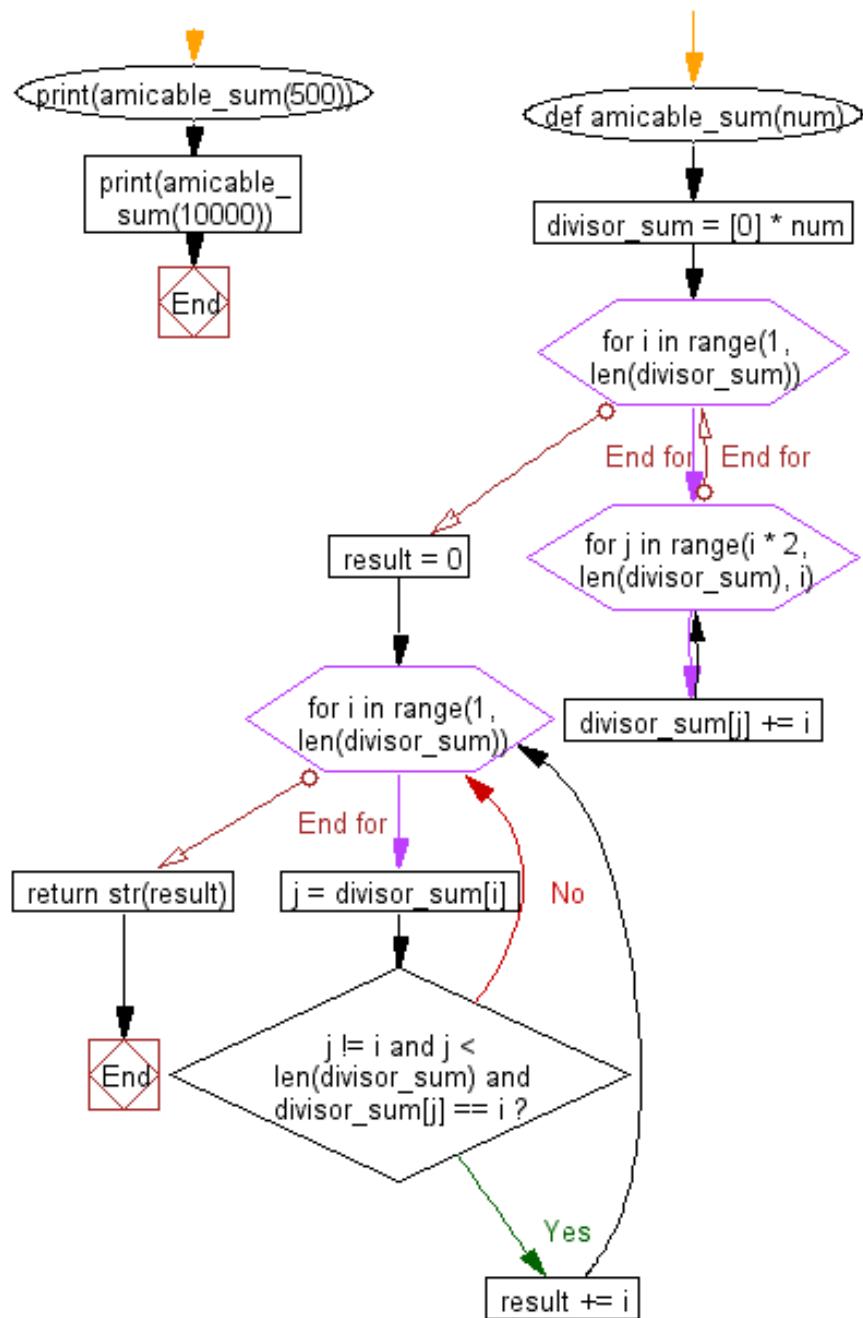
print(amicable_sum(500))
print(amicable_sum(10000))
```

Output



```
504
31626
```

Flowchart of the Code

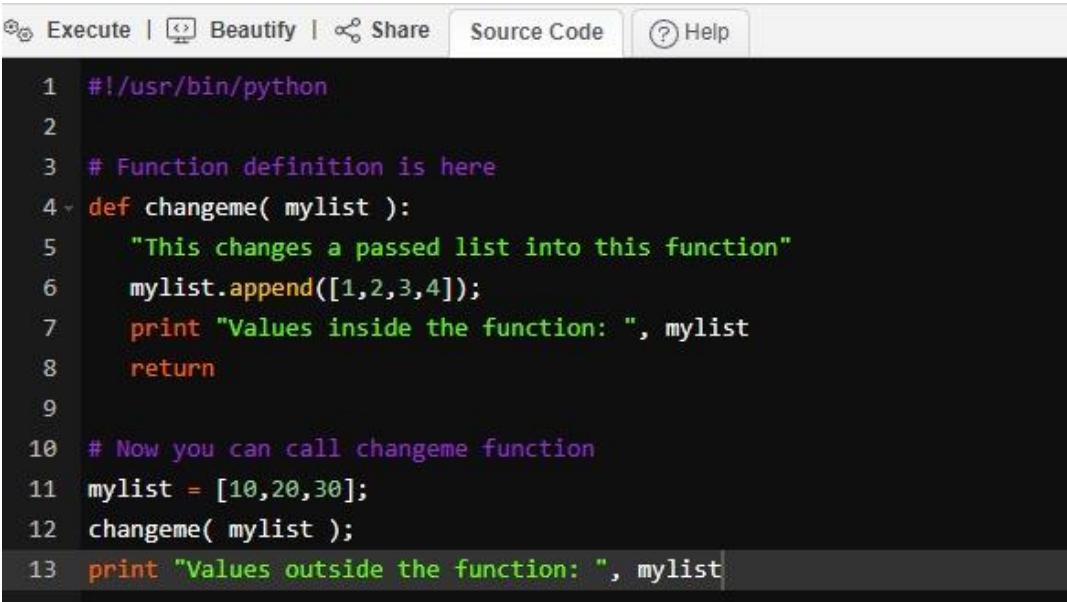


Unit 6- Function

In Python Programming language, the function is referred to the particular group of related statement which performed the particular tasks. Function helps programmers to break down the entire program code into the smaller sets of piece. As the program grows larger, the function makes the program more organised as well as more manageable. With the help of function, the programmer minimizes the repetition by making the entire code reusable.

Unit 6.1- Passing Arguments

The function in Python takes multiple arguments; this argument can be the function, objects or variable. All argument in Python is passed by the reference. It means that if the user changes what the particular parameter refers to within the same function, then the change will be reflect back to the calling function. If we take an example



```
Execute | Beautify | Share | Source Code | Help

1 #!/usr/bin/python
2
3 # Function definition is here
4 def changeme( mylist ):
5     "This changes a passed list into this function"
6     mylist.append([1,2,3,4]);
7     print "Values inside the function: ", mylist
8     return
9
10 # Now you can call changeme function
11 mylist = [10,20,30];
12 changeme( mylist );
13 print "Values outside the function: ", mylist
```

Here in this example, the user maintains reference of the passed objects and then it appends the value to the same objects. So it will give the output as

Terminal

```
Values inside the function: [10, 20, 30, [1, 2, 3, 4]]  
Values outside the function: [10, 20, 30, [1, 2, 3, 4]]
```

Execute | Beautify | Share

Source Code

Help

```
1 #!/usr/bin/python  
2  
3 # Function definition is here  
4 def changeme( mylist ):  
5     "This changes a passed list into this function"  
6     mylist = [1,2,3,4]; # This would assig new reference in mylist  
7     print "Values inside the function: ", mylist  
8     return  
9  
10 # Now you can call changeme function  
11 mylist = [10,20,30];  
12 changeme( mylist );  
13 print "Values outside the function: ", mylist
```

In this example, the parameter mylist is local to the function namely Changeme. So if the user changes the mylist within the function, that does not affect the mylist. So based on that, the respective function will accomplish nothing by simply producing the below mentioned output

```
Values inside the function: [1, 2, 3, 4]  
Values outside the function: [10, 20, 30]
```

Flexible Argument

- The flexible argument is the specific type of name in which the user has provided a name to the variable as it has passed to the function
- The developer uses the kwargs as the dictionary which maps every keyword to the specific value so when the user perform the iteration operation, then it has not seem to any other occurrence of order in which they are printed

Example

Input



```
def myFun(**kwargs):
    for key, value in kwargs.items():
        print("%s == %s" % (key, value))

# Driver code
myFun(first='CU', mid='for', last='CU')
```

Output

```
first == CU
mid == for
last == CU
```

Justification

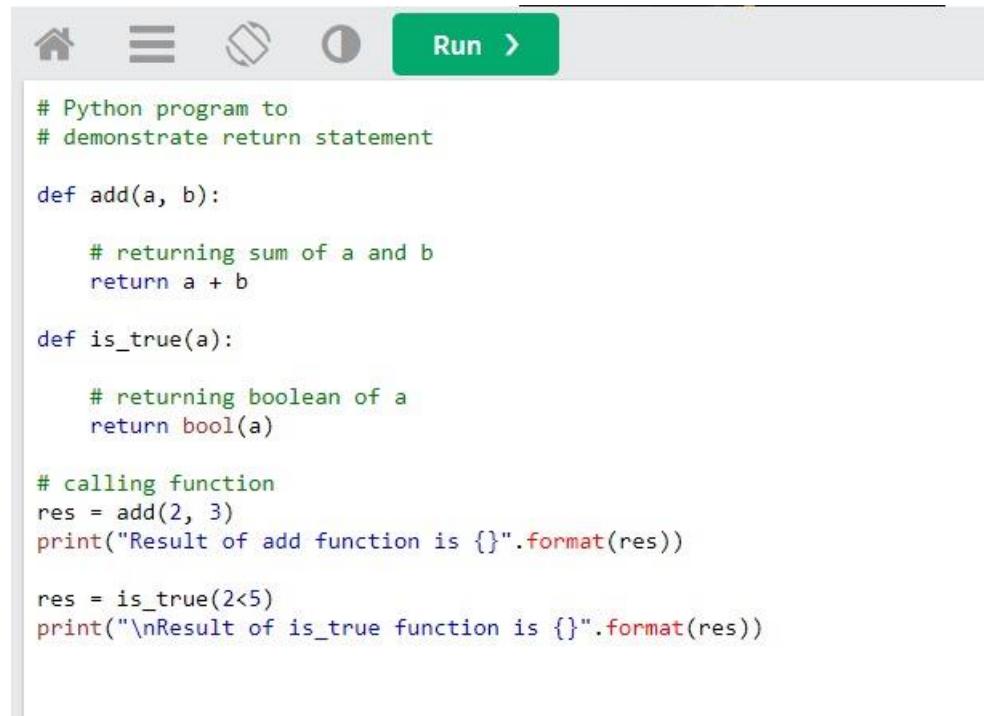
The above example illustrates the *kwargs for a specific variable to the total number of keyword arguments. So in this example, **kwargs has accept the keyword variable length based arguments that has passed by the specific function call. For the first= ‘CU’, based on this first is the key and the last ‘CU’ is the value.

Unit 6.2 Return Values

The return value is mainly used for ending the function call execution and, at the same time return the result of the function, which is followed by the return keyword. So the execution of specific statement after the return statement is not executed, if the particular return is used in the program without any type of expression, then it returned the specific value called none. So the return value is mainly used in the python program for invoking a function by that it can pass the statement that need to be executed

Example 1

Input



```
# Python program to
# demonstrate return statement

def add(a, b):
    # returning sum of a and b
    return a + b

def is_true(a):
    # returning boolean of a
    return bool(a)

# calling function
res = add(2, 3)
print("Result of add function is {}".format(res))

res = is_true(2<5)
print("\nResult of is_true function is {}".format(res))
```

Output

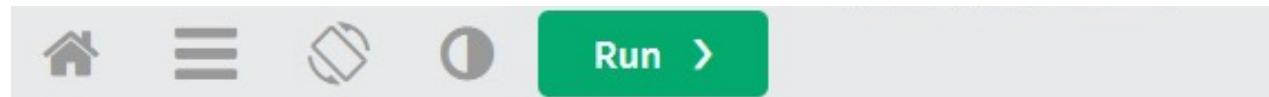
```
Result of add function is 5  
Result of is_true function is True
```

Justification

The above mention program demonstrates the return value or the return statement in the context of python programming

Example 2

Input



```
def my_function(x):  
    return 5 * x  
  
print(my_function(3))  
print(my_function(5))  
print(my_function(9))
```

Output

```
15  
25  
45
```

Justification

The above mention example lets a python function for returning a value with the help of the return statement

Unit 6.3 Passing a List

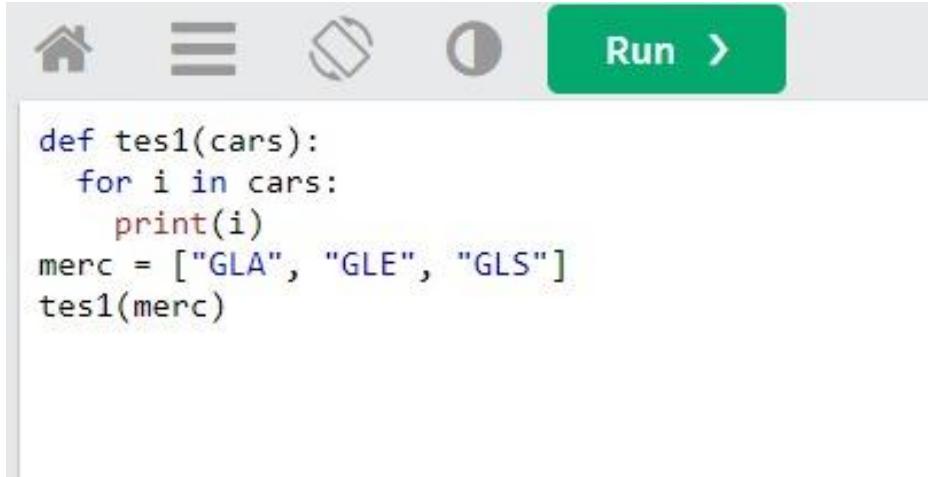
There are a total of three different ways in which this book describes passing a specific list to the python function

1. Usage of List and then pass it as arguments to the python function

Any type of argument passed to the function is considered the same data type inside the functions. So after called these data types inside the function, the list remains as the list and it is not changed to any data types

Example

Input



```
def tes1(cars):
    for i in cars:
        print(i)
merc = ["GLA", "GLE", "GLS"]
tes1(merc)
```

Output

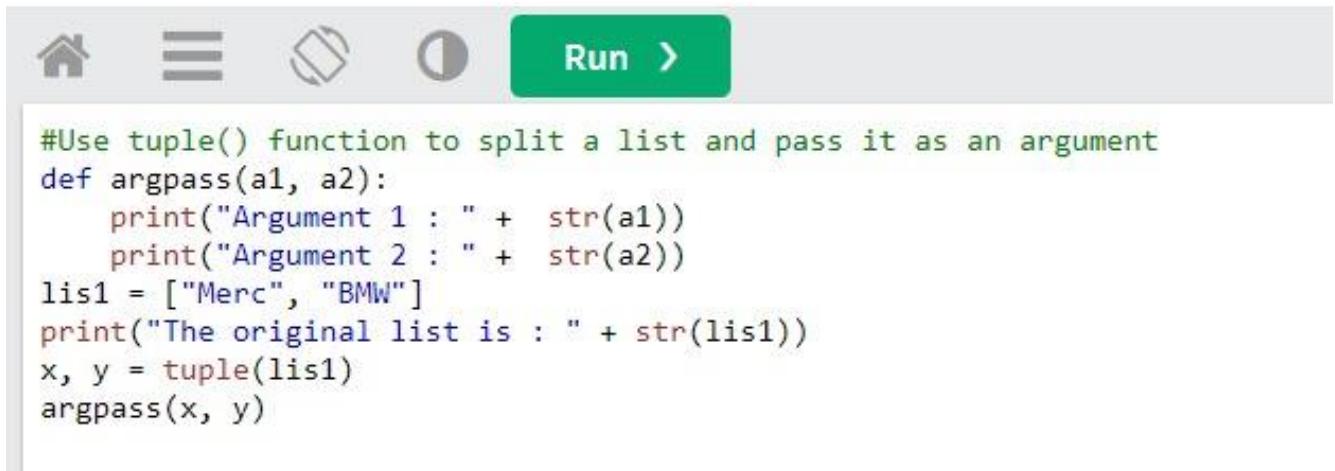
```
GLA
GLE
GLS
```

2. Tuple Function

The tuple function in python splitting the each elements of the particular list by concerting the entire list to the tuple and at the same time, this element is tackled as the separate variable that is passed as the arguments to the python function

Example

Input



```
#Use tuple() function to split a list and pass it as an argument
def argpass(a1, a2):
    print("Argument 1 : " + str(a1))
    print("Argument 2 : " + str(a2))
lis1 = ["Merc", "BMW"]
print("The original list is : " + str(lis1))
x, y = tuple(lis1)
argpass(x, y)
```

Output

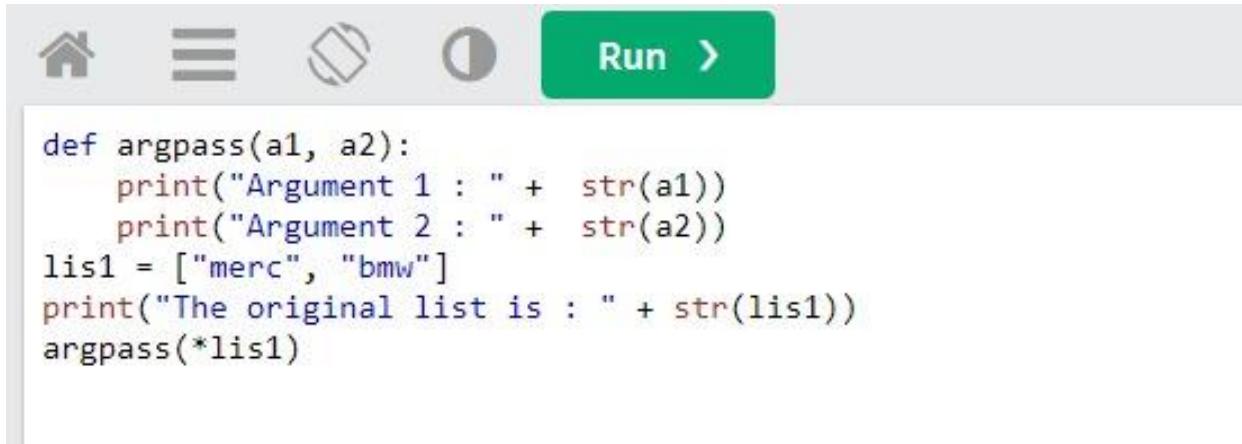
```
The original list is : ['Merc', 'BMW']
Argument 1 : Merc
Argument 2 : BMW
```

3. * Operator

The * operator is capable of unpacking the python list into the separate elements which has been tackled by the individuals variable, and then it is passed as the arguments in the python function

Example

Input



```
def argpass(a1, a2):
    print("Argument 1 : " + str(a1))
    print("Argument 2 : " + str(a2))
lis1 = ["merc", "bmw"]
print("The original list is : " + str(lis1))
argpass(*lis1)
```

Output

```
The original list is : ['merc', 'bmw']
Argument 1 : merc
Argument 2 : bmw
```

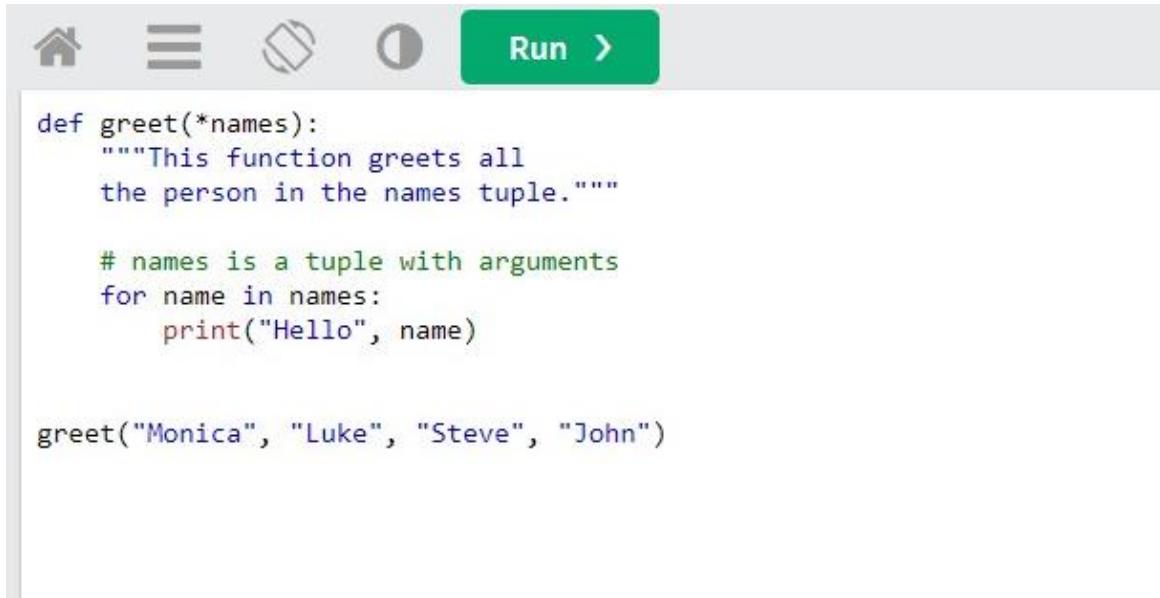
Unit 6.4 Passing an Arbitrary Number of Arguments

Sometimes, the python developer or the user does not know in advance that they need how many numbers of arguments are passed into the function. Python programming allows the users for handling different types of situation through the function call by the help of the arbitrary numbers of arguments.

In the context of the python function definition, the developer uses the asterisk before the name of the parameter to denote the arbitrary arguments.

Example

Input



The screenshot shows a Python code editor interface. At the top, there are several icons: a house, three horizontal lines, a circular arrow, and a refresh symbol. To the right of these is a green 'Run' button with a white arrow pointing right. Below the toolbar, the code is displayed:

```
def greet(*names):
    """This function greets all
    the person in the names tuple."""

    # names is a tuple with arguments
    for name in names:
        print("Hello", name)

greet("Monica", "Luke", "Steve", "John")
```

Output

```
Hello Monica
Hello Luke
Hello Steve
Hello John
```

Justification

In the above example, the developer calls this function with multiple type arguments. So these types of arguments have been wrapped into the python tuple before it has been passed to the respective function. So inside the function, the developer uses the for loop for retrieving the entire arguments block

Unit 6.5 Function in Modules

The module is the file container of python code that is executed for the user specified code. So the module is the specific Python oriented object with the arbitrarily named arguments in which the user can use it both for binding as well as referencing. The module is the specific type of file that contain of different Python code.

Dir () Function in Module

The dir () is the python most useful built-in function for listing the entire name of the function in a python module.

Example

Input



```
import platform

x = dir(platform)
print(x)
```

Output

```
['DEV_NULL', '_UNIXCONFDIR', 'WIN32_CLIENT_RELEASES', 'WIN32_SERVER_RELEASES', '
```

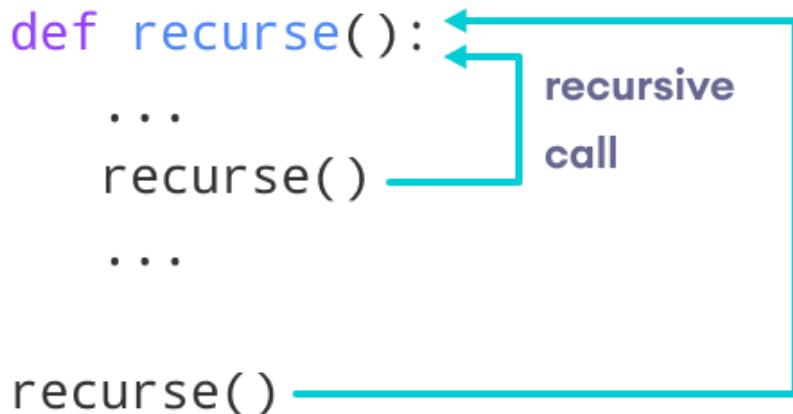
Justification

In this example, the list function list the all defined names that has belonged to the specific platform module.

Unit 6.6 Recursive Function

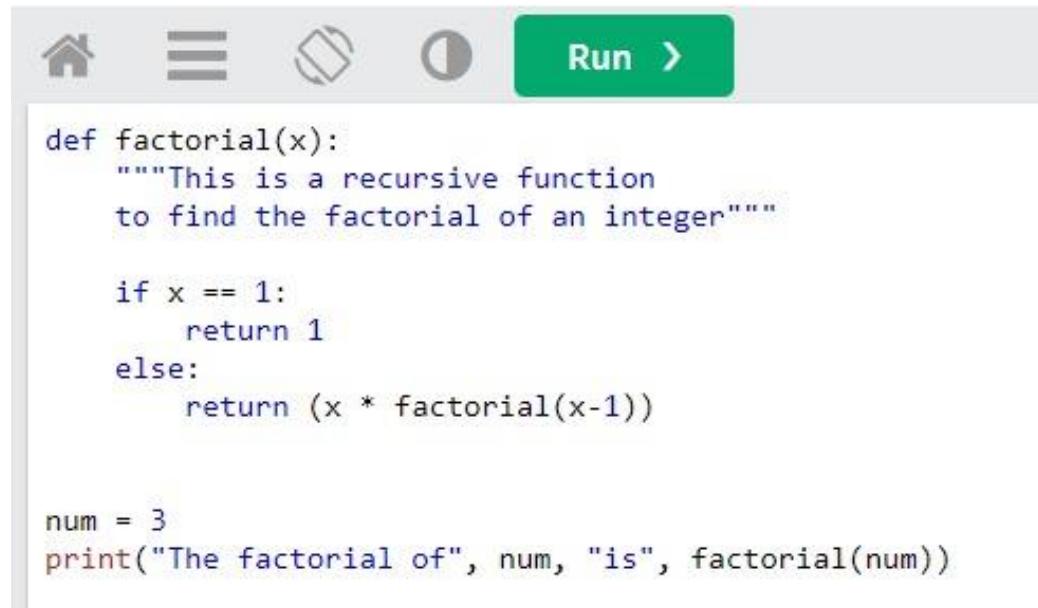
Definition of Recursive Function

In Python, one function is capable of calling the other functions and it is possible that the respective function even call for them. These types of special constructs are referred as the recursive function. In this below image, it shown the working functionality of the recursive function known as the recurse



Example

Input



The screenshot shows a Python code editor interface. At the top, there are several icons: a house, three horizontal lines, a circular arrow, and a play button. To the right of these is a green 'Run >' button. Below the toolbar, the code is displayed:

```
def factorial(x):
    """This is a recursive function
    to find the factorial of an integer"""

    if x == 1:
        return 1
    else:
        return (x * factorial(x-1))

num = 3
print("The factorial of", num, "is", factorial(num))
```

Output

```
The factorial of 3 is 6
```

Justification

In this example, the programmer performs the recursive function for finding out the factorials of the integer. In this above function, the factorial () is the specific recursive function that called by itself.

Unit 6.7 Nested Functions

Definition of Nested Function

Nested function is capable for accessing the different types of a variable inside the enclose scope. So the nested function is used so that it is protected from different aspects that are performed inside the outer functions. This entire process is known as Encapsulation.

Example

Input



```
# Python program to illustrate
# nested functions
def outerFunction(text):
    text = text

    def innerFunction():
        print(text)

    innerFunction()


```

Output

```
This code will return nothing when executed.
```

Justification

In this example, the innerFunction() is defined inside the outerFunction() by making it into the inner function. For calling the innerFunction(), the user first need to called the outerFunction(). Then the outerFunction() call the innerFunction() as it is define inside this function.

Unit 6.8 Default and Flexible Arguments

Unit 6.8.1- Default Arguments

Definition of Default Arguments

Default arguments in python occur when the developer assigns the respective default value as none, and then it checks in that function whether the expected list as well as arguments is none or not. If the respected value is none, then developer assigns the value with the list or the dictionary that are based on user requirement.

Example

Input



```
print('#list')
def appendItem(itemName, itemList=None):
    if itemList == None:
        itemList = []
    itemList.append(itemName)
    return itemList

print	appendItem('notebook'))
print	appendItem('pencil'))
print	appendItem('eraser'))

# using None as value of default parameter

print('\n\n#dictionary')
def addItemAtDictionary(itemName, quantity, itemList = None):
    if itemList == None:
        itemList = {}
    itemList[itemName] = quantity
    return itemList

print(addItemAtDictionary('notebook', 4))
print(addItemAtDictionary('pencil', 1))
print(addItemAtDictionary('eraser', 1))
```

Output

```
#list
['notebook']
['pencil']
['eraser']

#dictionary
{'notebook': 4}
{'pencil': 1}
{'eraser': 1}
```

Justification

As the readers clearly see from the above example is that in every time when the function is called as well as the list is not passing as the arguments to the respective function then it will develop a new list or new dictionary

Unit 6.8.2- Flexible Arguments

Flexible arguments occur in the Python function after the required arguments when the user defines the respective function. Flexible argument follows the **kwargs keyword which has passed to the particular function by providing the capability of optional arguments within the functions.

Example 1

Input



```
def myFun(arg1, **kwargs):
    for key, value in kwargs.items():
        print("%s == %s" % (key, value))

# Driver code
myFun("Hi", first='CU', mid='for', last='CU')
```

Output

```
first == CU
mid == for
last == CU
```

Justification

In this example, the user passed the non keyword based argument that is accepted by the specific positional arguments in this example it is arg1 in the myFun and user passed accepted by the **kwargs also.

Unit 6.9 Lambda Function

Definition of Lambda Function

This function is a specific function which accepts as well as stores the intended result of that expression

Example

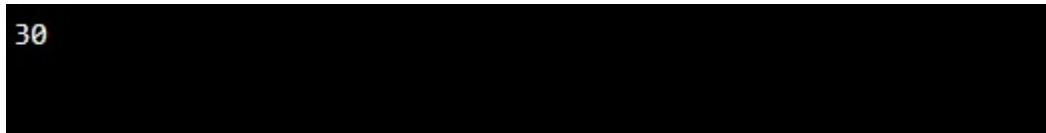
Input



A screenshot of a code editor interface. At the top, there is a toolbar with icons for file, edit, and run. A green 'Run' button is highlighted. Below the toolbar is a code editor window containing the following Python code:

```
x = lambda a, b: a * b
print(x(5, 6))
```

Output



A screenshot of a terminal window. The output of the previous code execution is displayed:

```
30
```

Justification

The lambda function can take any number of arguments. In this example, the lambda function multiplies the argument a with the argument b and then returns the result as the output

Unit 6.10 Map () Function

Definition of Map Function

This function is useful for every item in an iterable. Then the item sends to that function as a functional parameter.

Syntax

Map (fun, iter)

Example

Input



```
# Python program to demonstrate working
# of map.

# Return double of n
def addition(n):
    return n + n

# We double all numbers using map()
numbers = (1, 2, 3, 4)
result = map(addition, numbers)
print(list(result))
```

Output

```
[2, 4, 6, 8]
```

Unit 6.11 Filter Function

Definition of Filter Function

It can be applied to any type of iterable, for example, to create a new iterator, dictionary or list. Thus the new iterator filters out the particular elements based on the specific condition that the user provides it very efficiently

Syntax

Filter (function, sequence)

Example

Input



```
# function that filters vowels
def fun(variable):
    letters = ['a', 'e', 'i', 'o', 'u']
    if (variable in letters):
        return True
    else:
        return False

# sequence
sequence = ['g', 'e', 'e', 'j', 'k', 's', 'p', 'r']

# using filter function
filtered = filter(fun, sequence)

print('The filtered letters are:')
for s in filtered:
    print(s)
```

Output

```
The filtered letters are:  
e  
e
```

Justification

The above example demonstrates the filter function, which filters among the vowels

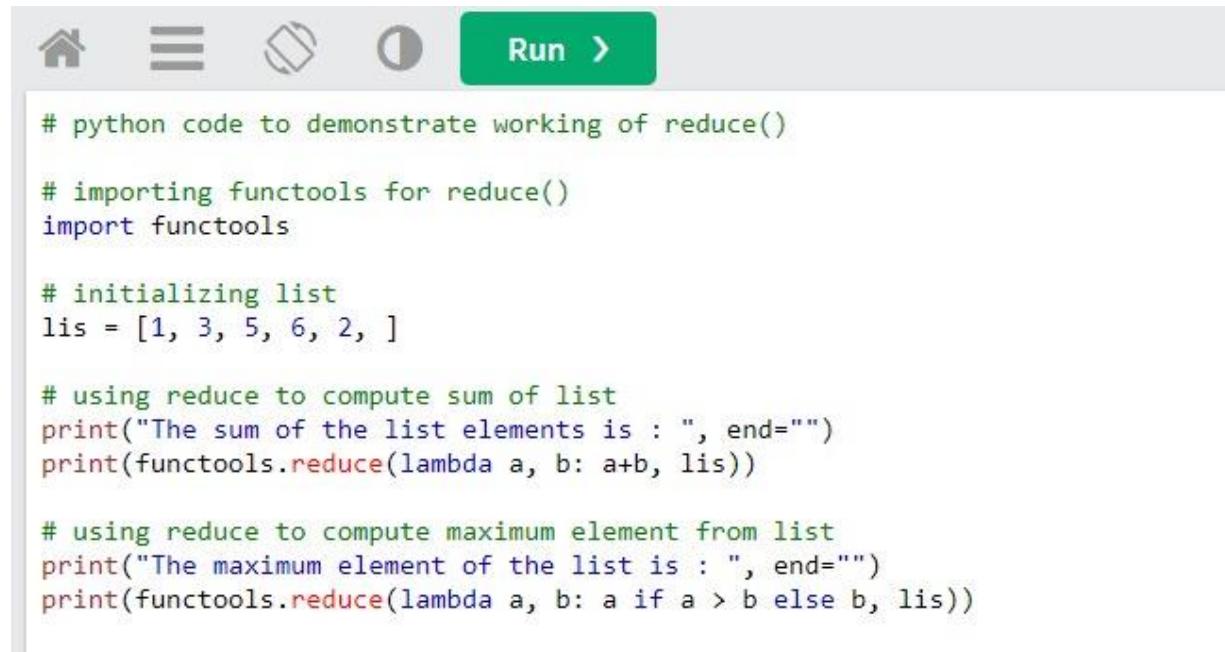
Unit 6.12 Reduce () Function

Definition of Reduce Function

Reduce () function in Python is mainly used for implementing the mathematical technique known as folding. This function is useful when the user applies that function to an iterable and then reduces it to the single cumulative value.

Example

Input



```
# python code to demonstrate working of reduce()

# importing functools for reduce()
import functools

# initializing list
lis = [1, 3, 5, 6, 2, ]

# using reduce to compute sum of list
print("The sum of the list elements is : ", end="")
print(functools.reduce(lambda a, b: a+b, lis))

# using reduce to compute maximum element from list
print("The maximum element of the list is : ", end="")
print(functools.reduce(lambda a, b: a if a > b else b, lis))
```

Output

```
The sum of the list elements is : 17
The maximum element of the list is : 6
```

Unit 6.13 Python Inbuilt Functions

Math Function

The Python Math function has handled different types of values within the specified range of integer as well as float types. So the, Python has utilized different math function that has allowed the users for performing the different mathematical based tasks on the numbers itself

PI () Function

This Function has returned the values of PI

Input



A screenshot of a Python code editor interface. At the top, there are several icons: a house, three horizontal lines, a clipboard, and a circular progress bar. To the right of these is a green 'Run' button with a white arrow. Below the toolbar, the code is displayed in a text area:

```
# Import math Library
import math

# Print the value of pi
print (math.pi)
```

Output

```
3.141592653589793
```

Min () and Max ()

These two function have been used for finding the lowest as well as highest possible values in the list of the arguments

Max ()

Input

```
# Python code to demonstrate the working of  
# max()  
  
# printing the maximum of 4,12,43.3,19,100  
print("Maximum of 4,12,43.3,19 and 100 is : ",end="")  
print (max( 4,12,43.3,19,100 ) )
```

Output

```
Maximum of 4,12,43.3,19 and 100 is : 100
```

Min ()

Input

```
# Python code to demonstrate the working of  
# min()  
  
# printing the minimum of 4,12,43.3,19,100  
print("Minimum of 4,12,43.3,19 and 100 is : ",end="")  
print (min( 4,12,43.3,19,100 ) )
```

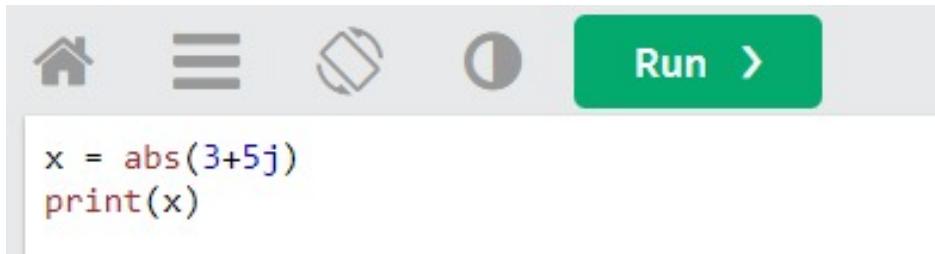
Output

```
Minimum of 4,12,43.3,19 and 100 is : 4
```

Abs ()

This function has returned the absolute values for any numbers

Input



A screenshot of a Python code editor interface. At the top, there are several icons: a house, three horizontal lines, a clipboard, and a circular progress bar. To the right of these is a green "Run" button with a white arrow. Below the toolbar, the code is displayed in a text area:

```
x = abs(3+5j)
print(x)
```

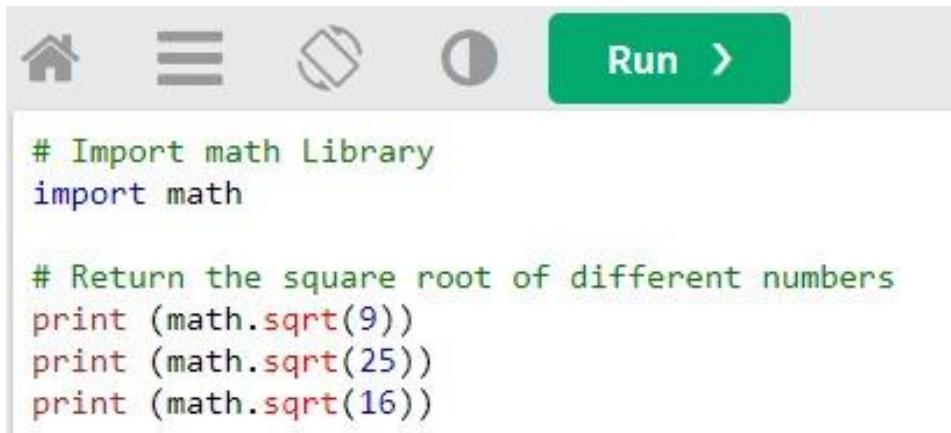
Output

```
5.830951894845301
```

Sqrt()

This function has returned the square root of the numbers

Input



A screenshot of a Python code editor interface. At the top, there are several icons: a house, three horizontal lines, a clipboard, and a circular progress bar. To the right of these is a green "Run" button with a white arrow. Below the toolbar, the code is displayed in a text area:

```
# Import math Library
import math

# Return the square root of different numbers
print (math.sqrt(9))
print (math.sqrt(25))
print (math.sqrt(16))
```

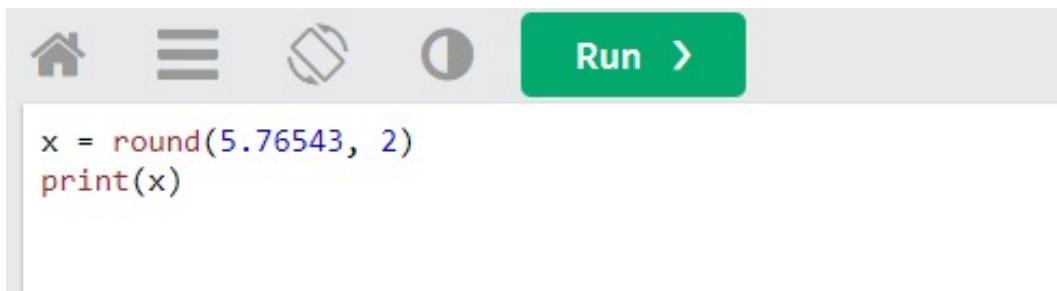
Output

```
3.0  
5.0  
4.0
```

Round ()

This function has rounded the specific floating point based numbers to the nearest integer

Input



A screenshot of a Python code editor interface. At the top, there are several icons: a house (Home), three horizontal lines (File), a square with a diagonal (Edit), and a circle (Run). To the right of these is a green 'Run' button with a white arrow. Below the toolbar, the code is displayed in a text area:

```
x = round(5.76543, 2)
print(x)
```

Output

```
5.77
```

MCQ

Question 1- Which of the following keyword is useful for the Function?

- a) def
- b) int
- c) float
- d) null

Question 2- What are most important two main function?

- a) Built Function
- b) User Defined Function
- c) Both
- d) None

Question 3- In which area of the code, the function is fully defined?

- a) Module
- b) Class
- c) All of the above
- d) None

Question 4- What is called when the function is defined within the class?

- a) Method
- b) Class
- c) Array
- d) Objects

Question 5- Which is the following referred to as the mathematical based function

- a) sqrt
- b) main
- c) pi
- d) none

Question 6- Lambda is the statement?

- a) False
- b) Does not know
- c) True
- d) Machine Dependent

Question 7- Output

```
def add(a, b):  
    return a+5, b+5  
  
result = add(3, 2)  
print(result)
```

- a) (8, 7)
- b) 13
- c) 12
- d) 5

Question 8- Python function always returns a specific Value

- a) False
- b) Does not know
- c) True
- d) Either True or False

Question 9- Which is the output of the following function call?

- a) Name Error
- b) 10
- c) 20
- d) 30

Case Studies

<https://python.plainenglish.io/python-beginner-tutorial-case-studies-for-data-scientists-to-start-right-away-b426f3ca950d>

<http://alumni.media.mit.edu/~tpminka/patterns/python/>

Reference

https://www.w3schools.com/python/python_functions.asp

<https://www.programiz.com/python-programming/function>

https://www.tutorialspoint.com/python/python_functions.htm

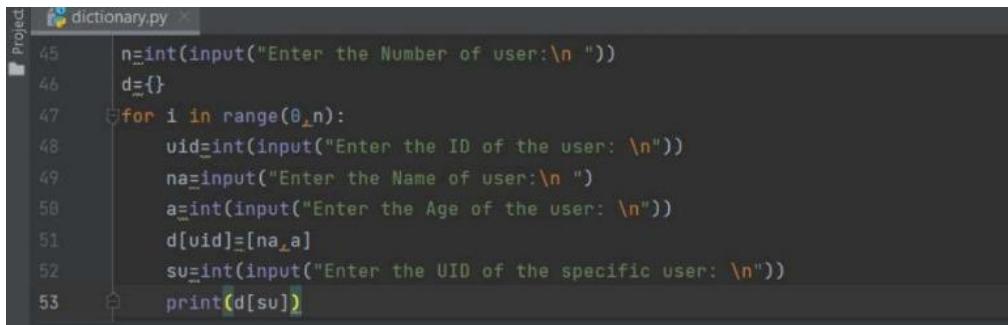
<https://www.geeksforgeeks.org/python-functions/>

Unit 7- Experiment no. 3

Unit 7.1- Program 4

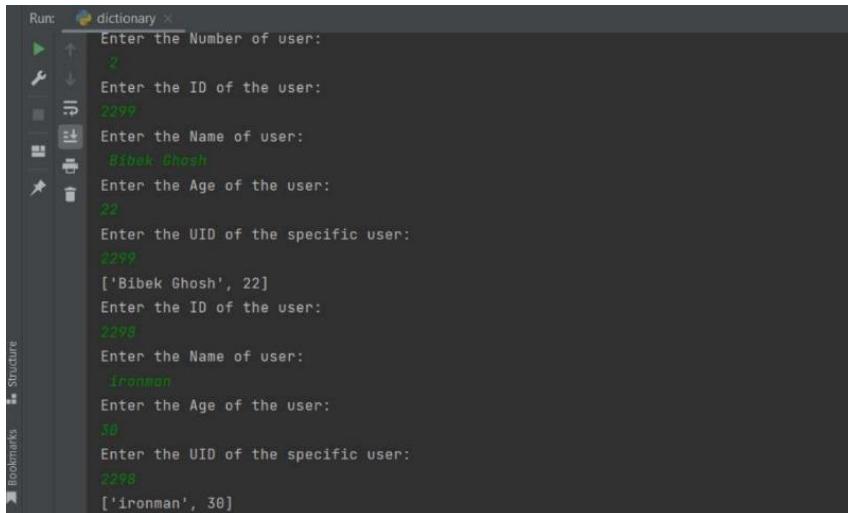
“WAP to get user id, user name, and user age from user and based on the entered id print the details for particular user. Hint: use dictionary”

Program Code



```
Project: dictionary.py
45     n=int(input("Enter the Number of user:\n "))
46     d={}
47     for i in range(0,n):
48         uid=int(input("Enter the ID of the user:\n "))
49         na=input("Enter the Name of user:\n ")
50         a=int(input("Enter the Age of the user: \n"))
51         d[uid]=[na,a]
52         su=int(input("Enter the UID of the specific user: \n"))
53     print(d[su])
```

Output



```
Run: dictionary.x
Enter the Number of user:
2
Enter the ID of the user:
2299
Enter the Name of user:
Bibek Ghosh
Enter the Age of the user:
22
Enter the UID of the specific user:
2299
['Bibek Ghosh', 22]
Enter the ID of the user:
2298
Enter the Name of user:
ironman
Enter the Age of the user:
30
Enter the UID of the specific user:
2298
['ironman', 30]
```

Unit 8- Class

Unit 8.1- Creating and using a class

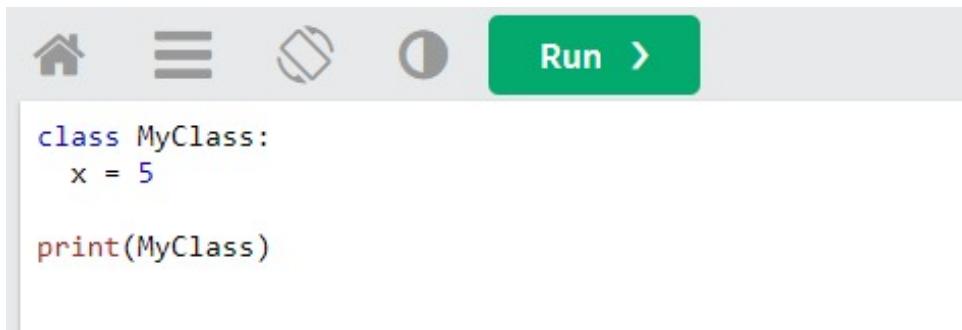
The class is the user defined based blueprint in which different types of object is developed. Class in Python is capable for both bundling data as well as function together. Developing the new class creates the new type of the object which allows different types of new object of that type to be created

Syntax

```
class ClassName:  
    # Statement
```

Example

Input



A screenshot of a Python code editor interface. At the top, there are several icons: a house (Home), three horizontal lines (File), a document (Edit), and a circular arrow (Run). To the right of these is a green 'Run' button with a white arrow. Below the toolbar, the code is displayed in a light gray text area:

```
class MyClass:  
    x = 5  
  
print(MyClass)
```

Output

```
<class '__main__.MyClass'>
```

Justification

In this example, the user created a class by naming it as MyClass with the specific property name X

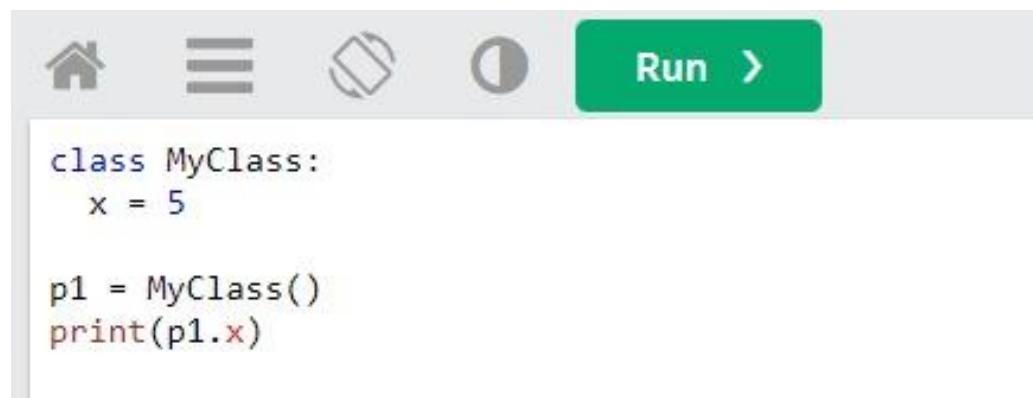
Unit 8.2- Working with classes and Instances

Instance

Based on the previous example, the user creates instances by the name of P1 and then prints the value of x

Example

Input



The screenshot shows a Python code editor interface. At the top, there are several icons: a house (Home), three horizontal lines (File), a circular arrow (Edit), and a circle with a dot (Run). To the right of these is a green button labeled "Run >". Below the toolbar, the code is displayed in a text area:

```
class MyClass:  
    x = 5  
  
p1 = MyClass()  
print(p1.x)
```

Output

5

Init () Function

For working with the both class as well as objects, all of the class has their inbuilt init () that has been executed when the particular class is initiated. The init () in Python is same as the C++

constructor in the context of object oriented based approach. The init () first lets the respective class to initialize the attributes of the objects and then it can serve with no other purposes because this function is only used within the class

Example

Input

```
class Person:  
    def __init__(self, name, age):  
        self.name = name  
        self.age = age  
  
p1 = Person("John", 36)  
  
print(p1.name)  
print(p1.age)
```

Output

```
John  
36
```

Justification

In this example, the user first develop the specific class name Person and then apply the init () function for assigning value to the both age as well as name

Init with Inheritance

Inheritance is the particular capability of the one class for deriving or inheriting the specific properties from the different class. In this below example, the author discuss about how init () with the inheritance

Example

Input

```
# Python program to
# demonstrate init with
# inheritance

class A(object):
    def __init__(self, something):
        print("A init called")
        self.something = something

class B(A):
    def __init__(self, something):
        # Calling init of parent class
        A.__init__(self, something)
        print("B init called")
        self.something = something

obj = B("Something")
```

Output

```
A init called
B init called
```

Justification

In this example, at first constructor of the parent class called first. But in the context of Python, it is not important that the constructor of the parent class will be called first. The specific order in which `init()` is called for the parent or respective child class has been modified. This step is execute by simply call the constructor of the parent class after the body of the child class constructor

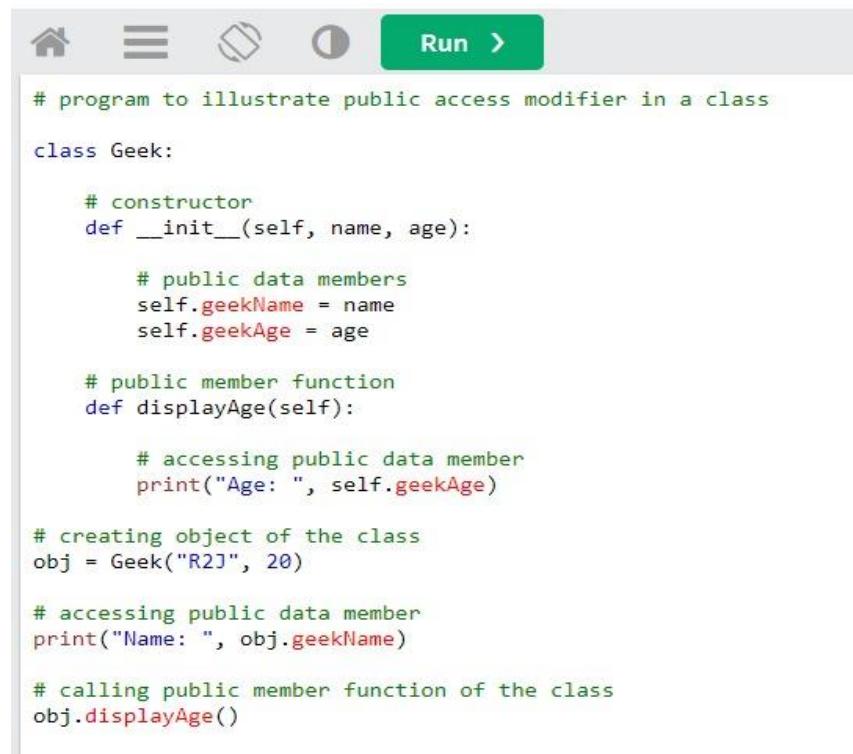
Unit 8.3- Public and Private Members

Public Members

The members of the class which is declared public are accessible very easily from any parts of the program. So all data members and respective member function of the specific class is public in default

Example

Input



A screenshot of a code editor window. At the top, there are several icons: a house (Home), three horizontal lines (File), a circular arrow (Edit), and a play button (Run). To the right of these is a green 'Run' button with a white arrow. Below the toolbar is a code editor area containing the following Python code:

```
# program to illustrate public access modifier in a class

class Geek:

    # constructor
    def __init__(self, name, age):

        # public data members
        self.geekName = name
        self.geekAge = age

    # public member function
    def displayAge(self):

        # accessing public data member
        print("Age: ", self.geekAge)

# creating object of the class
obj = Geek("R2J", 20)

# accessing public data member
print("Name: ", obj.geekName)

# calling public member function of the class
obj.displayAge()
```

Output

```
Name: R2J  
Age: 20
```

Justification

In this example, both geekName and the geekAge are the public data members

Private Members

The members of the class which is declared private are accessible within the specific class only not the part of the entire program.

Example

Input

```
# program to illustrate private access modifier in a class

class Geek:

    # private members
    __name = None
    __roll = None
    __branch = None

    # constructor
    def __init__(self, name, roll, branch):
        self.__name = name
        self.__roll = roll
        self.__branch = branch

    # private member function
    def __displayDetails(self):

        # accessing private data members
        print("Name: ", self.__name)
        print("Roll: ", self.__roll)
        print("Branch: ", self.__branch)
```

```
# public member function
def accessPrivateFunction(self):

    # accessing private member function
    self.__displayDetails()

# creating object
obj = Geek("R2J", 1706256, "Information Technology")

# calling public member function of the class
obj.accessPrivateFunction()
```

Output

```
Name: R2J
Roll: 1706256
Branch: Information Technology
```

Justification

In this example, name, branch and roll are the private members

Unit 8.4- Inheritance

Inheritance is the specific capability of the one class for deriving as well as inheriting to the specific properties from the another class

Importance of Inheritance

Inheritance represents different types of real world based relationship very easily

In addition, inheritance provides the code reusability

Inheritance is in transitive in nature for example if the class B has inherited from the class A then the all the respective subclass of the B will inherit from class A

Example 1

Input

```
# A Python program to demonstrate inheritance

class Person(object):

    # Constructor
    def __init__(self, name, id):
        self.name = name
        self.id = id

    # To check if this person is an employee
    def Display(self):
        print(self.name, self.id)

# Driver code
emp = Person("Satyam", 102) # An Object of Person
emp.Display()
```

Output

```
Satyam 102
```

Justification

This example implements a Person Class with the Display Methods

Example 2

Input

```
class Person(object):

    # Constructor
    def __init__(self, name):
        self.name = name

    # To get name
    def getName(self):
        return self.name

    # To check if this person is an employee
    def isEmployee(self):
        return False

# Inherited or Subclass (Note Person in bracket)
class Employee(Person):

    # Here we return true
    def isEmployee(self):
        return True

# Driver code
emp = Person("Geek1") # An Object of Person
print(emp.getName(), emp.isEmployee())

emp = Employee("Geek2") # An Object of Employee
print(emp.getName(), emp.isEmployee())
```

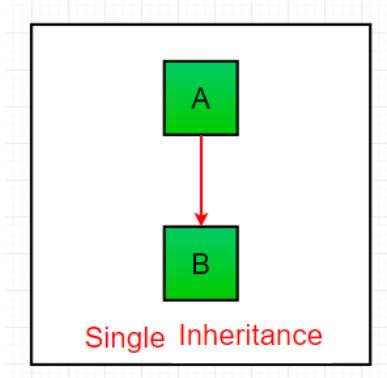
Output

```
Geek1 False
Geek2 True
```

Unit 8.5- Types of Inheritance

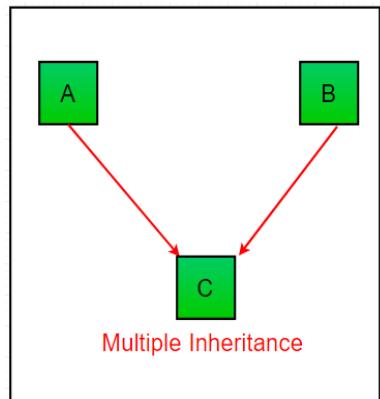
Single Inheritance

Single inheritance has enabled the derived class for inheriting the properties from the single parent class so by that it has maximized the reusability of the code



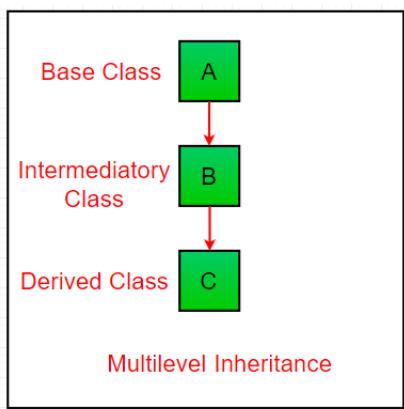
Multiple Inheritances

When the specific class has been derived from more than one base class then it has been referred as multiple inheritances



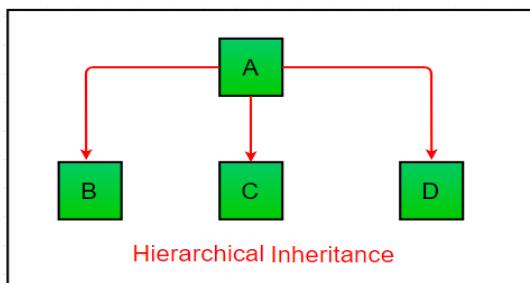
Multilevel Inheritance

In the context of multilevel inheritance, different features of the base class as well as respective derived class has inherited into the new derived class



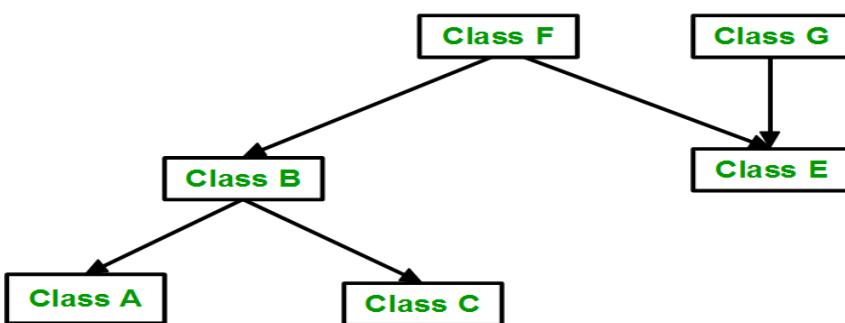
Hierarchical Inheritance

When more than one derived class has been created from the single base class, then it is called as the hierarchical inheritance



Hybrid Inheritance

The inheritance contains different types of inheritance that is called as the hybrid inheritance



Unit 8.6- Polymorphism

Polymorphism stated that the multiple form of the one objects. In the context of the programming, the word polymorphism referred as the same function name but different types of signature has used for the different types.

Example

Input

```
class India():
    def capital(self):
        print("New Delhi is the capital of India.")

    def language(self):
        print("Hindi is the most widely spoken language of India.")

    def type(self):
        print("India is a developing country")

class USA():
    def capital(self):
        print("Washington, D.C. is the capital of USA.")

    def language(self):
        print("English is the primary language of USA.")

    def type(self):
        print("USA is a developed country")

obj_ind = India()
obj_usa = USA()
for country in (obj_ind, obj_usa):
    country.capital()
    country.language()
    country.type()
```

Output

```
New Delhi is the capital of India.  
Hindi is the most widely spoken language of India.  
India is a developing country.  
Washington, D.C. is the capital of USA.  
English is the primary language of USA.  
USA is a developed country.
```

Justification

In this above example, Python use the two different class types in the same unique way

Unit 8.7- Importing Classes

There are total three way for importing a class from the another file in the context of Python

- Import the specific class with the help of import command
- Importation of the multiple classes from the one file with the help of the import command
- Importation of the entire class from the specific one file with the help of import* command

Unit 8.8- Python Standard Library

The standard library in Python is associated with the particular collection of different types of script module which is accessible to a Python program for simplifying the entire programming process as well as removing the need for rewriting the commonly used command. Examples of the python standard library in Python are Tensor Flow, Panda, SciPy, Scikit-Learn and Matplotlib

MCQ

Question 1- In Python, what is the method inside the class?

- a) Class
- b) Object
- c) Function
- d) None

Question 2- Which represent the entity to its behaviour as well as identify?

- a) Object
- b) Class
- c) String
- d) Method

Question 3- Which are used for creating the object?

- a) Constructor
- b) Destructor
- c) Both
- d) None

Question 4- What is the usage of setattr ()

- a) To set an attribute
- b) To add an attribute
- c) To delete an attribute
- d) To replace an attribute

Question 5- What is the usage of getattr ()

- a) To set an attribute
- b) To add an attribute
- c) To delete an attribute
- d) To access the attributes of the object

Question 6- What is the instantiation for OOP terminology?

- a) Creating an Instances of Class
- b) Add Class
- c) Delete Class
- d) None

Question 7- Parent class is the class that are being inherited from, known as the?

- a) Base Class
- b) Hybrid Class
- c) Both
- d) None

Question 8- When the child class is inherit from the only one parent class then it is clled as

- a) Single Inheritance
- b) Hybrid Inheritance
- c) Multiple Inheritance
- d) Hierarchical Inheritance

Question 9- When the respective inheritance is a blend from the more than one type of inheritance then it is called as the?

- a) Single Inheritance
- b) Hybrid Inheritance
- c) Multiple Inheritance
- d) Hierarchical Inheritance

Case Studies

<https://subscription.packtpub.com/book/application-development/9781784398781/1/ch01lvl1sec15/case-study>

<https://www.oreilly.com/library/view/python-3-object-oriented/9781784398781/ch04s02.html>

Reference

<https://www.javatpoint.com/python-oops-concepts>

<https://www.programiz.com/python-programming/object-oriented-programming>

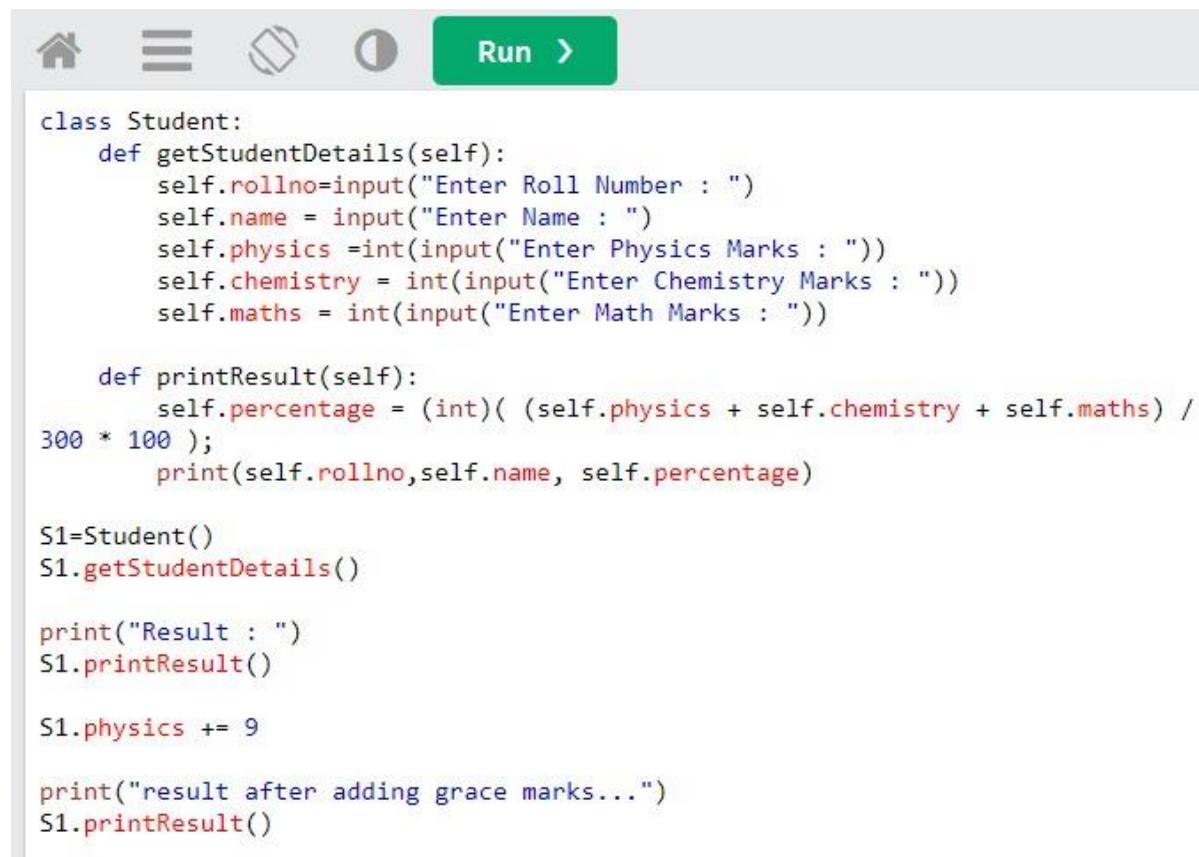
<https://realpython.com/python3-object-oriented-programming/>

Unit 9 -Experiment no. 4

Unit 9.1- Program 5

“Implement a student class with information such as roll no, name, class. The information must be entered by the user”

Program Code



```
class Student:
    def getStudentDetails(self):
        self.rollno=input("Enter Roll Number : ")
        self.name = input("Enter Name : ")
        self.physics =int(input("Enter Physics Marks : "))
        self.chemistry = int(input("Enter Chemistry Marks : "))
        self.maths = int(input("Enter Math Marks : "))

    def printResult(self):
        self.percentage = (int)((self.physics + self.chemistry + self.maths) /
300 * 100 );
        print(self.rollno,self.name, self.percentage)

S1=Student()
S1.getStudentDetails()

print("Result : ")
S1.printResult()

S1.physics += 9

print("result after adding grace marks...")
S1.printResult()
```

Output

```
Enter Roll Number : 001
Enter Name : John
Enter Physics Marks : 87
Enter Chemistry Marks : 45
Enter Math Marks : 95
Result :
001 John 75
result after adding grace marks...
001 John 78
```

Unit 9.2- Program 6

“Create a program to implement library management system using classes and objects”

Program Code

1. Enums and Constants



```
class BookFormat(Enum):
    HARDCOVER, PAPERBACK, AUDIO_BOOK, EBOOK, NEWSPAPER, MAGAZINE, JOURNAL = 1,
    2, 3, 4, 5, 6, 7

class BookStatus(Enum):
    AVAILABLE, RESERVED, LOANED, LOST = 1, 2, 3, 4

class ReservationStatus(Enum):
    WAITING, PENDING, CANCELED, NONE = 1, 2, 3, 4

class AccountStatus(Enum):
    ACTIVE, CLOSED, CANCELED, BLACKLISTED, NONE = 1, 2, 3, 4, 5

class Address:
    def __init__(self, street, city, state, zip_code, country):
        self.__street_address = street
        self.__city = city
        self.__state = state
        self.__zip_code = zip_code
        self.__country = country
```

```
class Person(ABC):
    def __init__(self, name, address, email, phone):
        self.__name = name
        self.__address = address
        self.__email = email
        self.__phone = phone
```

```
class Constants:
    self.MAX_BOOKS_ISSUED_TO_A_USER = 5
    self.MAX_LENDING_DAYS = 10
```

2. Account, Member, and Librarian



```
# For simplicity, we are not defining getter and setter functions. The
# reader can
# assume that all class attributes are private and accessed through their
# respective
# public getter methods and modified only through their public methods
# function.

from abc import ABC, abstractmethod

class Account(ABC):
    def __init__(self, id, password, person, status=AccountStatus.Active):
        self.__id = id
        self.__password = password
        self.__status = status
        self.__person = person

    def reset_password(self):
        None

class Librarian(Account):
    def __init__(self, id, password, person, status=AccountStatus.Active):
        super().__init__(id, password, person, status)

    def add_book_item(self, book_item):
        None
```

```

def block_member(self, member):
    None

def un_block_member(self, member):
    None

class Member(Account):
    def __init__(self, id, password, person, status=AccountStatus.ACTIVE):
        super().__init__(id, password, person, status)
        self.__date_of_membership = datetime.date.today()
        self.__total_books_checkedout = 0

    def get_total_books_checkedout(self):
        return self.__total_books_checkedout

    def reserve_book_item(self, book_item):
        None

    def increment_total_books_checkedout(self):
        None

    def renew_book_item(self, book_item):
        None

    def checkout_book_item(self, book_item):
        if self.get_total_books_checked_out() >=
            Constants.MAX_BOOKS_ISSUED_TO_A_USER:
            print("The user has already checked-out maximum number of books")
            return False
        book_reservation = BookReservation.fetch_reservation_details(
            book_item.get_barcode())
        if book_reservation != None and book_reservation.get_member_id() != self.get_id():
            # book item has a pending reservation from another user
            print("self book is reserved by another member")
            return False
        elif book_reservation != None:
            # book item has a pending reservation from the give member, update
            book_reservation.update_status(ReservationStatus.COMPLETED)

        if not book_item.checkout(self.get_id()):
            return False

        self.increment_total_books_checkedout()
        return True

```

```

def check_for_fine(self, book_item_barcode):
    book_lending = BookLending.fetch_lending_details(book_item_barcode)
    due_date = book_lending.get_due_date()
    today = datetime.date.today()
    # check if the book has been returned within the due date
    if today > due_date:
        diff = today - due_date
        diff_days = diff.days
        Fine.collect_fine(self.get_member_id(), diff_days)

def return_book_item(self, book_item):
    self.check_for_fine(book_item.get_barcode())
    book_reservation = BookReservation.fetch_reservation_details(
        book_item.get_barcode())
    if book_reservation != None:
        # book item has a pending reservation
        book_item.update_book_item_status(BookStatus.RESERVED)
        book_reservation.send_book_available_notification()
    book_item.update_book_item_status(BookStatus.AVAILABLE)

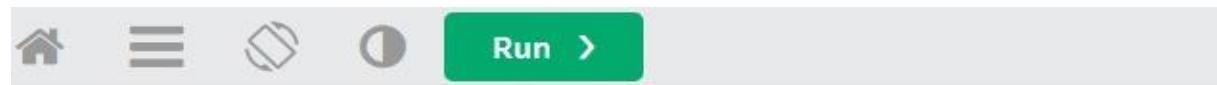
def renew_book_item(self, book_item):
    self.check_for_fine(book_item.get_barcode())
    book_reservation = BookReservation.fetch_reservation_details(
        book_item.get_barcode())
    # check if self book item has a pending reservation from another member
    if book_reservation != None and book_reservation.get_member_id() != self.get_member_id():
        print("self book is reserved by another member")
        self.decrement_total_books_checkedout()
        book_item.update_book_item_state(BookStatus.RESERVED)
        book_reservation.send_book_available_notification()
        return False
    elif book_reservation != None:
        # book item has a pending reservation from self member
        book_reservation.update_status(ReservationStatus.COMPLETED)
        BookLending.lend_book(book_item.get_bar_code(), self.get_member_id())
        book_item.update_due_date(
            datetime.datetime.now().AddDays(Constants.MAX_LENDING_DAYS))
    return True

```

3. BookReservation, BookLending, and Fine

```
class BookReservation:  
    def __init__(self, creation_date, status, book_item_barcode, member_id):  
        self.__creation_date = creation_date  
        self.__status = status  
        self.__book_item_barcode = book_item_barcode  
        self.__member_id = member_id  
  
    def fetch_reservation_details(self, barcode):  
        None  
  
  
class BookLending:  
    def __init__(self, creation_date, due_date, book_item_barcode, member_id):  
        self.__creation_date = creation_date  
        self.__due_date = due_date  
        self.__return_date = None  
        self.__book_item_barcode = book_item_barcode  
        self.__member_id = member_id  
  
    def lend_book(self, barcode, member_id):  
        None  
  
    def fetch_lending_details(self, barcode):  
        None  
  
  
class Fine:  
    def __init__(self, creation_date, book_item_barcode, member_id):  
        self.__creation_date = creation_date  
        self.__book_item_barcode = book_item_barcode  
        self.__member_id = member_id  
  
    def collect_fine(self, member_id, days):  
        None
```

4. BookItem



```
from abc import ABC, abstractmethod

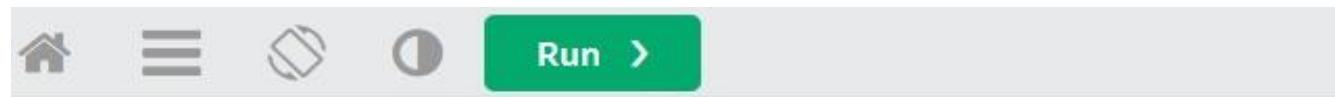
class Book(ABC):
    def __init__(self, ISBN, title, subject, publisher, language,
number_of_pages):
        self.__ISBN = ISBN
        self.__title = title
        self.__subject = subject
        self.__publisher = publisher
        self.__language = language
        self.__number_of_pages = number_of_pages
        self.__authors = []

    class BookItem(Book):
        def __init__(self, barcode, is_reference_only, borrowed, due_date, price,
book_format, status, date_of_purchase, publication_date, placed_at):
            self.__barcode = barcode
            self.__is_reference_only = is_reference_only
            self.__borrowed = borrowed
            self.__due_date = due_date
            self.__price = price
            self.__format = book_format
            self.__status = status
            self.__date_of_purchase = date_of_purchase
            self.__publication_date = publication_date
            self.__placed_at = placed_at

        def checkout(self, member_id):
            if self.get_is_reference_only():
                print("self book is Reference only and can't be issued")
                return False
            if not BookLending.lend_book(self.get_bar_code(), member_id):
                return False
            self.update_book_item_status(BookStatus.LOANED)
            return True

    class Rack:
        def __init__(self, number, location_identifier):
            self.__number = number
            self.__location_identifier = location_identifier
```

5. Search interface and Catalog



```
from abc import ABC, abstractmethod

class Search(ABC):
    def search_by_title(self, title):
        None

    def search_by_author(self, author):
        None

    def search_by_subject(self, subject):
        None

    def search_by_pub_date(self, publish_date):
        None

class Catalog(Search):
    def __init__(self):
        self.__book_titles = {}
        self.__book_authors = {}
        self.__book_subjects = {}
        self.__book_publication_dates = {}

    def search_by_title(self, query):
        # return all books containing the string query in their title.
        return self.__book_titles.get(query)

    def search_by_author(self, query):
        # return all books containing the string query in their author's name.
        return self.__book_authors.get(query)
```

Module 2: Data Analytics

Key Learning Outcome

To understand the core concepts and requirement in Python

- To analyze the comprehensive analysis about Numpy concepts such as features, environment setup, numpy Ndarray, array creation data types, array attributes and numpy operations in real time example
- To evaluated the comprehensive analysis about matplotlib concepts such as Figure class, Axes class, line plot, subplots, bar plot, histogram, scatter plot, pie chart, box plot, area chart and word cloud in real time example

Unit 10- Introduction to Data Analytics

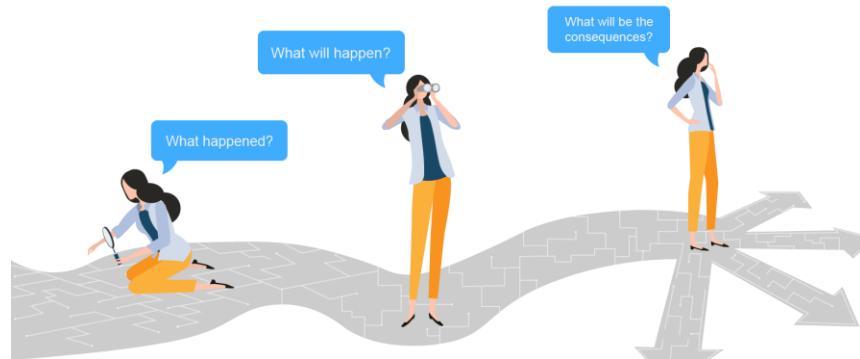
Unit 10.1- Data Analytics

Data analytics is the specific process of analyzing the different types of raw data for the statistics. Data analytics is mainly used for the interpretation as well as discovery of the different meaningful pattern in the form of raw data. So Data analytics is the scientific mechanism for evaluating the different types of raw data for making the conclusion about the information about that data

There are total four types of data analytics that we are going to discussed in this section

Descriptive Analytics

Descriptive analytics helps business organization or software developer for findings the answer of what happen. So with the help of this technique, developer summarized the large database for describing the different types of outcomes to the wide ranges of stakeholder. By implementing the KPI indicators, this strategy is mainly used for tracking the both success as well as failure of the projects.



Diagnostic Analytics

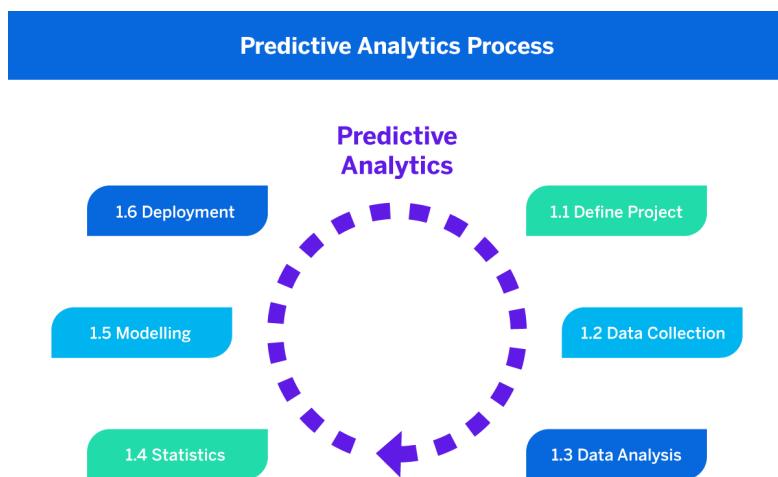
Diagnostic analytics helps business organization or software developer for findings the answer of why the things will happen. This analytic technique takes the basic findings from the descriptive analytic and then digs deeper for findings the root cause. Then the KPI indicator will analyze how the particular things got worse or got better. In this process, there are three types of investigation is carried for findings out the performance indicators about the data

- Identification of different types of anomalies in the respective data
- Specific data which has been related to the anomalies has collected successfully
- Different types of statistical technique is mainly used for findings out the both trends as well as relationship which explain the above mention anomalies



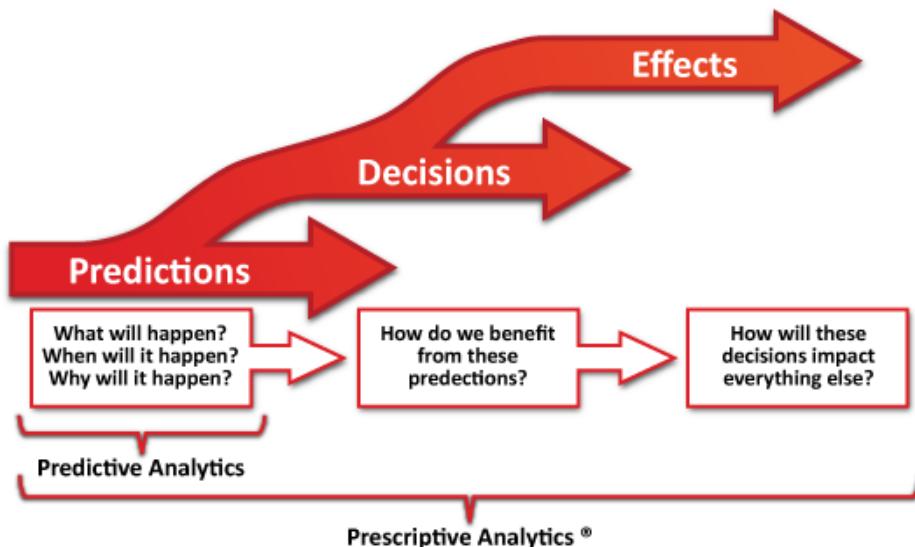
Predictive Analytics

Predictive analytic is capable for forecasting what will happen in the future. So predictive analysis uses the historical data for identifying different types of trends and then it will determine if this trends is likely to be recur or not



Prescriptive Analytics

This is the last data analytic technique that helps the organization to understand what should need to be done. By getting the insight from the specific predictive analytic technique, different types of data driven based decision is gather that helps business organization for making the different types of informed decision.



Unit 10.2-Requirements of Data Analytics in Python

- Learning the fundamental about the Python
- Practice with the different types of hands-on based learning
- Learning about the different types of data science library
- Development of a data science portfolio for getting the job
- Apply on different types of advance data science and data analytics technique

Unit 10.3-Introduction to NumPy

Definition of NumPy

NumPy usually known as the numerical Python is the specific python libraries that are mainly used for working with the different types of array. NumPy provides different types of working capability for Fourier transformation and linear algebra. So it is the general purpose based array processing package that is mainly aim to provides different types of high performance based multidimensional array and at the same time provides updated tool for working with the array

Usage of NumPy

- In Python programming, the developer has the different types of lists which serve as the main purpose of the array but the main problem with the list is that it is very slow processing
- NumPy is capable for providing an array objects which is 100 xs faster than the any types of Python lists
- The array object in the context of NumPy referred as the ndarray that provides different types of supporting function which makes to work with the ndarray very easily



Unit 10.4-Features of Numpy

- NumPy is capable of providing the high performance oriented N-Dimentional based array objects
- NumPy consists of the multidimensional based container for the computation on the generic data
- NumPy also consists of different types of tools which has integrated the code from the C++ and FORTRAN
- NumPy is capable for associated with the random number based capability and linear algebra
- Lastly, NumPy consisted of different types of broadcast function that help the developer to speed up the array processing

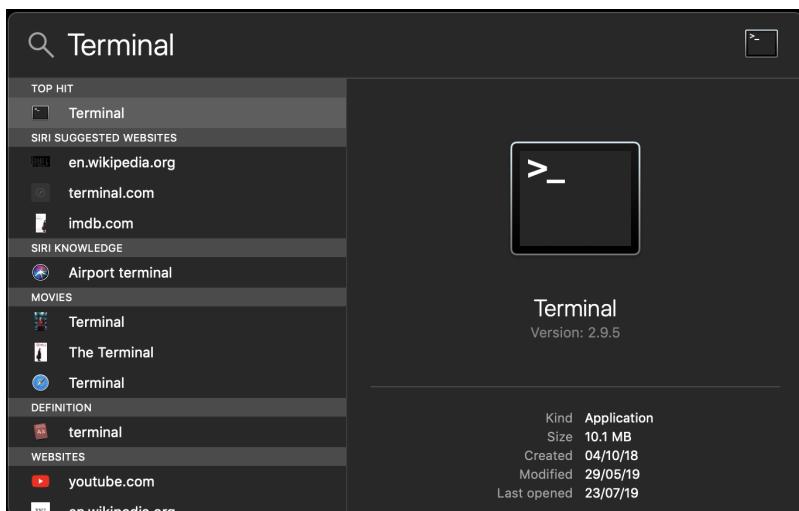


Unit 10.5-Environment setup of NumPy

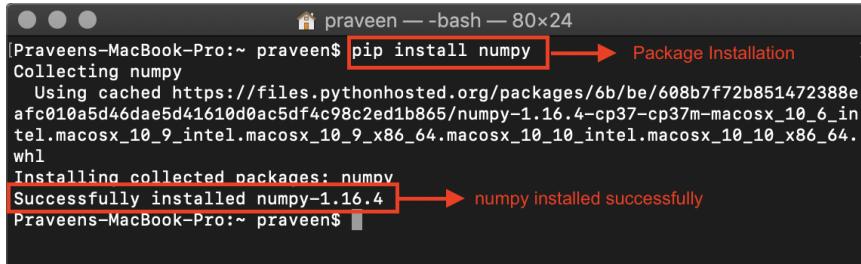
Installation of NumPy in Mac Operating System

Python 2.7

Firstly, the user needs to open the terminal of the Macbook and then he need to type python for getting into the prompt of the Python. The user needs to click on the (⌘) + Space Bar for opening the spotlight search

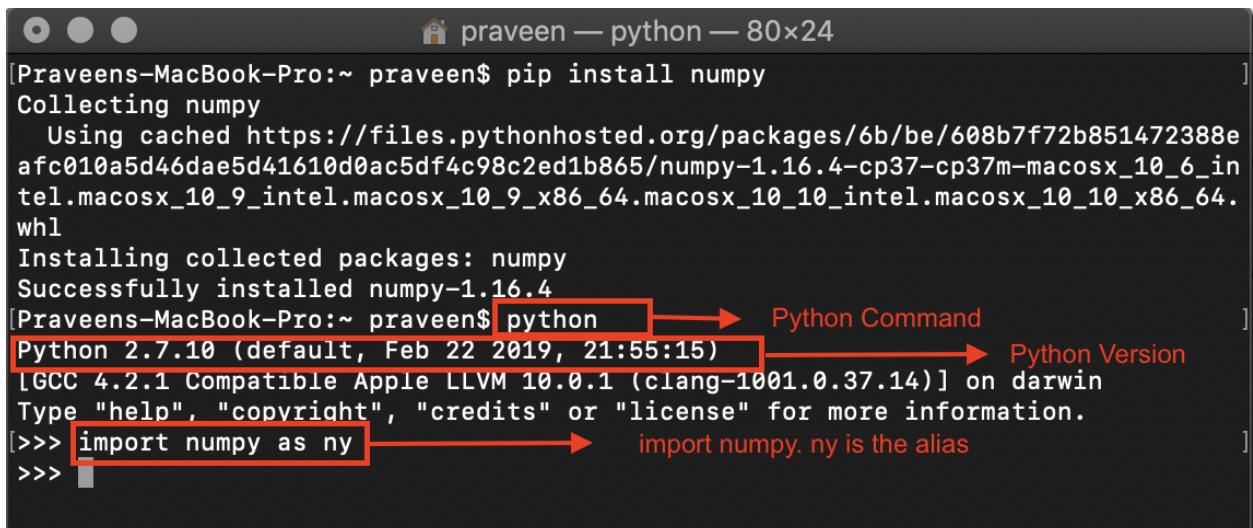


In the terminal, the user needs to use the pip command for installing the numpy package



```
[Praveens-MacBook-Pro:~ praveen$ pip install numpy
Collecting numpy
  Using cached https://files.pythonhosted.org/packages/6b/be/608b7f72b851472388e
afc010a5d46dae5d41610d0ac5df4c98c2ed1b865/numpy-1.16.4-cp37-cp37m-macosx_10_6_in
tel.macosx_10_9_intel.macosx_10_9_x86_64.macosx_10_10_intel.macosx_10_10_x86_64.
whl
Installing collected packages: numpy
Successfully installed numpy-1.16.4
Praveens-MacBook-Pro:~ praveen$ ]
```

Once the package has been successfully install then the user need to type Python for getting into the Python Prompt



```
[Praveens-MacBook-Pro:~ praveen$ pip install numpy
Collecting numpy
  Using cached https://files.pythonhosted.org/packages/6b/be/608b7f72b851472388e
afc010a5d46dae5d41610d0ac5df4c98c2ed1b865/numpy-1.16.4-cp37-cp37m-macosx_10_6_in
tel.macosx_10_9_intel.macosx_10_9_x86_64.macosx_10_10_intel.macosx_10_10_x86_64.
whl
Installing collected packages: numpy
Successfully installed numpy-1.16.4
[Praveens-MacBook-Pro:~ praveen$ python
Python 2.7.10 (default, Feb 22 2019, 21:55:15)
[GCC 4.2.1 Compatible Apple LLVM 10.0.1 (clang-1001.0.37.14)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
]>>> import numpy as ny
]>>> import numpy. ny is the alias
]>>> ]
```

Installation of NumPy in Windows Operating System

The python is not installed by default in the form of windows so the user need to download the python version from the official website and then user need to open the command prompt by using pip for installing NumPy

```
C:\Windows\system32\cmd.exe - python
C:\Users\praveen>pip3 install numpy
Collecting numpy
  Using cached https://files.pythonhosted.org/packages/ce/61/be72eee50f042db3acf0b1fb86650ad36d6c
0d9be9fc29f8505d3b9d6baa(numpy-1.16.4-cp37-cp37m-win_amd64.whl)
Installing collected packages: numpy
Successfully installed numpy-1.16.4

C:\Users\praveen>python
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 20:34:20) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy as np
>>>
```

Installation of NumPy in Ubuntu Operating System

For installing NumPy in the ubuntu, the user needs to incorporate the root privilege to the system for installing of the both pip as well as numpy. So the user needs to open the terminal in the ubuntu and then install the pip with the help of apt

```
root@parallels-vm:~# apt install python-pip python-pip3 → Install pip for Python2 and Python3
Reading package lists... done
Building dependency tree
Reading state information... Done
E: Unable to locate package python-pip3
root@parallels-vm:~# apt install python-pip python3-pip
Reading package lists... done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  libpython3-dev libpython3.5 libpython3.5-dev libpython3.5-minimal libpython3.5
-stdlib python3-dev python3-setuptools python3-wheel python3.5
  python3.5-dev python3.5-minimal
```

Once the pip has setup, the user can see the same command

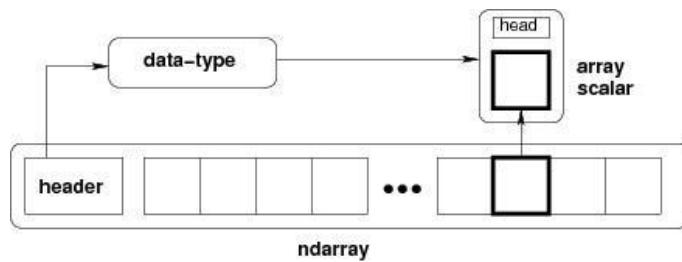
```
root@parallels-vm:~# pip3 install numpy → Install numpy for Python 3
Collecting numpy
  Using cached https://files.pythonhosted.org/packages/bb/ef/d5a21cbc094d3f4d5
b5336494dbc9550b70c766a8345513c7c24ed18418(numpy-1.16.4-cp35-cp35m-manylinux1
_x86_64.whl)
Installing collected packages: numpy
Successfully installed numpy-1.16.4 → numpy installed successfully
```

Installation of NumPy in Fedora Operating System

The user needs to use the pip command for installing the NumPy in the fedora operating system

Unit 10.6-Numpy Ndarray

The array class in the context of Numpy is referred as the ndarray. So different elements in the form of Numpy array has been accessed with the help of the square brackets and then it has been initialize with the help of nested Python based Lists. Ndarray describe the collection of the different items of the same data types. So the item in the respective collection has been accessed with the help of the zero-th index. Items that are extracted from the Ndarray with the help of the slicing operation has represented by the Python object of the one of the array of the scalar types. The following diagram shows the relationship between the Ndarray, array scalar type and respective dtype.



Example

Input

```
# Python program to demonstrate
# basic array characteristics
import numpy as np

# Creating array object
arr = np.array( [[ 1, 2, 3],
                 [ 4, 2, 5]] )

# Printing type of arr object
print("Array is of type: ", type(arr))

# Printing array dimensions (axes)
print("No. of dimensions: ", arr.ndim)

# Printing shape of array
print("Shape of array: ", arr.shape)

# Printing size (total number of elements) of array
print("Size of array: ", arr.size)

# Printing type of elements in array
print("Array stores elements of type: ", arr.dtype)
```

Output

```
Array is of type: <class 'numpy.ndarray'>
No. of dimensions: 2
Shape of array: (2, 3)
Size of array: 6
Array stores elements of type: int64
```

Indexing

Indexing in the NumPy is implemented with the help of the array in index. Numpy array has been indexed with the help of other types of array or other types of sequence with the tuple exception. The last element of the array indexed by the -1 then the second last is -2 and so on.

Example

Input

```
Python 
```

```
1 # Python program to demonstrate
2 # the use of index arrays.
3 import numpy as np
4
5 # Create a sequence of integers from
6 # 10 to 1 with a step of -2
7 a = np.arange(10, 1, -2)
8 print("\n A sequential array with a negative step: \n",a)
9
10 # Indexes are specified inside the np.array method.
11 newarr = a[np.array([3, 1, 2 ])]
12 print("\n Elements at these indices are:\n",newarr)
13
```

Output

```
A sequential array with a negative step:  
[10 8 6 4 2]
```

```
Elements at these indices are:  
[4 8 6]
```

Slicing

Slicing in the NumPy array is possible for extracting the particular ranges of elements from the array. Slicing in the NumPy array is executed in the same way as it is executed in the Python list

Input

```
import numpy as np  
  
a = np.array([[1,2],[3,4],[5,6]])  
  
print(a[0,1])  
print(a[2,0])
```

Output

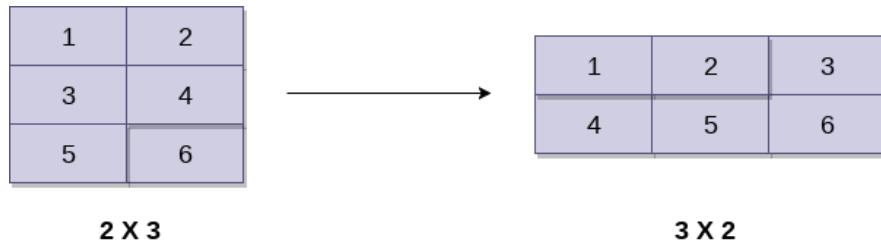
```
2  
5
```

This element prints the 2nd elements from the particular 0th index and then the 0th element from the respective 2nd index of the above array

Reshape ()

By shaping the array, we understand the total number of rows and respective columns of the multi-dimensional array. The reshape () that are associated with the ndarray is mainly used for reshaping the entire array by changing the total numbers of rows and respective columns of the multi dimensional array

Let's reshape the array in the given example



Input

```
import numpy as np
a = np.array([[1,2],[3,4],[5,6]])
print("printing the original array..")
print(a)
a=a.reshape(2,3)
print("printing the reshaped array..")
print(a)
```

Output

```
printing the original array..
[[1 2]
 [3 4]
 [5 6]]
printing the reshaped array..
[[1 2 3]
 [4 5 6]]
```

Arrange ()

The arrange method in Numpy return the array with the evenly space elements as per the interval. The interval is defined as the half opened that is [Start, Stop). So the Numpy arrange () creates a particular instance of the ndarray and then return a reference into it. The user can define the interval level and space between each of the values with the help of the arrange ()

Example

Input

```
# Python Programming illustrating
# numpy.arange method

import numpy as geek

print("A\n", geek.arange(4).reshape(2, 2), "\n")
print("A\n", geek.arange(4, 10), "\n")
print("A\n", geek.arange(4, 20, 3), "\n")
```

Output

```
A
[[0 1]
 [2 3]]


A
[4 5 6 7 8 9]


A
[ 4  7 10 13 16 19]
```

Unit 10.7-Array Creation Data Types

For creating an array of the numeric value in the Python, the user needs to import the array module

Example

Input

```
import array as arr
a = arr.array('d', [1.1, 3.5, 4.5])
print(a)
```

Output

```
array('d', [1.1, 3.5, 4.5])
```

Justification

In this example, the user implements the array of the float type because the letter d is the specific type code

Array Creation Data Types

Code	C Type	Python Type	Min bytes
b	signed char	int	1
B	unsigned char	int	1
u	Py_UNICODE	Unicode	2
h	signed short	int	2
H	unsigned short	int	2
i	signed int	int	2
I	unsigned int	int	2
l	signed long	int	4
L	unsigned long	int	4
f	float	float	4
d	double	float	8

Unit 10.8-Array Attributes

There are total five different Numpy array attributes that we are going to discuss in this book

Ndarray.ndim

Ndim is represented as the total number of the dimension that the axes in the form of ndarray

Ndarray.size

Size represents the total numbers of the elements in the respective ndarray

Ndarray.itemsize

The item size has returned the total size (in terms of the bytes) for each of the elements in the form of ndarray

Ndarray.shape

Shape is the specific tuple of the integer that represents the ndarray size in every dimension

Ndarray.dtype

Dtype specified the data types for each of the elements in the context of Numpy Array

Unit 10.9-Numpy Operations

Arithmetic Operation

Example

Input

```
import numpy as np

# Initializing the array
arr1 = np.arange(4, dtype = np.float_).reshape(2, 2)

print('First array:')
print(arr1)

print('\nSecond array:')
arr2 = np.array([12, 12])
print(arr2)

print('\nAdding the two arrays:')
print(np.add(arr1, arr2))

print('\nSubtracting the two arrays:')
print(np.subtract(arr1, arr2))

print('\nMultiplying the two arrays:')
print(np.multiply(arr1, arr2))

print('\nDividing the two arrays:')
print(np.divide(arr1, arr2))
```

Output

```
First array:  
[[ 0.  1.]  
 [ 2.  3.]]  
  
Second array:  
[12 12]  
  
Adding the two arrays:  
[[ 12.  13.]  
 [ 14.  15.]]  
  
Subtracting the two arrays:  
[[-12. -11.]  
 [-10. -9.]]  
  
Multiplying the two arrays:  
[[ 0.  12.]  
 [ 24.  36.]]  
  
Dividing the two arrays:  
[[ 0.          0.08333333]  
 [ 0.16666667  0.25        11]]
```

Unit 10.10-Mathematical and Statistical Functions

Numpy.reciprocal ()

This function in Numpy is capable for returning the reciprocal for the arguments in the form of element-wise

Example

Input

```
# Python code to perform reciprocal operation
# on NumPy array
import numpy as np
arr = np.array([25, 1.33, 1, 1, 100])

print('Our array is:')
print(arr)

print('\nAfter applying reciprocal function:')
print(np.reciprocal(arr))

arr2 = np.array([25], dtype = int)
print('\nThe second array is:')
print(arr2)

print('\nAfter applying reciprocal function:')
print(np.reciprocal(arr2))
```

Output

```
Our array is:
[ 25.        1.33       1.        1.        100.     ]

After applying reciprocal function:
[ 0.04        0.7518797   1.        1.        0.01      ]

The second array is:
[25]

After applying reciprocal function:
[0]
```

Numpy.power ()

This function is treated each of the elements in the form of first input based array as the base and then returns it raised to the power of the next elements in the form of second input based array

Example

Input

```
# Python code to perform power operation
# on NumPy array

import numpy as np

arr = np.array([5, 10, 15])

print('First array is:')
print(arr)

print('\nApplying power function:')
print(np.power(arr, 2))

print('\nSecond array is:')
arr1 = np.array([1, 2, 3])
print(arr1)

print('\nApplying power function again:')
print(np.power(arr, arr1))
```

Output

```
First array is:
```

```
[ 5 10 15]
```

```
Applying power function:
```

```
[ 25 100 225]
```

```
Second array is:
```

```
[1 2 3]
```

```
Applying power function again:
```

```
[ 5 100 3375]
```

Numpy.mod ()

This NumPy function returned the division remainder of the elements in the form of an input array

Example

Input

```
# Python code to perform mod function
# on NumPy array

import numpy as np

arr = np.array([5, 15, 20])
arr1 = np.array([2, 5, 9])

print('First array:')
print(arr)

print('\nSecond array:')
print(arr1)

print('\nApplying mod() function:')
print(np.mod(arr, arr1))

print('\nApplying remainder() function:')
print(np.remainder(arr, arr1))
```

Output

```
First array:
```

```
[ 5 15 20]
```

```
Second array:
```

```
[2 5 9]
```

```
Applying mod() function:
```

```
[1 0 2]
```

```
Applying remainder() function:
```

```
[1 0 2]
```

MCQ

Question 1- NumPy stands for?

- a) Numerical Python
- b) Number Python
- c) Both
- d) None

Question 2- NumPy is used alongside with the which Package

- a) Node.js
- b) Panda
- c) Matplotlib
- d) None

Question 3- The most important object that are defined in NumPy is an ND array type called as?

- a) ndarray
- b) aarray
- c) barray
- d) none

Question 4- If the specific dimension given as __ in a reshaping operation, then the other dimension is automatically calculated?

- a) Negative 1
- b) 2
- c) 0
- d) 1

Question 5- Which of the following sets the size of the buffer that are used in the ufuncs?

- a) setbuffer(size)
- b) buffer size=1
- c) buffer size=2
- d) buffer size=0

Question 6- Which of the following set the specific floating point error call back function?

- a) Setter Call
- b) Getter Call
- c) Both
- d) None

Question 7- Output

```
import numpy as np
a = np.array([1,2,3,5,8])
b = np.array([0,3,4,2,1])
c = a + b
c = c*a
print (c[2])
```

- a) 21
- b) 22
- c) 18
- d) 28

Question 8- Output

```
import numpy as np
ary = np.array([1,2,3,5,8])
ary = ary + 1
print (ary[1])
```

- a) 8
- b) 3
- c) 2
- d) 9

Question 9- Output

```
import numpy as np  
a = np.array([[1,2,3],[0,1,4]])  
print (a.size)
```

- a) 6
- b) 8
- c) 2
- d) 4

Case Studies

<https://numpy.org/case-studies/blackhole-image/>

<https://www.analyticsvidhya.com/blog/2020/04/the-ultimate-numpy-tutorial-for-data-science-beginners/>

Reference

https://www.w3schools.com/python/numpy/numpy_intro.asp

<https://www.geeksforgeeks.org/numpy-in-python-set-1-introduction/>

<https://www.tutorialspoint.com/numpy/index.htm>

<https://www.mygreatlearning.com/blog/python-numpy-tutorial/>

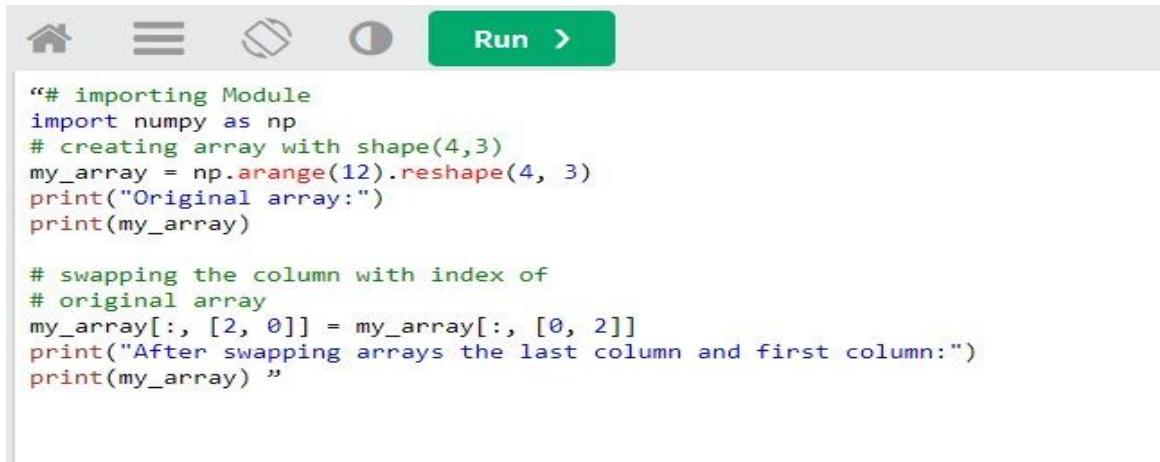
Unit 11- Experiment no. 5

Unit 11.1- Program 7

“Write an experiment to swap two columns in numpy array”

Technique 1: Swapping the column of an array

Program Code



```
# importing Module
import numpy as np
# creating array with shape(4,3)
my_array = np.arange(12).reshape(4, 3)
print("Original array:")
print(my_array)

# swapping the column with index of
# original array
my_array[:, [2, 0]] = my_array[:, [0, 2]]
print("After swapping arrays the last column and first column:")
print(my_array)
```

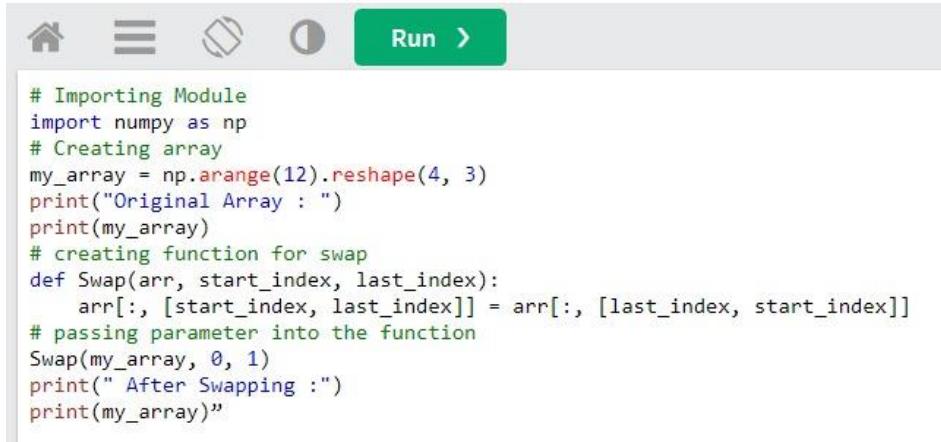
Output of Program Code

```
Original array:
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]

After swapping arrays the last column and first column:
[[ 2  1  0]
 [ 5  4  3]
 [ 8  7  6]
 [11 10  9]]
```

Technique 2: Swapping the column of an array with the user chooses

Program Code



```
# Importing Module
import numpy as np
# Creating array
my_array = np.arange(12).reshape(4, 3)
print("Original Array : ")
print(my_array)
# creating function for swap
def Swap(arr, start_index, last_index):
    arr[:, [start_index, last_index]] = arr[:, [last_index, start_index]]
# passing parameter into the function
Swap(my_array, 0, 1)
print(" After Swapping :")
print(my_array)
```

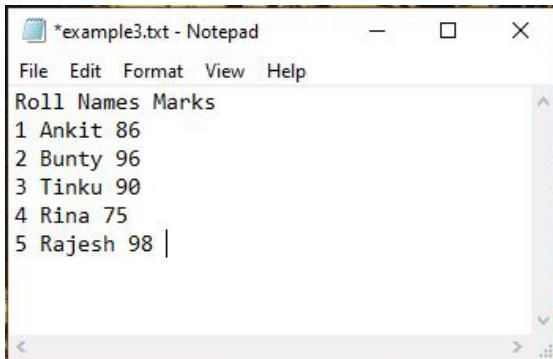
Output of Program Code

```
Original Array :
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
After Swapping :
[[ 1  0  2]
 [ 4  3  5]
 [ 7  6  8]
 [10  9 11]]
```

Unit 11.2- Program 8

“Write an experiment that imports a dataset with numbers and texts, keeping the text intact in python numpy?”

Importing text file into NumPy array by skipping first row



Roll	Names	Marks
1	Ankit	86
2	Bunty	96
3	Tinku	90
4	Rina	75
5	Rajesh	98

Program Code

```
import numpy as np

# only column1 data is imported into numpy
# array from text file
data = np.loadtxt("example3.txt", usecols=1, skiprows=1, dtype='str')

for each in data:
    print(each)
```

Output

```
Ankit
Bunty
Tinku
Rina
Rajesh
```

Explanation

In this example, the indexing in the NumPy array starts from 0. So the program code in the example shows that the roll column in the respective text file is situated in the 0th column, names column is situated in the first column and marks column is situated in the second column in the test file namely ‘example3.txt’.

Unit 12- Introduction to Matplotlib

Unit 12.1-Introduction to Matplotlib

Matplotlib is the updated visualization library in the form of Python for the 2D plot of different types of array. Matplotlib is written with the help of the Python language that makes the usage of the Numpy which is the numerical extension for the Python. Matplotlib provides the object oriented based API which helps to embed different type of plots in the Python GUI application

Features of Matplotlib

- Matplotlib provides semantic way for generating the complex and sub plot grid
- It sets the specific aspects ratio of the axes box with the help of the set_box_aspects () method
- Matplotlib provides custom visual style by implementing public quality plots
- Matplotlib used with the different user interface for example IPython Shells and Jupiter Notebook to develop numerous plot types

Usage of Matplotlib in Python

Matplotlib is the cross-platform based data visualization technique for Python and its associate NumPy. Matplotlib offers the important open source-based alternative to the MATLAB. Python developer uses the API of Matplotlib for embedding plot into different GUI applications. Matplotlib provides user effectiveness for visualizing the data with the help of different types of plots for making the entire data to be more understandable.

Unit 12.2-Figure Class

Definition

The figure class in the matplotlib provides the top level based container for the entire plot elements. So this class is mainly used for controlling the entire default space of the respective subplots as well as top-level containers for the entire plot elements. So this figure instances is supported callbacks with the help of the callback attributes that known as CallbackRegistry instance.

Syntax

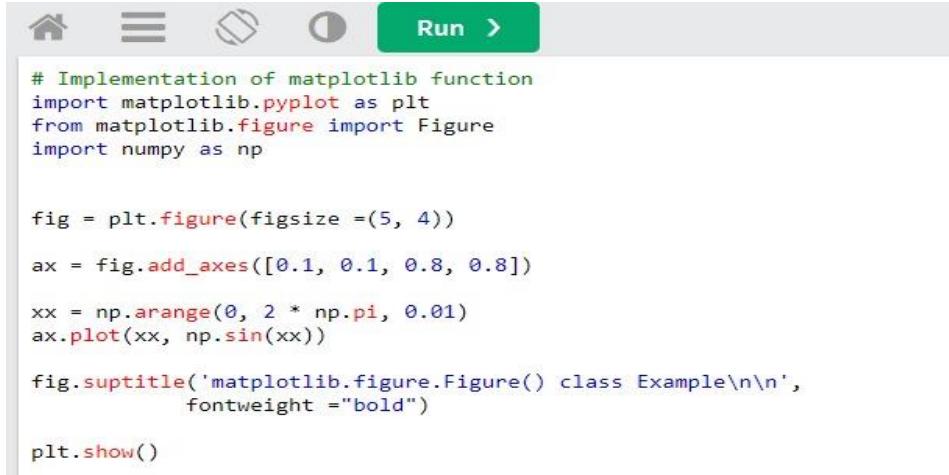
```
Syntax: class matplotlib.figure.Figure(figsize=None, dpi=None,  
facecolor=None, edgecolor=None, linewidth=0.0, frameon=None,  
subplotpars=None, tight_layout=None, constrained_layout=None)
```

Parameter

- **figsize** : This parameter is the Figure dimension (width, height) in inches.
- **dpi** : This parameter is the dots per inch.
- **facecolor** : This parameter is the figure patch facecolor.
- **edgecolor** : This parameter is the figure patch edge color.
- **linewidth** : This parameter is the linewidth of the frame.
- **frameon** : This parameter is the suppress drawing the figure background patch.
- **subplotpars** : This parameter is the Subplot parameters.
- **tight_layout** : This parameter is used to adjust subplot parameters.
- **constrained_layout** : This parameter is used to adjust positioning of plot elements.

Example

Input

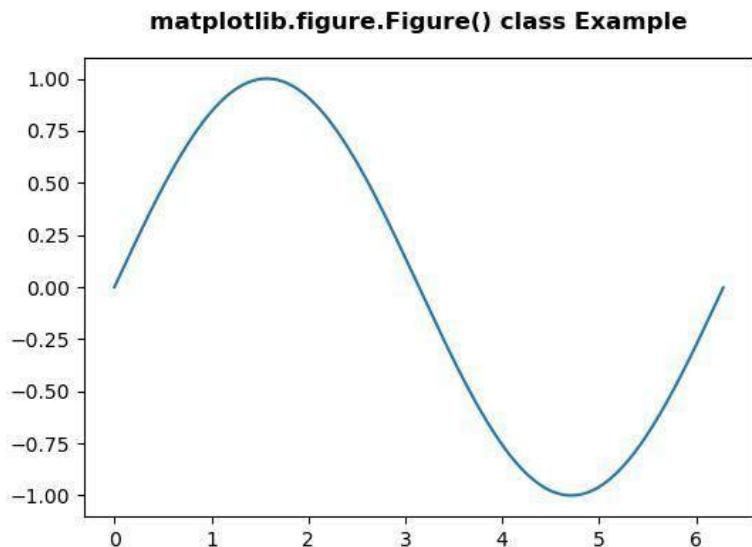


```
# Implementation of matplotlib function
import matplotlib.pyplot as plt
from matplotlib.figure import Figure
import numpy as np

fig = plt.figure(figsize =(5, 4))
ax = fig.add_axes([0.1, 0.1, 0.8, 0.8])
xx = np.arange(0, 2 * np.pi, 0.01)
ax.plot(xx, np.sin(xx))

fig.suptitle('matplotlib.figure.Figure() class Example\n\n',
fontweight ="bold")
plt.show()
```

Output



Justification

In this example, an illustration has been made to the `matplotlib.figure.Figure ()` Function

Unit 12.3-Axes Class

Definition

The axes are the most important part for developing the sub-plots. The axes in matplotlib allow the plot placement in any location. The axes function in the matplotlib develop the axes object with the arguments, in which the arguments is the list of total four elements

- Left
- Width
- Bottom
- Height

Syntax

```
axes([left, bottom, width, height])
```

Parameter

Parameters: `fig : Figure`

The Axes is built in the `Figure fig`.

`rect : [left, bottom, width, height]`

The Axes is built in the rectangle `rect`. `rect` is in `Figure` coordinates.

`sharex, sharey : Axes, optional`

The x or y `axis` is shared with the x or y axis in the input `Axes`.

`frameon : bool, default: True`

Whether the Axes frame is visible.

`box_aspect : float, optional`

Set a fixed aspect for the Axes box, i.e. the ratio of height to width.

Example

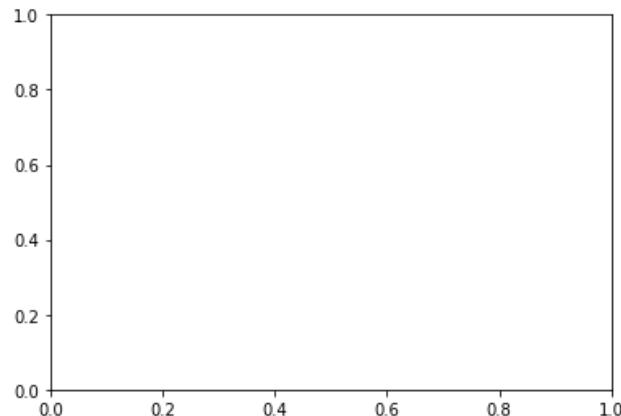
Input

```
import matplotlib.pyplot as plt

fig = plt.figure()

#[left, bottom, width, height]
ax = plt.axes([0.1, 0.1, 0.8, 0.8])
```

Output



Justification

In this diagram, it represents the axes ([0.1, 0.1, 0.8, and 0.8]) in which

The first 0.1 associated with distance measurement between the left sides of axes as well as the border for the figure window is near about 10 percent of the total numbers of width in the figure window. The second 0.1 associated with distance measurement between the bottom sides of axes as well as the border for the figure window is near about 10 percent of the total numbers of height in the figure window. The first 0.8 stated that width of the axes from the left to right is nearly 80 percent. The second 0.8 stated that height of the axes from the bottom to top is nearly 80 percent

Unit 12.4-Line Plot

Definition

Line plot in matplotlib is mainly used for representing the internal relationship between the two data X as well as Y on the different types of axis. So line plot displays the different types of numerical values on the one axis and different types of categorical values on the other axis

Syntax

```
matplotlib.pyplot.plot(*args, scalex=True, scaley=True, data=None, **kwargs)
```

Parameter

Parameters: **x, y : array-like or scalar**

The horizontal / vertical coordinates of the data points. x values are optional and default to `range(len(y))`.

Commonly, these parameters are 1D arrays.

They can also be scalars, or two-dimensional (in that case, the columns represent separate data sets).

These arguments cannot be passed as keywords.

fmt : str, optional

A format string, e.g. 'ro' for red circles. See the *Notes* section for a full description of the format strings.

Format strings are just an abbreviation for quickly setting basic line properties. All of these and more can also be controlled by keyword arguments.

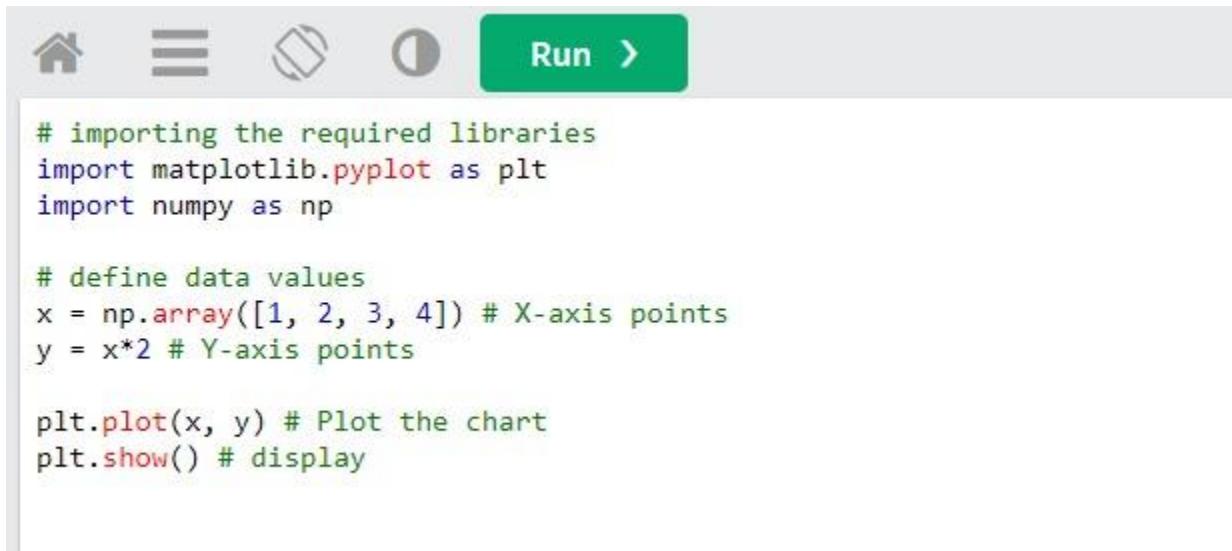
This argument cannot be passed as keyword.

data : indexable object, optional

An object with labelled data. If given, provide the label names to plot in x and y.

Example

Input



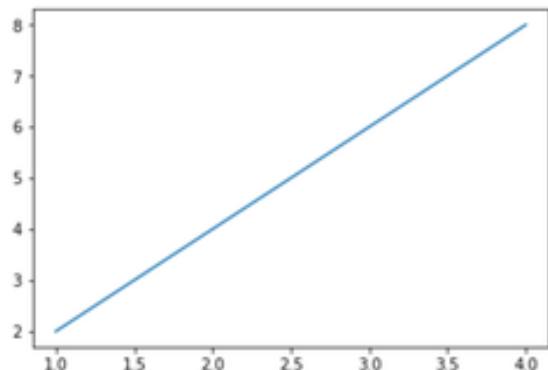
The screenshot shows a Jupyter Notebook interface with a toolbar at the top featuring icons for file, cell, run, and help. Below the toolbar is a code cell containing the following Python code:

```
# importing the required libraries
import matplotlib.pyplot as plt
import numpy as np

# define data values
x = np.array([1, 2, 3, 4]) # X-axis points
y = x*2 # Y-axis points

plt.plot(x, y) # Plot the chart
plt.show() # display
```

Output



Justification

Based on the above figure analysis, we can say that there is no proper level in the form of both X axis as well as Y axis. Since the labeling is most important for evaluating the dimension of the chart.

Unit 12.5-Subplots

Definition

Subplot in matplotlib means that the groups of different axes which are exist in the form of a single matplotlib based figure. The subplot () function in the context of matplotlib is mainly used for making a figure and associating different subplot sets.

Syntax

```
Syntax: matplotlib.pyplot.subplots(nrows=1, ncols=1, sharex=False,  
sharey=False, squeeze=True, subplot_kw=None, gridspec_kw=None,  
**fig_kw)
```

Parameter

- **nrows, ncols :** These parameter are the number of rows/columns of the subplot grid.
- **sharex, sharey :** These parameter controls sharing of properties among x (sharex) or y (sharey) axes.
- **squeeze :** This parameter is an optional parameter and it contains boolean value with default as True.
- **num:** This parameter is the pyplot.figure keyword that sets the figure number or label.
- **subplot_kw:** This parameter is the dict with keywords passed to the add_subplot call used to create each subplot.
- **gridspec_kw:** This parameter is the dict with keywords passed to the GridSpec constructor used to create the grid the subplots are placed on.

Example

Input

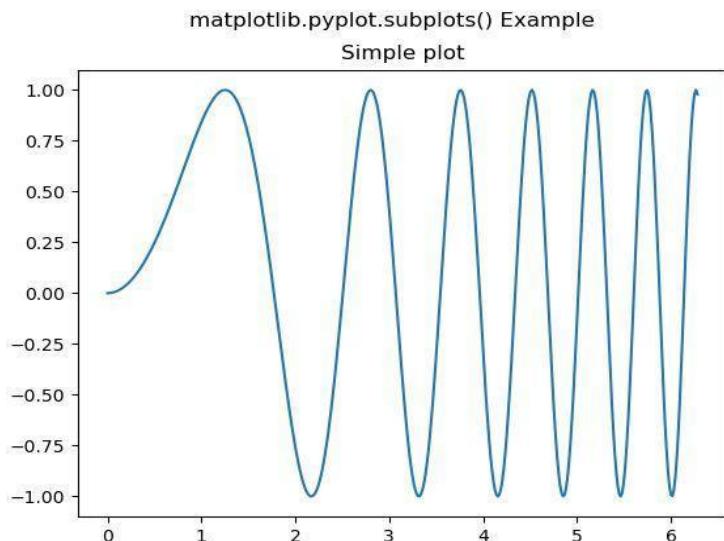
```
# Implementation of matplotlib function
import numpy as np
import matplotlib.pyplot as plt

# First create some toy data:
x = np.linspace(0, 2 * np.pi, 400)
y = np.sin(x**2)

fig, ax = plt.subplots()
ax.plot(x, y)
ax.set_title('Simple plot')

fig.suptitle('matplotlib.pyplot.subplots() Example')
plt.show()
```

Output



Justification

The above example depicted the `matplotlib.pyplot.subplots()` function in the `matplotlib.pyplot`

Unit 12.6-Bar Plot

Definition

The bar plot is the specific graph which represents the categorical data of a rectangular bar with the help of both length as well as height which are proportional to the representation value. The matplotlib API in Python programming associated with the bar () that has been used either in the form of MATLAB style as well as object oriented based API style.

Syntax

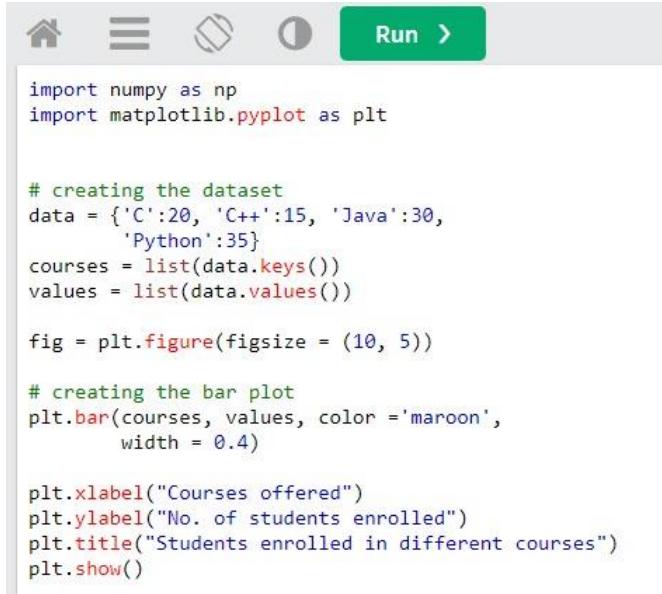
```
plt.bar(x, height, width, bottom, align)
```

Parameter

x	sequence of scalars representing the x coordinates of the bars. align controls if x is the bar center (default) or left edge.
height	scalar or sequence of scalars representing the height(s) of the bars.
width	scalar or array-like, optional. the width(s) of the bars default 0.8
bottom	scalar or array-like, optional. the y coordinate(s) of the bars default None.
align	{'center', 'edge'}, optional, default 'center'

Example

Input



```
import numpy as np
import matplotlib.pyplot as plt

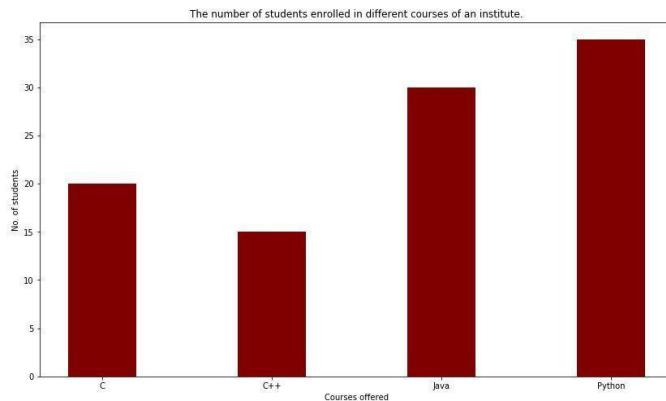
# creating the dataset
data = {'C':20, 'C++':15, 'Java':30,
        'Python':35}
courses = list(data.keys())
values = list(data.values())

fig = plt.figure(figsize = (10, 5))

# creating the bar plot
plt.bar(courses, values, color ='maroon',
        width = 0.4)

plt.xlabel("Courses offered")
plt.ylabel("No. of students enrolled")
plt.title("Students enrolled in different courses")
plt.show()
```

Output



Justification

In the below figure, the plt.bar (courses, values, color='maroon') is mainly used for specifying the respective bar chart need to be plotted with the help of the specific course column that are

specified in the X axis and the values that are specified in the Y axis. Moreover, the color attributes in this bar plot is mainly used for setting the bar color (In this example, it is maroon), plt.xlabel represents the total numbers of course offered and lastly plt.ylabel is mainly used for labeling the values of plot title.

Unit 12.7-Histogram

Definition

The histogram is mainly used for representing the specific types of data that is provided in the form of different groups. In general, this is the updated method for the graphical representation of different types of numerical-based data representation.

Syntax

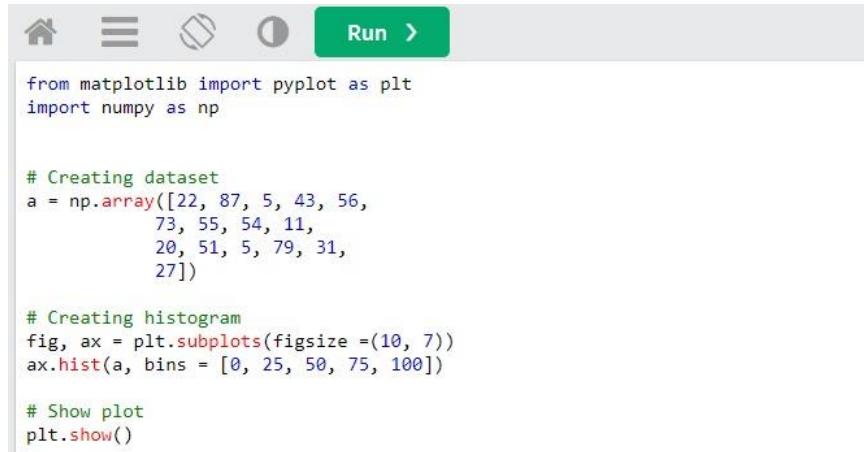
```
Syntax: matplotlib.pyplot.hist(x, bins=None, range=None, density=False,  
weights=None, cumulative=False, bottom=None, histtype='bar', align='mid',  
orientation='vertical', rwidth=None, log=False, color=None, label=None,  
stacked=False, *, data=None, **kwargs)
```

Parameter

Attribute	parameter
x	array or sequence of array
bins	optional parameter contains integer or sequence or strings
density	optional parameter contains boolean values
range	optional parameter represents upper and lower range of bins
histtype	optional parameter used to create type of histogram [bar, barstacked, step, stepfilled], default is "bar"
align	optional parameter controls the plotting of histogram [left, right, mid]
weights	optional parameter contains array of weights having same dimensions as x
bottom	location of the baseline of each bin
rwidth	optional parameter which is relative width of the bars with respect to bin width
color	optional parameter used to set color or sequence of color specs

Example

Input



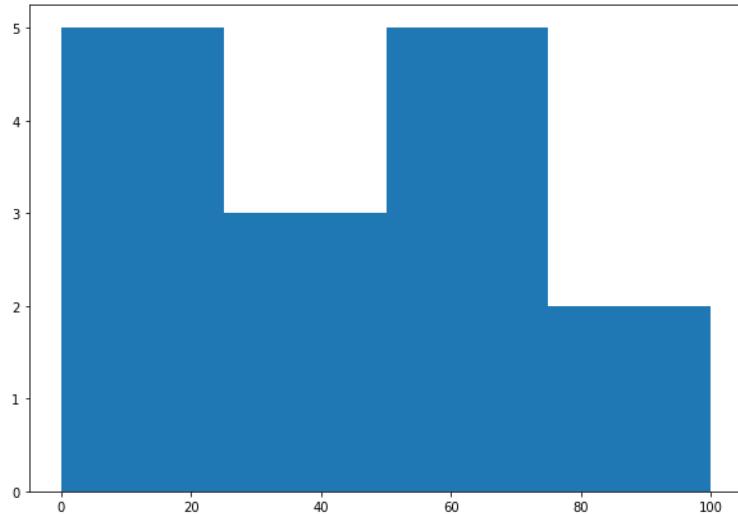
```
from matplotlib import pyplot as plt
import numpy as np

# Creating dataset
a = np.array([22, 87, 5, 43, 56,
              73, 55, 54, 11,
              20, 51, 5, 79, 31,
              27])

# Creating histogram
fig, ax = plt.subplots(figsize =(10, 7))
ax.hist(a, bins = [0, 25, 50, 75, 100])

# Show plot
plt.show()
```

Output



Justification

In this example, the user develops the histogram with the help of some random value.

Unit 12.8-Scatter Plot

Definition

Scatter plot is mainly used for analyzing the different types of relationship between the each of the variable in which the user dot demonstrates the internal relationship between them. The scatter () in the form of matplotlib library is mainly used for drawing the scatter plot

Syntax

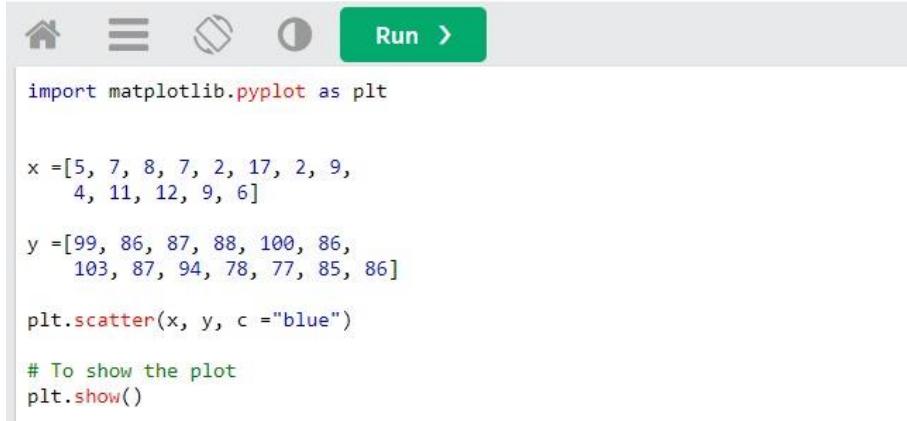
```
matplotlib.pyplot.scatter(x_axis_data, y_axis_data, s=None, c=None,  
marker=None, cmap=None, vmin=None, vmax=None, alpha=None,  
linewidths=None, edgecolors=None)
```

Parameter

- **x_axis_data**- An array containing x-axis data
- **y_axis_data**- An array containing y-axis data
- **s**- marker size (can be scalar or array of size equal to size of x or y)
- **c**- color of sequence of colors for markers
- **marker**- marker style
- **cmap**- cmap name
- **linewidths**- width of marker border
- **edgecolor**- marker border color
- **alpha**- blending value, between 0 (transparent) and 1 (opaque)

Example

Input



```
import matplotlib.pyplot as plt

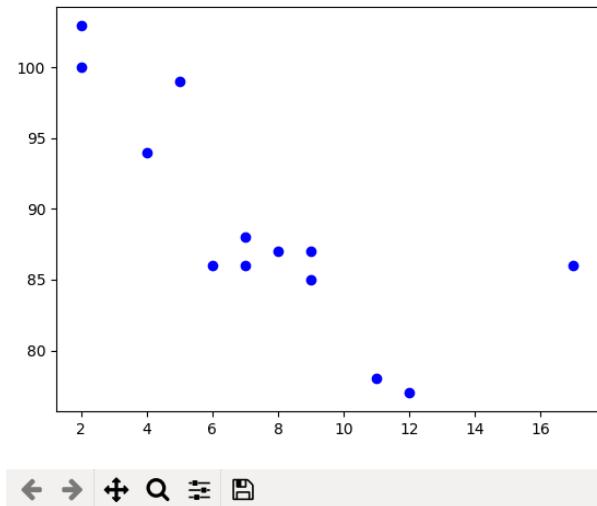
x =[5, 7, 8, 7, 2, 17, 2, 9,
    4, 11, 12, 9, 6]

y =[99, 86, 87, 88, 100, 86,
    103, 87, 94, 78, 77, 85, 86]

plt.scatter(x, y, c ="blue")

# To show the plot
plt.show()
```

Output



Justification

In the above example, the user sets all parameter is optional except the X and Y axis data in which the default value is designated to None

Unit 12.9-Pie Chart

Definition

The pie chart is the most effective circular statistical based plot which is mainly used for representing the only one series of data. Matplotlib API consists of pie () for developing a pie chart that represents the data in the array.

Syntax

Syntax: `matplotlib.pyplot.pie(data, explode=None, labels=None, colors=None, autopct=None, shadow=False)`

Parameter

data represents the array of data values to be plotted, the fractional area of each slice is represented by `data/sum(data)`. If `sum(data)<1`, then the data values returns the fractional area directly, thus resulting pie will have empty wedge of size `1-sum(data)`.

labels is a list of sequence of strings which sets the label of each wedge.

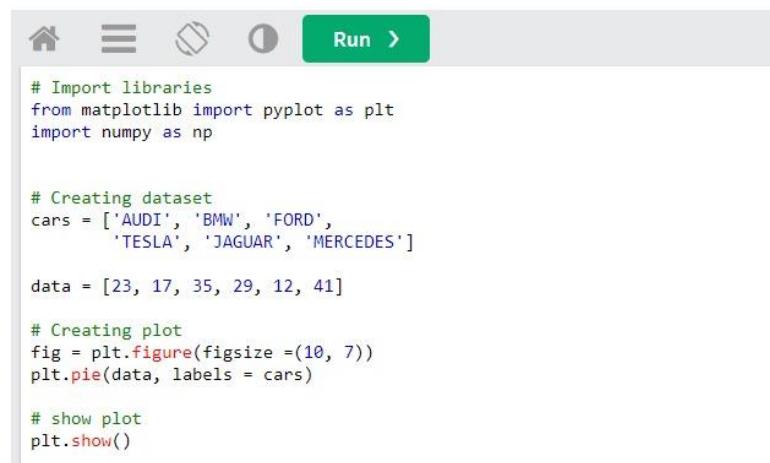
color attribute is used to provide color to the wedges.

autopct is a string used to label the wedge with their numerical value.

shadow is used to create shadow of wedge.

Example

Input



```
# Import libraries
from matplotlib import pyplot as plt
import numpy as np

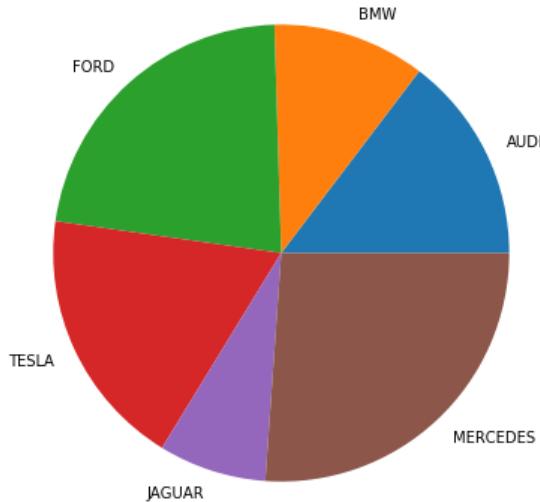
# Creating dataset
cars = ['AUDI', 'BMW', 'FORD',
        'TESLA', 'JAGUAR', 'MERCEDES']

data = [23, 17, 35, 29, 12, 41]

# Creating plot
fig = plt.figure(figsize =(10, 7))
plt.pie(data, labels = cars)

# show plot
plt.show()
```

Output



Justification

In this above figure, a simple pie chart is created with the help of the pie () function

Unit 12.10-Box Plot

Definition

The box plot in matplotlib is mainly created for displaying the summary of the given datasets value that has the different types of properties like minimum, median, maximum and first quartile. The matplotlib.pyplot module in the context of matplotlib library associated with the boxplot () function so that the user creates different types of box plot.

Syntax

```
matplotlib.pyplot.boxplot(data, notch=None, vert=None, patch_artist=None,  
widths=None)
```

Parameter

Attribute	Value
data	array or sequence of array to be plotted
notch	optional parameter accepts boolean values
vert	optional parameter accepts boolean values false and true for horizontal and vertical plot respectively
bootstrap	optional parameter accepts int specifies intervals around notched boxplots
usermedians	optional parameter accepts array or sequence of array dimension compatible with data
positions	optional parameter accepts array and sets the position of boxes
widths	optional parameter accepts array and sets the width of boxes
patch_artist	optional parameter having boolean values
labels	sequence of strings sets label for each dataset
meanline	optional having boolean value try to render meanline as full width of box
order	optional parameter sets the order of the boxplot

Example

Input



```
# Import libraries
import matplotlib.pyplot as plt
import numpy as np

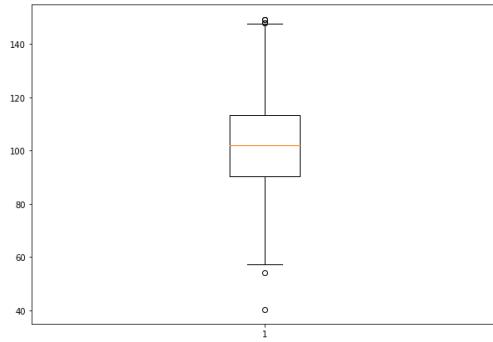
# Creating dataset
np.random.seed(10)
data = np.random.normal(100, 20, 200)

fig = plt.figure(figsize =(10, 7))

# Creating plot
plt.boxplot(data)

# show plot
plt.show()
```

Output



Justification

This example demonstrates the example of simple box plot with the usage of three random values.

Unit 12.11-Area Chart

Definition

The area plot in matplotlib displays the different types of quantitative data visually. So this function is capable for wrapping the entire area section of matplotlib. In matplotlib, the area chart is developed with the help of the `fill_between()` function

Syntax

```
Syntax: plotly.express.area(data_frame=None, x=None, y=None,  
line_group=None, color=None, hover_name=None, hover_data=None,  
custom_data=None, text=None, facet_row=None, facet_col=None,  
facet_col_wrap=0, animation_frame=None, animation_group=None,  
category_orders={}, labels={}, color_discrete_sequence=None,  
color_discrete_map={}, orientation=None, groupnorm=None, log_x=False,  
log_y=False, range_x=None, range_y=None, line_shape=None, title=None,  
template=None, width=None, height=None)
```

Parameter

data_frame: This argument needs to be passed for column names (and not keyword names) to be used. Array-like and dict are transformed internally to a pandas DataFrame.

x, y: Either a name of a column in data_frame, or a pandas Series or array_like object. Values from this column or array_like are used to position marks along the x and y axis in cartesian coordinates.

color: Either a name of a column in data_frame, or a pandas Series or array_like object. Values from this column or array_like are used to assign color to marks.

Example

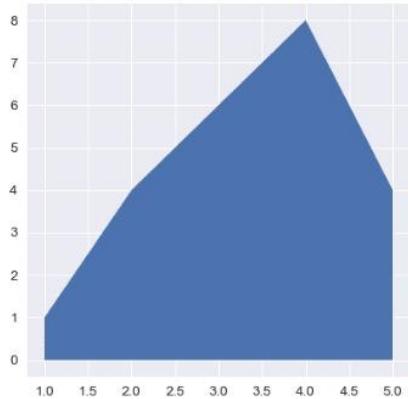
Input

```
# library
import numpy as np
import matplotlib.pyplot as plt

# Create data
x=range(1,6)
y=[1,4,6,8,4]

# Area plot
plt.fill_between(x, y)
plot.show()
```

Output



Justification

This example illustrated the common customization technique in the area chart

Unit 12.12-Word Cloud

Definition

The word cloud is the specific type of data virtualization based technique that is mainly used for representing the different text data in which the word size demonstrates the frequency. Different types of textual based data points have been represents with the help of word cloud.

Syntax

```
word_cloud = WordCloud(collocations = False, background_color = 'white').generate(text)
```

Parameter

```
Parameters
-----
font_path : string
    Font path to the font that will be used (OTF or TTF).
    Defaults to DroidSansMono path on a Linux machine. If you are on
    another OS or don't have this font; you need to adjust this path.

width : int (default=400)
    Width of the canvas.

height : int (default=200)
    Height of the canvas.

prefer_horizontal : float (default=0.90)
    The ratio of times to try horizontal fitting as opposed to vertical.
    If prefer_horizontal < 1, the algorithm will try rotating the word
    if it doesn't fit. (There is currently no built-in way to get only
    vertical words.)

mask : nd-array or None (default=None)
    If not None, gives a binary mask on where to draw words. If mask is not
    None, width and height will be ignored, and the shape of mask will be
    used instead. All white (#FF or #FFFFFF) entries will be considered
    "masked out" while other entries will be free to draw on. [This
    changed in the most recent version!]
```

Example

Input

```
# importing the necessary modules
from wordcloud import WordCloud
import matplotlib.pyplot as plt
import csv

# file object is created
file_ob = open(r"C:/Users/user/Documents/sample.csv")

# reader object is created
reader_ob = csv.reader(file_ob)

# contents of reader object is stored .
# data is stored in list of list format.
reader_contents = list(reader_ob)

# empty string is declare
text = ""

# iterating through list of rows
for row in reader_contents :

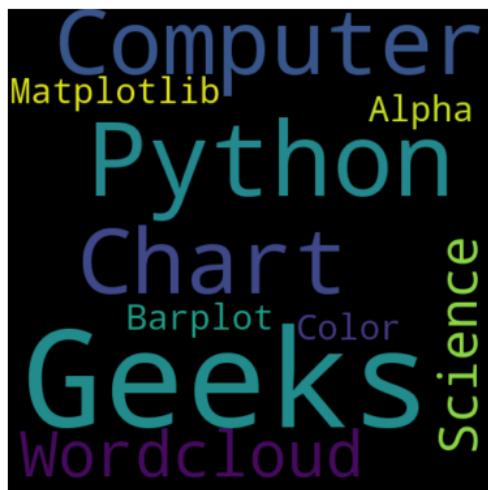
    # iterating through words in the row
    for word in row :

        # concatenate the words
        text = text + " " + word

# show only 10 words in the wordcloud .
wordcloud = WordCloud(width=480, height=480, max_words=10).generate(text)

# plot the WordCloud image
plt.figure()
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.margins(x=0, y=0)
plt.show()
```

Output



Justification

It is possible for setting a maximum number of words that the user wants to display on the respective tag cloud. The above example specifies the usage of the `max_words` keyword arguments of the `WordCloud` function for evaluating the total numbers of words in the tag cloud

Unit 12.13-Bee Swarm Plot

Definition

The point of `swarmplot()` has been represented with the help of the only categorical data in which they does not overlap. So this plot gives the better representation of distributed values for evaluating the representation of different observation

Syntax

```
Syntax: seaborn.swarmplot(x=None, y=None, hue=None, data=None,  
order=None, hue_order=None, dodge=False, orient=None, color=None,  
palette=None, size=5, edgecolor='gray', linewidth=0, ax=None, **kwargs)
```

Parameter

Parameters: `x, y, hue : names of variables in data or vector data, optional`
Inputs for plotting long-form data. See examples for interpretation.

`data : DataFrame, array, or list of arrays, optional`
Dataset for plotting. If `x` and `y` are absent, this is interpreted as wide-form. Otherwise it is expected to be long-form.

`order, hue_order : lists of strings, optional`
Order to plot the categorical levels in, otherwise the levels are inferred from the data objects.

`dodge : bool, optional`
When using `hue` nesting, setting this to `True` will separate the strips for different hue levels along the categorical axis.
Otherwise, the points for each level will be plotted in one swarm.

`orient : "v" / "h", optional`
Orientation of the plot (vertical or horizontal). This is usually inferred based on the type of the input variables, but it can be used to resolve ambiguity when both `x` and `y` are numeric or when plotting wide-form data.

`color : matplotlib color, optional`
Color for all of the elements, or seed for a gradient palette.

`palette : palette name, list, or dict`
Colors to use for the different levels of the `hue` variable. Should be something that can be interpreted by `color_palette()`, or a dictionary mapping hue levels to matplotlib colors.

Example

Input

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

plt.rcParams["figure.figsize"] = [7.50, 3.50]
plt.rcParams["figure.autolayout"] = True

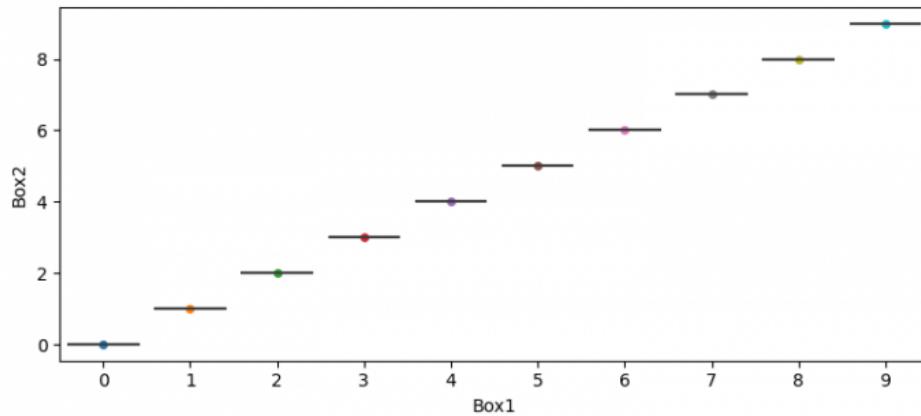
data = pd.DataFrame({"Box1": np.arange(10),
                     "Box2": np.arange(10)})

ax = sns.swarmplot(x="Box1", y="Box2", data=data, zorder=0)

sns.boxplot(x="Box1", y="Box2", data=data,
            showcaps=False, boxprops={'facecolor': 'None'},
            showfliers=False, whiskerprops={'linewidth': 0}, ax=ax)

plt.show()
```

Output



Justification

This example illustrates the bee swarm plot creation with Matplotlib

Unit 12.14-Violin Graph

Definition

The violin plot is the combination of both box plot as well as histogram. This plot portrays the median and distribution of wide ranges of data.

Syntax

```
Syntax: violinplot(dataset, positions=None, vert=True, widths=0.5,  
showmeans=False, showextrema=True, showmedians=False, quantiles=None,  
points=100,  
bw_method=None, *, data=None)
```

Parameter

Parameters:

dataset: Array or a sequence of vectors.

The input data.

positions: array-like, default = [1, 2, ..., n].

Sets the positions of the violins. The ticks and limits are automatically set to match the positions.

vert: bool, default = True.

If true, creates a vertical violin plot. Otherwise, creates a horizontal violin plot.

widths: array-like, default = 0.5

Either a scalar or a vector that sets the maximal width of each violin. The default is 0.5, which uses about half of the available horizontal space.

showmeans: bool, default = False

If True, will toggle rendering of the means.

showextrema: bool, default = True

If True, will toggle rendering of the extrema.

showmedians: bool, default = False

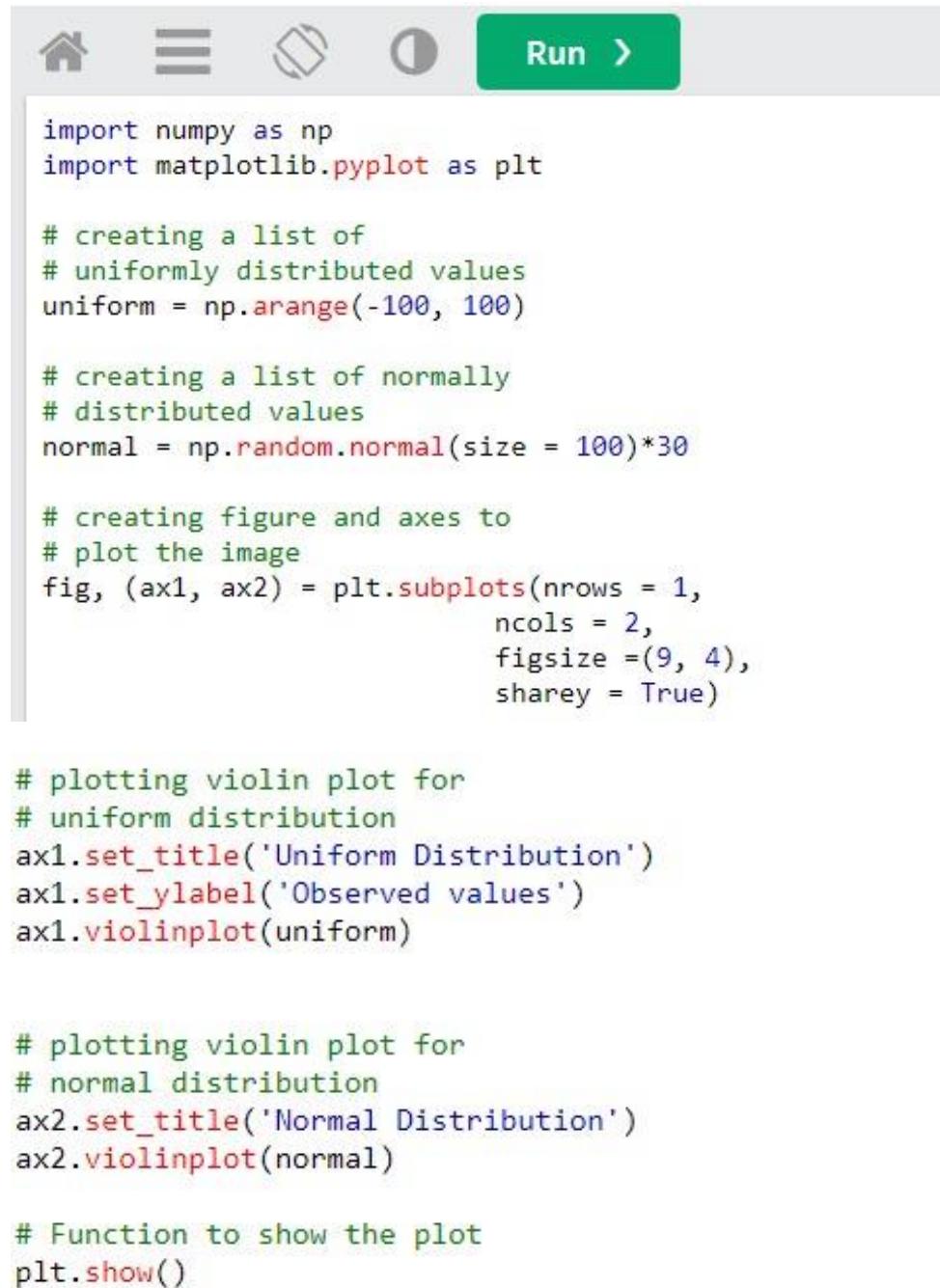
If True, will toggle rendering of the medians.

quantiles: array-like, default = None

If not None, set a list of floats in interval [0, 1] for each violin, which stands for the quantiles that will be rendered for that violin.

Example

Input



The screenshot shows a Jupyter Notebook interface with a toolbar at the top featuring icons for file, cell, and run, followed by a green "Run >" button. The main area contains the following Python code:

```
import numpy as np
import matplotlib.pyplot as plt

# creating a list of
# uniformly distributed values
uniform = np.arange(-100, 100)

# creating a list of normally
# distributed values
normal = np.random.normal(size = 100)*30

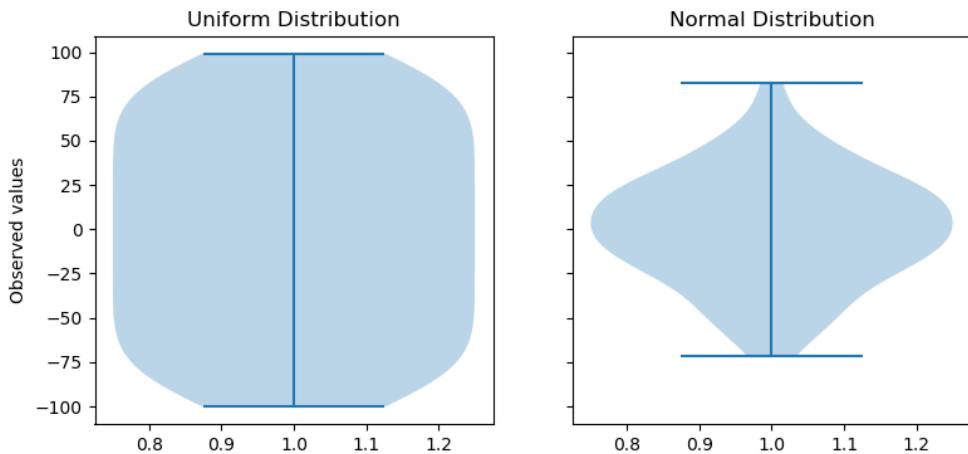
# creating figure and axes to
# plot the image
fig, (ax1, ax2) = plt.subplots(nrows = 1,
                               ncols = 2,
                               figsize =(9, 4),
                               sharey = True)

# plotting violin plot for
# uniform distribution
ax1.set_title('Uniform Distribution')
ax1.set_ylabel('Observed values')
ax1.violinplot(uniform)

# plotting violin plot for
# normal distribution
ax2.set_title('Normal Distribution')
ax2.violinplot(normal)

# Function to show the plot
plt.show()
```

Output



Justification

This example shows the creation of violin plot with the help of two distributions one is uniform distribution and next one is normal distribution

Unit 12.15-Working with Text

Definition

The Matplotlib.pyplot.text () function is mainly used for adding the text inside the plot with the help of mathematical expression. The syntax of this function adds different types of text in various arbitrary location of each of the axes

Syntax

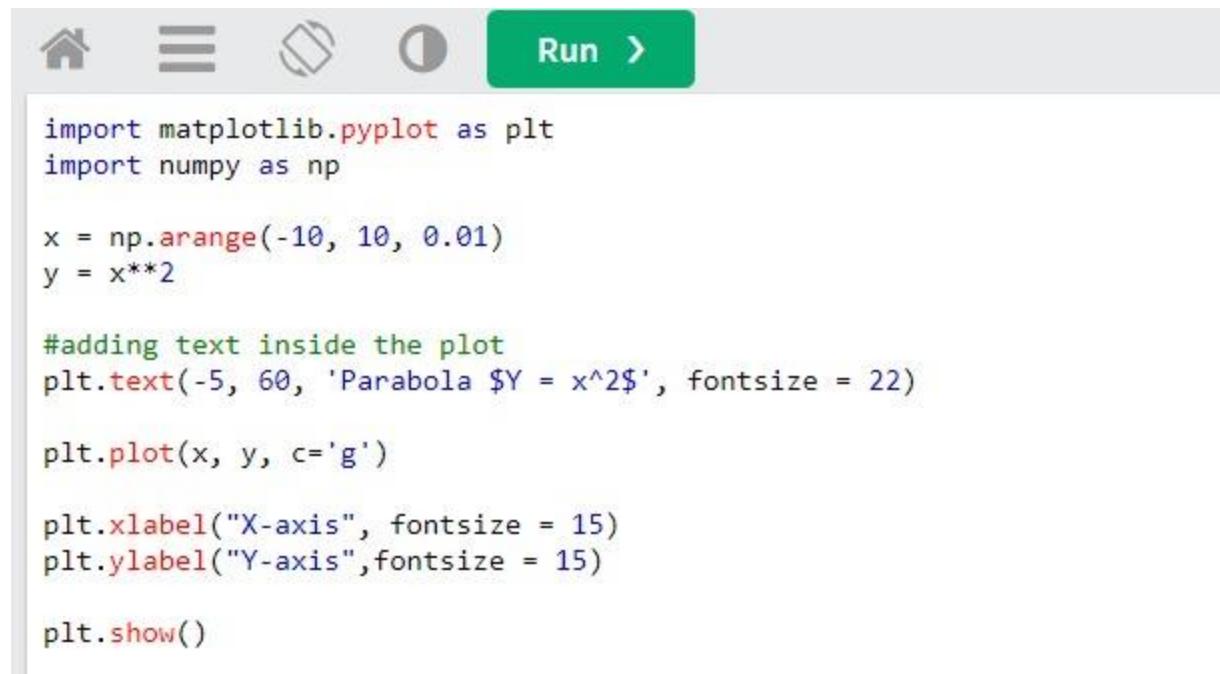
```
Syntax: matplotlib.pyplot.text(x, y, s, fontdict=None, **kwargs)
```

Parameter

text	Add text at an arbitrary location of the Axes.
annotate	Add an annotation, with an optional arrow, at an arbitrary location of theAxes.
xlabel	Add a label to the Axes's x-axis.
ylabel	Add a label to the Axes's y-axis.
title	Add a title to the Axes.
figtext	Add text at an arbitrary location of the Figure.
suptitle	Add a title to the Figure.

Example

Input



The screenshot shows a Jupyter Notebook interface with a toolbar at the top featuring icons for file, cell, cell type, cell execution, and a green 'Run' button. Below the toolbar is a code cell containing the following Python code:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.arange(-10, 10, 0.01)
y = x**2

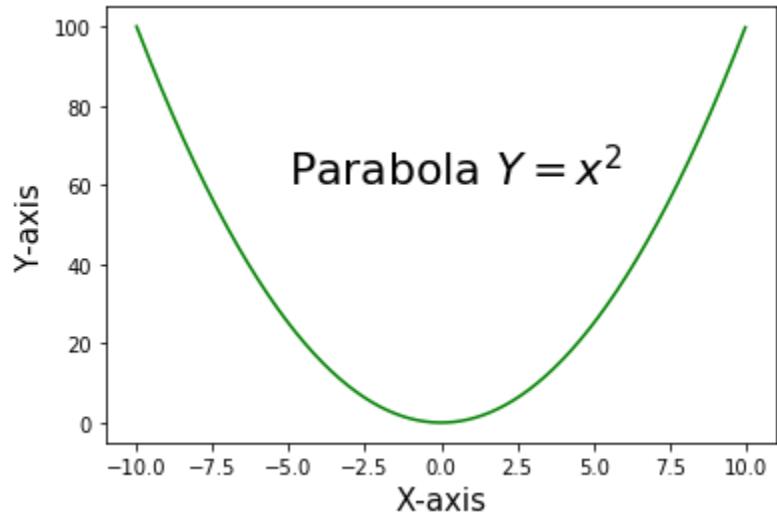
#adding text inside the plot
plt.text(-5, 60, 'Parabola $Y = x^2$', fontsize = 22)

plt.plot(x, y, c='g')

plt.xlabel("X-axis", fontsize = 15)
plt.ylabel("Y-axis", fontsize = 15)

plt.show()
```

Output



Justification

This example adds the mathematical equation inside the plot

MCQ

Question 1- Which of below mention could not use for data visualization?

- a) Maps
- b) Graphs
- c) Shape
- d) Chart

Question 2- Which of the following is not widely supported by the pyplot?

- a) Box Plot
- b) Histogram
- c) Pie Chart
- d) All

Question 3- In the below chart, the box that are surrounded by the red colour known as?



- a) Pie
- b) Bar
- c) Legend
- d) Line

Question 4- Plot which is used for the statistical summary known as?

- a) Box
- b) Line
- c) Bar
- d) Scatter

Question 5- Which of the following is not the parameter of the plot () of pyplot?

- a) Line Height
- b) Color
- c) Market
- d) Line Style

Question 6- To compare data, we can use __

- a) Bar
- b) Line
- c) Pie
- d) Scatter

Question 7- The plot is generated as per the data series which plotted irrespective of the xlim ()

- a) True
- b) False
- c) Cannot Determine

Question 8- It is not possible for plotting multiple series of value in the same bar graph

- a) True
- b) False
- c) Cannot Determine

Question 9- The line chart is plotted with the help of plot () function

- a) True
- b) False
- c) Cannot Determine

Case Studies

<https://www.kaggle.com/code/suhasvs/matplotlib-case-study>

<https://jovian.ai/aryab936/matplotlib-case-study>

Reference

https://www.w3schools.com/python/matplotlib_pyplot.asp

<https://pypi.org/project/matplotlib/>

<https://www.geeksforgeeks.org/python-introduction-matplotlib/>

<https://www.activestate.com/resources/quick-reads/what-is-matplotlib-in-python-how-to-use-it-for-plotting/>

Unit 13- Experiment no. 6

Unit 13.1- Program 10

“Write a python program to generate a simple bar graph using matplotlib. The graph should be properly labeled”

Program Code

```
import matplotlib.pyplot as pyplot

# Manual data setup
labels = ('Python', 'Java', 'JavaScript', 'C#', 'PHP', 'C,C++', 'R')
index = (1, 2, 3, 4, 5, 6, 7) # provides locations on x axis
sizes = [29.9, 19.1, 8.2, 7.3, 6.2, 5.9, 3.7]

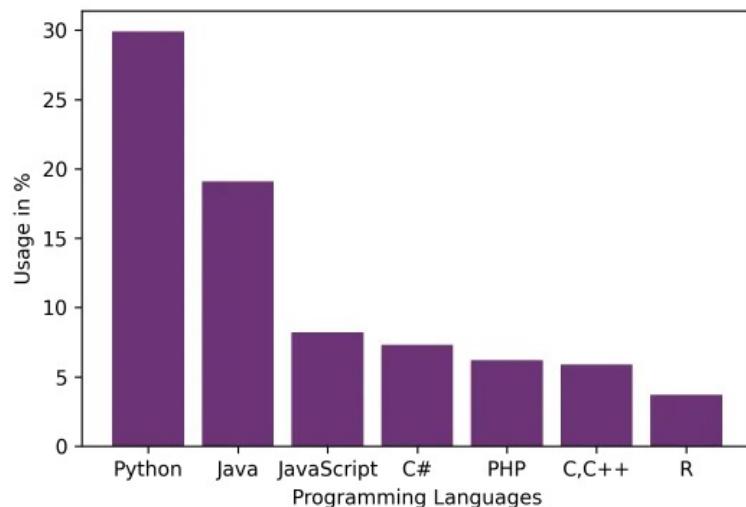
# bar chart setup
pyplot.bar(index, sizes, color="#6c3376", tick_label=labels)

# layout configuration
pyplot.ylabel('Usage in %')
pyplot.xlabel('Programming Languages')

# Save the chart file
#pyplot.savefig('filename.png', dpi=300)

# Print the chart
pyplot.show()
```

Output



Unit 13.2- Program 11

“Write a python program to generate Pie-chart using matplotlib. The graph should be properly labeled”

Program Code

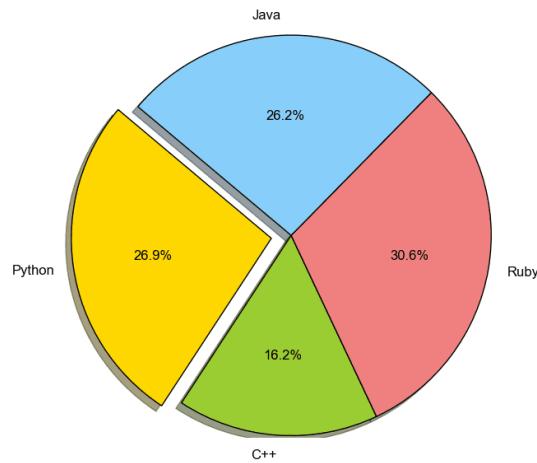
```
import matplotlib.pyplot as plt

# Data to plot
labels = 'Python', 'C++', 'Ruby', 'Java'
sizes = [215, 130, 245, 210]
colors = ['gold', 'yellowgreen', 'lightcoral', 'lightskyblue']
explode = (0.1, 0, 0, 0) # explode 1st slice

# Plot
plt.pie(sizes, explode=explode, labels=labels, colors=colors,
autopct='%1.1f%%', shadow=True, startangle=140)

plt.axis('equal')
plt.show()
```

Output



Unit 13.3- Program 12

“Write a Python program to plot the function $y = x^2$ using the matplotlib libraries”

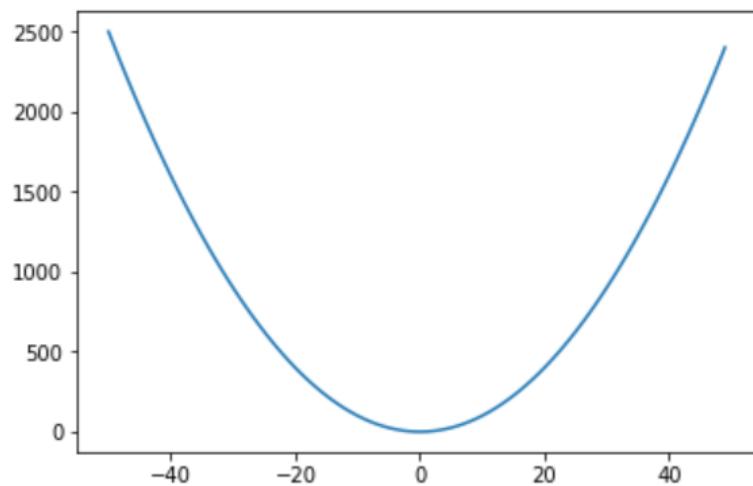
Program Code

The screenshot shows a Jupyter Notebook interface with a toolbar at the top featuring icons for file operations, cell selection, and execution. A green "Run" button is visible on the right. Below the toolbar, the Python code for generating a parabola is displayed:

```
import matplotlib.pyplot as plt
x_cords = range(-50,50)
y_cords = [x*x for x in x_cords]

plt.plot(x_cords, y_cords)
plt.show()
```

Output

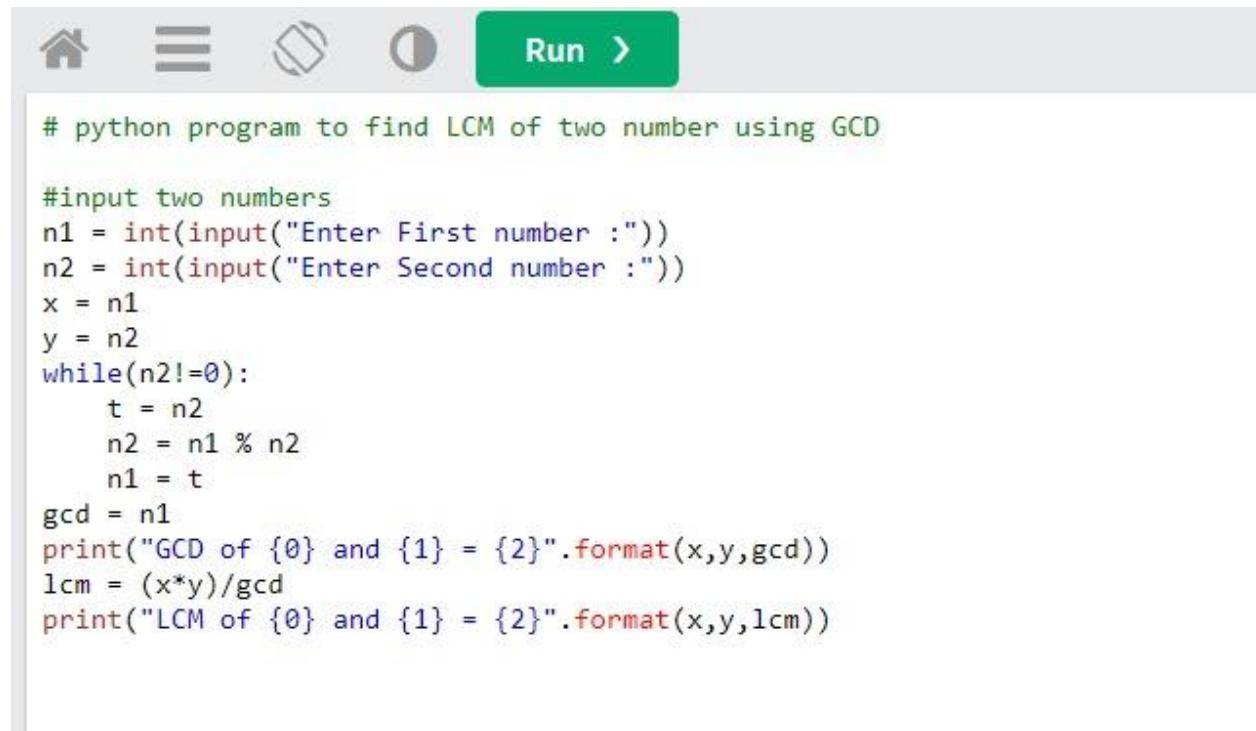


Unit 14- Experiment no. 7

Unit 14.1- Program 13

“Write a program in Python to compute the greatest common divisor and the least common multiple of two integers”

Program Code



```
# python program to find LCM of two number using GCD

#input two numbers
n1 = int(input("Enter First number :"))
n2 = int(input("Enter Second number :"))
x = n1
y = n2
while(n2!=0):
    t = n2
    n2 = n1 % n2
    n1 = t
gcd = n1
print("GCD of {0} and {1} = {2}".format(x,y,gcd))
lcm = (x*y)/gcd
print("LCM of {0} and {1} = {2}".format(x,y,lcm))
```

Output

```
Enter First number :54
Enter Second number :24
GCD of 54 and 24 = 6
LCM of 54 and 24 = 216.0
```

Unit 14.2- Program 14

“Write a program in Python to test if a number is equal to the sum of the cubes of its digits. Find the smallest and largest such numbers in the range of 100 to1000”

“Test if a number is equal to the sum of the cubes of its digits”

Program Code

```
n=input(" Enter a number ")
sum=0
for i in range(0,len(n)):
    sum=sum+pow(int(n[i]),3)

print("Sum of cube of digits : ",sum)
if(sum==int(n)):
    print("Digits are matching to cube : ", n)
else:
    print("Digits are NOT matching to cube : ", n)
```

Output

```
Enter a number 371
Sum of cube of digits : 371
Digits are matching to cube : 371
```

“Find the smallest and largest such numbers”

Program Code

```
my_list=[]
for n in range(10,100000): # increase this range
    sum=0
    my_str=str(n)
    k=len(my_str)
    for i in range(0,k):
        sum=sum+pow(int(my_str[i]),3)

    if(sum==int(n)):
        print("This is an matching number: ",n)
        my_list.append(n)
print("Highest number : ",max(my_list))
print("Lowest number : ",min(my_list))
```

Output

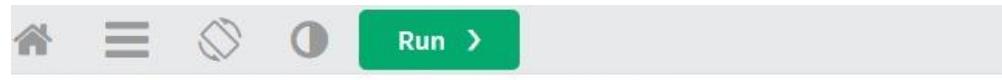
```
This is an matching number: 153
This is an matching number: 370
This is an matching number: 371
This is an matching number: 407
Highest number : 407
Lowest number : 153
```

Unit 15- Experiment no. 8

Unit 15.1- Program 15

“Write a program in python to read sort a list of integer elements using the bubble sort method. Display the sorted element on the screen”

Program Code



```
# Python program for implementation of Bubble Sort

def bubbleSort(arr):
    n = len(arr)
    # optimize code, so if the array is already sorted, it doesn't need
    # to go through the entire process
    swapped = False
    # Traverse through all array elements
    for i in range(n-1):
        # range(n) also work but outer loop will
        # repeat one time more than needed.
        # Last i elements are already in place
        for j in range(0, n-i-1):

            # traverse the array from 0 to n-i-1
            # Swap if the element found is greater
            # than the next element
            if arr[j] > arr[j + 1]:
                swapped = True
                arr[j], arr[j + 1] = arr[j + 1], arr[j]

    if not swapped:
        # if we haven't needed to make a single swap, we
        # can just exit the main loop.
    return

# Driver code to test above
arr = [64, 34, 25, 12, 22, 11, 90]

bubbleSort(arr)

print("Sorted array is:")
for i in range(len(arr)):
    print("% d" % arr[i], end=" ")
```

Output

```
Sorted array is:  
11 12 22 25 34 64 90
```

Unit 15.2- Program 16

“Write a program in python to find out the frequency of each element in a list using a dictionary”

Program Code



```
# Python program to count the frequency of  
# elements in a list using a dictionary  
  
def CountFrequency(my_list):  
  
    # Creating an empty dictionary  
    freq = {}  
    for item in my_list:  
        if (item in freq):  
            freq[item] += 1  
        else:  
            freq[item] = 1  
  
    for key, value in freq.items():  
        print ("% d : % d"%(key, value))  
  
# Driver function  
if __name__ == "__main__":  
    my_list =[1, 1, 1, 5, 5, 3, 1, 3, 3, 1, 4, 4, 4, 2, 2, 2]  
  
    CountFrequency(my_list)
```

Output

```
1 : 5
2 : 4
3 : 3
4 : 3
5 : 2
```

Module 3: Statistics

Key Learning Outcome

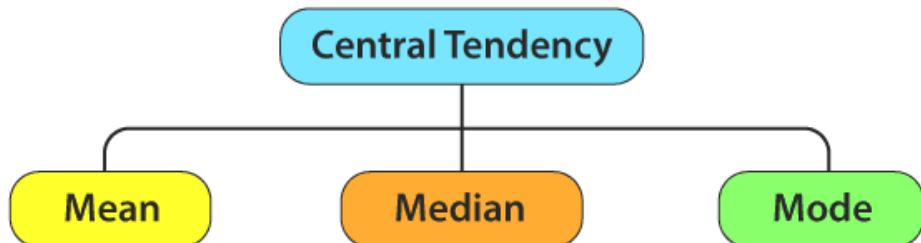
- To understand the core concepts of the Statestics such as Handling data frame, read csv, xls files. Handling null values or missing data and group by in real life example
- To evaluate the core concepts of the visualization using seaborn such as relational plots, categorical plots, distribution plots and matrix plots in real life example
- To demonstrate the core concepts of the visualization using plotly such as Line, bar, scatter, histogram, violin, Gantt, heatmap and 3D graph in real life example

Unit 16- Statistics

Unit 16.1- Central Tendency

Central tendency could not able to provide the specific information regarding the individual's data from the respective datasets, but it is associated with the datasets summary.

Measurement of Central Tendency



Mean

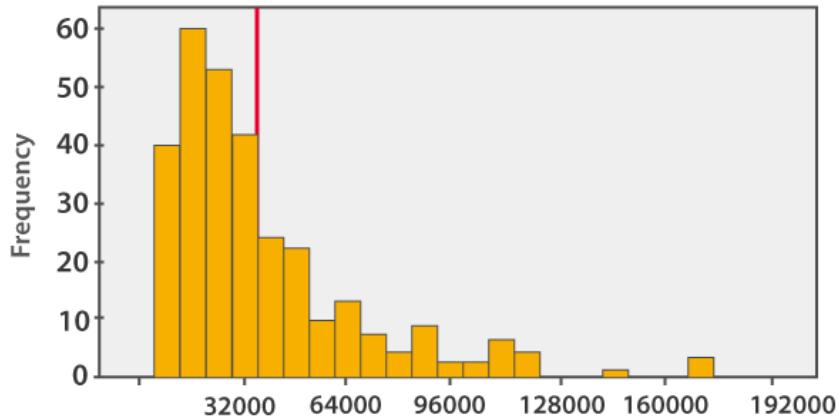
The mean value is representing dataset's particular average values. There is some of the measurement of mean that is mainly used for finding the optimum central tendency.

- Harmonic Mean
- Weighted Mean
- Geometric Mean

It has been observe that if the entire value in the respective datasets is same then GM, HM and WM will be the same.

Histogram of skewed continuous

Mean 36624



Median

Median is calculation of respective dataset's middle value so that entire datasets has been arranged both in the form of ascending order as well as descending order. When the respective datasets consists of the even value, then the median of the respective datasets has been found after the calculation of mean value. For example, there has different odd number of observation that arranged descending based order “23, 21, 18, 16, 15, 13, 12, 10, 9, 7, 6, 5, and 2”

Median odd
23
21
18
16
15
13
12
10
9
7
6
5
2

In this example, 12 is the median that have total 6 values above it and the 6 values under it

So based on this datasets, the two middle value obtained are 27 and 29. If we want to find the mean then it will be $(27+29)/2$ which yield the median for the data distribution is 28

Mode

The mode measurement estimates the occurring value of frequency in the respective datasets. For example in the below mention datasets “5, 4, 2, 3, 2, 1, 5, 4, 5”

Mode
5
5
5
4
4
3
2
2
1

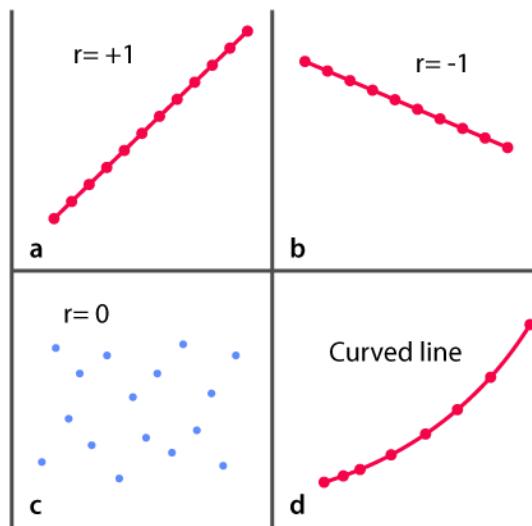
In the above example the most repeated value or mode is 5

Unit 16.2- Measure of Dispersion

The various measures of central value give us one single figure that represents the entire data. But the average alone cannot adequately describe a set of observations, unless all the observations are the same. Measures of dispersion help us in studying this important characteristic of a distribution. There are five measures of dispersion: Range, Inter-quartile range or Quartile Deviation, Mean deviation, Standard Deviation, and Lorenz curve. Among them, the first four are mathematical methods and the last one is the graphical method.

Unit 16.3- Correlation and Regression

The degree in which association is measured by “r” after its originator and the measurement of the linear association. The other types of complicated measurement is used if the respective curved line represents the internal relationship between two



The other graph represents the internal correlation. The coefficient of the correlation has measured on the scale which varies from the ranges of +1 to the -1 through the 0

If we take the X and Y is the random variable then the population correlation coefficient is formulated by

$$\rho_{X,Y} = \text{corr}(X, Y) = \frac{\text{cov}(X, Y)}{\sigma_X \sigma_Y} = \frac{E[(X - \mu_X)(Y - \mu_Y)]}{\sigma_X \sigma_Y}$$

Where,

ρ_{XY} = Population correlation coefficient between X and Y

μ_X = Mean of the variable X

μ_Y = Mean of the variable Y

σ_X = Standard deviation of X

σ_Y = Standard deviation of Y

E = Expected value operator

Cov = Covariance

The above formula has been written as

$$\rho_{X,Y} = \frac{\mathbb{E}(XY) - \mathbb{E}(X)\mathbb{E}(Y)}{\sqrt{\mathbb{E}(X^2) - \mathbb{E}(X)^2} \cdot \sqrt{\mathbb{E}(Y^2) - \mathbb{E}(Y)^2}}$$

The sample correlation coefficient formula is

$$r_{xy} = \frac{n \sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n \sum x_i^2 - (\sum x_i)^2} \sqrt{n \sum y_i^2 - (\sum y_i)^2}}$$

Example

Calculate the correlation coefficient for the given datasets

Person	Hand	Height
A	17	150
B	15	154
C	19	169
D	17	172
E	21	175

Solution

Pers on	Han d	Hei ght	$(x_i - \bar{x})$	$y_i - \bar{y}$	$(x_i - \bar{x})(y_i - \bar{y})$	$(x_i - \bar{x})^2$	$(y_i - \bar{y})^2$
A	17	150	-0.8	-14.0	11.2	0.6	196.0
B	15	154	-2.8	-10.0	28.0	7.8	100.0
C	19	169	1.2	5.0	6.0	1.4	25.0
D	17	172	-0.8	8.0	-6.4	0.6	64.0
E	21	175	3.2	11.0	35.2	10.2	121.0
Ave rage	17.8	164		Total	74.0	20.8	506.0

If we applied the formula

$$r_{xy} = \frac{\sum_1^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_1^n (x_i - \bar{x})^2 \sum_1^n (y_i - \bar{y})^2}}$$

$$= 0.72$$

Based on that, we stated that the data has the higher positive correlation

Unit 16.4- Introduction to Pandas

Introduction

The panda is the open source based library which is mainly implementing to work with the relational database for easily accessible. Pandas provide the different types of data structure as respective business operation to manipulate the numerical data. Panda is develop on the top of the another types of package known as the Numpy that provides technical supports for different multi dimensional based array

Usage of Panda in Python

Panda is one of the most efficient use tools for the both data science as well as machine learning. Handling different types of datasets using Python Panda is very fast and efficient with the help of the Panda series as well as data frame. These two data structure of Panda will help the Python Developer for manipulating the entire data in different types of ways. Based on the specific feature available in the Panda, we could say that Panda is the best choice for handing different types of data. In addition, Panda handle the record of different missing data, supports different types of file format and cleaning up the data so the user is capable to read as well as load the respective data in different types file format for example CSV and Excel format

Advantage

- Efficient and accessible for both manipulating as well as analyzing the data
- Data from various file objects has been loaded
- Easily handled of different types of missing data
- Provides the both time series based functionality as well as group by functionality

Unit 16.5- Data Structure

In panda, it supports total two types of data structure for manipulating the both numerical data as well as time series data.

- Series
- Dataframe

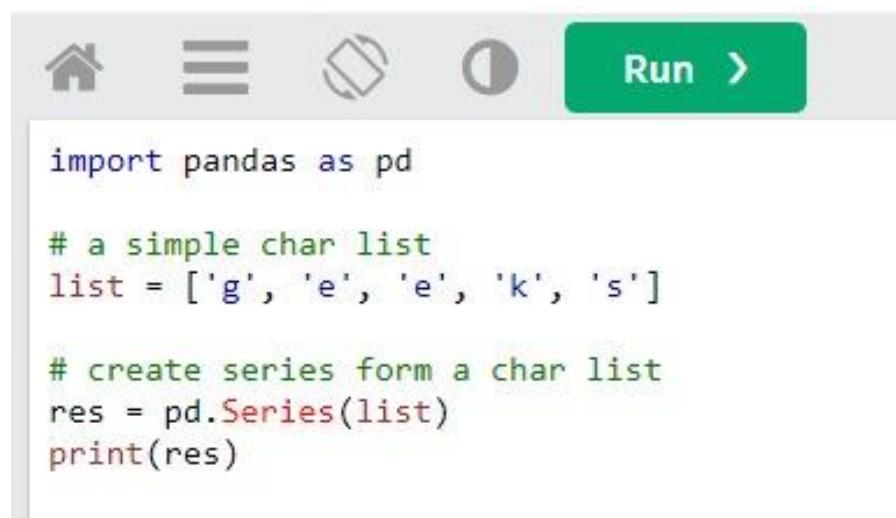
Series Data Structure

Panda is the one dimensional based labeled array that are capable of holding different types of data types.

Syntax of Series Data Structure

Example 1

Input



The image shows a screenshot of a Jupyter Notebook interface. At the top, there is a toolbar with icons for file, cell, kernel, and help, followed by a green 'Run' button with a right-pointing arrow. Below the toolbar is a code cell containing the following Python code:

```
import pandas as pd

# a simple char list
list = ['g', 'e', 'e', 'k', 's']

# create series form a char list
res = pd.Series(list)
print(res)
```

Output

```
0    g
1    e
2    e
3    k
4    s
dtype: object
```

Justification

This example shows the Python series that hold the char data types

Example 2

Input



```
import pandas as pd

dic = { 'Id': 1013, 'Name': 'MOhe',
        'State': 'Maniput', 'Age': 24}

res = pd.Series(dic)
print(res)
```

Output

```
Id      1013
Name    Mohe
State   Maniput
Age     24
dtype: object
```

Justification

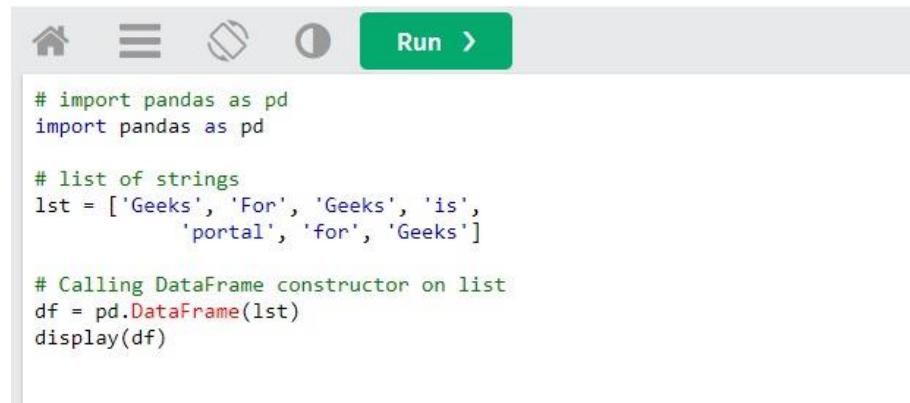
This example shows the Python series that hold the dictionary

Data Frame Data Structure

The data frame in Pandas is the 2D size mutable and heterogeneous based data structure with the help of the labeled axes (combination of row and columns). Thus in Data Frame Data Structure, the respective data has been aligned in the form of tabular data in the form of row and Column.

Example 1

Input



```
# import pandas as pd
import pandas as pd

# list of strings
lst = ['Geeks', 'For', 'Geeks', 'is',
       'portal', 'for', 'Geeks']

# Calling DataFrame constructor on list
df = pd.DataFrame(lst)
display(df)
```

Output

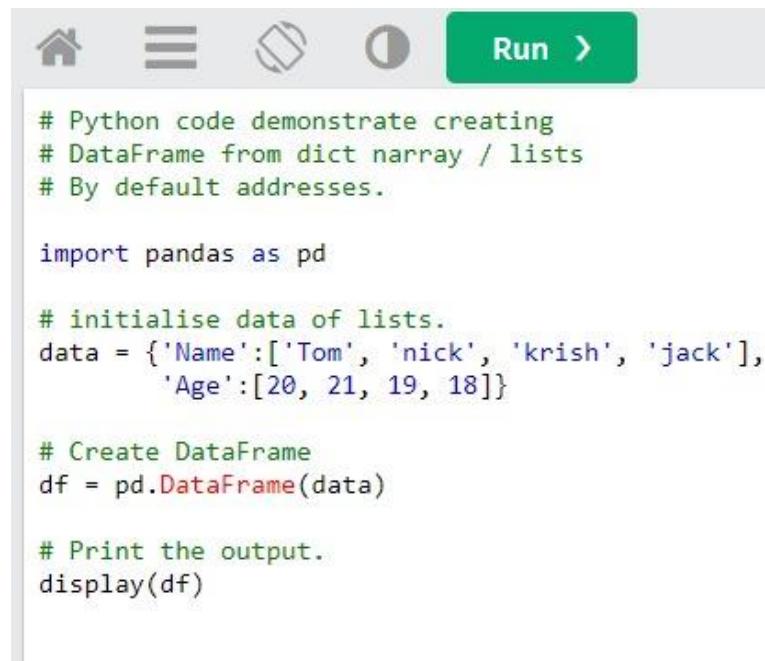
	0
0	Geeks
1	For
2	Geeks
3	is
4	portal
5	for
6	Geeks

Justification

In the above example, the data frame has been developed with the help of the single list or the list of data

Example 2

Input



The screenshot shows a Jupyter Notebook interface with a toolbar at the top featuring icons for home, file, cell, and run. A green 'Run >' button is highlighted. Below the toolbar is a code cell containing the following Python code:

```
# Python code demonstrate creating
# DataFrame from dict narray / lists
# By default addresses.

import pandas as pd

# initialise data of lists.
data = {'Name':['Tom', 'nick', 'krish', 'jack'],
        'Age':[20, 21, 19, 18]}

# Create DataFrame
df = pd.DataFrame(data)

# Print the output.
display(df)
```

Output

	Name	Age
0	Tom	20
1	nick	21
2	krish	19
3	jack	18

Justification

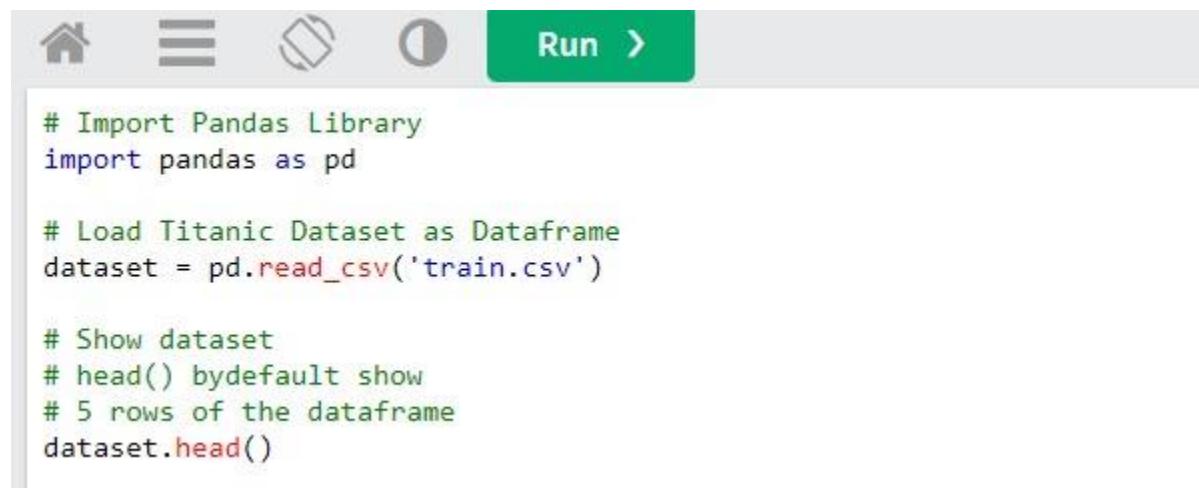
In this example, the developer created a data frame from the dict of the lists

Unit 16.6- Implement Statistics

Performing different types of statistical operation in the Python programming has been minimized to the form of single command line with the help of Panda. So in Python, Panda is used for implementing different types of statistical operation for processing. In this book, we take the example as Titanic Survival Datasets for implementing statistical operation (<https://drive.google.com/file/d/17XszLX8k8O0hYx9kfYH07NDzKJKsk9WS/view>)

Example

Input



```
# Import Pandas Library
import pandas as pd

# Load Titanic Dataset as Dataframe
dataset = pd.read_csv('train.csv')

# Show dataset
# head() bydefault show
# 5 rows of the dataframe
dataset.head()
```

Output

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	Cumings, Mrs. John Bradley (Florence Briggs Th... Heikkinen, Miss. Laina	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3 Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	Allen, Mr. William Henry	male	35.0	1	0	113803	53.1000	C123	S
4	5	0	3			0	0	373450	8.0500	NaN	S

1. Mean

Syntax

```
Syntax: DataFrame/Series.mean(self, axis=None, skipna=None, level=None, numeric_only=None, **kwargs)
```

Example

Input



The screenshot shows a Jupyter Notebook interface with a toolbar at the top featuring icons for file, cell, and run. Below the toolbar is a code cell containing the following Python code:

```
# Calculate the Mean
# of 'Age' column
mean = dataset['Age'].mean()

# Print mean
print(mean)
```

Output

```
29.69911764705882
```

Justification

The program calculates the mean value of the above datasets with the help of Data Frame or Series.mean () method

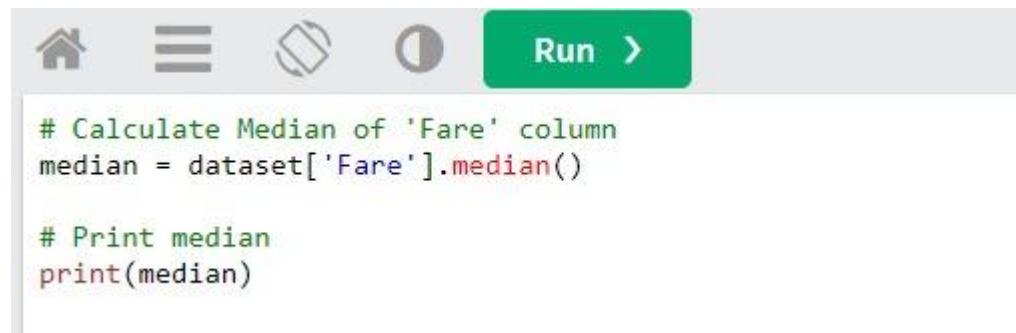
2. Median

Syntax

```
Syntax: DataFrame/Series.median(self, axis=None, skipna=None, level=None,  
numeric_only=None, **kwargs)
```

Example

Input



```
# Calculate Median of 'Fare' column  
median = dataset['Fare'].median()  
  
# Print median  
print(median)
```

Output

```
14.4542
```

Justification

The program calculates the median value of the above datasets with the help of Data frame or series. Median () method

3. Mode

Syntax

```
Syntax: DataFrame/Series.mode(self, axis=0, numeric_only=False, dropna=True)
```

Example

Input

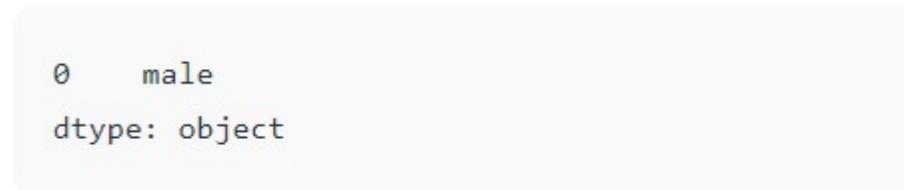


The screenshot shows a Jupyter Notebook interface with a toolbar at the top featuring icons for file, cell, and run. Below the toolbar is a code cell containing the following Python code:

```
# Calculate Mode of 'Sex' column
mode = dataset['Sex'].mode()

# Print mode
print(mode)
```

Output



```
0    male
dtype: object
```

Justification

It calculates the mode or most frequency based value with the help of the Data Frame. Mode () method

4. Count

Syntax

```
Syntax: DataFrame/Series.count(self, axis=0, level=None,  
numeric_only=False)
```

Example

Input



```
# Calculate Count of 'Ticket' column  
count = dataset['Ticket'].count()  
  
# Print count  
print(count)
```

Output

```
891
```

Justification

Calculation of the count or frequency of the not null based value is calculated with the help of Data Frame or Series.count () method

5. Standard Deviation

Syntax

```
Syntax: DataFrame/Series.std(self, axis=None, skipna=None, level=None,  
ddof=1, numeric_only=None, **kwargs)
```

Example

Input



The screenshot shows a Jupyter Notebook interface. At the top, there are several icons: a house (Home), three horizontal lines (File), a document (Cell), and a circular arrow (Kernel). To the right of these is a green 'Run' button with a white arrow. Below the toolbar, the code is displayed in a cell:

```
# Calculate Standard Deviation  
# of 'Fare' column  
std = dataset['Fare'].std()  
  
# Print standard deviation  
print(std)
```

Output

```
49.693428597180905
```

Justification

Calculation of the standard deviation of the value is calculated with the help of Data Frame or Series.std () method

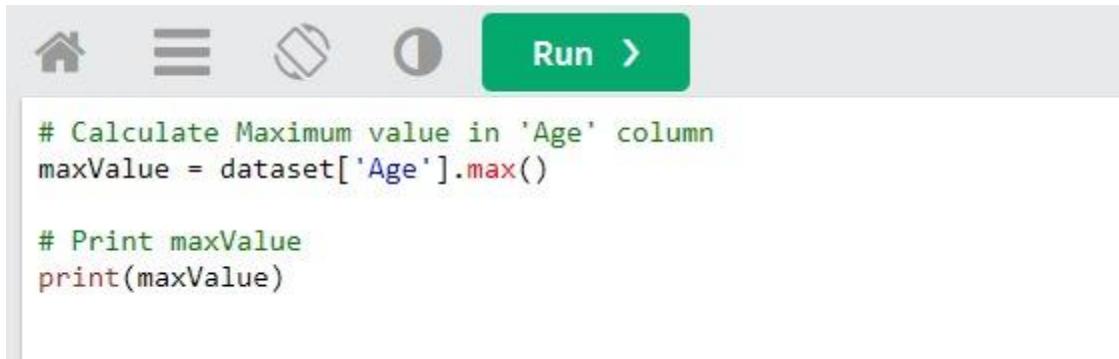
6. Max

Syntax

```
Syntax: DataFrame/Series.max(self, axis=None, skipna=None, level=None,  
numeric_only=None, **kwargs)
```

Example

Input



```
# Calculate Maximum value in 'Age' column  
maxValue = dataset['Age'].max()  
  
# Print maxValue  
print(maxValue)
```

Output

```
80.0
```

Justification

Calculation of the maximum value with the help of Data Frame or Series.max () method

7. Min

Syntax

```
Syntax: DataFrame/Series.min(self, axis=None, skipna=None, level=None,  
numeric_only=None, **kwargs)
```

Example

Input



```
# Calculate Minimum value in 'Fare' column  
minValue = dataset['Fare'].min()  
  
# Print minValue  
print(minValue)
```

Output

```
0.0000
```

Justification

Calculation of the minimum value with the help of Data Frame or Series.min () method

8. Describe

Syntax

```
Syntax: DataFrame/Series.describe(self: ~FrameOrSeries, percentiles=None,  
include=None, exclude=None)
```

Example

Input



```
# Statistical summary  
dataset.describe()
```

Output

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

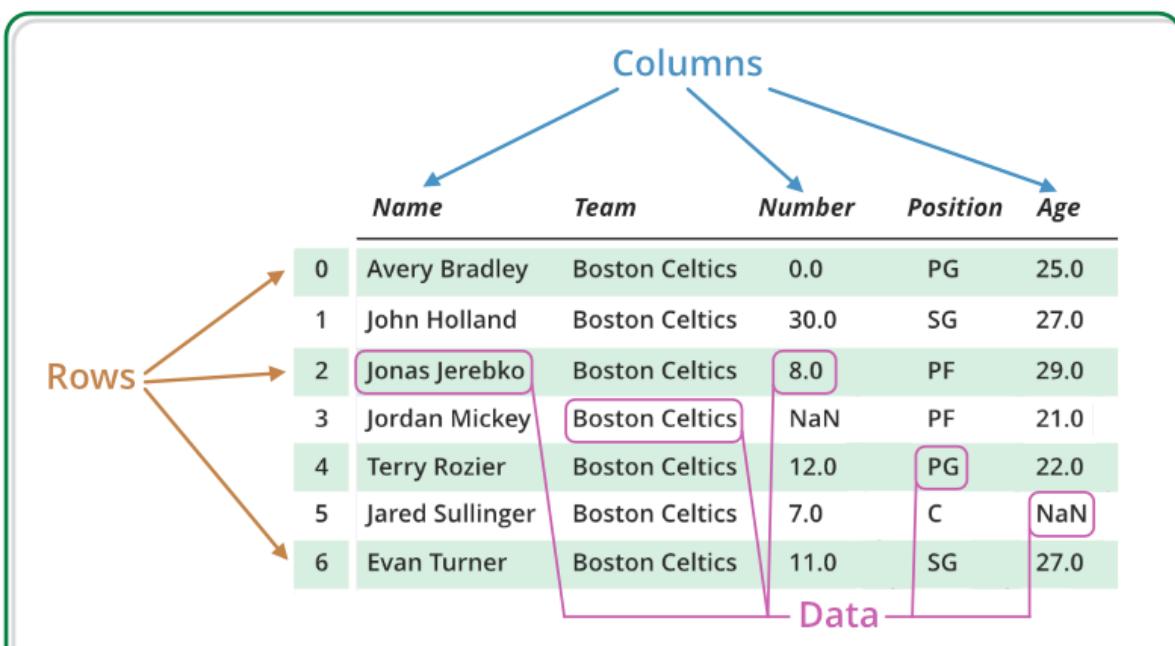
Justification

Summarization of the descriptive statistics with the help of Data Frame or Series.describe () method

Unit 16.7- Handling Data Frame

Data frame is 2D based data structure in which the respective data has been aligned in the form of tabular fashion with the help of both row as well as column. The data frame in Panda is consists of three of the most important element

- Data
- Column
- Row



Creation of Panda Data Frame

Creation of Panda Data Frame using Lists

Example

Input

```
# import pandas as pd
import pandas as pd

# list of strings
lst = ['Geeks', 'For', 'Geeks', 'is',
       'portal', 'for', 'Geeks']

# Calling DataFrame constructor on list
df = pd.DataFrame(lst)
print(df)
```

Output

0
0 Geeks
1 For
2 Geeks
3 is
4 portal
5 for
6 Geeks

Creation of Panda Data Frame from dict of ndarray

Example

Input

```
# Python code demonstrate creating  
# DataFrame from dict narray / lists  
# By default addresses.  
  
import pandas as pd  
  
# intialise data of lists.  
data = {'Name': ['Tom', 'nick', 'krish', 'jack'],  
        'Age': [20, 21, 19, 18]}  
  
# Create DataFrame  
df = pd.DataFrame(data)  
  
# Print the output.  
print(df)
```

Output

	Name	Age
0	Tom	20
1	nick	21
2	krish	19
3	jack	18

Column Selection

For selecting the column in the Panda data frame, the user either needs to access the entire column with the help of the column name

Example

Input

```
# Import pandas package
import pandas as pd

# Define a dictionary containing employee data
data = {'Name':['Jai', 'Princi', 'Gaurav', 'Anuj'],
        'Age':[27, 24, 22, 32],
        'Address':['Delhi', 'Kanpur', 'Allahabad', 'Kannauj'],
        'Qualification':['Msc', 'MA', 'MCA', 'Phd']}

# Convert the dictionary into DataFrame
df = pd.DataFrame(data)

# select two columns
print(df[['Name', 'Qualification']])
```

Output

	Name	Qualification
0	Jai	Msc
1	Princi	MA
2	Gaurav	MCA
3	Anuj	Phd

Row Selection

Panda provided the updated method for retrieving the row from the data frame with the help of the Data Frame.loc []

Example

Input

```
# importing pandas package
import pandas as pd

# making data frame from csv file
data = pd.read_csv("nba.csv", index_col ="Name")

# retrieving row by loc method
first = data.loc["Avery Bradley"]
second = data.loc["R.J. Hunter"]

print(first, "\n\n\n", second)
```

Output

```
Team      Boston Celtics
Number          0
Position        PG
Age            25
Height         6-2
Weight         180
College        Texas
Salary    7.73034e+06
Name: Avery Bradley, dtype: object

Team      Boston Celtics
Number          28
Position        SG
Age            22
Height         6-5
Weight         185
College       Georgia State
Salary    1.14864e+06
Name: R.J. Hunter, dtype: object
```

Indexing Data Frame using Indexing Operator []

The indexing operator is mainly used for referring to the square brackets for the single column selection

Example

Input

```
# importing pandas package
import pandas as pd

# making data frame from csv file
data = pd.read_csv("nba.csv", index_col ="Name")

# retrieving columns by indexing operator
first = data["Age"]

print(first)
```

Output

Name	
Avery Bradley	25.0
Jae Crowder	25.0
John Holland	27.0
R.J. Hunter	22.0
Jonas Jerebko	29.0
Amir Johnson	29.0
Jordan Mickey	21.0
Kelly Olynyk	25.0
Terry Rozier	22.0
Marcus Smart	22.0
Jared Sullinger	24.0
Isaiah Thomas	27.0
▪	▪
▪	▪
▪	▪
Joe Ingles	28.0
Chris Johnson	26.0
Trey Lyles	20.0
Shelvin Mack	26.0
Raul Neto	24.0
Tibor Pleiss	26.0
Jeff Withey	26.0
NaN	NaN

Name: Age, Length: 458, dtype: float64

Selection of Data Frame

For selecting the single row with the help of .iloc[], the user pass the single integer to the .iloc[] function

Example

Input

```
# importing pandas package
import pandas as pd

# making data frame from csv file
data = pd.read_csv("nba.csv", index_col ="Name")

# retrieving columns by indexing operator
first = data["Age"]

print(first)
```

Output

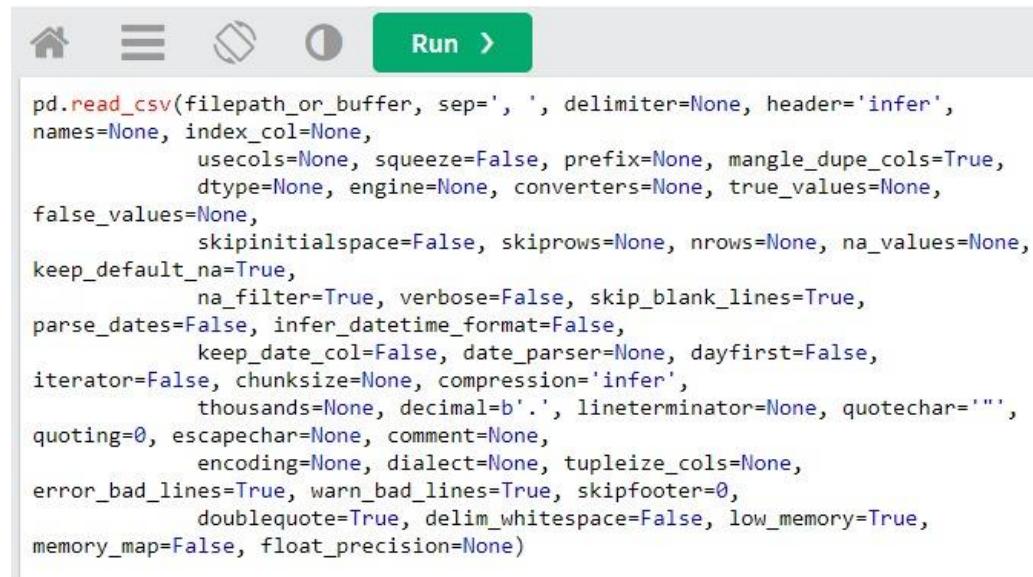
Name	Age
Avery Bradley	25.0
Jae Crowder	25.0
John Holland	27.0
R.J. Hunter	22.0
Jonas Jerebko	29.0
Amir Johnson	29.0
Jordan Mickey	21.0
Kelly Olynyk	25.0
Terry Rozier	22.0
Marcus Smart	22.0
Jared Sullinger	24.0
Isaiah Thomas	27.0
...	...
...	...
Joe Ingles	28.0
Chris Johnson	26.0
Trey Lyles	20.0
Shelvin Mack	26.0
Raul Neto	24.0
Tibor Pleiss	26.0
Jeff Withey	26.0
NaN	NaN
Name: Age, Length: 458, dtype: float64	

Unit 16.8- Read csv, xls files

Unit 16.8.1- Read CSV files

Most of the data that are used for the internal analysis is available on the form of CSV file. For accessing the specific data from the CSV file, the user need to implement the read _csv () which retrieved the data in the form of the data frame

Syntax



```
pd.read_csv(filepath_or_buffer, sep=', ', delimiter=None, header='infer',
            names=None, index_col=None,
            usecols=None, squeeze=False, prefix=None, mangle_dupe_cols=True,
            dtype=None, engine=None, converters=None, true_values=None,
            false_values=None,
            skipinitialspace=False, skiprows=None, nrows=None, na_values=None,
            keep_default_na=True,
            na_filter=True, verbose=False, skip_blank_lines=True,
            parse_dates=False, infer_datetime_format=False,
            keep_date_col=False, date_parser=None, dayfirst=False,
            iterator=False, chunksize=None, compression='infer',
            thousands=None, decimal=b'.', lineterminator=None, quotechar='',
            quoting=0, escapechar=None, comment=None,
            encoding=None, dialect=None, tupleize_cols=None,
            error_bad_lines=True, warn_bad_lines=True, skipfooter=0,
            doublequote=True, delim_whitespace=False, low_memory=True,
            memory_map=False, float_precision=None)
```

Parameter

Filepath_	This is the location of the file that need to be retrieved with the help of that function
Sep	Sep stands for the separator
Header	It accepts the both int and row number for the usage of column name as well as starting of the data
usecols	This parameter is mainly used for retracing the selected column from the csv file

nrow	This parameter represents the total numbers of rows that user will display alongside with the records
Index_col	If none so there are no proper index number will display with the respective records
Squeeze	If true and only the one column is passed then it will return the Panda series
Skiprows	It skipped passed row in the new data frame
Names	This parameter allows for retrieving the column with the new names

Example

```

# importing Pandas library
import pandas as pd

pd.read_csv(filepath_or_buffer = "pokemon.csv")

# makes the passed rows header
pd.read_csv("pokemon.csv", header =[1, 2])

# make the passed column as index instead of 0, 1, 2, 3....
pd.read_csv("pokemon.csv", index_col ='Type')

# uses passed cols only for data frame
pd.read_csv("pokemon.csv", usecols =[["Type"]])

# returns pandas series if there is only one column
pd.read_csv("pokemon.csv", usecols =[["Type"]], squeeze = True)

# skips the passed rows in new series
pd.read_csv("pokemon.csv", skiprows = [1, 2, 3, 4])

```

Justification

This example shows the successful retrieval of pokemon.csv file

Unit 16.8.2- Read XLS files

In python Panda, the user can read the xls file from the python datasets with the help of the specific method

Step

In the first step, the developer needs to use the read_excel method that loads the respective xls data file into the data frame of Panda

```
read_excel(filename)
```

In the next step, the user reads the Excel file with the help of Python Panda. The below mention code reads the respective excel data into the Python Datasets

```
from pandas import DataFrame, read_csv
import matplotlib.pyplot as plt
import pandas as pd

file = r'data/Presidents.xls'
df = pd.read_excel(file)
print(df['Occupation'])
```

The data frame has been used, as it can seen in the below figure

```
from pandas import DataFrame, read_csv
import matplotlib.pyplot as plt
import pandas as pd

file = r'data/Presidents.xls'
df = pd.read_excel(file)

# remove messy data
df = df[df['Years in office'] != 'n/a']

# show data
print('Min: ', df['Years in office'].min())
print('Max: ', df['Years in office'].max())
print('Sum: ', df['Years in office'].sum())
```

The datasets that are used in this book is gathered from the
https://qrc.depaul.edu/Excel_Files/Presidents.xls

A	B	C	D	E	F	G	H	I	J	K	L
President	Years in office	Year first inaugurated	Age at inauguration	State elected from	# of electoral votes	# of popular votes	National total votes	Total electoral votes	Rating points	Political Party	Occupation
George Washington	8	1789	57	Virginia	69 NA()	NA()		69	842	None	Planter
John Adams	4	1797	61	Massachusetts	132 NA()	NA()		139	598	Federalist	Lawyer
Thomas Jefferson	8	1801	57	Virginia	73 NA()	NA()		137	711	Democratic-Republican	Planter, Lawyer
James Madison	8	1809	57	Virginia	122 NA()	NA()		176	567	Democratic-Republican	Lawyer
James Monroe	8	1817	58	Virginia	183 NA()	NA()		221	602	Democratic-Republican	Lawyer
John Quincy Adams	4	1825	57	Massachusetts	84 NA()	NA()		261	564	Democratic-Republican	Lawyer
Andrew Jackson	8	1829	61	Tennessee	178	642,553	1,148,018	261	632	Democrat	Lawyer
Martin Van Buren	4	1837	54	New York	170	764,176	1,503,534	294	429	Democrat	Lawyer
William Henry Harrison	0.8	1841	68	Ohio	234	1,275,390	2,411,808	294	329	Whig	Soldier
James K. Polk	4	1845	49	Tennessee	170	1,339,494	2,703,659	275	632	Democrat	Lawyer
Zachary Taylor	1	1849	64	Louisiana	163	1,361,393	2,879,184	290	447	Whig	Soldier
Franklin Pierce	4	1853	48	New Hampshire	254	1,607,510	3,161,830	296	286	Democrat	Lawyer
James Buchanan	4	1857	65	Pennsylvania	174	1,836,072	4,054,647	296	259	Democrat	Lawyer
Abraham Lincoln	4	1861	52	Illinois	180	1,865,908	4,685,561	303	900	Republican	Lawyer
Ulysses S. Grant	8	1869	46	Illinois	214	3,013,650	5,722,440	294	403	Republican	Soldier
Rutherford B. Hayes	4	1877	54	Ohio	185	4,034,311	8,413,101	369	477	Republican	Lawyer
James A. Garfield	0.5	1881	49	Ohio	214	4,446,158	9,210,420	369	444	Republican	Lawyer
Grover Cleveland	4	1885	47	New York	219	4,874,621	10,049,754	401	576	Democrat	Lawyer
Benjamin Harrison	4	1889	55	Indiana	233	5,443,892	11,383,320	401	426	Republican	Lawyer
Grover Cleveland	4	1893	55	New York	277	5,551,883	12,056,097	444	576	Democrat	Lawyer
William McKinley	4	1897	54	Ohio	271	7,108,480	13,935,738	447	601	Republican	Lawyer
William Howard Taft	4	1909	51	Ohio	321	7,676,258	14,882,734	483	491	Republican	Lawyer
Woodrow Wilson	8	1913	56	New Jersey	435	6,293,152	15,040,963	531	723	Democrat	Educator
Warren G. Harding	2	1921	55	Ohio	404	16,133,314	26,753,766	531	326	Republican	Editor
Herbert Hoover	4	1929	54	California	444	21,411,991	36,790,364	531	400	Republican	Engineer

Unit 16.9- Handling Null Values or Missing Data

Missing data is occurred when there is no proper information is provided for the one or more than the one unit

Checking the Missing Value Using notnull () and isnull ()

For checking the missing value, both of the function checks whether the respective value is NaN or not

Example

Input

```
# importing pandas as pd
import pandas as pd

# importing numpy as np
import numpy as np

# dictionary of lists
dict = {'First Score':[100, 90, np.nan, 95],
        'Second Score': [30, 45, 56, np.nan],
        'Third Score':[np.nan, 40, 80, 98]}

# creating a dataframe from list
df = pd.DataFrame(dict)

# using isnull() function
df.isnull()
```

Output

	First Score	Second Score	Third Score
0	False	False	True
1	False	False	False
2	True	False	False
3	False	True	False

Checking the Missing Value Using fillna(), replace() and interpolate()

Both the above mention three function helps programmer for filling the null values in the datasets. For example, Interpolate () function filling the NA value in the respective data frame and at the same time, this function uses the different types of interpolation technique for filling the different types of missing values

Example

Input

```
# importing pandas as pd
import pandas as pd

# importing numpy as np
import numpy as np

# dictionary of lists
dict = {'First Score':[100, 90, np.nan, 95],
        'Second Score': [30, 45, 56, np.nan],
        'Third Score':[np.nan, 40, 80, 98]}

# creating a dataframe from dictionary
df = pd.DataFrame(dict)

# filling missing value using fillna()
df.fillna(0)
```

Output

	First Score	Second Score	Third Score
0	100.0	30.0	0.0
1	90.0	45.0	40.0
2	0.0	56.0	80.0
3	95.0	0.0	98.0

Unit 16.10- Group by

The data frame. Group by () function in Panda data frame is mainly used for splitting the respective data into the integrated groups based on the different types of criteria.

Syntax

```
Syntax: DataFrame.groupby(by=None, axis=0, level=None, as_index=True,
sort=True, group_keys=True, squeeze=False, **kwargs)
```

Example

Input



```
# importing pandas as pd
import pandas as pd

# Creating the dataframe
df = pd.read_csv("nba.csv")

# Print the dataframe
df
```

Output

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
1	Jae Crowder	Boston Celtics	99.0	SF	25.0	6-6	235.0	Marquette	6796117.0
2	John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston University	NaN
3	R.J. Hunter	Boston Celtics	28.0	SG	22.0	6-5	185.0	Georgia State	1148640.0
4	Jonas Jerebko	Boston Celtics	8.0	PF	29.0	6-10	231.0	NaN	5000000.0
5	Amir Johnson	Boston Celtics	90.0	PF	29.0	6-9	240.0	NaN	12000000.0
6	Jordan Mickey	Boston Celtics	55.0	PF	21.0	6-8	235.0	LSU	1170960.0
7	Kelly Olynyk	Boston Celtics	41.0	C	25.0	7-0	238.0	Gonzaga	2165160.0
8	Terry Rozier	Boston Celtics	12.0	PG	22.0	6-2	190.0	Louisville	1824360.0
9	Marcus Smart	Boston Celtics	36.0	PG	22.0	6-4	220.0	Oklahoma State	3431040.0
10	Jared Sullinger	Boston Celtics	7.0	C	24.0	6-9	260.0	Ohio State	2569260.0
11	Isaiah Thomas	Boston Celtics	4.0	PG	27.0	5-9	185.0	Washington	6912869.0
12	Evan Turner	Boston Celtics	11.0	SG	27.0	6-7	220.0	Ohio State	3425510.0
13	James Young	Boston Celtics	13.0	SG	20.0	6-6	215.0	Kentucky	1749840.0
14	Tyler Zeller	Boston Celtics	44.0	C	26.0	7-0	253.0	North Carolina	2616975.0
15	Bojan Bogdanovic	Brooklyn Nets	44.0	SG	27.0	6-8	216.0	NaN	3425510.0

Justification

In the following example, the developer uses the groupby () function for grouping the different types of data that are based on a team.

Unit 16.11- Data visualization with Different Existing Dataset

Basic Plotting and Plot

The functionality for both series as well as Data Frame wrap under the matplotlib library plot () method

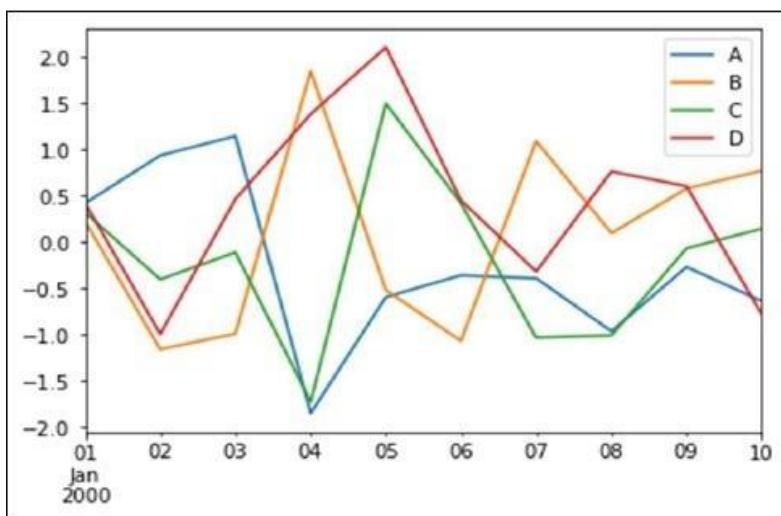
Input

```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.randn(10,4),index=pd.date_range('1/1/2000',
   periods=10), columns=list('ABCD'))

df.plot()
```

Output



Justification

If the above index consists of only date, then it will call the gct().autofmt_xdate () for formatting the entire X-axis that has shown in the above figure.

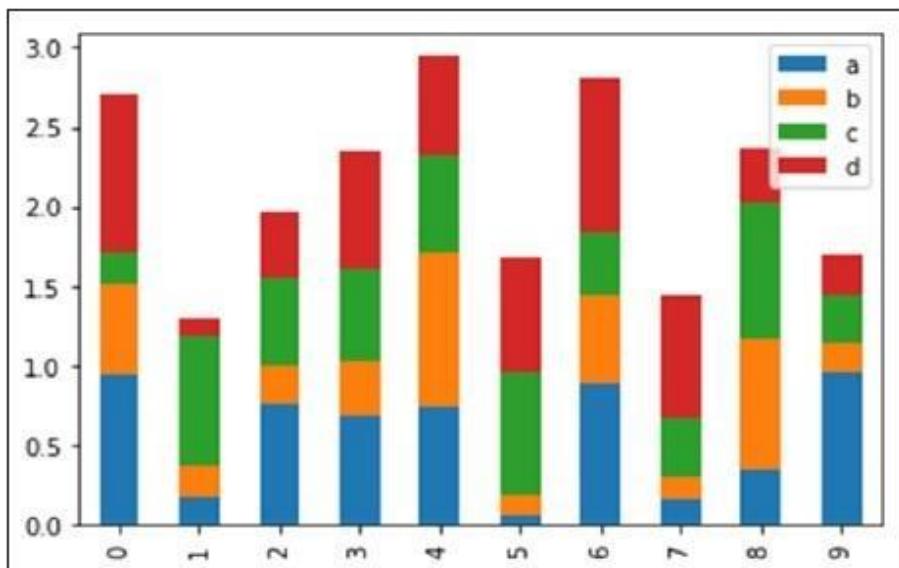
Bar Plot

Input

```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.rand(10,4),columns=['a','b','c','d'])
df.plot.bar()
```

Output

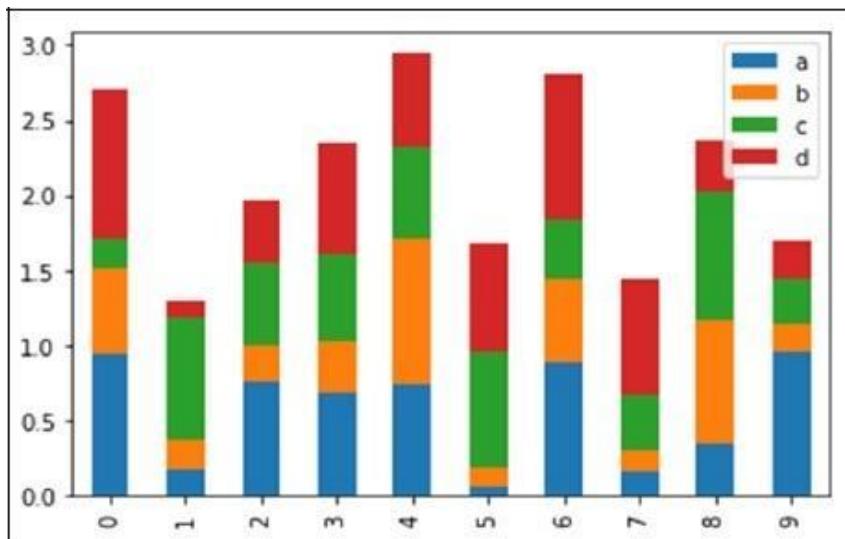


For producing the stacked bar plot, the user need to pass Stacked = True-

Input

```
import pandas as pd
df = pd.DataFrame(np.random.rand(10,4),columns=['a','b','c','d'])
df.plot.bar(stacked=True)
```

Output



For using the horizontal bar plots, the user need to use the barh method

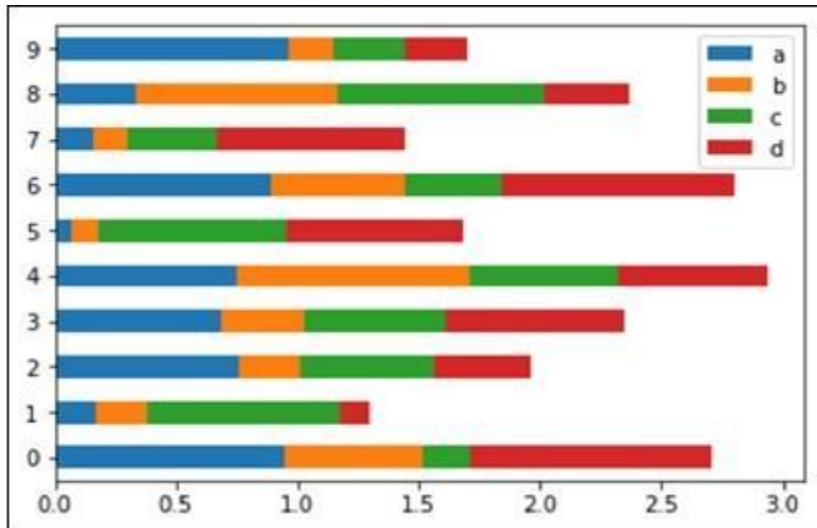
Input

```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.rand(10,4),columns=['a','b','c','d'])

df.plot.barh(stacked=True)
```

Output



Histogram

Histogram has been plotted with the help of the plot.hist () method. In this program, we could specified the total number of bin

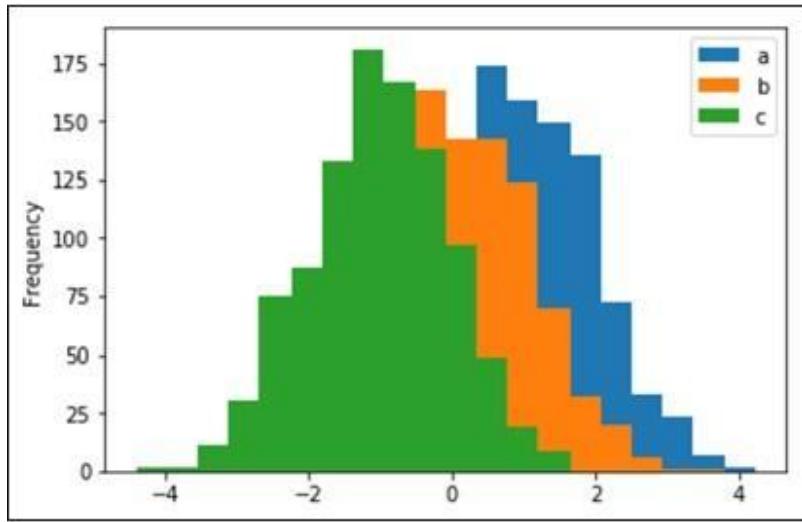
Input

```
import pandas as pd
import numpy as np

df = pd.DataFrame({'a':np.random.randn(1000)+1,'b':np.random.randn(1000),'c':np.random.randn(1000) - 1}, columns=['a', 'b', 'c'])

df.plot.hist(bins=20)
```

Output



For plotting the different types of histogram for every column, the developer needs to use the following code

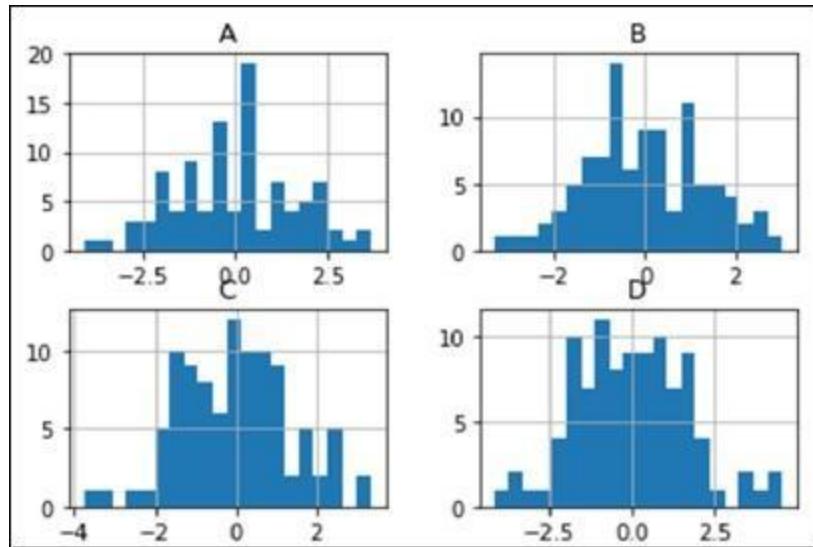
Input

```
import pandas as pd
import numpy as np

df=pd.DataFrame({'a':np.random.randn(1000)+1,'b':np.random.randn(1000),'c':np.random.randn(1000) - 1}, columns=['a', 'b', 'c'])

df.diff.hist(bins=20)
```

Output



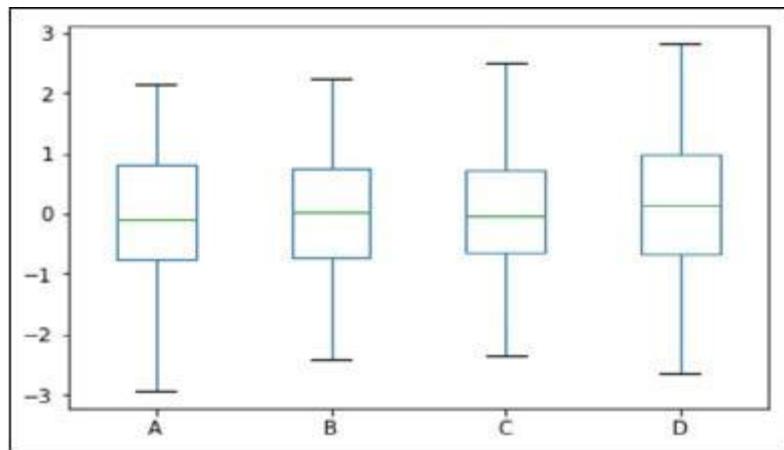
Box Plot

The box plot has been drawn by calling the both Series.box.plot () as well as Data Frame.box.plot () for visualizing the entire value distribution for every column

Input

```
import pandas as pd
import numpy as np
df = pd.DataFrame(np.random.rand(10, 5), columns=['A', 'B', 'C', 'D', 'E'])
df.plot.box()
```

Output



Area Plot

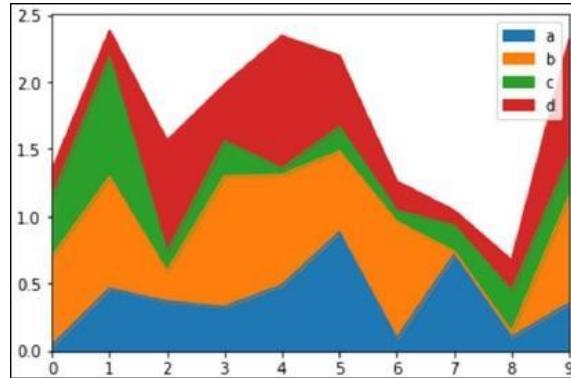
The area plot is implemented with the help of either Series.plot.area () or DataFrame.plot.area () methods

Input

```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.rand(10, 4), columns=['a', 'b', 'c', 'd'])
df.plot.area()
```

Output



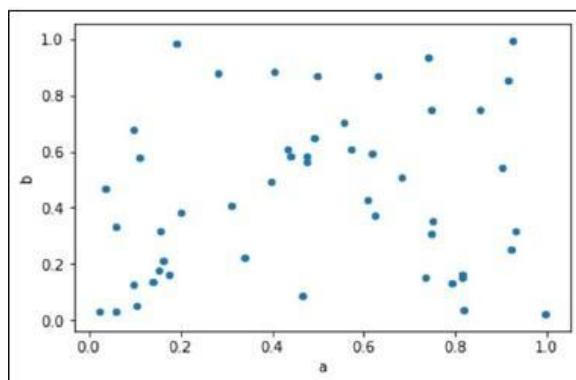
Scatter Plot

The scatter plot has been implemented with the help of the DataFrame.plot.scatter () methods

Input

```
import pandas as pd
import numpy as np
df = pd.DataFrame(np.random.rand(50, 4), columns=['a', 'b', 'c', 'd'])
df.plot.scatter(x='a', y='b')
```

Output



Pie Chart

Pie chart is developed with the help of the DataFrame.plot.pie () method

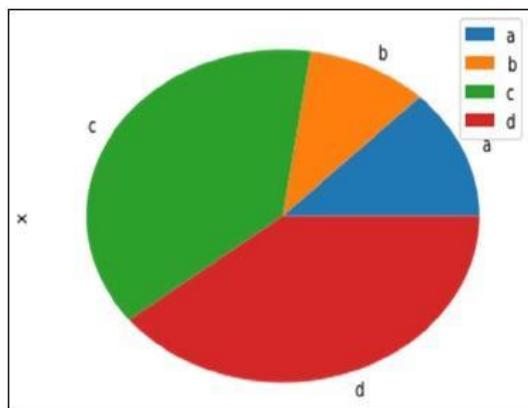
Input



```
import pandas as pd
import numpy as np

df = pd.DataFrame(3 * np.random.rand(4), index=['a', 'b', 'c', 'd'], columns=['x'])
df.plot.pie(subplots=True)
```

Output



Unit 16.12- Random Walks

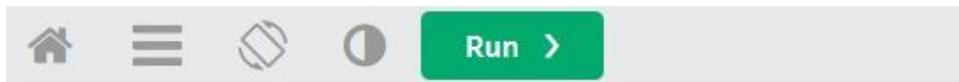
The random walk is the specific mathematical based object which described a path which consists of the succession of the different random step on some of the mathematical space such as integer. The example of random walk is the random walk for the integer number line which starts at 0 and then in each of the steps it moves to the +1 or the -1.

Library used in Random Walk

- Numpy
- Matplotlib
- Random

Example

Input



```
# Python code for 1-D random walk.
import random
import numpy as np
import matplotlib.pyplot as plt

# Probability to move up or down
prob = [0.05, 0.95]

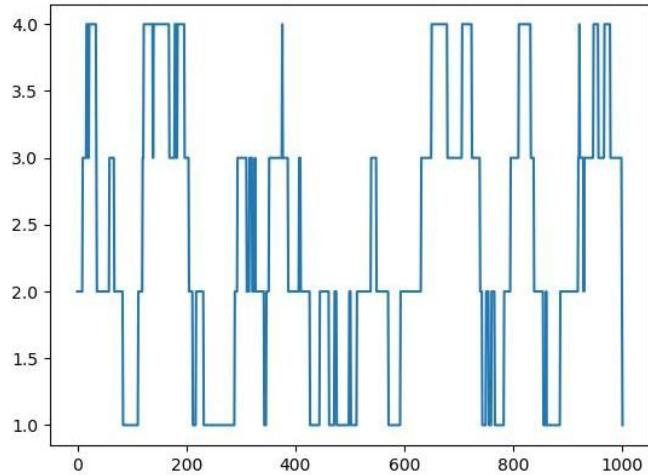
# statically defining the starting position
start = 2
positions = [start]

# creating the random points
rr = np.random.random(1000)
downp = rr < prob[0]
upp = rr > prob[1]

for idownp, iupp in zip(downp, upp):
    down = idownp and positions[-1] > 1
    up = iupp and positions[-1] < 4
    positions.append(positions[-1] - down + up)

# plotting down the graph of the random walk in 1D
plt.plot(positions)
plt.show()
```

Output



Justification

This is the example of 1-D random walk in the Python Panda in which the initiation of the integer number is 0, in each of the step it moves to +1 or -1 with the same equal probability

Unit 16.13- PDF

The user can also read different types of table from the PDF and then convert the table into the Data frame of Pandas. Python has its own tabula-py that has enabled the user for converting the PDF file into the respective CSV files. The user can also work with the preexisting based PDF in the python with the help of the PyPDF2 based package that has used for the different types of PDF operation.

MCQ

Question 1- The individual run score by the batsman in 5 ODI are 31, 97, 112, 63 and 12. So what is the standard deviation?

- a) 25.79
- b) 25.01
- c) 25.654
- d) 25.34

Question 2- Find out the Mode of the call received on the 7 consecutive days 11,13,13,17,19,23,25?

- a) 11
- b) 12
- c) 13
- d) 14

Question 3- Find out the Median of the call received on the 7 consecutive days 11,13,17,13,23,25,19

- a) 18
- b) 17
- c) 14
- d) 16

Q 4- When a specific mean of a number is 18 then what the Mean is of the sampling distribution?

- a) 11
- b) 13
- c) 15
- d) 18

Question 5- If the probability of the hitting object is 0.8 then what is the variance?

- a) 0.16
- b) 0.12
- c) 0.11
- d) 0.10

Question 6- Find the mean of tossing 4 coin?

- a) 1
- b) 2
- c) 3
- d) 4

Question 7- Output

```
import pandas as pd

s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])

print s['a']
```

- a) 4
- b) 5
- c) 6
- d) 1

Question 8- Output

```
import pandas as pd
import numpy as np

s = pd.Series(np.random.randn(4))

print s.ndim
```

- a) 1
- b) 5
- c) 7
- d) 9

Question 9- Output

```
import pandas as pd  
  
import numpy as np  
  
s = pd.Series(np.random.randn(2))  
  
print s.size
```

- a) 12
- b) 6
- c) 9
- d) 2

Case Studies

<https://www.learndatasci.com/tutorials/data-science-statistics-using-python/>

Reference

<https://realpython.com/python-statistics/#:~:text=Python's%20statistics%20is%20a%20built,%2D%20and%20multi%2Ddimensional%20arrays.>

https://www.w3schools.com/python/module_statistics.asp

<https://www.geeksforgeeks.org/statistics-with-python/>

<https://scipy-lectures.org/packages/statistics/index.html>

Unit 17- Experiment no. 9

Unit 17.1- Program 17

“Perform different statistics operations on dataset taken from kaggle or datagov”

IMDB Movie data Analysis

<https://www.kaggle.com/code/robinjrjr/imdb-movie-data-analysis>

About The Datasets

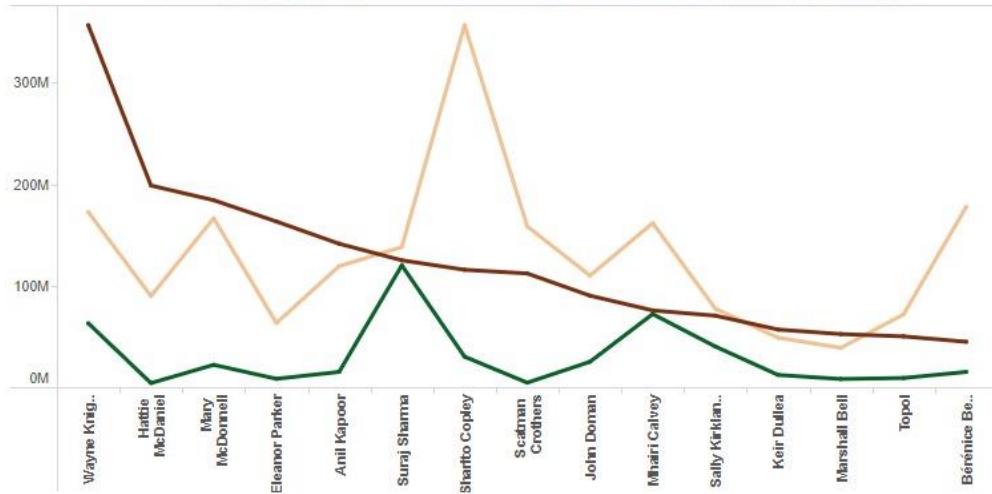
The dataset used for the visualization comes from 5000+ movies reviews and details housed in the IMDB website. This dashboard attempts to visualize how the highest grossing, highly reviewed movies characteristics looks like when it comes to its reviews and gross revenue. This dashboard also attempts to provide a visual understanding of the topmost genres and top keywords used in the movie description that do becomesuperhit earning millions !!

Import

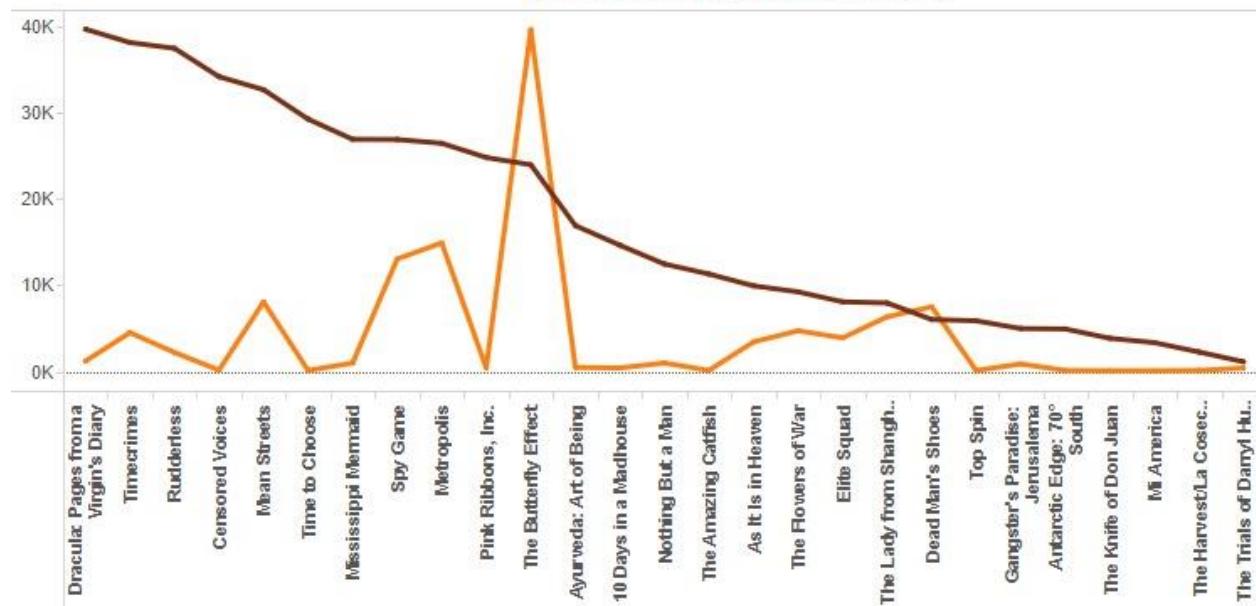
```
import os
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(style="white", color_codes=True)
# Any results you write to the current directory are saved as output.
```

Data Visualization

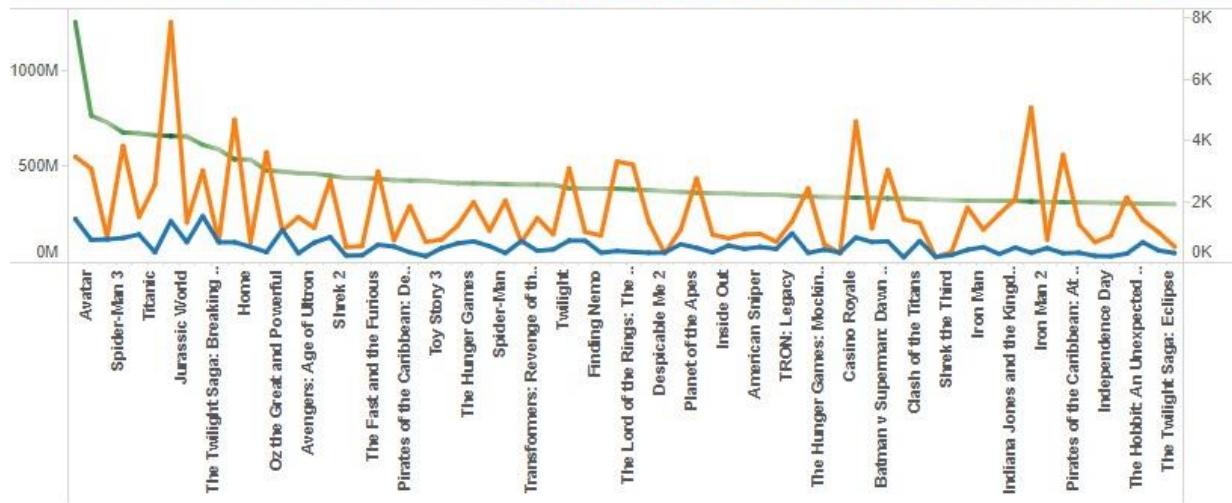
Lowest Gross revenue Actors /less Actor's facebook likes/ less Sum of Budget



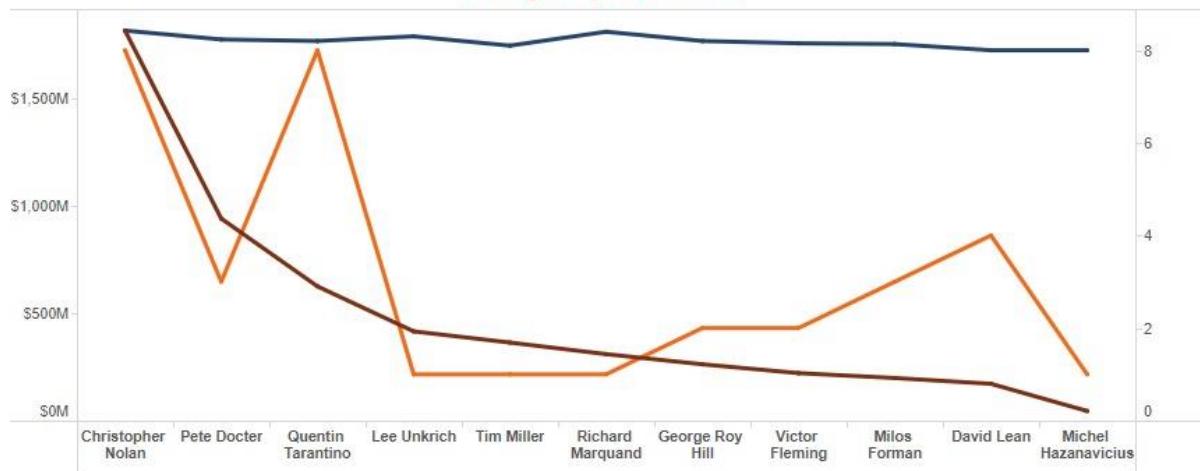
Less Gross revenue on films directly proportional to the low num of users they had for reviews



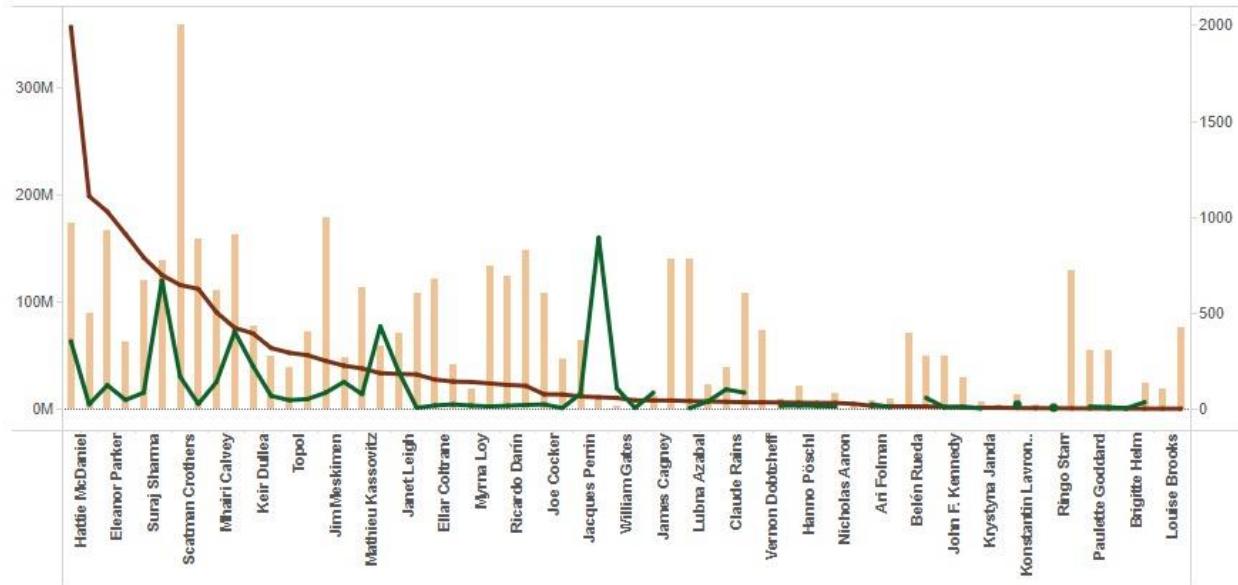
High Grossing Movies were correlated with large number of users and higher amongst other critics reviews



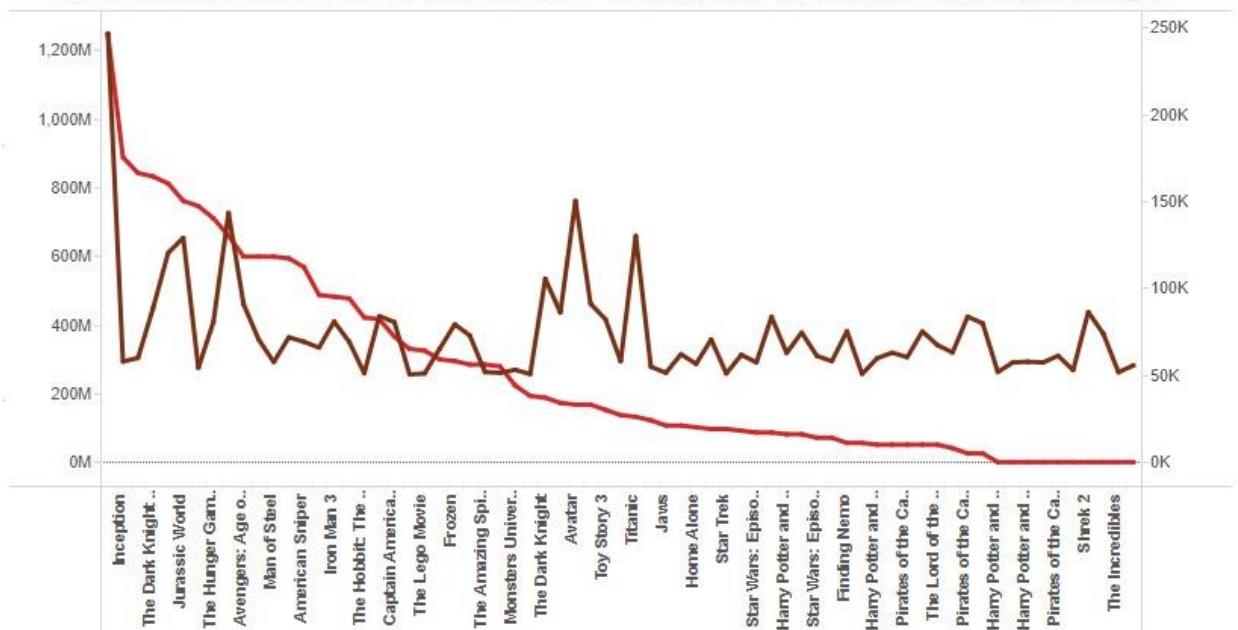
Highest Grossing Directors directed the highest number of movies and had the highest Avg Imdb Score collecting the highest gross revenue



Highest Grossing Actors had an average Actor's facebook likes (they were not the actors with the most highest facebook likes) and an average amount of budget invested on them but the movies they were associated made the highest Gross revenue



Gross revenue on films directly proportional to the number of movie facebook likes that film got.



Unit 18- Data visualization using seaborn and plotly

Unit 18.1- Introduction to Seaborn

Seaborn is the most effective virtualization library for different statistical graph plotting in the Python. Seaborn provides different types of default style and associate color pallettes for making the statistical plot even more attractive. So it has been developed to the top of the matplotlib by integrating the specific data structure from the Pandas. Seaborn is capable for making the entire virtualization technique lots easier for exploring the data. In addition, seaborn provides the datasets based API so that the user can switch between the different types of virtual representation for the same type of datasets for easy processing

In this Unit, we are going to use the ‘tips’ datasets
(<https://www.kaggle.com/code/sanjanabasu/tips-dataset>)

The tips datasets is consisted of different types of information about the individuals who had their respective food in the restaurant and at the same time whether or not they left a tip.

Unit 18.2- Relational Plots

Relational plot is mainly used to visualize the entire statistical relationship between the each of the data points. This plot provide the different types of access capability to the various access level based function which shows the internal relationship between the two of the variable with the help of semantic mapping of respective subsets.

Parameter

Parameter	Value	Use
x, y	numeric	Input data variables
Data	Dataframe	Dataset that is being used.
hue, size, style	name in data; optional	Grouping variable that will produce elements with different colors.
kind	scatter or line; default : scatter	defines the type of plot, either scatterplot() or lineplot()
row, col	names of variables in data; optional	Categorical variables that will determine the faceting of the grid.
col_wrap	int; optional	"Wrap" the column variable at this width, so that the column facets span multiple rows.
row_order, col_order	lists of strings; optional	Order to organize the rows and columns of the grid.

Example

Input

Datasets Import



```
# importing the library
import seaborn as sns

# reading the dataset
data = sns.load_dataset('tips')

# printing first five entries
print(data.head())
```

Output

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

For drawing the relational plot in seaborn, it provides the total three function that we are going to discussed in this book

Relplot ()

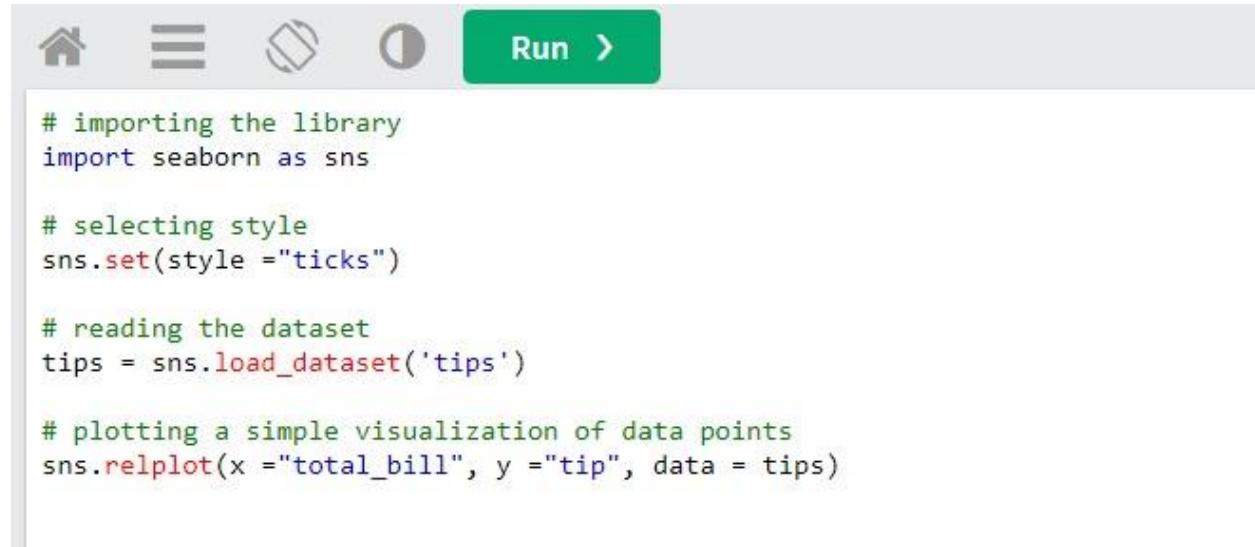
The relplot () function provides special capability for the user to access some of the different types of axes level based function that shows the internal relationship between the two of the variable with the help of subset's semantic mapping

Syntax

```
seaborn.relplot(x=None, y=None, data=None, **kwargs)
```

Example 1

Input



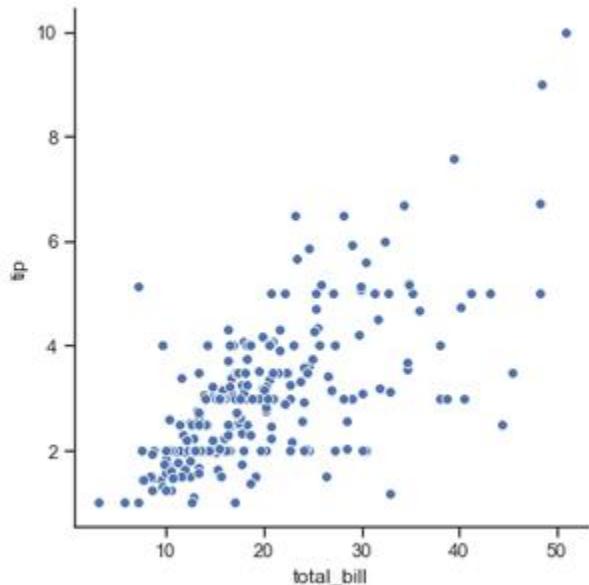
```
# importing the library
import seaborn as sns

# selecting style
sns.set(style ="ticks")

# reading the dataset
tips = sns.load_dataset('tips')

# plotting a simple visualization of data points
sns.relplot(x = "total_bill", y = "tip", data = tips)
```

Output



Justification

This above example visualize the most basic plot for showing the entire data points in the form of tips datasets

Example 2

Input



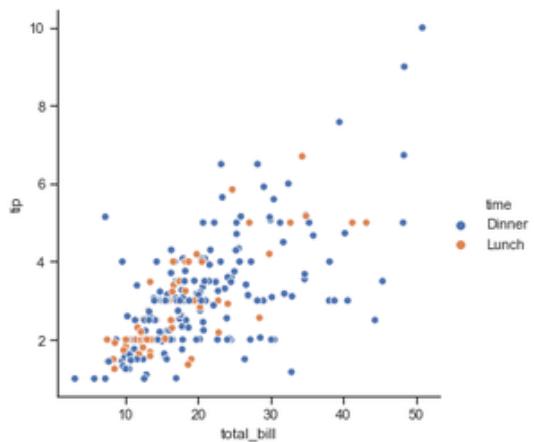
```
# importing the library
import seaborn as sns

# selecting style
sns.set(style = "ticks")

# reading the dataset
tips = sns.load_dataset('tips')

sns.relplot(x="total_bill",
            y="tip",
            hue="time",
            data=tips)
```

Output



Justification

This example performs the group operation for the data points that are based on the specific category (time)

Example 3

Input



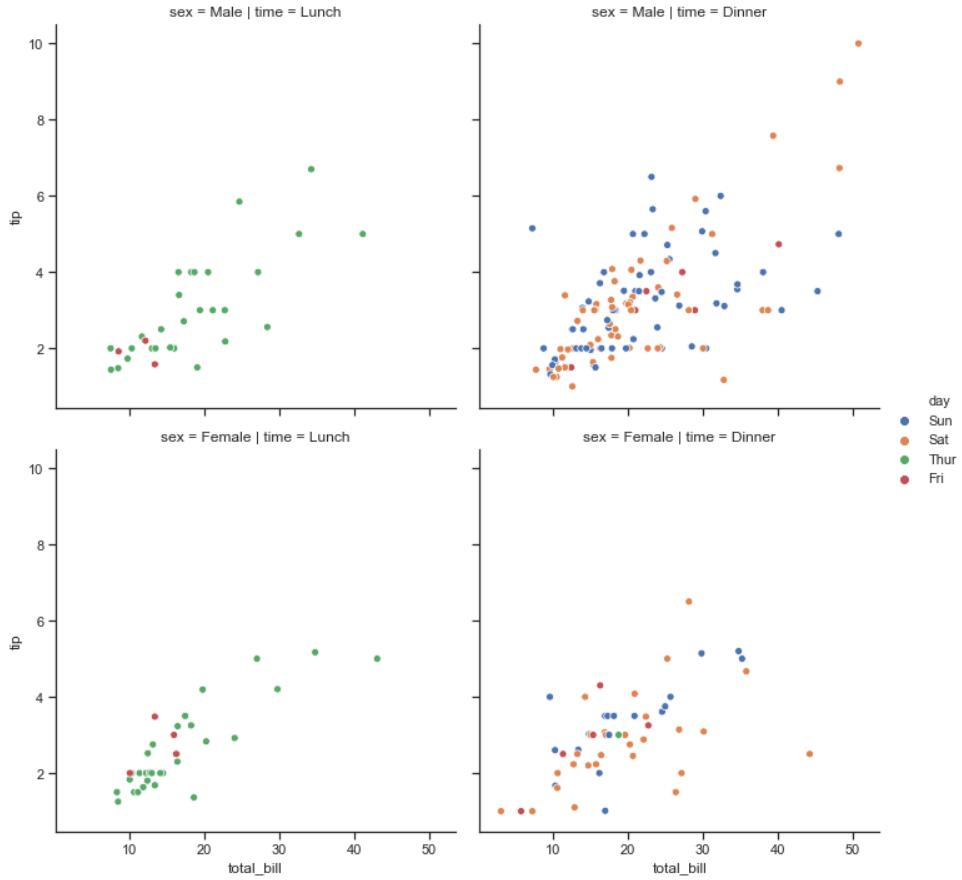
```
# importing the library
import seaborn as sns

# selecting style
sns.set(style = "ticks")

# reading the dataset
tips = sns.load_dataset('tips')

sns.relplot(x="total_bill",
             y="tip",
             hue="day",
             col="time",
             row="sex",
             data=tips)
```

Output

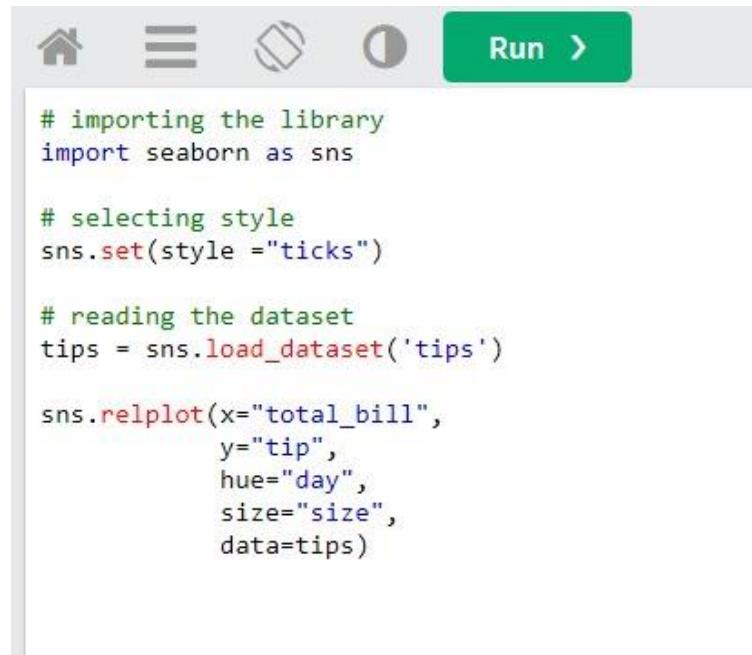


Justification

This example uses both sex and time to determine the facet grid.

Example 4

Input



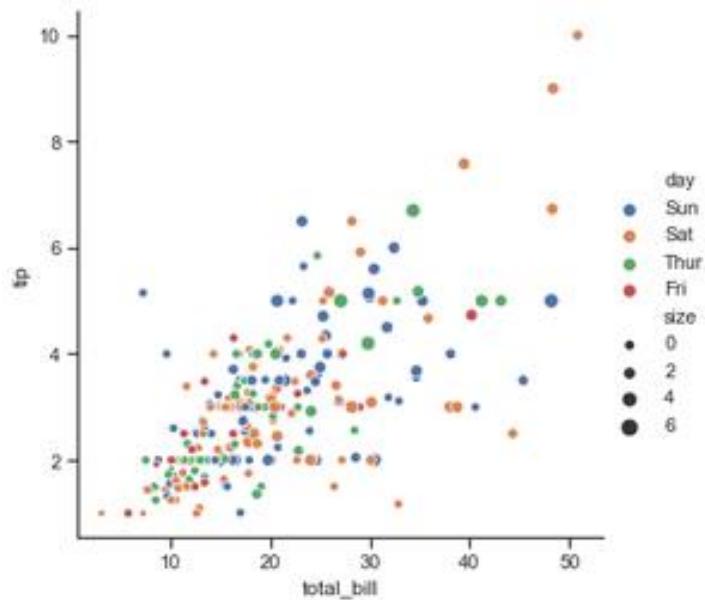
```
# importing the library
import seaborn as sns

# selecting style
sns.set(style = "ticks")

# reading the dataset
tips = sns.load_dataset('tips')

sns.relplot(x="total_bill",
             y="tip",
             hue="day",
             size="size",
             data=tips)
```

Output



Justification

This example takes the size attributes (data points that has different size)

Scatter Plot ()

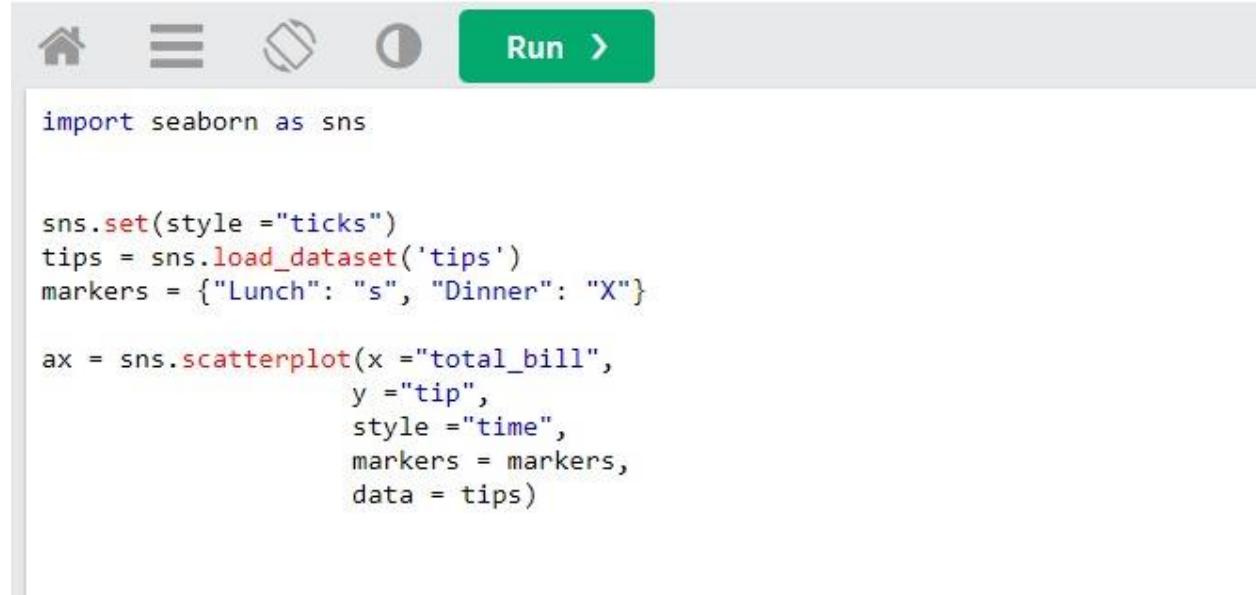
The scatter plot is the updated virtualization technique that depicts the joint distribution between the two of the variable with the help of cloud of points in which each of the point mainly represents the observation in the datasets

Syntax

```
seaborn.scatterplot(x=None, y=None, data=None, **kwargs)
```

Example 1

Input



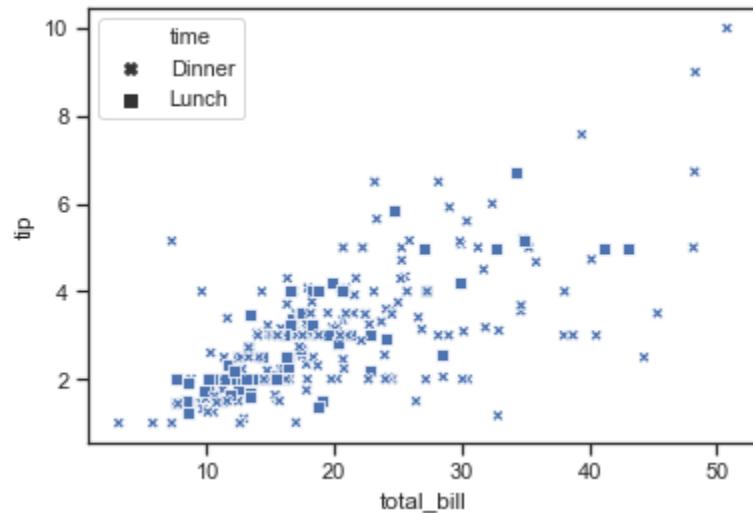
The screenshot shows a Jupyter Notebook interface with a toolbar at the top featuring icons for file, cell, and run. Below the toolbar is a code cell containing Python code for generating a scatter plot. The code imports seaborn, sets the style to "ticks", loads the "tips" dataset, defines markers for "Lunch" and "Dinner", and then creates a scatterplot with "total_bill" on the x-axis and "tip" on the y-axis, using the specified style and markers.

```
import seaborn as sns

sns.set(style = "ticks")
tips = sns.load_dataset('tips')
markers = {"Lunch": "s", "Dinner": "X"}

ax = sns.scatterplot(x ="total_bill",
                     y ="tip",
                     style ="time",
                     markers = markers,
                     data = tips)
```

Output



Justification

This above example plotting the scatter plot with the help of the market to differentiate the timing in which people visited the restaurant

Example 2

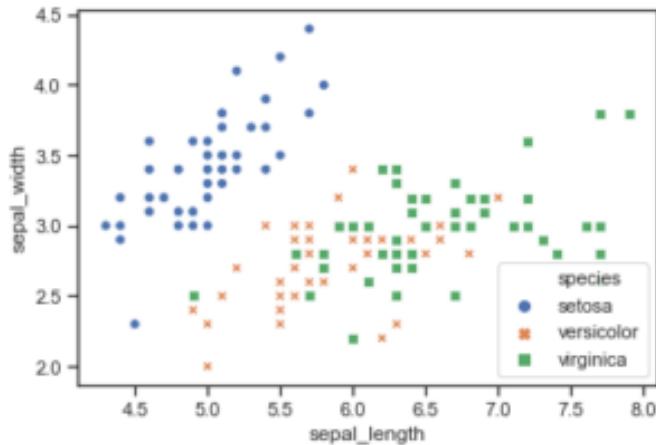
Input

```
import seaborn as sns

iris = sns.load_dataset("iris")

sns.scatterplot(x = iris.sepal_length,
                 y = iris.sepal_width,
                 hue = iris.species,
                 style = iris.species)
```

Output



Justification

This example shows how to pass the respective data vector instead of the data frame's name

Line Plot ()

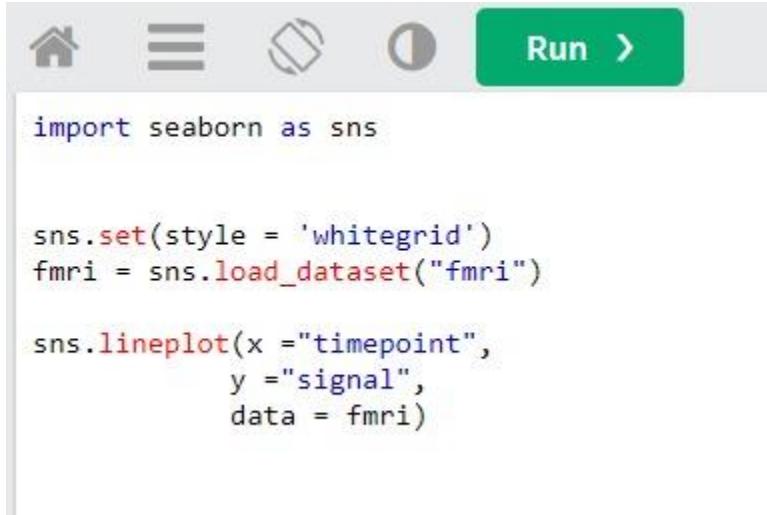
The scatter plot is highly effective, and there are no proper universal optimization techniques for the virtualization. For different datasets, the user wants to consider some of the changes as a function of time in the context of one variable with the help of the line plot function

Syntax

```
seaborn.lineplot(x=None, y=None, data=None, **kwargs)
```

Example 1

Input

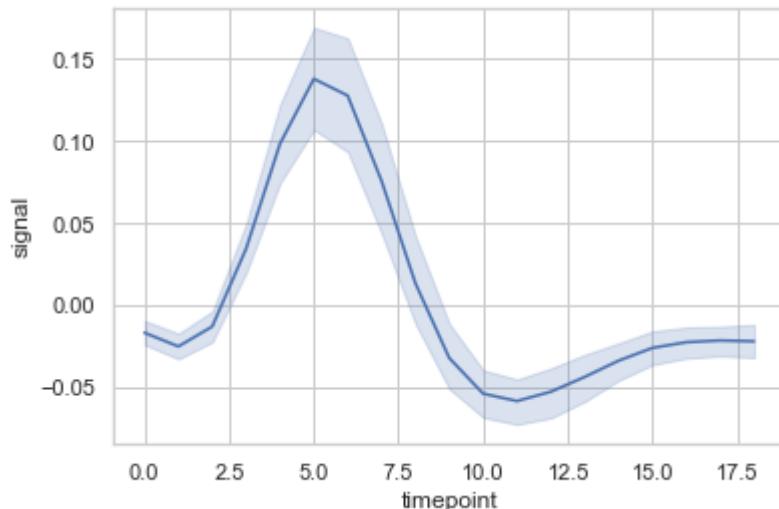


```
import seaborn as sns

sns.set(style = 'whitegrid')
fmri = sns.load_dataset("fmri")

sns.lineplot(x ="timepoint",
             y ="signal",
             data = fmri)
```

Output

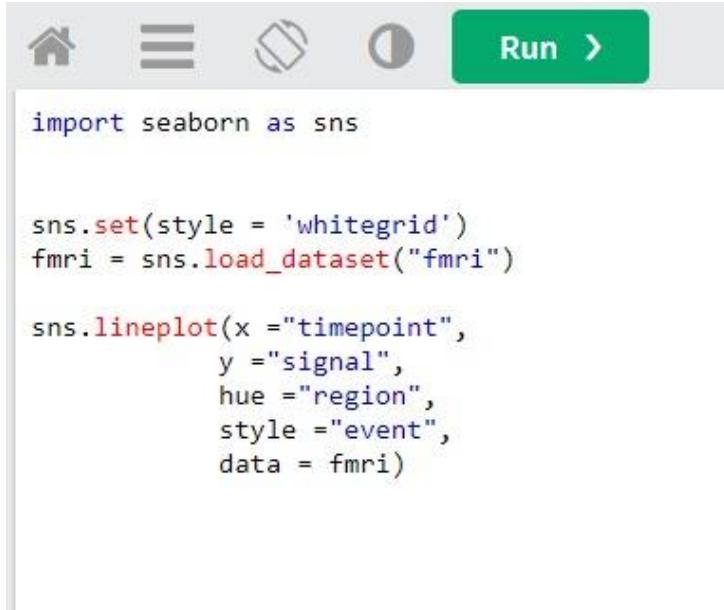


Justification

This example provides the basic visualization technique of “fmri” datasets with the help of the line plot ()

Example 2

Input

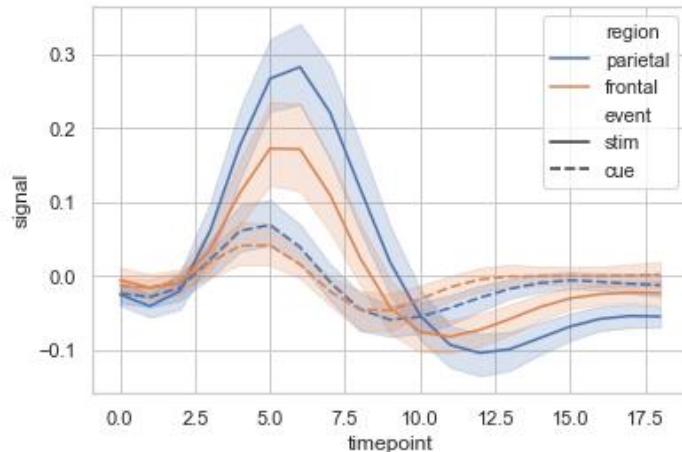


```
import seaborn as sns

sns.set(style = 'whitegrid')
fmri = sns.load_dataset("fmri")

sns.lineplot(x = "timepoint",
              y = "signal",
              hue = "region",
              style = "event",
              data = fmri)
```

Output

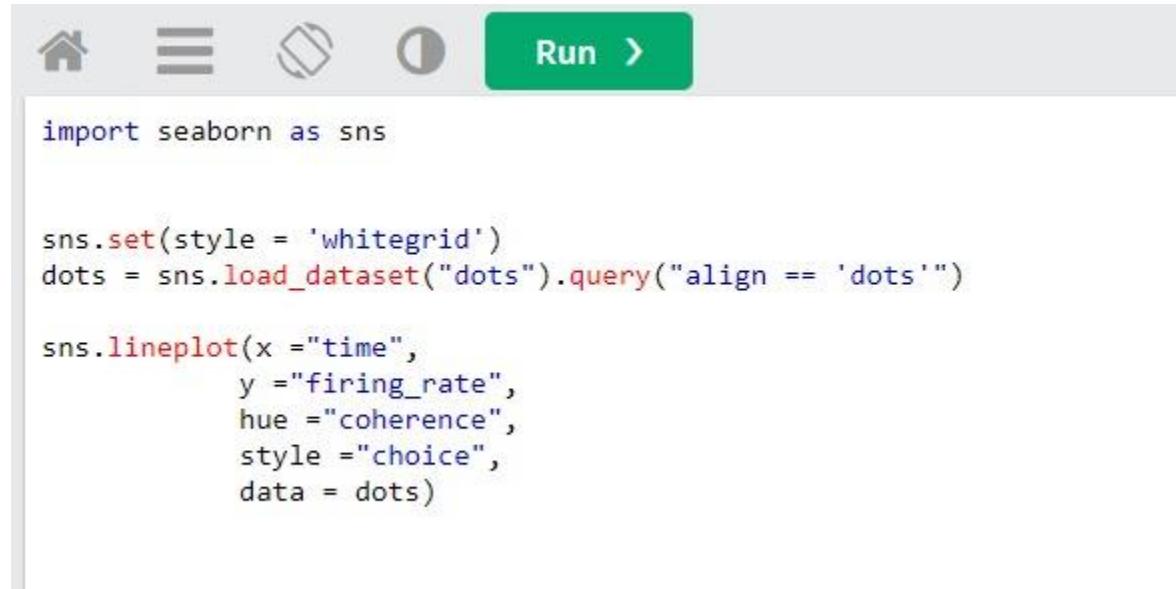


Justification

This example performs the group operation for the data points that are based on the specific category (event and time)

Example 3

Input

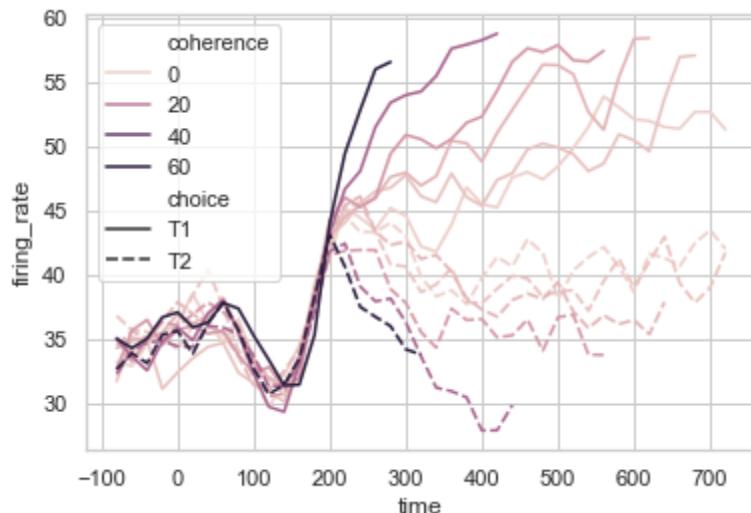


```
import seaborn as sns

sns.set(style = 'whitegrid')
dots = sns.load_dataset("dots").query("align == 'dots'")

sns.lineplot(x = "time",
              y = "firing_rate",
              hue = "coherence",
              style = "choice",
              data = dots)
```

Output



The last example plotting the “dots” datasets for showing the internal capability of seaborn with the help of quantitative color mapping based technique

Unit 18.3- Categorical Plots

This plot has associated with the different types of categorical variable and analyzed how they will visualized properly

Load the Datasets

Input

```
# import the seaborn library
import seaborn as sns

# import done to avoid warnings
from warnings import filterwarnings

# reading the dataset
df = sns.load_dataset('tips')

# first five entries of the dataset
df.head()
```

Output

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

Now in the next step, the user needs to proceed to the plot for visualizing the categorical variable.

Bar Plot

The bar plot is mainly used for aggregating different types of categorical data based on some methods. For using the bar plot, the user need to choose the categorical column in the form of X axis and numerical column in the form of y axis so that it has developed a plot by taking mean per categorical column

Syntax

```
barplot([x, y, hue, data, order, hue_order, ...])
```

Example

Input

```
# set the background style of the plot
sns.set_style('darkgrid')

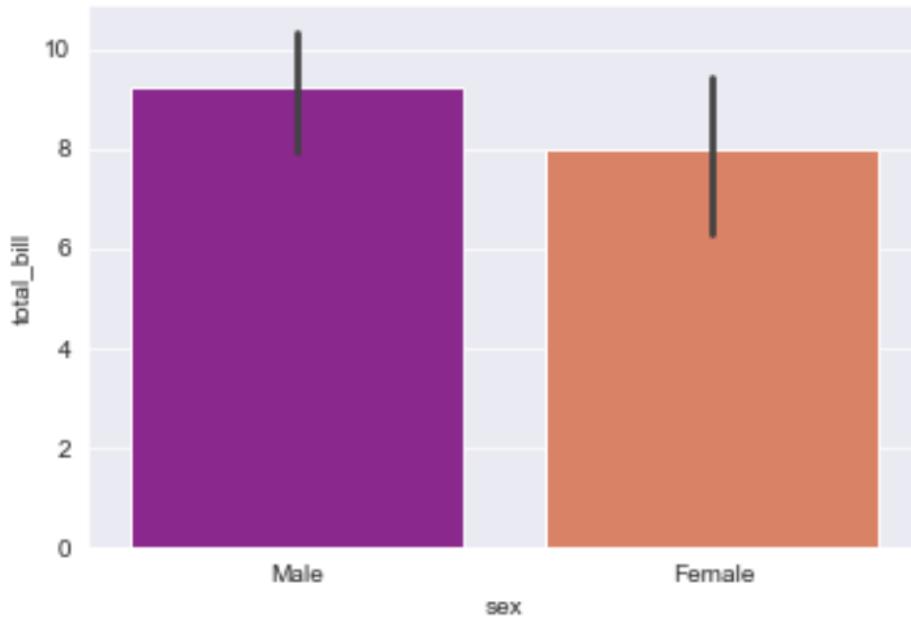
# plot the graph using the default estimator mean
sns.barplot(x ='sex', y ='total_bill', data = df, palette ='plasma')

# or

import numpy as np

# change the estimator from mean to standard deviation
sns.barplot(x ='sex', y ='total_bill', data = df,
            palette ='plasma', estimator = np.std)
```

Output



Explanation

It has been stated from the above analysis is that average total_bill for male is more than the female

Count Plot

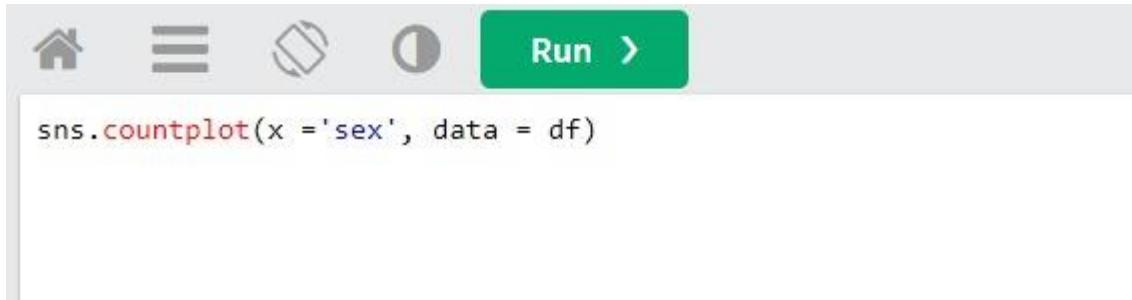
The count plot is capable for counting the respective category and then returns the count of occurrence

Syntax

```
countplot([x, y, hue, data, order, ...])
```

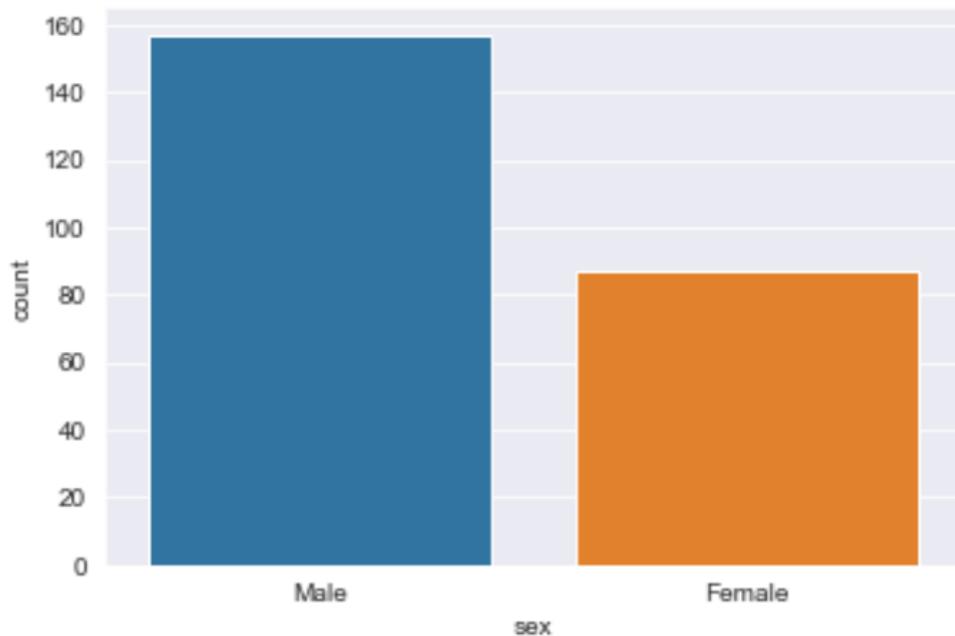
Example

Input



A screenshot of a Jupyter Notebook interface. At the top, there are icons for file operations (Home, New, Open, Save, Run Cell, etc.). Below the toolbar, a green "Run >" button is visible. The main area contains the Python code: `sns.countplot(x = 'sex', data = df)`.

Output



Box Plot

The box plot shows the distribution of different types of quantitative data which represents the comparison for each of the variable

Syntax

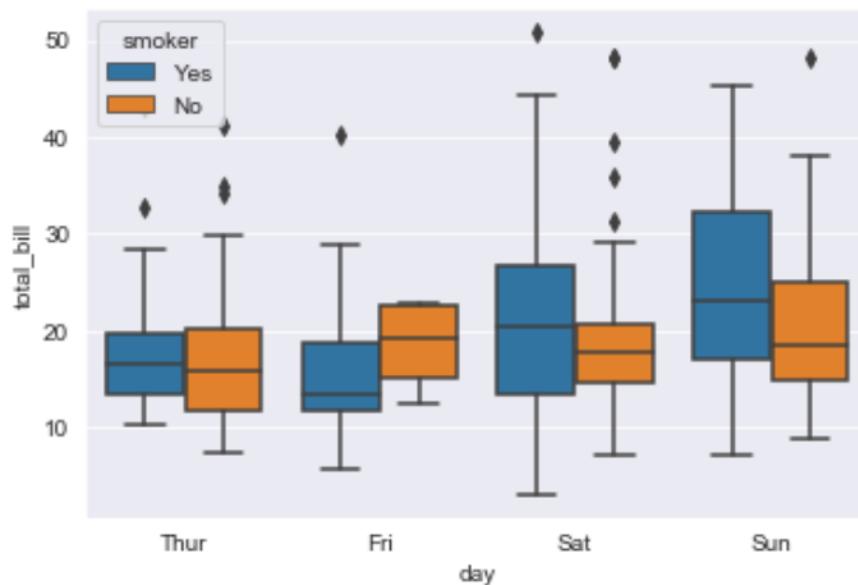
```
boxplot([x, y, hue, data, order, hue_order, ...])
```

Example

Input

A screenshot of a Jupyter Notebook cell. At the top, there are icons for file, cell, and run, followed by a green "Run >" button. Below the button is the Python code: `sns.boxplot(x = 'day', y = 'total_bill', data = df, hue = 'smoker')`.

Output



Explanation

X represents the categorical column, and y represents the numerical column. So it has been seen from this figure that total bill spending by the customer in each day in which “hue “parameter is mainly used for adding the categorical separation

Violin Plot

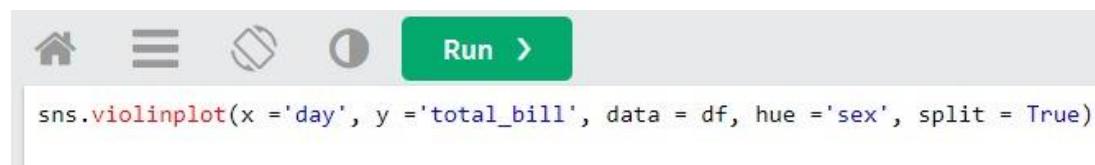
The violin plot provides more advanced form of virtualization technique with the help of kernel density-based estimation for the distribution of data

Syntax

```
violinplot([x, y, hue, data, order, ...])
```

Example

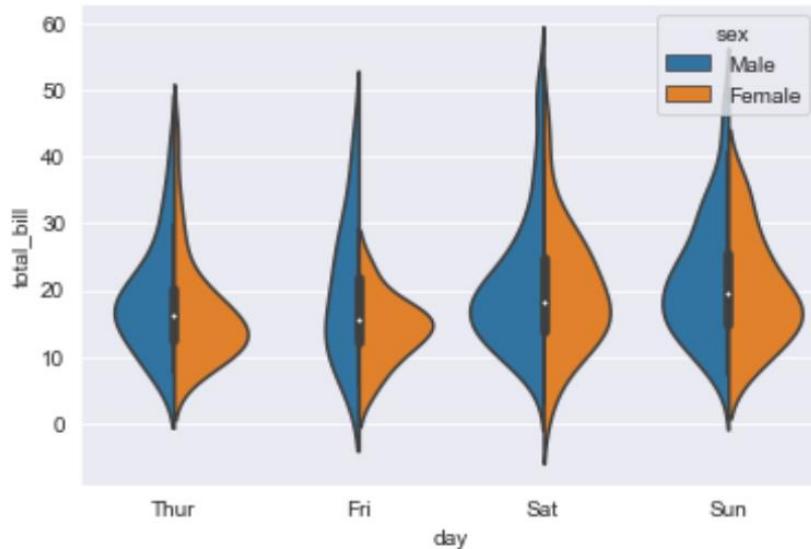
Input



A screenshot of a Jupyter Notebook interface. At the top, there are several icons: a home icon, a three-line menu icon, a refresh icon, and a run button labeled "Run >". Below the toolbar, the code cell contains the following Python command:

```
sns.violinplot(x ='day', y ='total_bill', data = df, hue ='sex', split = True)
```

Output



Explanation

It has been stated that “hue” is mainly used for separating the data with the help of the sex category

Strip Plot

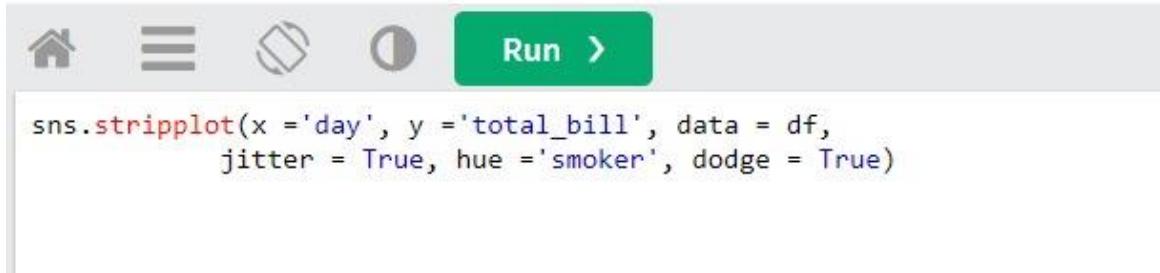
This plot is mainly used for developing the scatter plot that are based on a specific category

Syntax

```
stripplot([x, y, hue, data, order, ...])
```

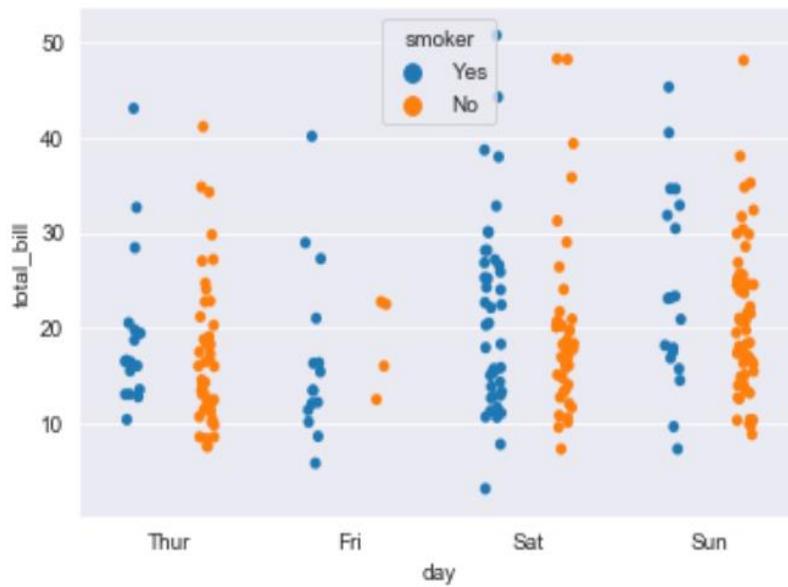
Example

Input



```
sns.stripplot(x = 'day', y = 'total_bill', data = df,
                jitter = True, hue = 'smoker', dodge = True)
```

Output



Explanation

If the condition `split = True` then it has been used for drawing the separate kinds of strip plot based on the “hue” parameterized category

Swarm Plot

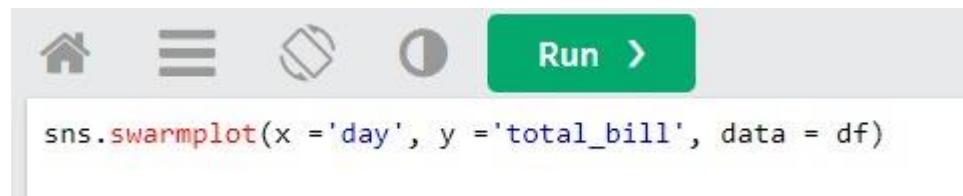
This plot is same like strip plot but the main difference is that the point is adjusted so that it cannot be overlap

Syntax

```
swarmplot([x, y, hue, data, order, ...])
```

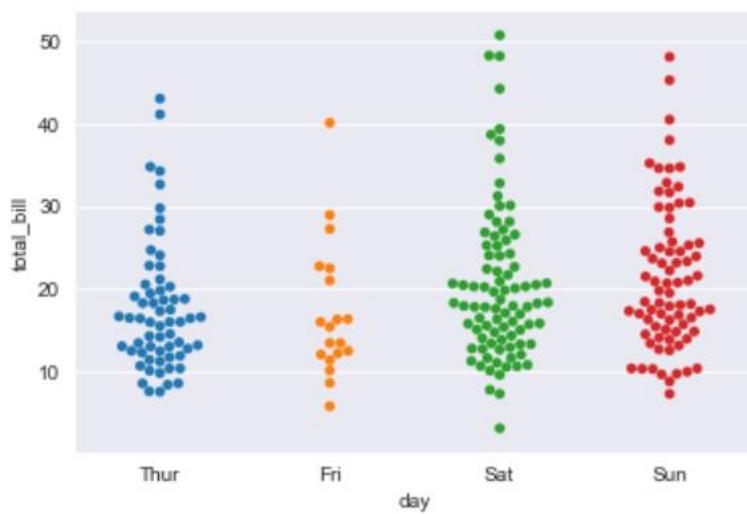
Example 1

Input



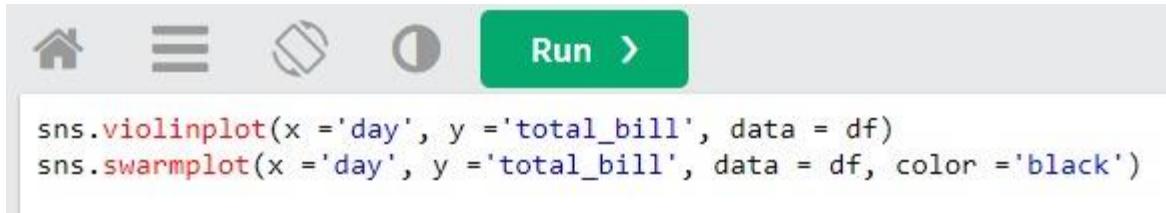
```
sns.swarmplot(x ='day', y ='total_bill', data = df)
```

Output



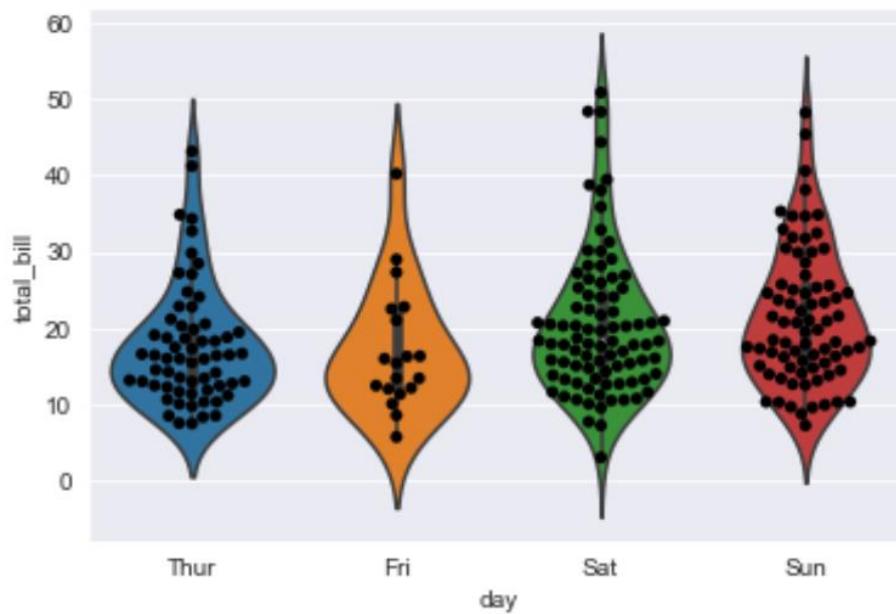
Example 2

Input



```
sns.violinplot(x = 'day', y = 'total_bill', data = df)
sns.swarmplot(x = 'day', y = 'total_bill', data = df, color = 'black')
```

Output



Factor Plot

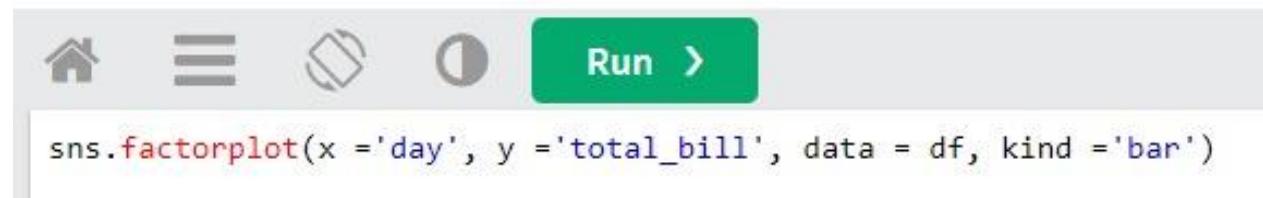
This is the most general plot among the others plot by providing the specific parameter so that the user can choose the type of plot separately

Syntax

```
sns.factorplot([x, y, hue, data, row, col, ...])
```

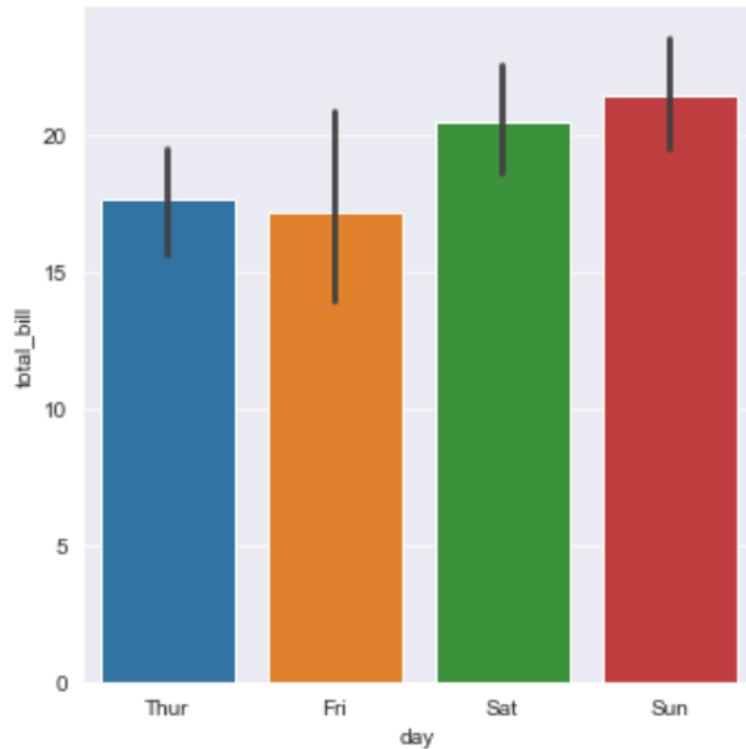
Example

Input



```
sns.factorplot(x ='day', y ='total_bill', data = df, kind ='bar')
```

Output



Unit 18.4- Distribution Plots

The distribution plot is mainly used to examine the univariate as well as the bivariate distribution. In this book, we are going to the discussed total of four types of distribution plot

Displot

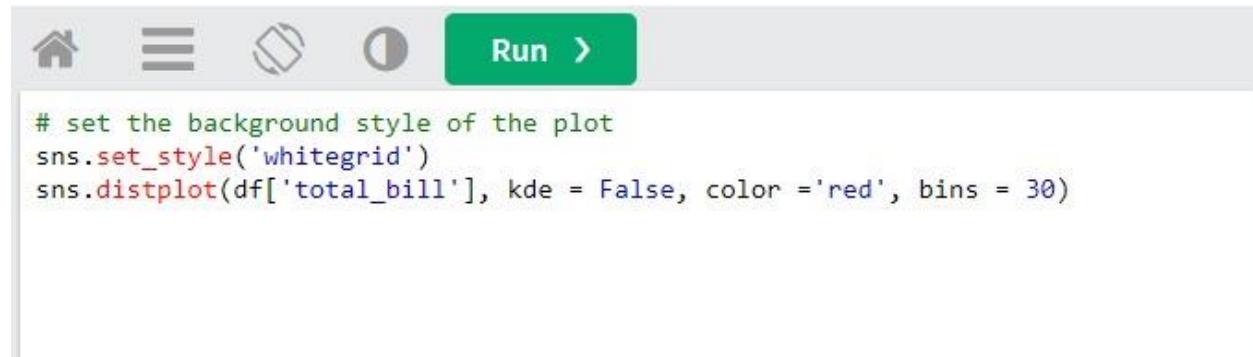
This plot is mainly used for the univariate sets of observation that are visualize with the help of histogram

Syntax

```
distplot(a[, bins, hist, kde, rug, fit, ...])
```

Example

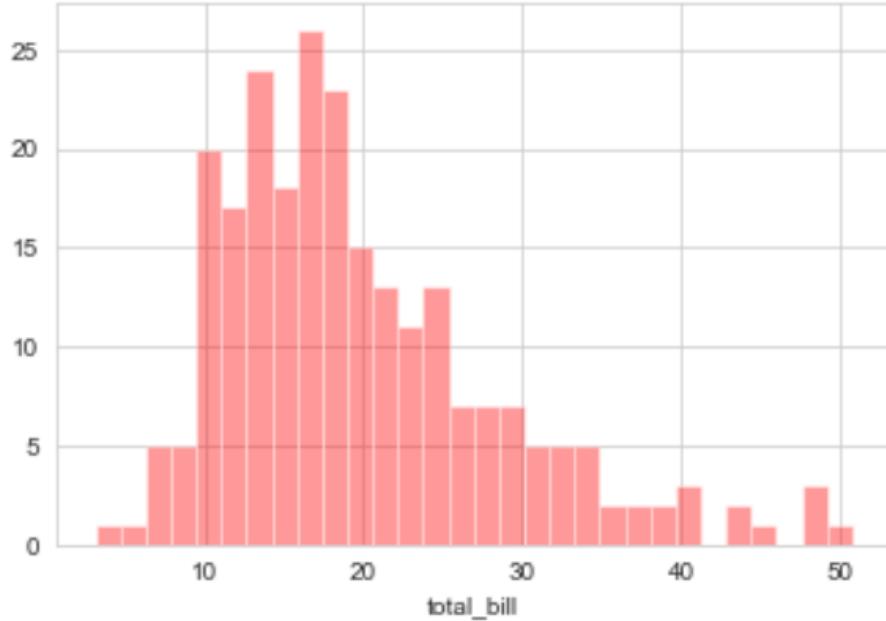
Input



A screenshot of a Jupyter Notebook interface. At the top, there are icons for file, cell, and kernel, followed by a green 'Run' button with a right-pointing arrow. Below the toolbar, the code for generating a distribution plot is displayed:

```
# set the background style of the plot
sns.set_style('whitegrid')
sns.distplot(df['total_bill'], kde = False, color ='red', bins = 30)
```

Output



Justification

In this figure, bin is mainly used for setting the total numbers of bin that the user is wanted to plot. It has been stated from the above analysis is that, the total bills lies in between the 10 and 20

Join Plot

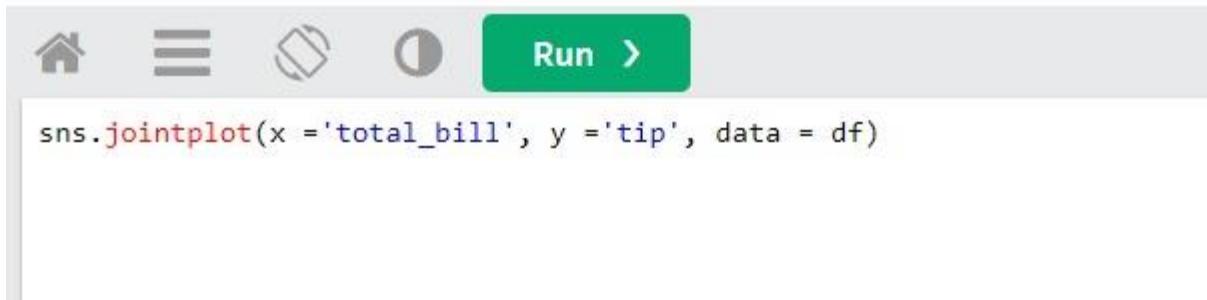
This plot is mainly used for drawing the plot of two of the variable with the help of both bivariate graph as well as univariate graph

Syntax

```
jointplot(x, y[, data, kind, stat_func, ...])
```

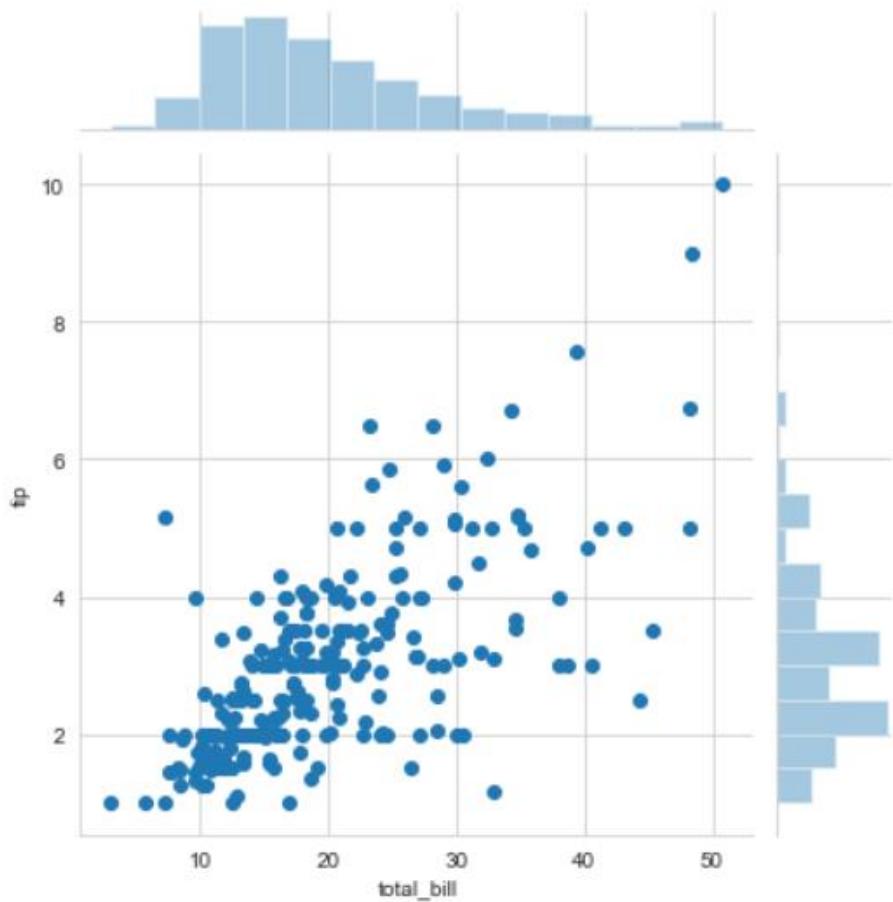
Example

Input

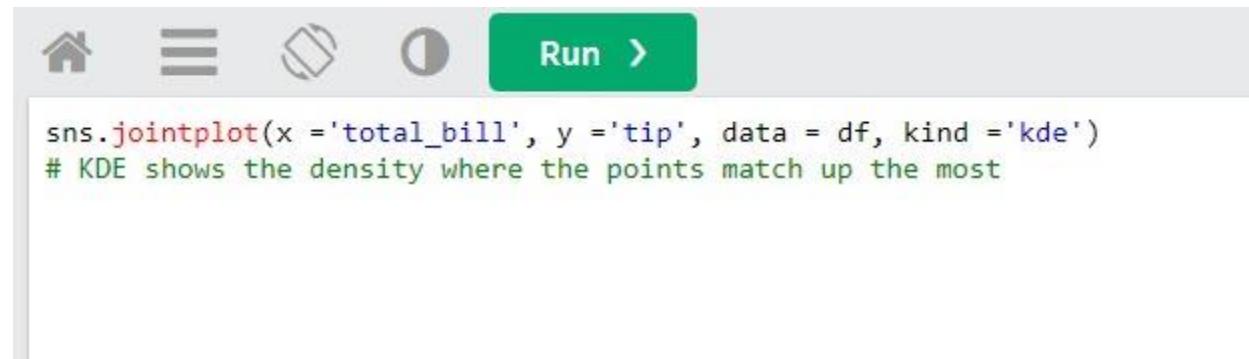


```
sns.jointplot(x = 'total_bill', y = 'tip', data = df)
```

Output

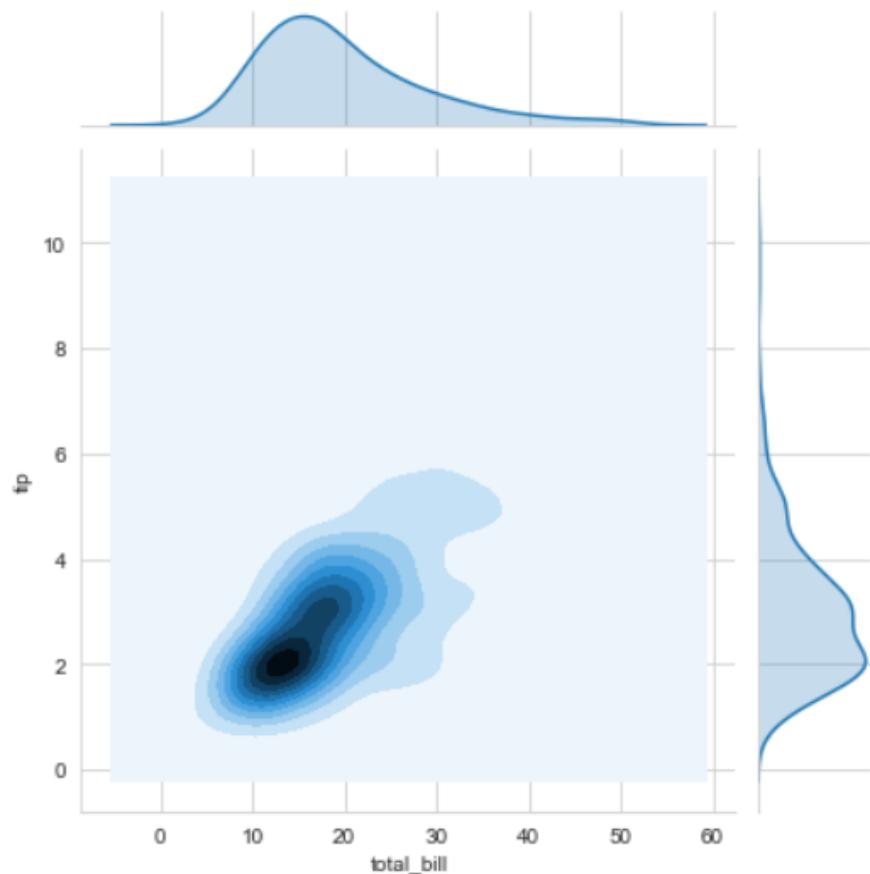


Input



```
sns.jointplot(x = 'total_bill', y = 'tip', data = df, kind = 'kde')
# KDE shows the density where the points match up the most
```

Output



Justification

Based on the tip of the Y axis and the total bill on the X axis, it can be stated that the total bill will increase with the tips.

Pair Plot

This plot demonstrates the pair-wise based relation with the help of the entire data frame and at the same time, the pair plot supports the specific arguments known as hue to the separation of categorical data.

Syntax

```
pairplot(data[, hue, hue_order, palette, ...])
```

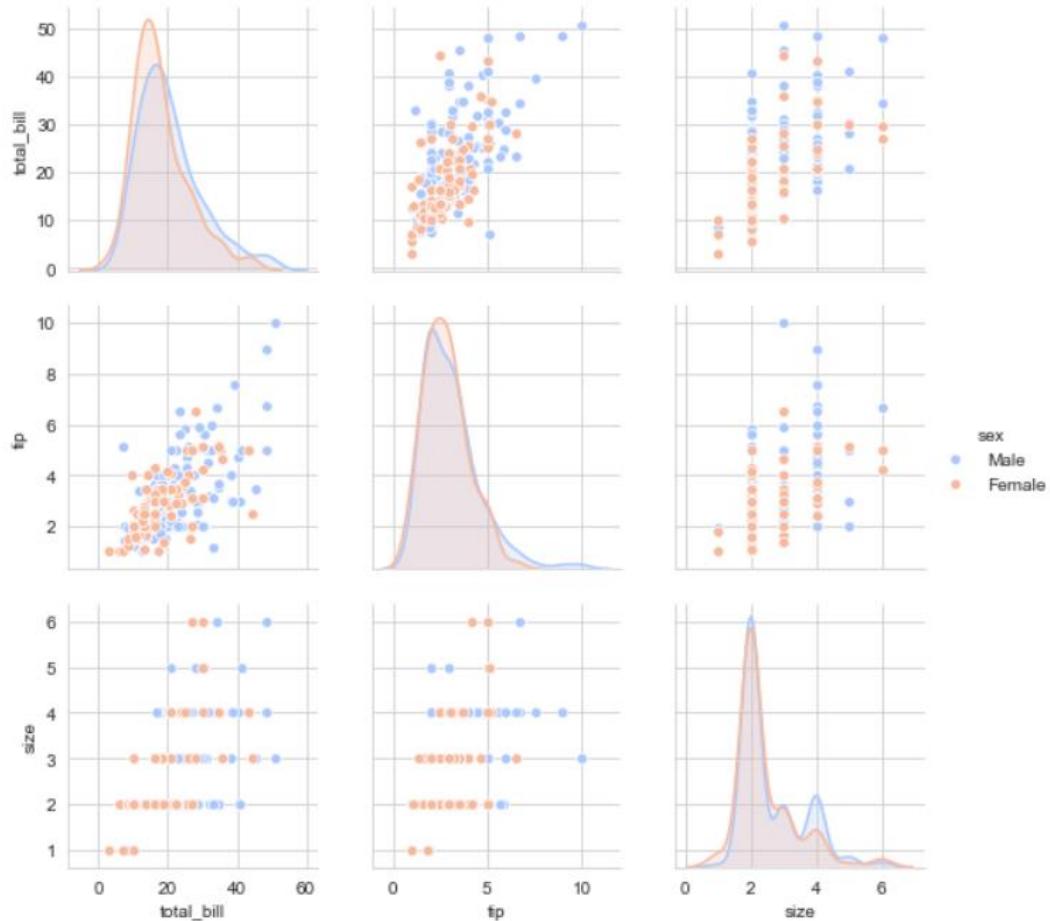
Example

Input



```
sns.pairplot(df, hue ="sex", palette ='coolwarm')
```

Output



Justification

Based on the diagram, it has been stated that the hue represents the categorical separation between every entries.

Rug Plot

Rug plot data points in the form of array as it sticks on the axis by taking only one single column.

Syntax

```
rugplot(a[, height, axis, ax])
```

Example

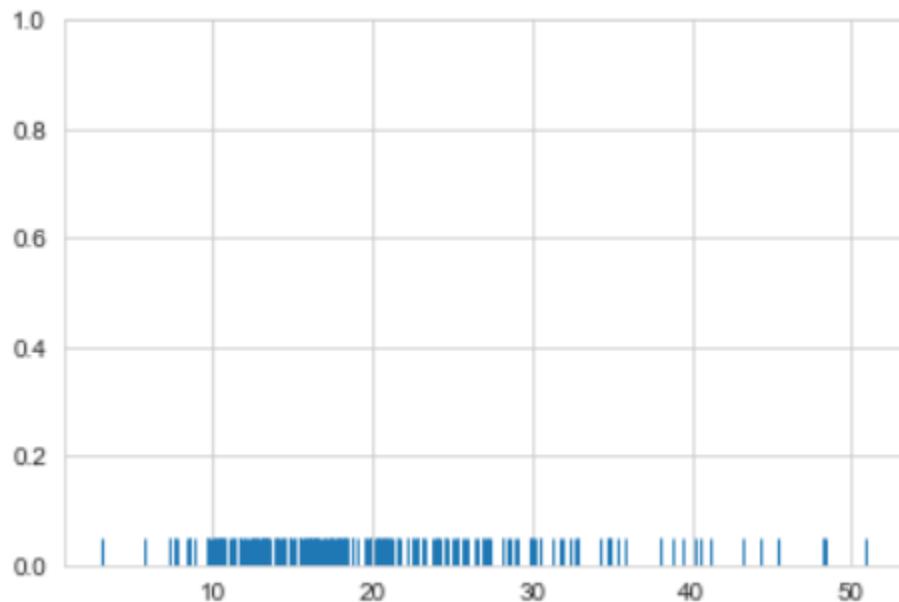
Input



The image shows a screenshot of a Jupyter Notebook interface. At the top, there is a toolbar with icons for file operations (Home, New, Open, Save, Run, etc.). Below the toolbar, a green button labeled "Run >" is visible. The main area contains a single line of Python code:

```
sns.rugplot(df['total_bill'])
```

Output



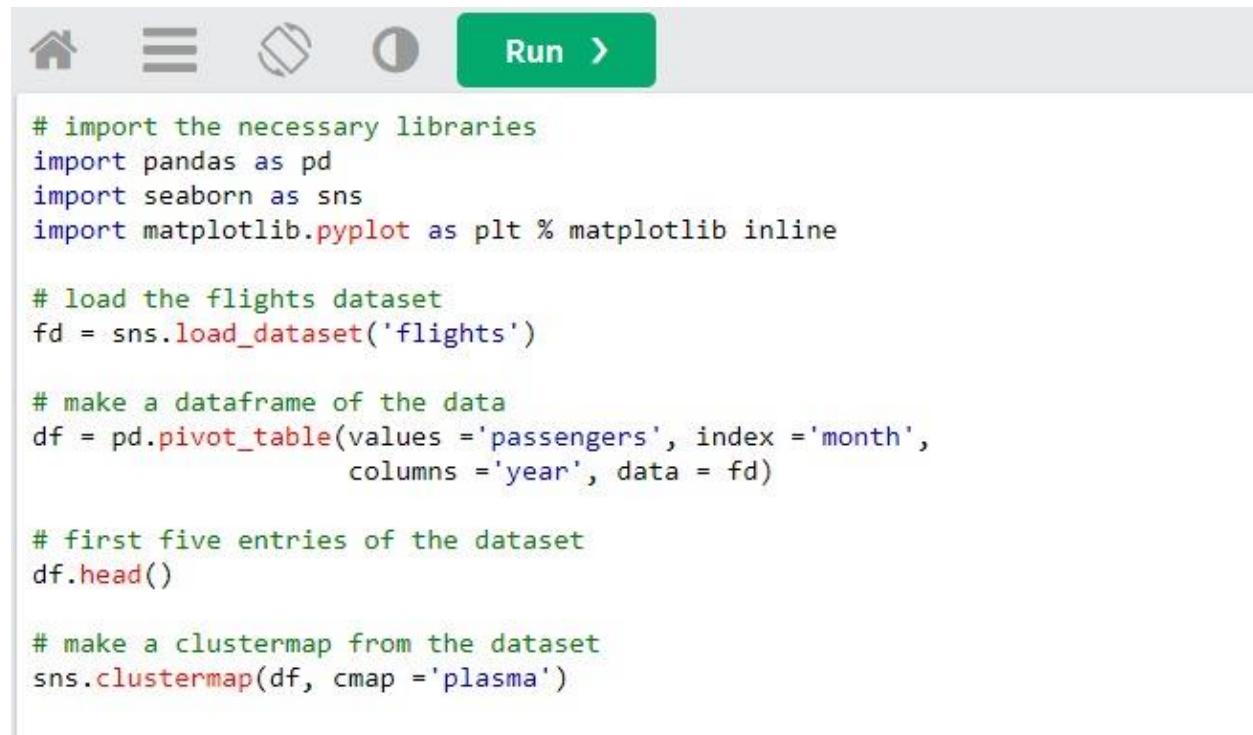
Unit 18.5- Matrix plots

The matrix plot is the array of different types of scatter plot

Example – Cluster Maps

Cluster map uses the hierarchical based clustering by performing the clustering operation that are totally based on the similarity of the columns as well as rows

Input



```
# import the necessary libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt %matplotlib inline

# load the flights dataset
fd = sns.load_dataset('flights')

# make a dataframe of the data
df = pd.pivot_table(values ='passengers', index ='month',
                     columns ='year', data = fd)

# first five entries of the dataset
df.head()

# make a clustermap from the dataset
sns.clustermap(df, cmap ='plasma')
```

Output

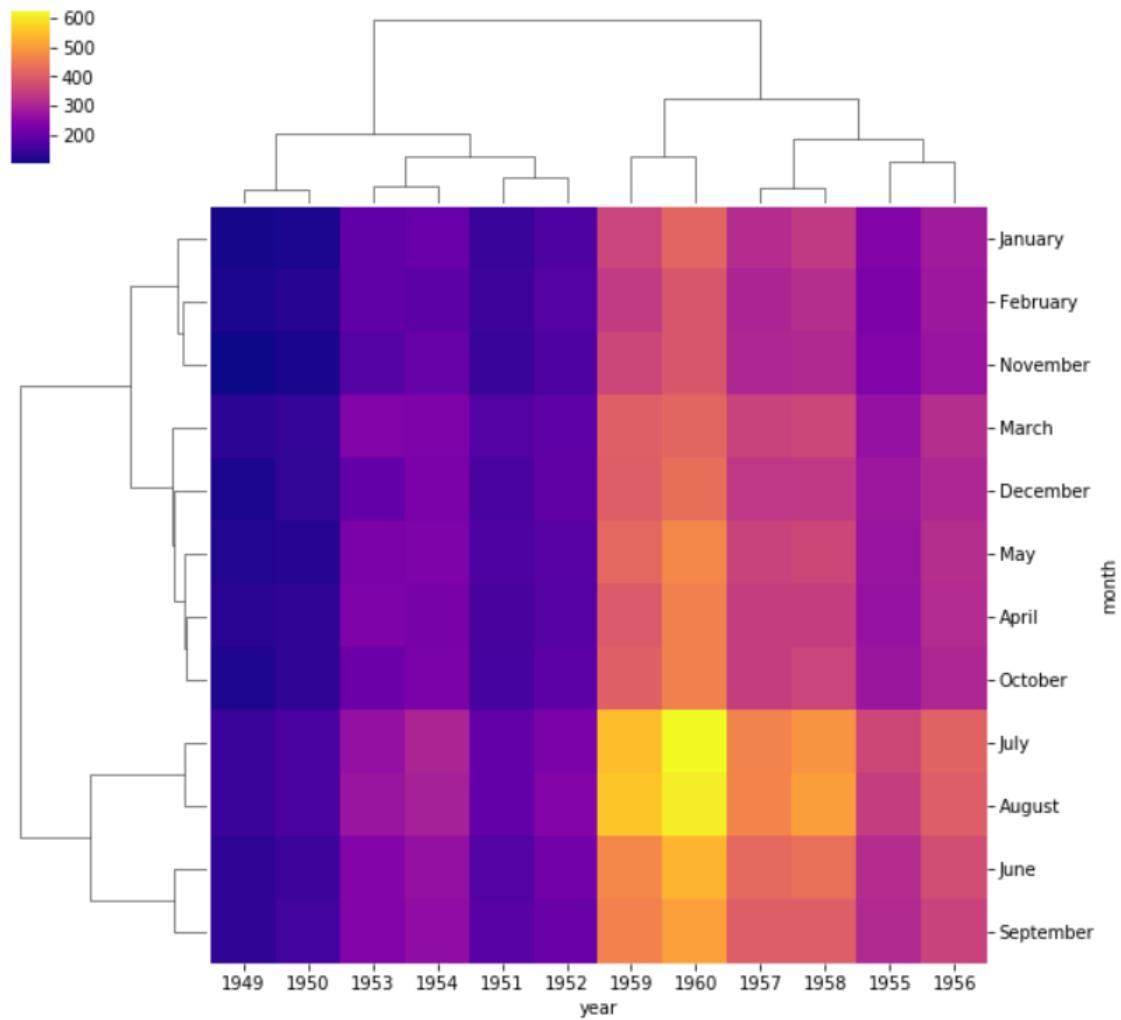
	year	month	passengers
0	1949	January	112
1	1949	February	118
2	1949	March	132
3	1949	April	129
4	1949	May	121

It shows the first five entries of the datasets

	year	1949	1950	1951	1952	1953	1954	1955	1956	1957	1958	1959	1960
	month												
	January	112	115	145	171	196	204	242	284	315	340	360	417
	February	118	126	150	180	196	188	233	277	301	318	342	391
	March	132	141	178	193	236	235	267	317	356	362	406	419
	April	129	135	163	181	235	227	269	313	348	348	396	461
	May	121	125	172	183	229	234	270	318	355	363	420	472

The matrix is implemented with the help of pivot table (for the first five entries)

<seaborn.matrix.ClusterGrid at 0x218f7060860>



Clustering from the Given Data

Input



```
# import the necessary libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt %matplotlib inline

# load the flights dataset
fd = sns.load_dataset('flights')

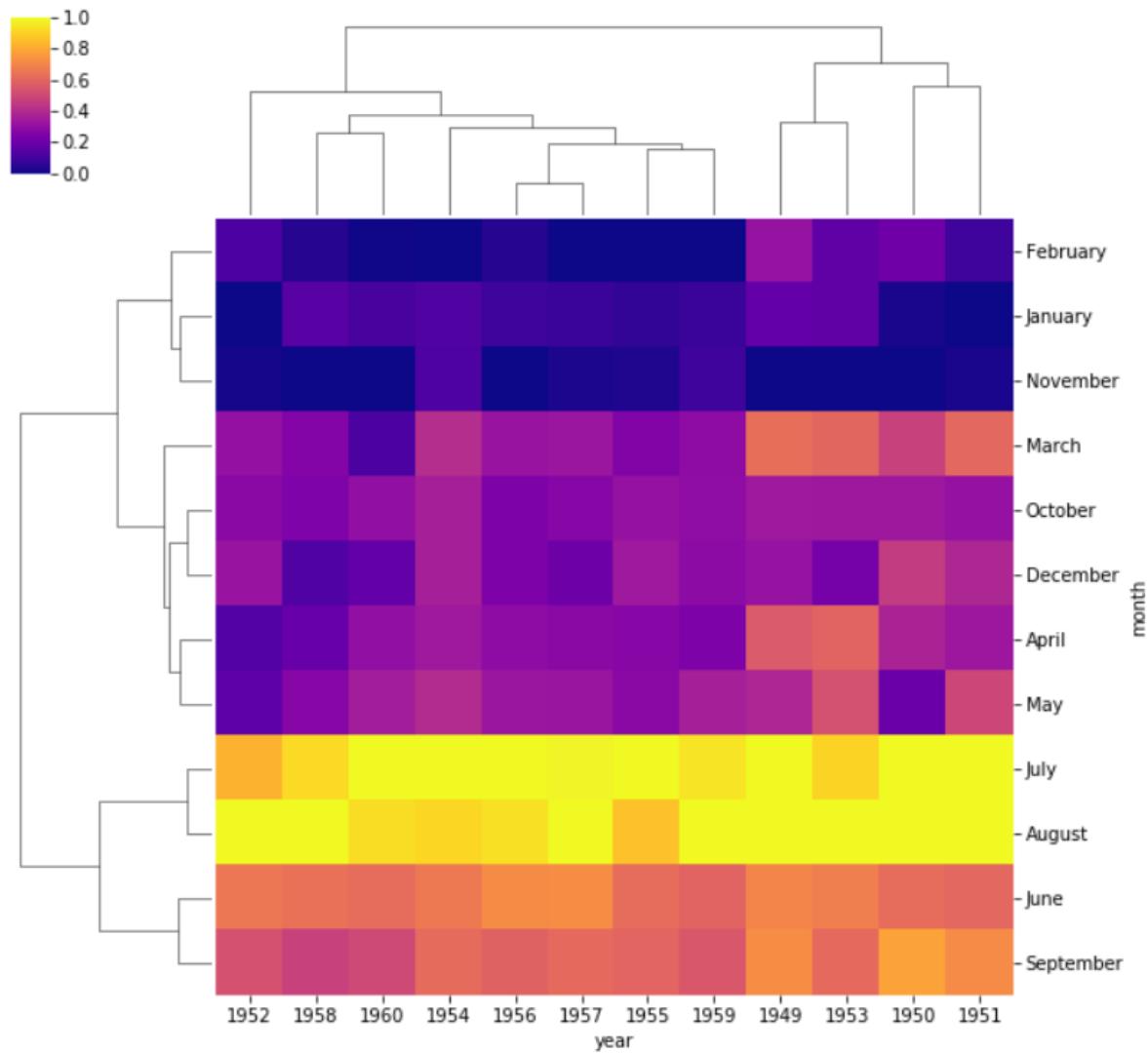
# make a dataframe of the data
df = pd.pivot_table(values = 'passengers',
                     index ='month', columns ='year', data = fd)

# first five entries of the dataset
df.head()

# make a clustermap from the dataset
sns.clustermap(df, cmap ='plasma', standard_scale = 1)
```

Output

```
<seaborn.matrix.ClusterGrid at 0x218f68cc390>
```



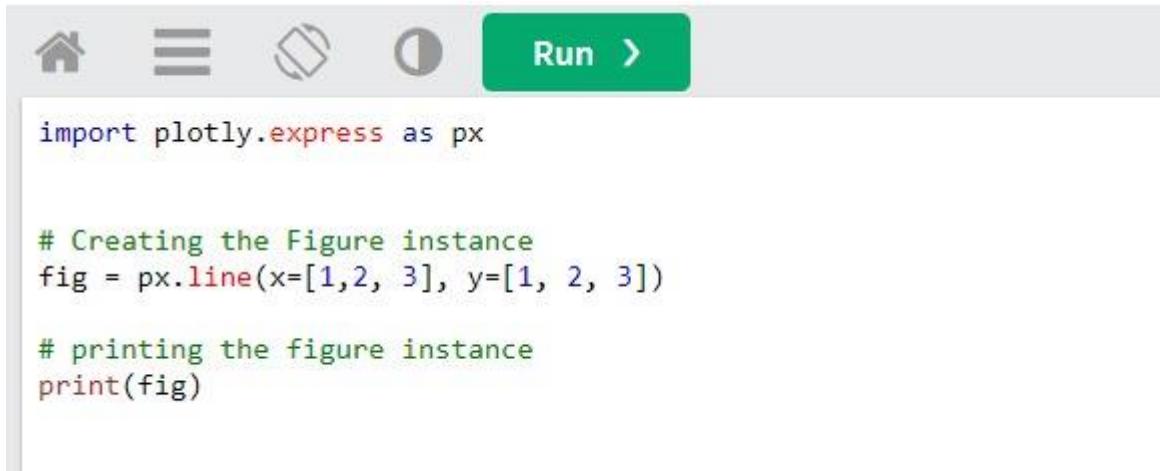
Justification

Cluster map after using the standard scaling `standard_scale = 1` that normalizes the entire data from the ranges of 0 to the 1. Based on the above output, it has stated that month and years is no longer is in the proper order when the similarity of the cluster maps clusters them.

Unit 18.6- Plotly: Line

Import Data

Input



```
import plotly.express as px

# Creating the Figure instance
fig = px.line(x=[1,2, 3], y=[1, 2, 3])

# printing the figure instance
print(fig)
```

Output

```
Figure({
    'data': [{'hovertemplate': 'x=%{x}<br>y=%{y}</extra></extra>',
              'legendgroup': '',
              'line': {'color': '#636efa', 'dash': 'solid'},
              'mode': 'lines',
              'name': '',
              'orientation': 'v',
              'showlegend': False,
              'type': 'scatter',
              'x': array([1, 2, 3]),
              'xaxis': 'x',
              'y': array([1, 2, 3]),
              'yaxis': 'y'}],
    'layout': {'legend': {'tracegroupgap': 0},
               'margin': {'t': 60},
               'template': '...',
               'xaxis': {'anchor': 'y', 'domain': [0.0, 1.0], 'title': {'text': 'x'}},
               'yaxis': {'anchor': 'x', 'domain': [0.0, 1.0], 'title': {'text': 'y'}}}
})
```

Line

Definition

The line chart in the Plotly is widely accessible as well as illustrious annexation to the Plotly that has managed the variation of different types of data. With the help of the px.line, every position of the data has been represented in the form of the vertex of the polyline mark in the respective 2D space

Example

Input



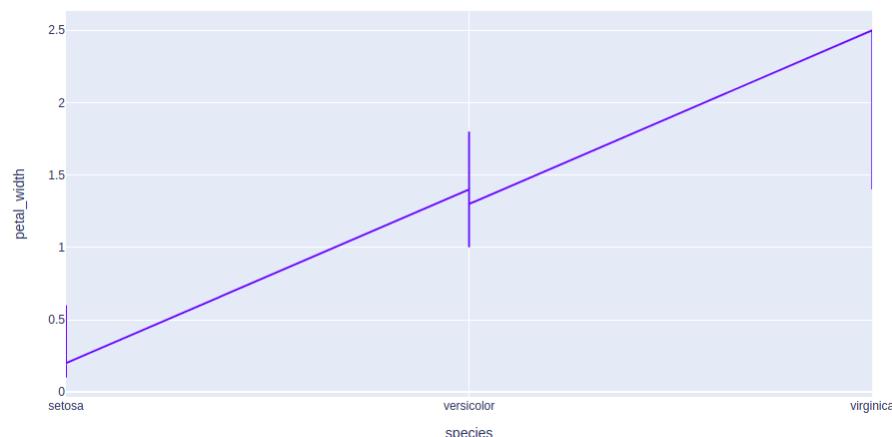
```
import plotly.express as px

# using the iris dataset
df = px.data.iris()

# plotting the line chart
fig = px.line(df, x="species", y="petal_width")

# showing the plot
fig.show()
```

Output



Unit 18.7- Bar

Definition

The bar chart is the specific type for the pictorial representation of different data which presents the categorical data with rectangular bars with the height or the length that are proportional to the data values. Thus the datasets consist of the different numerical values for the variable which represent either the height or the length

Example

Input



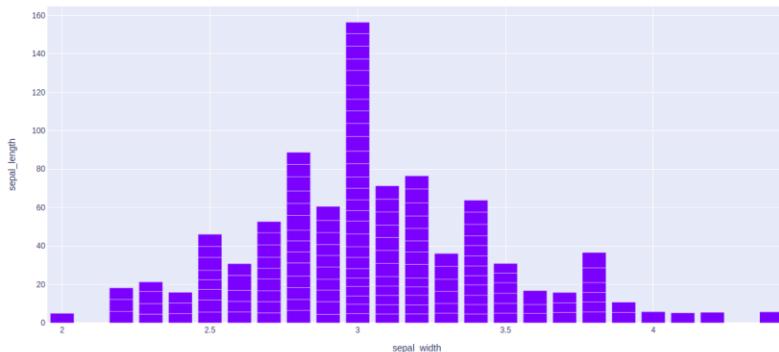
```
import plotly.express as px

# using the iris dataset
df = px.data.iris()

# plotting the bar chart
fig = px.bar(df, x="sepal_width", y="sepal_length")

# showing the plot
fig.show()
```

Output



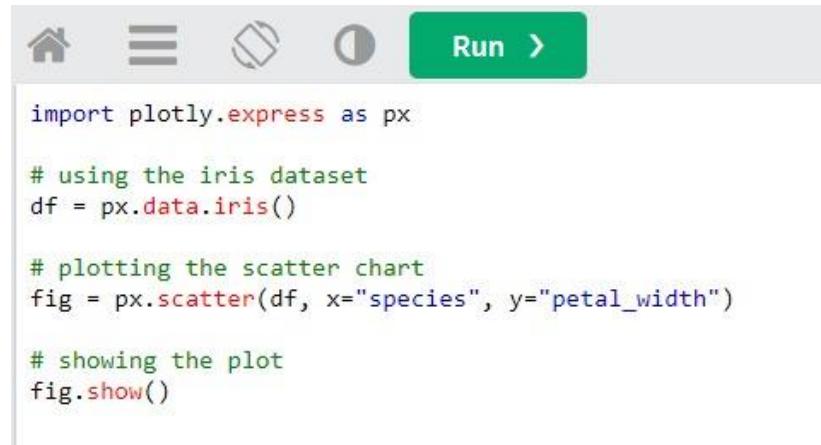
Unit 18.8- Scatter

Definition

The scatter plot is the particular set of different types of dotted points for representing the spome portion of the data is in the form of both vertical axis as well as the horizontal axis. The graph in which the specific values for the two variables are plotted alongside with the both X-axis as well as Y-axis, so the pattern for the resulting points associated with the correlation pattern between the each axis

Example

Input



```
import plotly.express as px

# using the iris dataset
df = px.data.iris()

# plotting the scatter chart
fig = px.scatter(df, x="species", y="petal_width")

# showing the plot
fig.show()
```

Output



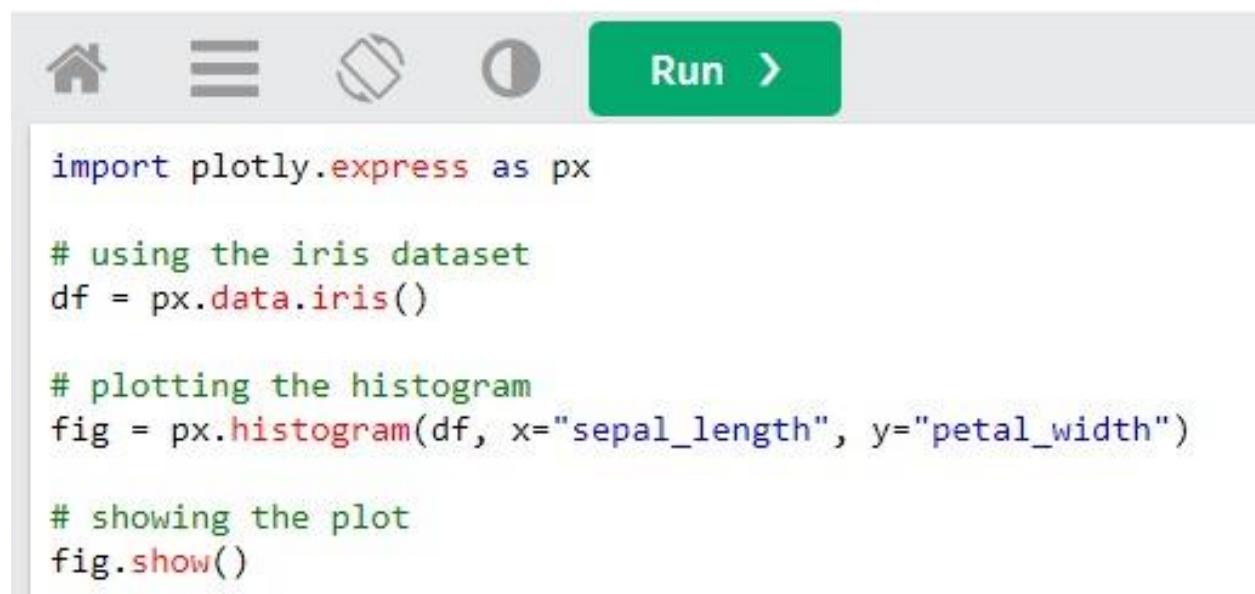
Unit 18.9- Histogram

Definition

The histogram consists of a rectangular area for displaying the specific statistical information that is proportional to the frequency of the variable as well as its width in successive numerical-based intervals. The graphical representation has managed the particular groups of data points into the various specified range. One of the most useful features of a histogram is no gaps between each bar.

Example

Input



The screenshot shows a Jupyter Notebook interface with a toolbar at the top featuring icons for file, cell, and run, followed by a large green "Run" button. Below the toolbar is a code cell containing the following Python code:

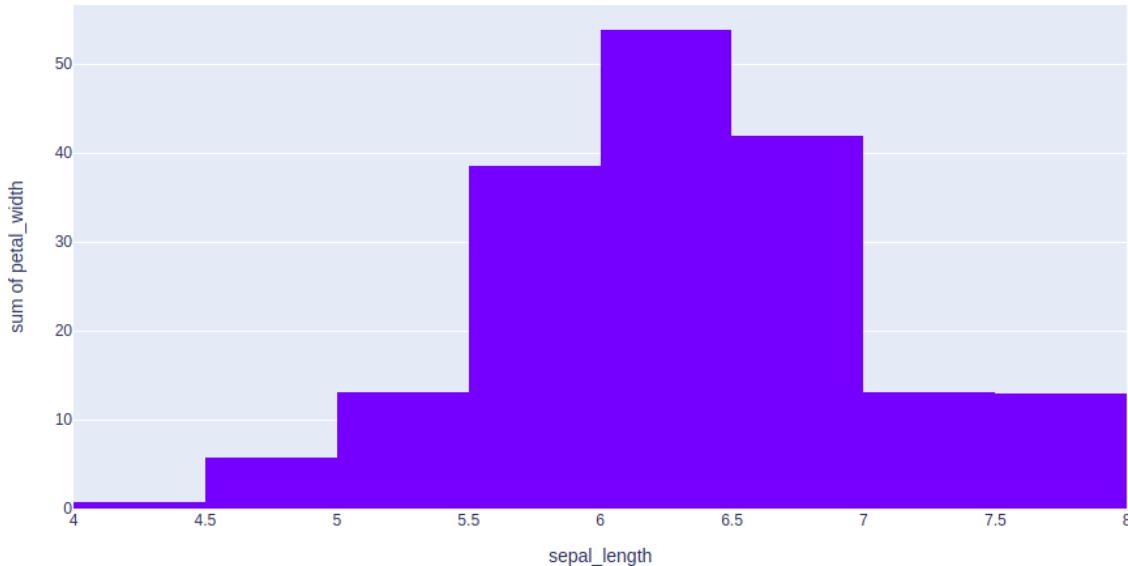
```
import plotly.express as px

# using the iris dataset
df = px.data.iris()

# plotting the histogram
fig = px.histogram(df, x="sepal_length", y="petal_width")

# showing the plot
fig.show()
```

Output



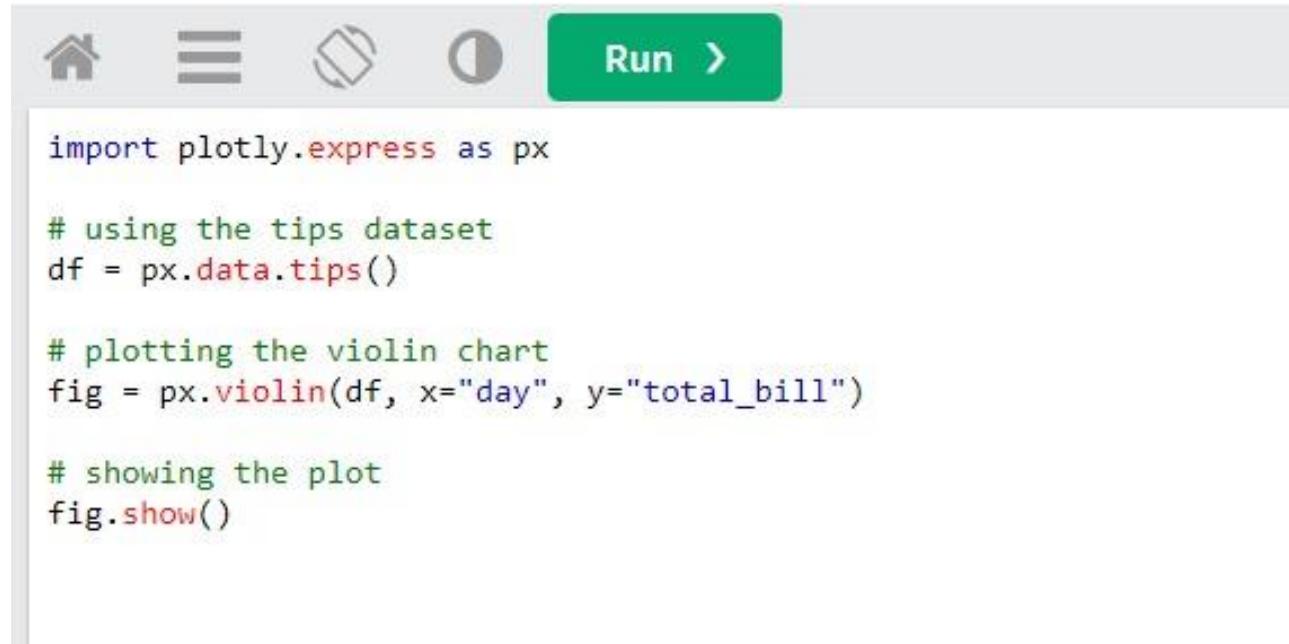
Unit 18.10- Violin

Definition

Violin plot is the specific method for visualizing the entire distribution of the numerical data of a different variable. Violin plot follows the same mechanism to the Box Plot, but one major difference is that in the violin plot there has been a rotated plot for every side, which gives more accurate information for the density estimation to the respective y-axis. The developer mainly used the violin plot because this plot shows specific nuances in the respective distribution which is not perceptible in the respective box plot

Example

Input



The screenshot shows a Jupyter Notebook interface with a toolbar at the top featuring icons for home, file, cell, and run, followed by a green "Run >" button. Below the toolbar is a code cell containing Python code for generating a violin plot:

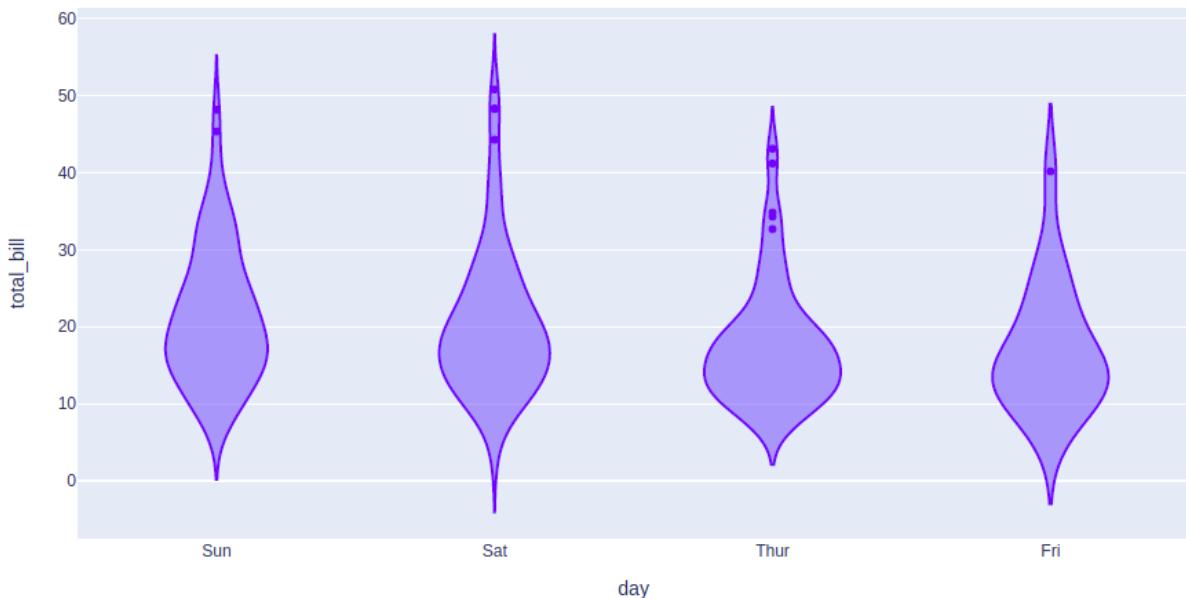
```
import plotly.express as px

# using the tips dataset
df = px.data.tips()

# plotting the violin chart
fig = px.violin(df, x="day", y="total_bill")

# showing the plot
fig.show()
```

Output



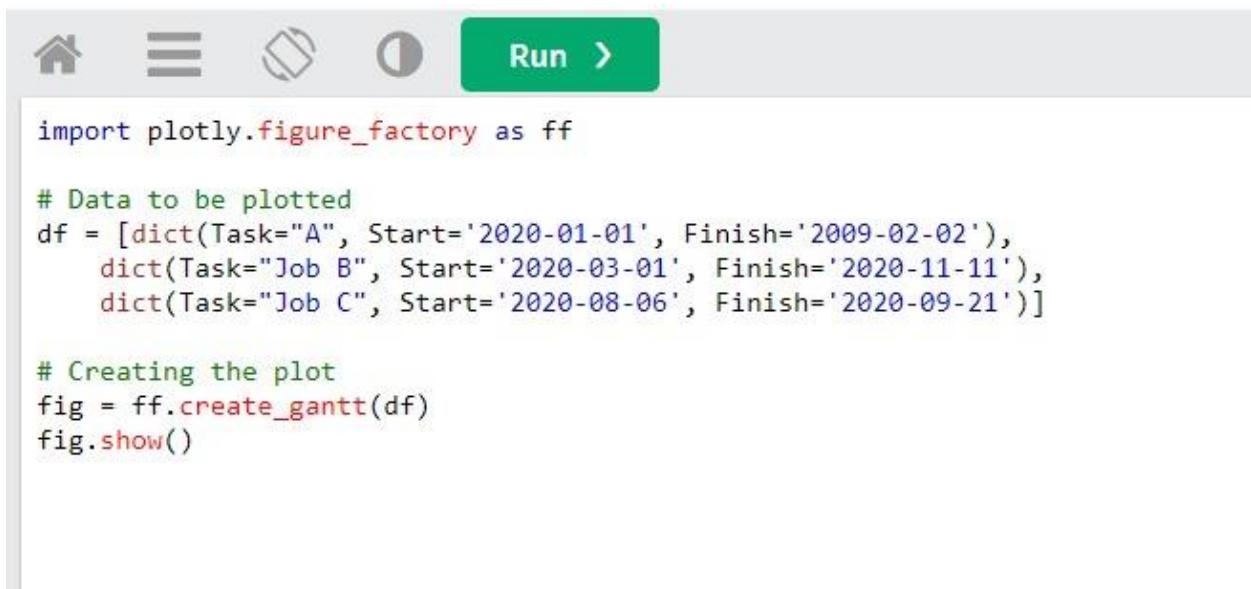
Unit 18.11- Gantt

Definition

The Gantt chart is the specific chart in which specific series of the horizontal line has been presented which shows the total amount of the work done as well as completion of production within the specific timeframe for the planned projects amount

Example

Input



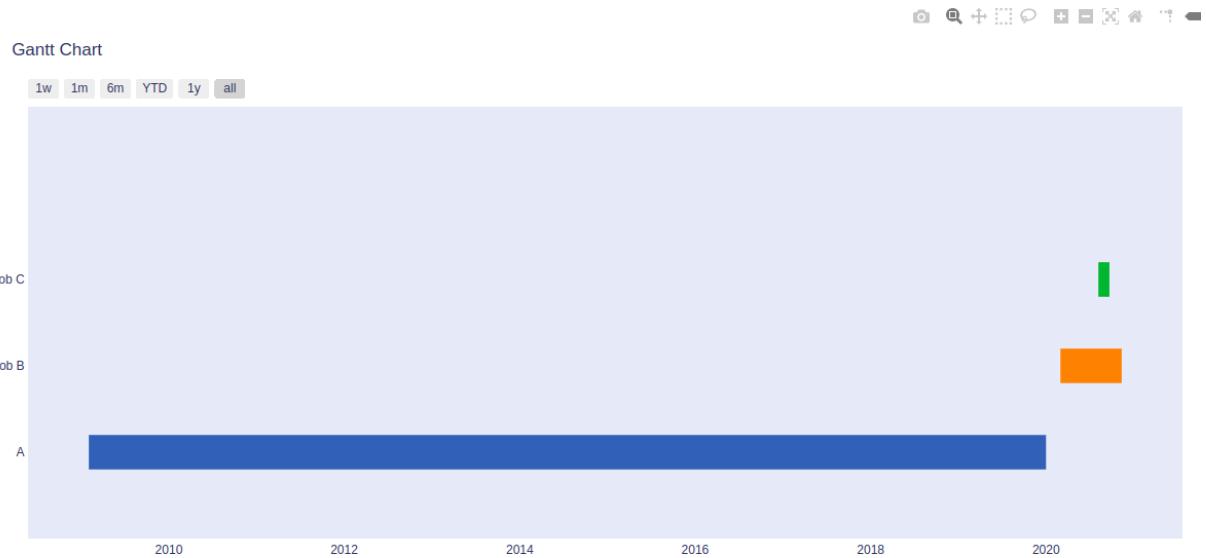
The screenshot shows a Jupyter Notebook interface with a toolbar at the top featuring icons for file, cell, and run, followed by a green 'Run >' button. Below the toolbar is a code cell containing the following Python code:

```
import plotly.figure_factory as ff

# Data to be plotted
df = [dict(Task="A", Start='2020-01-01', Finish='2020-02-02'),
      dict(Task="Job B", Start='2020-03-01', Finish='2020-11-11'),
      dict(Task="Job C", Start='2020-08-06', Finish='2020-09-21')]

# Creating the plot
fig = ff.create_gantt(df)
fig.show()
```

Output



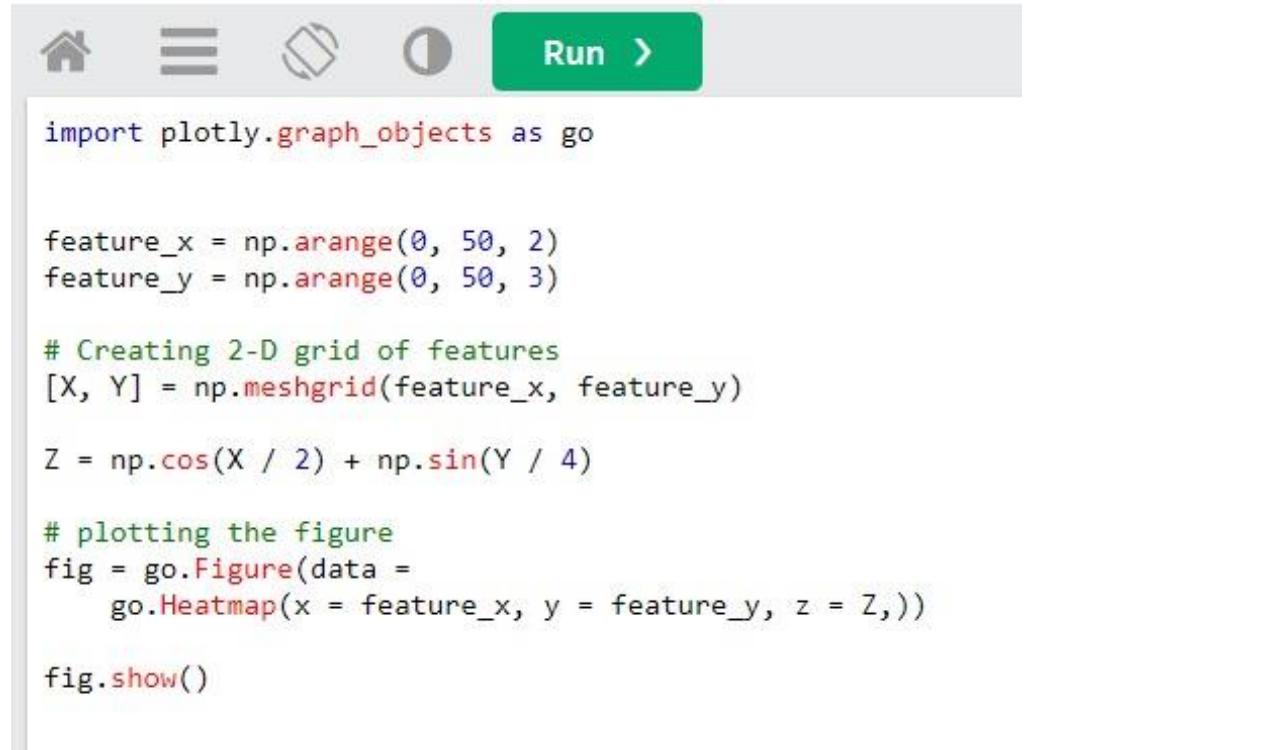
Unit 18.12- Heatmap

Definition

The heatmap is referred as the specific graphical representation of different types of data with the help of color for visualizing the matrix value. For representing the more of the common values as well as higher activities, the reddish color is used; on the other hand, for representing the less common based colour, the darker color is always preferred.

Example

Input



The screenshot shows a Jupyter Notebook interface with a toolbar at the top featuring icons for file, cell, and run, followed by a green "Run >" button. Below the toolbar is a code cell containing the following Python code:

```
import plotly.graph_objects as go

feature_x = np.arange(0, 50, 2)
feature_y = np.arange(0, 50, 3)

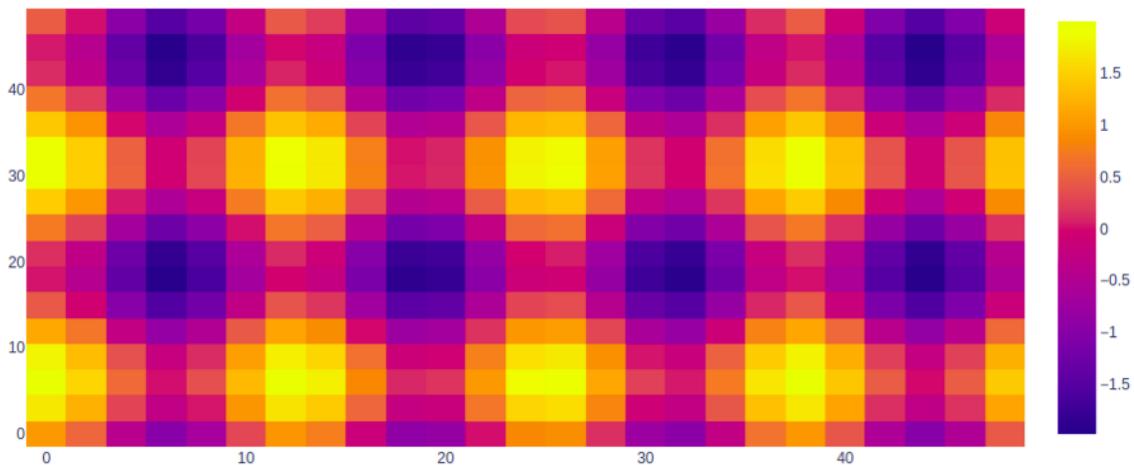
# Creating 2-D grid of features
[X, Y] = np.meshgrid(feature_x, feature_y)

Z = np.cos(X / 2) + np.sin(Y / 4)

# plotting the figure
fig = go.Figure(data =
    go.Heatmap(x = feature_x, y = feature_y, z = Z))

fig.show()
```

Output



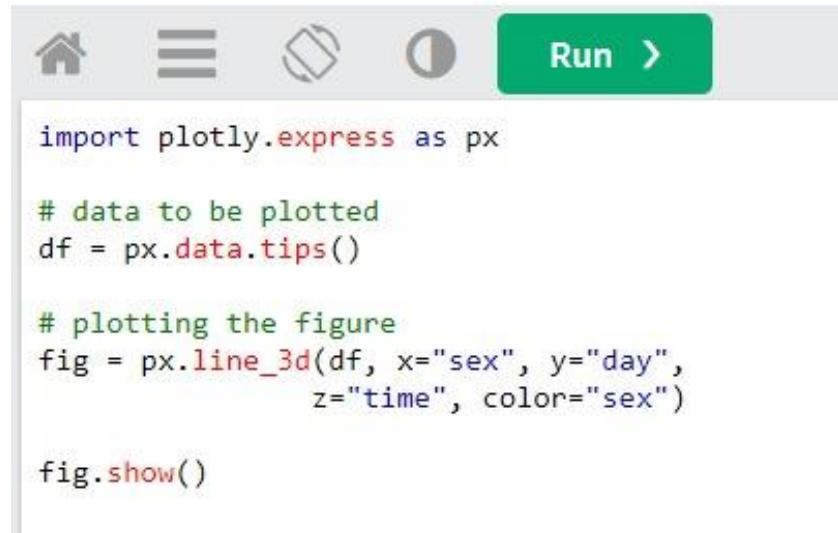
Unit 18.13- 3D graph

Definition

3D graph in the context of Plotly provides the different types of illustration for managing the different types of data variety. With the help of the px.line_3d, in every data position has represented in the form of vertex.

Example

Input



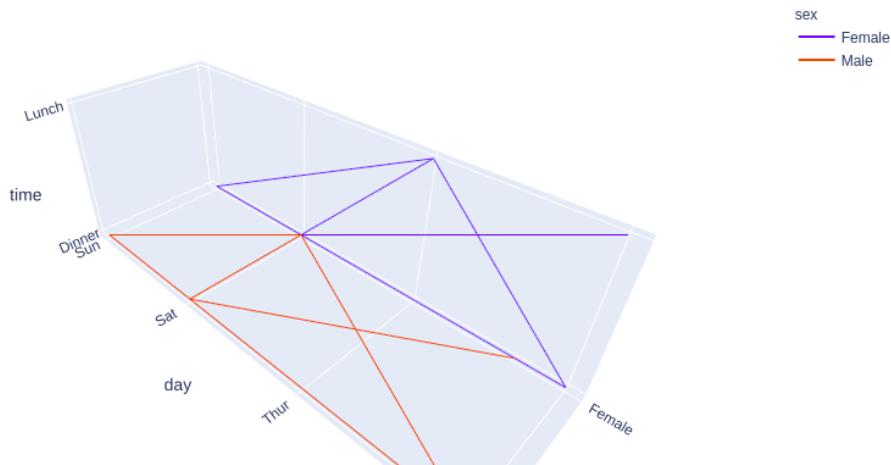
```
import plotly.express as px

# data to be plotted
df = px.data.tips()

# plotting the figure
fig = px.line_3d(df, x="sex", y="day",
                  z="time", color="sex")

fig.show()
```

Output



MCQ

Question 1- One of the most popular data visualization libraries in Python?

- a) Pip
- b) Matplotlib
- c) Both
- d) None

Question 2- Which Plot display the data distribution based on the five number summaries?

- a) Line
- b) Box
- c) Chart
- d) Scatter

Question 3- Which of the following do not visualize data?

- a) Map
- b) Graph
- c) Shape
- d) Chart

Question 4- Identification of the Following chart type?

- a) Scatter
- b) Bubble
- c) Pie Chart
- d) Frequency Polygon

Question 5- Plotly is __ plotting Library?

- a) 4D
- b) 3D
- c) 2D
- d) 1D

Question 6- Data __ refers to the graphical representation of data

- a) Visualization
- b) Plotting
- c) All
- d) None

Case Studies

<https://towardsdatascience.com/data-visualization-with-python-using-seaborn-and-plotly-gdp-per-capita-life-expectency-dataset-cd5426e7ab3b>

<https://ashutoshtripathi.com/2020/08/23/data-visualization-using-plotly-matplotlib-seaborn-and-squarify/>

Reference

<https://www.kaggle.com/code/mehmetkasap/data-visualization-tools-seaborn-and-plotly>

<https://analyticsindiamag.com/plotly-vs-seaborn-compari/>

<https://towardsdatascience.com/matplotlib-vs-seaborn-vs-plotly-f2b79f5bddb>

<https://www.coursereport.com/blog/intro-to-3-data-viz-tools-matplotlib-seaborn-and-plotly>

Unit 19- Experiment no. 10

Unit 19.1- Program 18

“Visualize dataset using plotly and create heatmap of the correlation between different columns”

Importation of the library

```
import sklearn
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Finding the internal correlation between the two of the variable

Input

```
y = pd.Series([1, 2, 3, 4, 3, 5, 4])
x = pd.Series([1, 2, 3, 4, 5, 6, 7])
correlation = y.corr(x)
correlation
```

Output

```
0.8603090020146067
```

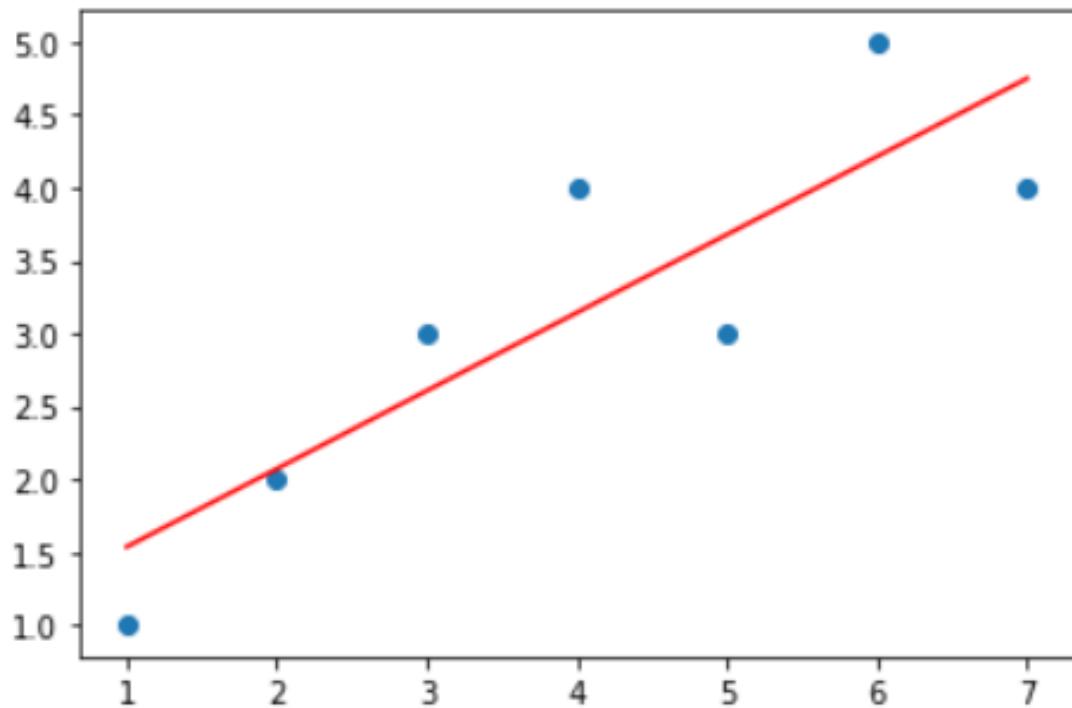
Plotting the graph with the help of scatter plot

Input

```
# plotting the data
plt.scatter(x, y)

# This will fit the best line into the graph
plt.plot(np.unique(x), np.poly1d(np.polyfit(x, y, 1))
         (np.unique(x)), color='red')
```

Output



Adding title and respective labels in the respective graph

Input

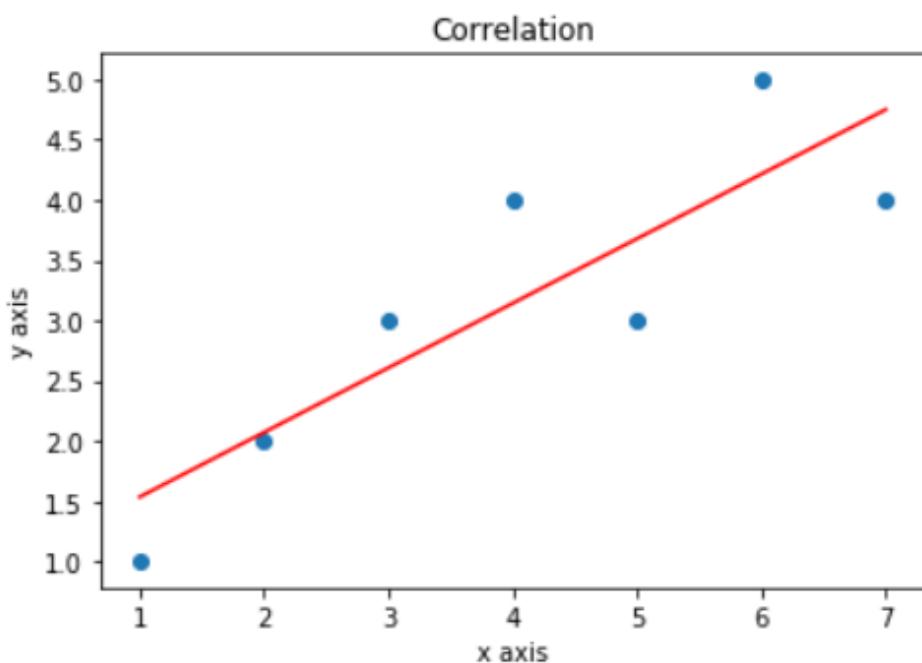
```
# adds the title
plt.title('Correlation')

# plot the data
plt.scatter(x, y)

# fits the best fitting line to the data
plt.plot(np.unique(x),
          np.poly1d(np.polyfit(x, y, 1))
          (np.unique(x)), color='red')

# Labelling axes
plt.xlabel('x axis')
plt.ylabel('y axis')
```

Output



Plotting the Graph using Heat Maps

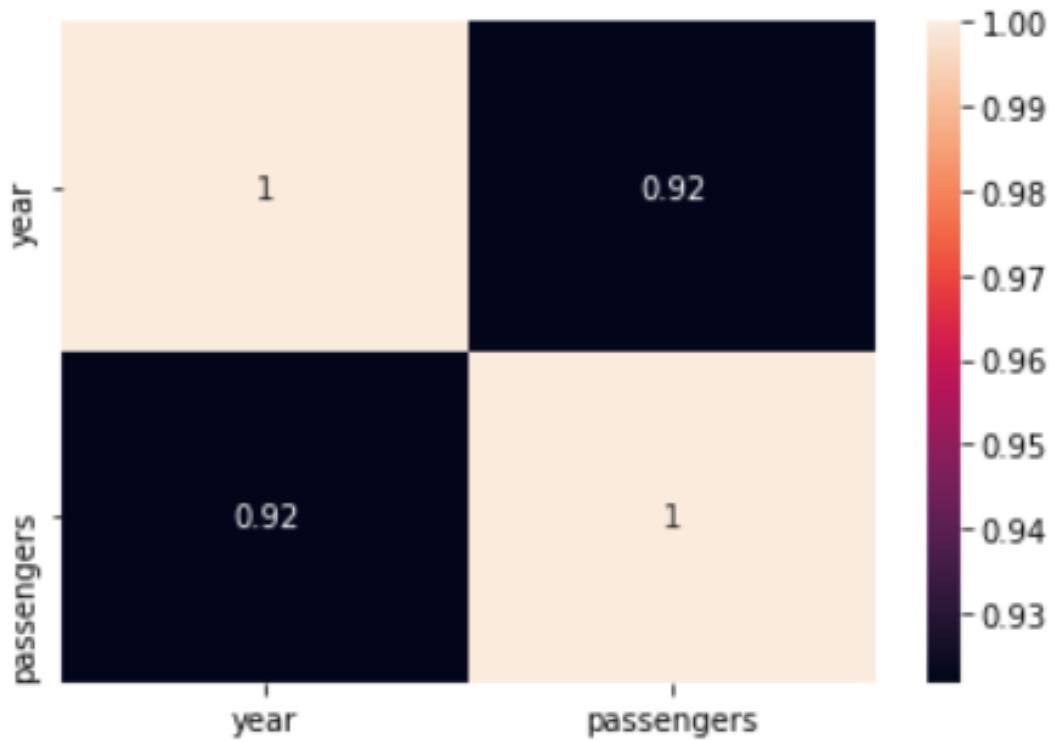
Input

```
import seaborn as sns

# checking correlation using heatmap
#Loading dataset
flights = sns.load_dataset("flights")

#ploting the heatmap for correlation
ax = sns.heatmap(flights.corr(), annot=True)
```

Output



Case Studies

<https://www.geeksforgeeks.org/covid-19-analysis-and-visualization-using-plotly-express/>

<https://k21academy.com/datasience/python/data-visualization-using-plotly/>

<https://www.analyticsvidhya.com/blog/2021/07/interactive-data-visualization-plots-with-plotly-and-cufflinks/>

Reference

<https://towardsdatascience.com/better-heatmaps-and-correlation-matrix-plots-in-python-41445d0f2bec>

<https://stackoverflow.com/questions/29432629/plot-correlation-matrix-using-pandas>

<https://www.stackvidhya.com/plot-correlation-matrix-in-pandas-python/>

<https://plotly.com/python/v3/ipython-notebooks/big-data-analytics-with-pandas-and-sqlite/>