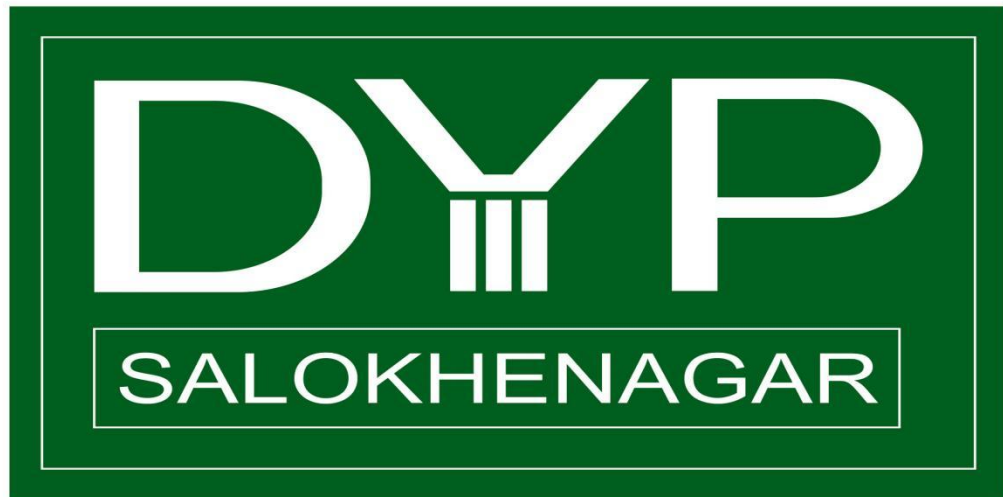


**Dr.D. Y. PATIL PRATISHTHAN'S COLLEGE OF ENGINEERING,  
SALOKHENAGAR, KOLHAPUR**



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING (DATA SCIENCE)


LABORATORY MANUAL

PYTHON FOR DATA SCIENCE

SECOND YEAR

SEM-IV

YEAR: 2022-23

	Dr. D. Y. Patil Pratishthan's College of Engineering, Salokhe Nagar, Kolhapur.	
	Doc. No.: DYP-ACAD-FRM	Rev. No:00
	PAGE 1 of	Rev. Date:
	Department of CSE(Data Science)	Year:2022-23
	Subject-Python for Data Science	Class- SY

Department: CSE(Data Science)

Academic year: 2022-23

Class: SY

Lab Name : Python for Data Science

Date:

Semester: IV

Division: --

Lab In-charge: Prof. P. R. Kamble

Sr. No	Title of Experiment	Source
01	Program to demonstrate data types and their Methods.	SUK CURRICULUM
02	Program to demonstrate Looping- Loop Control statement.	
03	Program to demonstrate User defined functions	
04	Program to demonstrate Files: File manipulations	
05	Program to demonstrate Exception Handling	
06	Program to demonstrate Class & Objects and Inheritance	
07	Program to demonstrate String Manipulation	
08	Program to demonstrate to use pandas Data Structures	
09	Program to demonstrate Plotting with pandas	
10	Program to demonstrate to use NumPy	
11	Program to demonstrate to use SciPy Library	
12	Program to demonstrate Data processing	

## Experiment No.1

**Title:-**Program to demonstrate data types and their Methods.

**Aim:-**To demonstrate data types and their methods in Python.

**Objective:-**To provide a comprehensive understanding of the different data types available in Python and their associated methods

### Theory:-

#### What is a Data Type in python?

A data type represents the type of the data stored into a variable or memory.

By using **type(p)** function we can check the data type of each variable.

- **type(parameter1)** is predefined function in python.
- By using this we can check the type of the variables.

There are **two type** of data types.

1. Built-in data types
2. User defined data types

- **Built-in data types:**

The data types which are already existing in python are called built- in data types.

1. **Numeric types**
  - i. int
  - ii. float
2. **bool (boolean type)**
3. **None**
4. **Complex**
5. **Sequence**
  - i. Str
  - ii. List
  - iii. Set
  - iv. Tuple
  - v. Dict
  - vi. Range
  - vii. Bytes
  - viii. Bytearray
  - ix. frozenset

- **User defined data types :**

The datatype which are created by the programmers are called 'user- defined' data types, example is class, module, array etc.

Datatype	Description	Is Immutable	Example
Int	We can use to represent the whole/integral numbers	Immutable	<pre>&gt;&gt;&gt; a=10 &gt;&gt;&gt; type(a) &lt;class 'int'&gt;</pre>
Float	We can use to represent the decimal/floatingpoint numbers	Immutable	<pre>&gt;&gt;&gt; b=10.5 &gt;&gt;&gt; type(b) &lt;class 'float'&gt;</pre>
Complex	We can use to represent the complex numbers	Immutable	<pre>&gt;&gt;&gt; c=10+5j &gt;&gt;&gt; type(c) &lt;class 'complex'&gt; &gt;&gt;&gt; c.real 10.0 &gt;&gt;&gt; c.imag 5.0</pre>
Bool	We can use to represent the logical values(Only allowed values are True and False)	Immutable	<pre>&gt;&gt;&gt; flag=True &gt;&gt;&gt; flag=False &gt;&gt;&gt; type(flag) &lt;class 'bool'&gt;</pre>
Str	To represent sequence of Characters	Immutable	<pre>&gt;&gt;&gt; s='Suresh' &gt;&gt;&gt; type(s) &lt;class 'str'&gt; &gt;&gt;&gt; s="Suresh" &gt;&gt;&gt; s="IT Software Solutions ... Pune" &gt;&gt;&gt; type(s) &lt;class 'str'&gt;</pre>
bytes	To represent a sequence of byte values from 0-255	Immutable	<pre>&gt;&gt;&gt; list=[1,2,3,4] &gt;&gt;&gt; b=bytes(list) &gt;&gt;&gt; type(b) &lt;class 'bytes'&gt;</pre>
bytearray	To represent a sequence of byte values from 0-255	Mutable	<pre>&gt;&gt;&gt; list=[10,20,30] &gt;&gt;&gt; ba=bytearray(list) &gt;&gt;&gt; type(ba) &lt;class 'bytearray'&gt;</pre>
range	To represent a range of values	Immutable	<pre>&gt;&gt;&gt; r=range(10) &gt;&gt;&gt; r1=range(0,10) &gt;&gt;&gt; r2=range(0,10,2)</pre>
list	To represent an ordered collection of objects	Mutable	<pre>&gt;&gt;&gt; l=[10,11,12,13,14,15] &gt;&gt;&gt; type(l) &lt;class 'list'&gt;</pre>
tuple	To represent an ordered collections of objects	Immutable	<pre>&gt;&gt;&gt; t=(1,2,3,4,5) &gt;&gt;&gt; type(t) &lt;class 'tuple'&gt;</pre>
set	To represent an unordered collection of unique objects	Mutable	<pre>&gt;&gt;&gt; s={1,2,3,4,5,6} &gt;&gt;&gt; type(s) &lt;class 'set'&gt;</pre>

Here are some of the built-in data types in Python and some common methods associated with them:

**String:** A string is a sequence of characters.

1. **upper()** - converts a string to uppercase
2. **lower()** - converts a string to lowercase
3. **strip()** - removes whitespace from the beginning and end of a string.
4. **split(sep=None, maxsplit=-1):** Returns a list of words in the string, separated by the specified separator sep. If no separator is specified, it splits on whitespace. maxsplit is an optional argument that limits the number of splits to be made. Example: "hello world".split() would return ['hello', 'world'].
5. **replace(old, new, count=-1):** Returns a copy of the string with all occurrences of the substring old replaced with new. count is an optional argument that limits the number of replacements to be made. Example: "hello world".replace("o", "e") would return "helle world".
6. **find(sub, start=0, end=-1):** Returns the lowest index in the string where the substring sub is found. start and end are optional arguments that limit the search to a slice of the string. Example: "hello world".find("o") would return 4.
7. **index(sub, start=0, end=-1):** Similar to find(), but raises a ValueError if the substring is not found. Example: "hello world".index("o") would return 4
8. **join(iterable):** Returns a string that is a concatenation of the strings in iterable, separated by the string on which it was called. Example: ".join(['hello', 'world']) would return "hello world".

**List:** A list is a collection of values that are ordered and changeable.

1. **append(item)** - adds an element to the end of a list
2. **insert(index, item)** - adds an element at a specified index in a list
3. **remove()** - removes the first occurrence of an element in a list
4. **pop(index)** - removes the element at a specified index in a list
5. **sort()** - sorts the elements in a list
6. **reverse():** Reverses the order of the items in the list.
7. **count(item):** Returns the number of times the specified item appears in the list.
8. **index(item):** Returns the index of the first occurrence of the specified item in the list.
9. **clear():** Removes all items from the list.

**Tuple:** A tuple is a collection of values that are ordered and unchangeable.

1. **tuple()** - creates a new tuple
2. **count()** - returns the number of times a specified value appears in a tuple
3. **index()** - returns the index of the first occurrence of a specified value in a tuple

**Set:** A set is a collection of values that are unordered and unindexed.

1. **set()** - creates a new set
2. **add()** - adds an element to a set
3. **remove()** - removes a specified element from a set
4. **pop()** - removes an arbitrary element from a set
5. **len()** - returns the number of elements in a set
6. **union()** - returns the union of two sets
7. **intersection()** - returns the intersection of two sets
8. **difference()** - returns the difference between two sets

**Dictionary:** A dictionary is a collection of key-value pairs that are unordered, changeable, and indexed.

1. **dict()** - creates a new dictionary
2. **get()** - returns the value of a specified key
3. **keys()** - returns a list of all the keys in a dictionary
4. **values()** - returns a list of all the values in a dictionary
5. **clear()**: Removes all key-value pairs from the dictionary.
6. **copy()**: Returns a shallow copy of the dictionary.
7. **items()**: Returns a view object that contains the key-value pairs of the dictionary as tuples.
8. **popitem()**: Removes and returns an arbitrary key-value pair from the dictionary as a tuple. Raises a KeyError if the dictionary is empty.

**Conclusion:**

Hence we studied different data types and their methods in python.

## Experiment No.2

**Title:-** Program to demonstrate Looping- Loop Control statement.

**Aim:-**Program to demonstrate Looping- Loop Control statement.

**Objective:-**To provide a comprehensive understanding of the dLooping- Loop Control statement.

### Theory:-

Loops are an essential part of programming, and Python provides several loop constructs that allow you to perform repetitive tasks efficiently.

### Advantages of using loops in Python:

1. **Simplify repetitive tasks:** Loops allow you to repeat a block of code multiple times without having to write it out manually each time.
2. **Improved readability and maintainability:** By encapsulating the repetitive code in a loop, you can make your code more readable and easier to maintain.
3. **Process large amounts of data**
4. **Flexibility:** Python allow you to customize the behavior of your loops to fit your specific needs
5. **Enhanced functionality:** With loops, you can perform a wide range of tasks, such as searching for specific items in a list, calculating statistical measures, or generating complex patterns.

In Python, there are two main types of loops: the **for loop** and the **while loop**.

1. **for loop:** A for loop is used to iterate over a sequence (e.g., a list, tuple, or string) or other iterable object.

#### Syntax:

**for variable in sequence:**

**# statements to be executed inside the loop**

2. **while loop:** A while loop is used to repeatedly execute a block of code as long as a specified condition is true.

#### Syntax:

**while condition:**

**# statements to be executed inside the loop**

- **if-elif-else statement:**

In Python, the **if** statement is used to execute a block of code only if a specified condition is true.

The if statement can also be combined with the **else** keyword to execute a block of code if the condition is false.

Additionally, the if-else statement can be extended to include multiple conditions using the **elif** (short for "else if") keyword

**Syntax:**

```
if condition1:
    # statements to be executed if condition1 is true
elif condition2:
    # statements to be executed if condition2 is true and condition1 is false
else:
    # statements to be executed if both condition1 and condition2 are false
```

The if statement can be used in combination with loops (e.g., for and while loops) and other control flow statements (e.g., break, continue, and pass) to create more complex programs that perform specific tasks

There are three types of loop control statements in Python: break, continue, and pass.

1. **break:** Terminates the loop prematurely. When a break statement is executed, the loop is exited and the program continues with the statement following the loop.

**Example:**

```
for i in range(10):
    if i == 5:
        break
    print(i)
```

This will print numbers from 0 to 4, and then terminate the loop when i becomes

2. **continue:** Skips the rest of the current iteration of the loop and continues with the next iteration.

**Example :**

```
for i in range(10):
    if i == 5:
        continue
    print(i)
```



This will print numbers from 0 to 9, except for 5.

3. **pass:** It is used as a placeholder when a statement is required syntactically but you don't want to execute any code.

**Example:**

```
for i in range(10):  
    if i == 5:  
        pass  
    else:  
        print(i)
```

This will print numbers from 0 to 9, except for 5, without doing anything special when i is 5.

Note that loop control statements can be used with for loops, while loops, and nested loops. They are powerful tools that allow you to customize the behavior of your loops to fit your specific needs.

**Conclusion :**

Thus we studied looping (for, while, if-elif-else) - loop control (break, continue, pass) statements in Python.

### Experiment No.3

**Title:-** Program to demonstrate User defined functions.

**Aim:-** Program to demonstrate User defined functions.

**Objective:-** To provide a comprehensive understanding of User defined functions.

**Theory:-**

In Python, a function is a block of reusable code that performs a specific task. A function in Python can take zero or more arguments as input and return zero or more values as output.

**In Python, there are several types of functions**

1. **Built-in functions:** These are pre-defined functions in Python that are available for use without the need for import statements. Examples include **print()**, **len()**, and **max()**.
2. **User-defined functions:** These are functions defined by the user to perform specific tasks. They are created using the **def keyword**, followed by the function name, parameters (if any), and the body of the function.
3. **Anonymous functions (Lambda functions):** These are small, one-line functions that can be defined without a name. They are created using the **lambda keyword**, followed by the parameters and the expression to be evaluated.

**Syntax:**

**lambda** argument\_list: expression

**map(p1, p2) function:-** map(fun, iterable)

**filter(p1, p2) function:-** filter(function, sequence)

**reduce(p1, p2) function:-** reduce(function, sequence)

A function can contain mainly two parts,

1. Defining function
2. Calling function

**1. Defining function :**

**Syntax:**

**def function\_name(parameters):**

```
"""docstring"""  
# function code  
return [expression]
```

- The **def** keyword is used to define a function.
- **function\_name** is the name of the function.
- **parameters** are the input arguments to the function. These are optional and can be omitted if the function doesn't take any arguments.
- The **docstring** is an optional description of what the function does. It should be enclosed in triple quotes.
- The **return** statement is used to return a value from the function. This is also optional and can be omitted if the function doesn't return anything.

## 2. Calling function:

- After function is created then we need to call that function to execute the function body.
- While calling the function, function name should be match otherwise we will get error.

Example:

```
# function creation  
def display():  
    print("welcome to function concept")  
  
# function calling  
display()  
  
Output: welcome to function concept
```

### ● **Types of arguments/Parameters in function:**

Based on Parameters: Functions are two types:

#### ● **Function without parameters (or) No parameterised function**

If a function having no parameters then that function is called as, a function without parameters.

#### ● **Function with parameters (or) Parameterised function**

If a function having parameters then that function called as function with parameters.

In programming, there are typically four types of arguments that can be passed to a function:

1. **Positional arguments:** These are arguments that are passed to a function based on their position or order. For example, if a function takes three positional arguments, then the first argument will be assigned to the first parameter of the function, the second argument will be assigned to the second parameter, and so on.
2. **Keyword arguments:** These are arguments that are passed to a function using a key-value syntax. With keyword arguments, the order of the arguments doesn't matter, as long as the keys are correct. For example, if a function takes a keyword argument named "color", you could call the function like this: `my_function(color='red')`.
3. **Default arguments:** These are arguments that have a default value specified in the function definition. If a value is not provided for a default argument when the function is called, the default value will be used instead. For example, if a function has a default argument `x=5`, you can call the function with `my_function()` and it will use the default value of `x`.
4. **Variable-length arguments:** These are arguments that can accept a variable number of inputs. In Python, you can define variable-length arguments using the `*args` syntax. For example, `def my_function(*args)` would allow you to pass any number of arguments to the function, and they would be stored in a tuple called `args`.

In Python, **\*\*kwargs** is a special syntax used in function definitions to allow the function to accept an arbitrary number of keyword arguments. The **\*\* operator** before the keyword **kwargs** allows you to pass a dictionary of keyword arguments to the function, which can then be accessed in the function as a dictionary.

### **Return Statement in function:**

In programming, the **return** statement is used in a function to specify the value that the function should return. When a function is called, it may perform some operations or computations and then return a value to the caller. The return statement allows you to specify what that value should be.

### **Example:-**

```
def add(a, b):  
    result = a + b  
    return result
```

```
sum = add(3, 4)
```

The variable **sum** will now contain the value **7**, which is the result of adding **3** and **4**.

**Conclusion:**

Hence, we studied predefined and user defined function in python.

## Experiment No.4

**Title:-** Program to demonstrate files: file manipulations

**Aim:-** To demonstrate files: file manipulations in Python.

**Objective:-** To provide a comprehensive understanding of files: file manipulations methods

**Theory:**

Python provides inbuilt function for creating, working and reading files.

**These are two types of files that can be handled by python:**

- 1) Normal text file and
- 2) Binary files

**Text File:**

In this type of file, each line of text is terminated with a special character called EOL (End Of line) which is the new line character ('\n') in python by default.

In this, we will study the opening, closing, reading and writing data in a text file.

**File access Mode:**

Access modes govern the type of operation possible in the opened files. It refers to how the file will be used once it is opened.

These modes also define the location of the file handle in the file. File handle is like a cursor, which defines from where the data has to be read or write in the file. There is some access mode in python.

### ● Files access Mode

**Read modes:**

#### 1. r:

Open a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.

#### 2. rb:

Open a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.

#### 3. rb+:

Opens a file for both reading and writing in binary format. The file pointer is placed at the beginning of the file.

**Write mode:****1. W:**

Opens a file for writing only other overwrites the file if the file exists.If the file does not exist creates a new file waiting.

**2. Wb:**

Open a file for writing only in binary format.

**● The seek() and tell() methods:**

- tell():** ==>We can use tell() method to return current position of the cursor(file pointer) from beginning of the file. [ can you please tell current cursor position] The position(index) of first character in files is zero just like string index.

**2. seek() :**

It is used to start reading data from particular position.

**Open a file in read mode of particular position:**

```
text_file=open("Path of file","r")
complete_text.seek(40)
complete_text.read()
```

**● Reading a file by line:**

```
text_file=open("Path of file","r")
lines=text_file.readlines()
```

**readlines:**

line by line code is read.

**● Writing to file**

```
my_file=open("Path","w")
e.g:
lines=['line1','line2','line3']
my_life.writeline(lines)
my_life.write()
```

**1. Writeline():**

write multiple lines.

**2. Write():**

write single file.

**3. W+:**

Open a file for both writing and reading

**4. Wb+:**

Open a file for both writing and reading in binary format

**● Append Mode:****1. a:**

Open a file for appending. The file Pointer is at the end of the file.

**2. ab:**

Opens a file for appending in binary file.

**3. a+:**

Opens a file for appending and reading.

**4. ab+:**

Opens a file for both appending and reading in binary format.

**● Open a file in read mode:**

```
txt.file=open("Path of file","r")
```

**● To close file:**

```
Text_file.close()
```

**● Append to file:**

```
my.file=Open("Path","a+")
```

**example:**

```
lines=['line1','line2','line3']  
my_life.writeline(lines)  
my_life.write()
```

**Conclusion:**

In this experiment we demonstrated program on the file manipulation.



## Experiment No. 5

**Title:-**Program to demonstrate Exception Handling

**Aim:-**To demonstrate Exception Handling in Python.

**Objective:-**To provide a comprehensive understanding of Exception Handling methods

**Theory:**

In Python exception can be defined as an unusual condition in a program resulting in the interruption in the flow of program.

Whenever, an exception occurs the program stop the execution and thus the further code is not executed. Therefore, an exception is the runtime error that are unable to handle to python script. An exception is a python object that represents an error.

Python provides a way to handle the exception so the code, can be executed without any interruption.

Python has many built-in exceptions that enable our program to run without interruption and given the output.

**Keywords Used in Exception Handling**

It is possible to write a program that handle selected exception using the keyword:

- i. try
- ii. except
- iii. else
- iv. finally

● **Try:**

The try block contains the statement which may cause the error.

**Syntax:** try:

    #block of code

● **except:**

When exception occur in try block is handled by except block.

**Syntax:** try:

    #block of code

except:

    #block of code

● **else:**

If there is no execution in program will execute else block.

**Syntax:** try:

```
        #block of code
    except:
        #block of code
    else:
        #block of code
```

- **finally:**

The statement which are in finally block always executed whether error occurs or not in program statements.

**Syntax:**

```
finally:
    #block of code.
```

- **Error:**

It is abnormal condition whenever it occurs execution of program will be stopped. On the other hand, exception are raised when some internal events occur which changes the normal flow of the program.

- **The errors are classified into:**

1. Syntax Error
2. Semantic Error
3. Logical Error
4. Run-time Error

### **1. Syntax Error:**

It refers to rules and regulations of programming languages. It occurs when rules of programming languages are misused.

Example: `def add(a,b)`

`c=a+b`

`print(c)`

error: Colon missing

### **2. Semantic Error:**

It occurs when statement are not meaningful. The statement should be semantically correct and it has some meaning.

Example: `a+b=c`

The syntax is correct but it shows semantic error because expression can't be on left side, of an assignment statement.

### **3. Logical Error:**

The logical error occurs when the logic program is incorrect and it produce incorrect output.

Example:  $10+20=3$

### **4. Run-time Error:**

Run time error occurs during program execution.

Example:  $10/0$

An expression trying to divide a number by 0 it shows run time error as Divide by Zero.

### **Conclusion:**

In this experiment, we understood the basic concept of exception handling and demonstrated programs.

## Experiment No.6

**Title:-**Program to demonstrate Class,Object and Inheritance

**Aim:-**To demonstrate Class,Object and Inheritance in Python.

**Objective:-**To provide a comprehensive understanding of Class,Object and Inheritance methods

**Theory:**

### **Class and objects in programming:**

A Class describes to things - the forms and Object created from it will take and functionality it will have.

**for example:** A bird class may specify the form in terms of weight, color, number of feathers etc and functionality in term of flying, hopping, chirping, eating etc.

Thus class is generic in nature,an object is specific in nature.

Multiple objects can be created from a class.The process of creation of an object from a class is called instantiation.

In python everything is a class so int, float, complex, bool, str, list, tuple, set, dic are all classes.

A class has a name, where as objects are nameless. Since object do not have names. They are referred using their addresses in memory.

### **More programatic example of classes and objects**

```
a=3.14 # a is an object of float class.
```

```
s=sudesh #s is an object of str class.
```

```
lst=[10,20,30] #lst is an object of list class.
```

The specific data in an object is often called instance data or properties of the object or state of the object or attributes of the object.

### ● **Some Points on Python Class:**

Classes are created by keyword **class**.

**Attributes** are the variables that belong to a class.

Attributes are always public and can be accessed using the **dot(.) operator**.

**Example:** myclass.myattribute

Class Definition syntax:

Class Classname:

    #Statement

    .

    .

    .

    #Statement\_n

### **Class Objects:**

- ✓ An object is an instance of a class.
- ✓ A Class is like a blueprint while an instance is a copy of the class with actual values.

An object consists of:

- ✓ State
- ✓ Behavior
- ✓ Identity

Identity	State/Attribute	Behavior
Name of dog	Breed	Dark
	Age	Sleep
	Color	Eat

### ● **Inheritance in Python:**

Inheritance is the capability of one class to derive or inherit the properties from another class.

- ✓ **Benefits:**
- It prevents reall-world relationship well

- It provides reusability of a code we don't have to write the same code again and again.
- It is transitive in nature, which means that if class B inherits from another class A, then all the subclasses of B would automatically inherit from class A.

- **Different Forms of Inheritance:**

- ✓ **Single Inheritance:**

When a child class inherits from only one parent class, it is called single inheritance

- ✓ **Multiple Inheritance:**

When a child class inherits from multiple class, it is called single inheritance

- ✓ **Multilevel Inheritance:**

When we have a child and grandchild relationship.

- ✓ **Hierarchical Inheritance:**

More than one derived classes are created from a single base.

- ✓ **Hybrid Inheritance:**

This form combines more than one form of inheritance. Basically It is a blend of more than one type of inheritance.

**Conclusion:**

We demonstrated the programs related to Class, Object and Inheritance in Python.

## Experiment No.7

**Title:-**Program to demonstrate String manipulation.

**Aim:-**To demonstrate String manipulation in Python.

**Objective:-**To provide a comprehensive understanding of String manipulation methods.

**Theory:**

- **Python string:**

1. Python String is the collection of characters surrounded by single quotes,double quotes,triple quotes.
2. The computer does not understand the characters internally it stores manipulated characters as the combination of 0's and 1's.
3. Each character is enclosed in the ASCII or unicode character.so we can say that python string are also called the collection of unicode characters.

**for e.g.:-** `str="Hipython!"`

- **String indexing and splitting:**

1. Like other languages,the indexing of python string starts from 0.
2. The slice operator `[]` is used to access the individual characters of string.However,we can use the `(:)` colon operator in python to access the sub string from the given string.

- **Reassigning the String:**

1. Updating the control of the string is as easy as assigning it to a new string.The string object doesn't support item assignment i.e. A string since its content cannot be partially replaced.
2. Strings are immutable in python.

**for e.g.:**

```
str="Hello"  
str[0]='H'  
print(str)
```

**● Deleting the String:**

1. As we know that string are immutable. We cannot delete or remove the characters from the list.
2. But we can delete the entire string using the del keyword.

**for e.g.:**

```
str1="Python"
```

```
del str1
```

**● String Operator:**

Operator	Description
+	It is known as concatenation operator used to join the strings given either side of the operator.
*	It is known as repetition operator. It concatenate the multiple copies of the same string.
[]	It is known as slice operator. It is used to access the sub-string of a particular string.
[:]	It is known as range slice operator. It is used to access the characters from the specified range.
in	It is known as membership operator does not extract reverse of it. It returns true if a particular substring is present in the specified string.
not in	It is also a membership operator does not extract reverse of it. It returns true if a particular substring is not present in the specified string.

**● Methods in str class:**

- ✓ str is a predefined class.
- ✓ We can check these **methods** by using **dir(parameter1)** predefined function.
- ✓ str class can contain methods because methods can be created inside of class only.
- ✓ all str class methods we can access by using str object

**● Important methods in str class:**

- ✓ upper()



- ✓ lower()
- ✓ strip()
- ✓ count(parameter)
- ✓ replace(parameter1, parameter2)
- ✓ split(parameter)

### 1. upper():

This method converts lower case letters into upper case letters

**Eg:**

```
Name= "dypsn"
```

```
Print("Converting in capital letter:",name.upper())
```

**Output:**

```
Converting in capital letter: DYPSN
```

### 2. lower():

converts upper case letters into lower case letters

### 3. strip() method:

This method removes left and right side spaces of string

**Eg:**

```
course= "Python   "
```

```
x=course.strip()
```

```
print("After strip:",x)
```

**Output:**

```
Python
```

### 4. count(p) method:

By using count() method we can find the number of occurrences of substring present in the string.

**Eg:**

```
s="Python programming language, Python is easy"
```

```
print(s.count("Python"))
```

```
print(s.count("Hello"))
```

**Output:**

```
2
0
```

### **5. replace(p1, p2) method:**

We can replace old string with new string by using replace(p1, p2) method.

**Eg:**

```
s1="Java programming language"
s2=s1.replace("Java", "Python")
print(s1)
print(s2)
```

**Output:**

```
Java programming language
Python programming language
```

### **Conclusion:**

Thus we demonstrated programs on string manipulation.

## Experiment No.8

**Title:-**Program to demonstrate to use pandas Data Structure.

**Aim:-**To demonstrate the use of pandas Data Structure in Python.

**Objective:-**To provide a comprehensive understanding of the use of pandas Data Structure methods

### Theory:

#### ● **Pandas Data Structure:**

1. Pandas is an open-source library that uses for working with relational or labeled data both easily and intuitively.
2. It provides various data structures and operations for manipulating numerical data and time.
3. It is most popular python library is used for data analysis.
4. It supports Three data structure:
  - i. Series
  - ii. Dataframe
  - iii. Panel (A Panel is a group of DataFrames of data.)

The most widely used data structures are the Series and the DataFrame.

#### i. **Series:**

A series can be seen as a one-dimensional array. The data structure can hold any data type, that is including strings, integers, floats and python objects.

##### **Syntax:**

```
pandas series(data=None,index=None,dtype=None,Name=None,fastpath=false)
```

#### ii. **Dataframe:**

- ✓ Dataframe is a dimensional data structure size mutable potentially heterogeneous tabular data structure with labeled axes.
- ✓ Pandas Data frame consist of three principle components the data rows and columns.
- ✓ Basic operation that can be applied on pandas Data frame are shown in below:

1. Creating data frame
2. Performing operations on rows and columns
3. Data selection:addition,deletion
4. Working with missing data
5. Renaming the columns or indices of a data frame

### **1. Creating a data frame:**

- ✓ The pandas data frame can be created by loading the dataframe the external,existing storage like database SQL or CSV files.
- ✓ The pandas dataframe can be also created from the list,dictionary etc.

### **2. Performing operations on rows and Columns:**

- ✓ Data frame is a two-dimensional data structure data is stored in rows and columns.
- ✓ Below we can perform some operation on rows and columns.
  - 1) Selecting a column
  - 2) Selecting a row

### **3. Data selection:addition,deletion:**

- ✓ You can treat a dataframe semantically like a dictionary of like indexed series object.
- ✓ Getting setting and deleting columns works with the same syntax as the analogous dictionary operations.

### **4. Working with missing data:**

- ✓ Missing data occurs a lot of time when we are accessing big data set.It occurs often Non(Not a Number).
- ✓ In Order to fill those values,we can use 'is null()' method.

**5. Renaming the column or indices of a data frame:**

- ✓ To give the columns of the indx dataframe a different value, its best to use the rename() method.

**● Series object Attributes:**

- ✓ The series attributes is defined as any information related to the series object such as size, data type etc.
- ✓ Below are some attributes that you can use to get information about series object.

Attributes	Description
Series.index	Define the index of the series.
Series.shape	It returns the tuple of the data.
Series.dtype	It returns the datatype of the data.
Series.size	It defines the size of the data..

**● Series Functions:**

Functions	Description
Pandas series.map()	Map the value from two series that have a common column
Pandas series.std()	Calculate the standard derivation of the given set of numbers. Dataframe:column,rows.
Pandas series to -frame()	Convert the series object to the dataframe
Pandas series value.counts()	Returns a series that contain count

	of unique values
--	------------------

- **DataFrame Functions:**

Functions	Descriptions
Pandas Dataframe append()	Add the row of other data frame to the end of the given dataframe
Pandas Dataframe assign()	Add new column into a data frame
Pandas Dataframe Count()	Count the number of non-NA cells for each column or row.
Pandas Dataframe drop-duplicates()	Remove duplicate value from the dataframe
Pandas Dataframe Concat()	Perform Conactination Operation along on axis in the dataframe
Pandas Dataframe merge()	Merge the two Dataframe together into one.

**Conclusion:**

We understood the concepts of pandas and Demonstrated program using Panda Data Structure.

## Experiment No. 09

**Title:-** Program to demonstrate Plotting with Pandas.

**Aim:-** To demonstrate Plotting with Pandas in Python.

**Objective:-** To provide a comprehensive understanding of Plotting with Pandas.

### Theory:

#### Plotting:

- ✓ Pandas uses the plot() method to create diagram.
- ✓ We can pyplot a sub-meth module of the matplotlib library to visualize the diagram on the screen.

**for e.g.:**

```
import Pandas as pd

import matplotlib.pyplot as plt

df=pd.read_csv('data.csv')

df.plot()

plt.show()
```

### Plots:

There are number of plots variable to interpret the data. Each graph is used for a purpose.

#### 1. Scatter Plot:

- ✓ To get the scatterplot of a dataframe all we have to do is to just call the plot() method by specifying some parameters.
- ✓ In scatter plot each value in the data set is represented by a dot.
- ✓ By using this plot we can understand the relationship between two variables.

```
kind='scatter',x='some_column',y='some_column',color='somecolor'
```

## 2. Bar Plot:

- ✓ Similarly, we have to specify the some parameters for plot() method to get the bar plot.
- ✓ The bar graph is the graphical representation of categorical data.
- ✓ For horizontal bar graph use barh() function and vertical bar chart use bar() function.

```
kind='bar',x='some_column',y='some_column',color='somecolor'
```

## 3. Line Plot:

- ✓ The line plot of a single column is not always useful to get more insights we have to plot multiple columns on the same graph. To do we have to reuse the axes.
- ✓ A line chart or line graph is a type of chart which displays information as a series of data points connected by straight line .
- ✓ A line chart is often used to visualize a trend in data over intervals of time.
- ✓ label x axis and y axis by using xlabel and ylabel

```
kind='line',x='some_column',y='some_column',color='somecolor' ax='someaxes'.
```

## 4. Histogram:

- ✓ A histogram is the graphical representation of quantitative data.
- ✓ This displays the frequency/count of numerical data in bars.

## 5. Pie Chart :

- ✓ This is a circular plot that has been divided into slices displaying numerical proportions.
- ✓ Every slice in the pie chart shows the proportion of the element to the whole.
- ✓ A large category means that it will occupy a larger portion of the pie chart.

## 6. Box Plots:

- ✓ Box plots help us measure how well data in a dataset is distributed.



- ✓ The graph shows the maximum, minimum, median, first quartile and third quartiles of the dataset.
- ✓ Use a boxplot when you need to get the overall statistical information about the data distribution.
- ✓ It is a good tool for detecting outliers in a dataset.

**Box plot explanation :**

- ✓ The line dividing the box into two shows the median of the data.
- ✓ The end of the box represents the upper quartile (75%) while the start of the box represents the lower quartile (25%).
- ✓ The part between the upper quartile and the lower quartile is known as the Inter Quartile Range (IQR) and helps in approximating 50% of the middle data.

**7. Stack Plot :**

- ✓ The idea of stack plots is to show “parts to a whole” over time
- ✓ Stack plots are generated by plotting different datasets vertically on top of one another rather than overlapping with one another.
- ✓ Suppose you need to compare the sales scored by three different months per year over the last 8 years.

**Conclusion:**

Hence we demonstrated the programs plotting with pandas.

## Experiment No.10

**Title:-**Program to demonstrate to use Numpy.

**Aim:-**To demonstrate to use Numpy in Python.

**Objective:-**To provide a comprehensive understanding of to use Numpy methods.

### Theory:

- ✓ Numpy stands for numeric python which is a python Package for the computation and processing of the multi-dimensional and single dimensional array element.
- ✓ Numpy provides various powerful data structure implementing multi-dimensional array and matrices. These data structures are used for the optional computations regarding arrays and matrices.

### Advantages of Using Numpy for data analysis:

- Numpy performs array-oriented computing.
- It efficiently implement the multi-dimensional array.
- It perform scientific computations.
- Numpy provides the in-built functions for linear algebra and radom number generation.

### Data types in Numpy:

i=integer	b=boolean
u=unsigned integer	f=float
c=complex float	m=timedata
M=datetime	o=object
s=string	u=unicode string
v=fixed chunk of memory for other type(voice)	

**Common methods in NumPy:**

1. **numpy.array():** This method is used to create a NumPy array from a Python list or tuple.

**Ex.**

```
import numpy as np

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

print(arr)
```

**O/P:**

```
[1 2 3 4 5]
```

2. **numpy.zeros():** This method is used to create an array filled with zeros.

**Ex.**

```
import numpy as np

arr = np.zeros((3, 4))

print(arr)
```

**O/P:**

```
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

3. **numpy.ones():** This method is used to create an array filled with ones.

**Ex.**

```
import numpy as np

arr = np.ones((2, 3))

print(arr)
```

**O/P:**

```
[[1. 1. 1.]  
 [1. 1. 1.]]
```

4. **numpy.random.rand():** This method is used to create an array of random numbers between 0 and 1.

**Ex.**

```
import numpy as np  
  
arr = np.random.rand(3, 4)  
  
print(arr)
```

**O/P:**

```
[[0.23014491 0.83769004 0.77134061 0.43555515]  
 [0.65711713 0.42420229 0.35320463 0.86882823]  
 [0.96808698 0.33974663 0.16079247 0.34446171]]
```

5. **numpy.arange():** This method is used to create an array with evenly spaced values within a specified range.

**Ex.**

```
import numpy as np  
  
arr = np.arange(0, 10, 2)  
  
print(arr)
```

**O/P:**

```
[0 2 4 6 8]
```

6. **min() method :** By using this we can check minimum value from the array.

**Ex.**

```
import numpy as np  
  
details = np.array([[10, 20, 30], [40, 50, 60]])  
  
sales = np.array(details)
```

```
print(sales.max())
```

**O/P:**

60

7. **max() method:** By using this we can check maximum value from the array.
8. **sum() method :**By using this we can get sum of all values from array.
9. **reshape() method :**By using this we can change the shape of an array

**Ex.:**

```
import numpy as np

details = np.array([[10, 20, 30], [40, 50, 60]])

sales = np.array(details)

print(sales)

print("Reshape Array is: ")

print(sales.reshape(3, 2))
```

**O/P:**

```
array([[10, 20, 30],
       [40, 50, 60]])

Reshape Array is:

[[10 20]
 [30 40]
 [50 60]]
```

10. **sort() method:** By using this we can sort values in array.
11. **flatten() method :** This method keeps all values in one dimension array.

**Creating arrays with defined datatype:**

We use the `array()` function to create arrays. This function can take an optional argument: `d` type that allows us to define the expected data type of the array elements

**Shape of array:**

- The shape of array can be defined as the number of elements in each dimension
- Dimension is the number of indices or subscripts that we required in order to specify an individual element of an array.

**Syntax:**

**`numpy.shape(array_name)`**

**Joining of array:**

- Joining means putting content of two or more arrays in a single array.

For the array we use `concanicate` function

**Syntax:**

**`np.concanicate((array1,array2.....),axis=0)`**

**Splitting of an array:**

- Splitting is reverse operation of joining
- Splitting breaks one array into multiple
- We use `array-split()` for splitting arrays. we pass it the array we want to split and the number of split.

**Sorting of array:**

- Sorting means putting elements in an ordered sequence.
- The Numpy and array object has a function called `sort()` that will sort the specified array.

**Accessing array elements:****1. Indexing:**

Indexing is used to access the individual elements from an array.

But it can be used to obtain entire row, columns.

**Example:**

```
import numpy as np

al=np.array([1,2,3,4])

print(al[0])
```

**O/P:**

1

**2. Slicing:**

In numpy array,slicing is basically the way to extract a range of elements from an array.

**Example:**

```
import numpy as np

al=np.array([1,2,3,4])

b=al[1:3]

print(b)
```

**O/P:**

[2 3]

**Conclusion:**

We understood and Demonstrated the program using Numpy.

## Experiment No. 11

**Title:-**Program to demonstrate the use of Scipy library.

**Aim:-**To demonstrate the use of Scipy library in Python.

**Objective:-**To provide a comprehensive understanding of Scipy library methods.

**Theory:**

**Scipy:**

Is an open-source scientific library of python that is distributed under a BSD license. It is used to solve the complex scientific and mathematical problems. It is built on top of the Numpy extension which means if we import the Scipy, there is no need to import Numpy.

The Scipy is pronounced as Sigh pi, and it depends on the Numpy, including the Appropriate and Fast N-dimension, array manipulation and Optimization.

**The Scipy Library Supports:**

1. Integration gradient optimization, special functions, ordinary differential equation solves, parallel programming and many more. We can say that Scipy implementation exists in every complex numerical computation.
2. The Scipy is a data-processing and system prototyping environment as similar to MATLAB
3. It is easy to use and provides great flexibility to scientists and engineers.

**Scipy-installation and environment set up:**

```
python-m pip install-- user numpy scipy
```

**Special Function Package:**

Scipy special function includes cubic root, exponential, lambert, permutation and combinations, Gamma, Bessel, hypergeometric, kelvin, beta, parabolic cylinder, Relative error Exponential.



**1. Cubic Root Function:**

Syntax:

`Scipy.special.cbrt(x)`**2. Exponential Function:**`from Scipy.special import exp10``exp=exp10([1,10])`**3. Permutations and Combinations:**Combinations-`Scipy.special.comb(N,K)``from Scipy.Special import comb``com=comb(5,2,exact=false,repitition=True)``print(com)`Permutations- `Scipy.special.perm(N,K)`**4. Log sum Exponential Function:**

Syntax:

`Scipy.special.logsumexp(x)`**5. Bassel Function:**

Syntax:

`Scipy.special.jn()`**6. Inverse Function:**

Syntax:

`Scipy.inalg.inr()`**● Numpy Vs Scipy:**

Numpy and Scipy both are used for mathematical and numerical analysis. Numpy is suitable for basic operations such as sorting, indexing and many more. because it contains array data, where as Scipy consists of all the numeric data.

Package Name	Description
Scipy.io	File input/output
Scipy.special	Special Function
Scipy.linalg	Linear Algebra Operation
Scipy.Stats	Statistics and random numbers
Scipy.integrate	Numerical Integration
Scipy. optimize	Optimization and fit

**Ex:-**

```
import numpy as np
from scipy import stats

# Define the dataset
data = [3, 2, 6, 1, 8, 9, 10, 12, 15]

# Calculate the mean
mean = np.mean(data)

# Calculate the median
median = np.median(data)

# Calculate the mode
mode = stats.mode(data)

# Calculate the range
range_val = np.max(data) - np.min(data)

# Print the results
print("Mean:", mean)
print("Median:", median)
print("Mode:", mode.mode[0])

print("Range:", range_val)
```

**O/P:-**

Mean: 7.333333333333333

Median: 8.0

Mode: 1

Range: 14

**Ex:-**

```
from scipy.special import cbrt
print(cbrt(27))
print(cbrt([-8, -3, 0.125, 1.331]))
```

**O/P:-**

3.0

[-2. -1.44224957 0.5 1.1 ]

**Conclusion:**

We demonstrated programs using Scipy library.

## Experiment No.12

**Title:-**Program to demonstrate Data Processing.

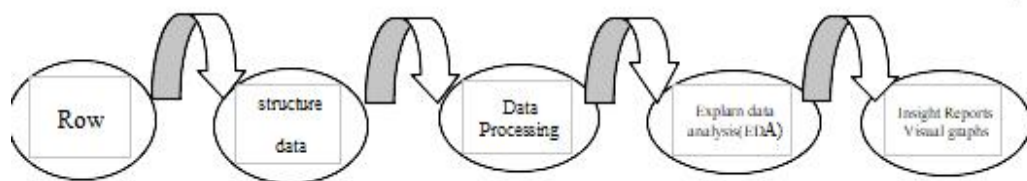
**Aim:-**To demonstrate Data Processing in Python.

**Objective:-**To provide a comprehensive understanding of Data Processing methods.

**Theory:**

- **Data Processing in Python:**

1. Per-Processing refers to the transformation applied to our data before feeding it to the algorithm.
2. Data Processing is a technique that is used to convert the row data into clean data set.
3. In other words, whenever the data is gathered from different sources it is collected in row format which is not feasible for the analysis.



- **Need of Data Processing:**

1. For achieving better results from the applied model in machine learning projects the format of the data has to be in a proper manner.
2. Some specified machine learning model need information in a specified format

3. Another aspect is that the data set should be formatted in such a way that more than one machine learning and deep learning algorithm are executed in one data set and best out of them is chosen.

- **Data processing involves below steps:**

1. **Getting dataset:**

**Ex1:-**

```
import pandas as pd

# Load the dataset from a library

dataset = pd.read_csv('dataset_name.csv')
```

**Ex2:-**

```
import pandas as pd

# Load the dataset from a URL

url = 'https://example.com/dataset.csv'

dataset = pd.read_csv(url)
```

2. **Importing libraries:**

```
import numpy # Importing the NumPy library for numerical computations

import pandas # Importing the Pandas library for data manipulation and analysis

import matplotlib.pyplot as plt # Importing the Matplotlib library for data
visualization

# You can also import specific modules or functions from a library

from sklearn.linear_model import LinearRegression # Importing only the
LinearRegression class from the scikit-learn library
```

**3. Importing datasets:****CSV files with the Pandas library:**

```
import pandas as pd

# Importing a CSV file

data = pd.read_csv('filename.csv')
```

**Excel files with the Pandas library:**

```
import pandas as pd

# Importing an Excel file

data = pd.read_excel('filename.xlsx')
```

**JSON files with the built-in json library:**

```
import json

# Importing a JSON file

with open('filename.json', 'r') as file:

    data = json.load(file)
```

**Image files with the Pillow library (Python Imaging Library):**

```
from PIL import Image

# Importing an image file

image = Image.open('filename.jpg')
```

**Text files with the built-in file handling capabilities:**

```
# Importing a text file

with open('filename.txt', 'r') as file:

    data = file.read()
```

**4. Finding Missing Data:**

```
import pandas as pd

import numpy as np

# Create a sample dataframe with missing values

data = {'Name': ['John', 'Alice', 'Bob', 'Charlie', 'David'],
        'Age': [25, np.nan, 30, 35, np.nan],
        'Gender': ['M', 'F', np.nan, 'M', 'M'],
        'Salary': [50000, 60000, np.nan, 70000, np.nan]}

df = pd.DataFrame(data)

# Check for missing values

print(df.isnull())
```

**OUTPUT:**

	Name	Age	Gender	Salary
0	John	25	M	50000
1	Alice	NaN	F	60000
2	Bob	30	NaN	NaN
3	Charlie	35	M	70000
4	David	NaN	M	NaN

**5. Encoding Categorical Data :**

Encoding categorical data is an important step in preparing a dataset for machine learning algorithms. Categorical data refers to variables that represent discrete values or categories, such as color, gender, or type

**One-Hot Encoding:**

```
import pandas as pd
```

```
# Create a sample dataframe with a categorical column

data = {'Color': ['Red', 'Blue', 'Green', 'Red', 'Blue']}

df = pd.DataFrame(data)

# Perform one-hot encoding

df_encoded = pd.get_dummies(df, columns=['Color'])

print(df_encoded)
```

**OUTPUT:**

	Color_Blue	Color_Green	Color_Red
0	0	0	1
1	1	0	0
2	0	1	0
3	0	0	1
4	1	0	0

**Label Encoding:**

```
import pandas as pd

from sklearn.preprocessing import LabelEncoder

# Create a sample dataframe with a categorical column

data = {'Size': ['Small', 'Large', 'Medium', 'Small', 'Large']}

df = pd.DataFrame(data)

# Perform label encoding

label_encoder = LabelEncoder()

df['Size_encoded'] = label_encoder.fit_transform(df['Size'])
```



```
print(df)
```

**OUTPUT:**

```
   Size Size_encoded
0  Small           2
1  Large           1
2  Medium          0
3  Small           2
4  Large           1
```

**6. Splitting dataset into training and test set:**

```
import pandas as pd

from sklearn.model_selection import train_test_split

# Load your dataset (assuming it's stored in a DataFrame called 'data')
data = pd.read_csv('your_dataset.csv')

# Separate the features (X) and the target variable (y)
X = data.drop('target_variable', axis=1) # Replace 'target_variable' with the actual
column name
y = data['target_variable']

# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

**7. Feature Scaling:**

Feature scaling is a preprocessing technique used to normalize or standardize the numerical features of a dataset. It helps in improving the performance and

stability of machine learning models. There are primarily two common methods of feature scaling: normalization (or min-max scaling) and standardization (or z-score scaling).

**Min-Max Scaling (Normalization):** Min-Max scaling scales the features to a specific range, typically between 0 and 1. The formula to perform min-max scaling on a feature is:

$$X\_scaled = (X - X\_min) / (X\_max - X\_min)$$

**Standard Scaling (Standardization):** Standard scaling transforms the features to have zero mean and unit variance. The formula to perform standard scaling on a feature is:

$$X\_scaled = (X - X\_mean) / X\_std$$

### **Conclusion:**

We understood basic concept of data processing in python.