



Shri Vile Parle Kelavani Mandal's

DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING

(Autonomous College Affiliated to the University of Mumbai)

NAAC Accredited with "A" Grade (CGPA : 3.18)



PYTHON LAB EXPERIMENTS

Name	Junaid Girkar
Sap ID	60004190057
Division	A
Batch	A4
Branch	Computer Engineering
Semester	5
Subject	Programming Laboratory –II (Python)
Subject Code	DJ19CEL505



INDEX

Experiment No.	Experiment Title	Page No.
1	Exploring basics of python like data types (strings, list, array, dictionaries, set, tuples) and control statements.	3
2	Creating functions, classes and objects using python	19
3	Menu driven program for data structure using built in function for link list, stack and queues.	35
4	Demonstrate exception handling.	50
5	Python program to explore different types of Modules	56
6	Demonstrate File handling and Directories a. Python program to append data to existing file and then display the entire file. b. Python program to count number of lines, words and characters in a file. c. Python program to display file available in current directory	69
7	Make use of RE module to perform text processing	75
8	Creating GUI with python containing widgets such as labels, textbox, radio, checkboxes and custom dialog boxes.	84
9	Program to demonstrate CRUD (create, read, update and delete) operations on database (SQLite/MySQL) using python.	93
10	Implementation of simple socket programming for message exchange between server and client	96
11	Make use of advance modules of Python like OpenCV, Matplotlib, NumPy	101
12	Creating web application using Django web framework to demonstrate functionality of user login and registration (also validating user detail using regular expression).	108



EXPERIMENT - 1

GROUP MEMBERS NAME:

Harshal Jain - 60004190041

Junaid Girkar - 60004190057

Khushi Chavan - 60004190061

Megh Dedhia - 60004190067

AIM: Exploring basics of python like data types (strings, list, array, dictionaries, set, tuples) and control statements.

THEORY:

Every value in Python has a datatype. Since everything is an object in Python programming, data types are actually classes and variables are instances (object) of these classes.

There are various data types in Python. Some of the important types are listed below.

Integers: You can use an integer to represent numeric data, and more specifically, whole numbers from negative infinity to infinity, like 4, 5, or -1.

CODE:

```
integer_1 = 100
integer_2 = 50

# various operations on integers
print("TYPE: "+str(type(integer_1)))
print ("Multiplication: ", integer_1*integer_2)
print ("Addition: ",integer_1+integer_2)
print ("Subtraction: ", integer_1-integer_2)
print ("Division: ",integer_1/integer_2)
```

OUTPUT:

```
TYPE: <class 'int'>
Multiplication: 5000
Addition: 150
Subtraction: 50
```



Division: 2.0

Float: "Float" stands for 'floating point number'. You can use it for rational numbers, usually ending with a decimal figure, such as 1.11 or 3.14.

CODE:

```
float_1 = 12.539
float_2 = 6.78

# various operations on floats
print("TYPE: "+str(type(float_1)))
print ("Multiplication: ", float_1*float_2)
print ("Addition: ",float_1+float_2)
print ("Subtraction: ", float_1-float_2)
print ("Division: ",float_1/float_2)
```

OUTPUT:

```
TYPE: <class 'float'>
Multiplication: 85.01442
Addition: 19.319
Subtraction: 5.7589999999999995
Division: 1.849410029498525
```

Boolean: This built-in data type that can take up the values: True and False, which often makes them interchangeable with the integers 1 and 0. Booleans are useful in conditional and comparison expressions, just like in the following examples:

CODE:

```
has_passed = False #Default Value
marks = 80
if (marks > 50):
```



```
# true has been assigned to has_passed (a boolean)
has_passed = True
print("Type: ", str(type(has_passed)))
print (has_passed)
```

OUTPUT:

```
Type: <class 'bool'>
True
```

Strings: Strings are arrays of bytes representing Unicode characters. However, Python does not have a character data type, a single character is simply a string with a length of 1. Square brackets can be used to access elements of the string. Strings in Python can be created using single quotes or double quotes or even triple quotes. individual characters of a String can be accessed by using the method of Indexing. Indexing allows negative address references to access characters from the back of the String, e.g. -1 refers to the last character, -2 refers to the second last character and so on.

CODE:

```
#strings

s = 'Hello world!'

print(type(s))

print(s)

print("s[4] = ", s[4])

print("s[6:11] = ", s[6:11])

# Generates error
# Strings are immutable in Python - TypeError
s[5] = 'd'
```

OUTPUT:



```
<class 'str'>
Hello world!
s[4] = o
s[6:11] = world
```

```
-----
TypeError                                Traceback (most recent call last)
<ipython-input-20-54c802e3dd27> in <module>
    14 # Generates error
    15 # Strings are immutable in Python
--> 16 s[5] = 'd'
```

TypeError: 'str' object does not support item assignment

Lists: Lists in Python are one of the most versatile collection object types available. The other two types are dictionaries and tuples, but they are really more like variations of lists.

- Python lists do the work of most of the collection data structures found in other languages and since they are built-in, you don't have to worry about manually creating them.
- Lists can be used for any type of object, from numbers and strings to more lists.
- They are accessed just like strings (e.g. slicing and concatenation) so they are simple to use and they're variable length, i.e. they grow and shrink automatically as they're used.
- In reality, Python lists are C arrays inside the Python interpreter and act just like an array of pointers.

CODE:

```
#lists
l = [10,20,30,40,50,60,70,80]

print(type(l))

print(l)

print("l[2] = ", l[2])
```



```
print("l[0:3] = ", l[0:3])
```

```
print("l[5:] = ", l[5:])
```

OUTPUT:

```
<class 'list'>
[10, 20, 30, 40, 50, 60, 70, 80]
l[2] = 30
l[0:3] = [10, 20, 30]
l[5:] = [60, 70, 80]
```

Array: An array is a collection of items stored at contiguous memory locations. The idea is to store multiple items of the same type together. This makes it easier to calculate the position of each element by simply adding an offset to a base value, i.e., the memory location of the first element of the array.

A user can treat lists as arrays. However, users cannot constraint the type of elements stored in a list. If you create arrays using the array module, all elements of the array must be of the same type.

Arrays in Python can be created by importing array modules. `array(data_type, value_list)` is used to create an array with data type and value list specified in its arguments.

In-built functionalities of an array:

- `insert()` - Add one or more data elements in any position
- `append()` - Add the value at the end of the array
- Index operator `[]` - Used to access the array items.
- `remove()` - Remove one array item from any position
- `pop()` - Removes one array item from the end
- Slicing operator `[Start Index:End Index]` - This creates a sub-array list from the main array.
- `sort()` - Sorts the list

CODE:

```
# Python program to demonstrate Creation of Array
```



```
# importing "array" for array creations
import array as arr

# creating an array with integer type
a = arr.array('i', [1, 2, 3])

# printing original array
print ("The new created array is : ", end = " ")
for i in range (0, 3):
    print (a[i], end = " ")
print()

# array with float type
b = arr.array('d', [2.5, 3.2, 3.3])

print ("Array before insertion : ", end = " ")
for i in range (0, 3):
    print (b[i], end = " ")
print()

# adding an element using append()
b.append(4.4)

print ("Array after insertion : ", end = " ")
for i in (b):
    print (i, end = " ")
print()

# accessing element of array
print("Access element index 0: ", a[0])

# initializing array with array values and signed integers
arr = arr.array('i', [1, 2, 3, 1, 5, 6, 8, 10, 12, 55, 99, 2])

# printing original array
print ("The new created array is : ", end = "")
for i in range (0, 5):
```




```
print (arr[i], end = " ")
print ("\r")

# using pop() to remove element at 2nd position
print ("The popped element is : ", end = "")
print (arr.pop(2))

# printing array after popping
print ("The array after popping is : ", end = "")
for i in range (0, 4):
    print (arr[i], end = " ")
print ("\r")

# using remove() to remove 1st occurrence of 1
arr.remove(1)

# printing array after removing
print ("The array after removing is : ", end = "")
for i in range (0, 3):
    print (arr[i], end = " ")
```

OUTPUT:

```
The new created array is : 1 2 3
Array before insertion : 2.5 3.2 3.3
Array after insertion : 2.5 3.2 3.3 4.4
Access element index 0: 1
The new created array is : 1 2 3 1 5
The popped element is : 3
The array after popping is : 1 2 1 5
The array after removing is : 2 1 5
```

Dictionary: In python, dictionaries are similar to hash or maps in other languages. It consists of key value pairs. The value can be accessed by a unique key in the dictionary.

- Keys are unique & immutable objects.



Syntax:

```
dictionary = {"key name": value}
```

CODE:

```
#dictionaries

d = {1:'val','key':2}

print(type(d))

print(d)

print("d[1] = ", d[1])

print("d['key'] = ", d['key'])

# Generates error - KeyError
print("d[2] = ", d[2])
```

OUTPUT:

```
<class 'dict'>
{1: 'val', 'key': 2}
d[1] = val
d['key'] = 2
-----
KeyError                                Traceback (most recent call last)
<ipython-input-23-d7fc8db7cd00> in <module>
    12
    13 # Generates error - KeyError
--> 14 print("d[2] = ", d[2])
KeyError: 2
```



Tuple: Python tuples work exactly like Python lists except they are immutable, i.e. they can't be changed in place. They are normally written inside parentheses to distinguish them from lists (which use square brackets), but as you'll see, parentheses aren't always necessary. Since tuples are immutable, their length is fixed. To grow or shrink a tuple, a new tuple must be created.

CODE:

```
#tuples

t = (1,'program', 3+4j)

print(type(t))

print(t)

print("t[1] = ", t[1])

print("t[0:3] = ", t[0:3])

# Generates error
# Tuples are immutable - TypeError
t[0] = 10
```

OUTPUT:

```
<class 'tuple'>
(5, 'program', (1+3j))
t[1] = program
t[0:3] = (5, 'program', (1+3j))

-----
TypeError                                Traceback (most recent call last)
<ipython-input-22-79b203763893> in <module>
    13 # Generates error
    14 # Tuples are immutable - TypeError
--> 15 t[0] = 10
```



TypeError: 'tuple' object does not support item assignment

Sets: Unordered collection of unique objects.

- Set operations such as union (|), intersection(&), difference(-) can be applied on a set.
- Frozen Sets are immutable i.e once created further data can't be added to them
- {} are used to represent a set. Objects placed inside these brackets would be treated as a set.

CODE:

```
#sets

s = {1, 2, 3, 4, 5}

print(type(s))

print(s)

#Generates Error - TypeError

print(s[1])
```

OUTPUT:

```
<class 'set'>
{1, 2, 3, 4, 5}

-----
TypeError                                Traceback (most recent call last)
<ipython-input-24-043a1c74b72c> in <module>
      8
      9 #Generates Error - TypeError
--> 10 print(s[1])
```



TypeError: 'set' object is not subscriptable

If Statement:

CODE:

```
a=False
if a:
    print("Inside if")
else:
    print("In else")
```

OUTPUT:

In else

Elif statement:

The elif keyword is Python's way of saying "if the previous conditions were not true, then try this condition".

CODE:

```
# printing the biggest of 3

a=int(input())
b=int(input())
c=int(input())
```



```
if(a>=b and a>=c):  
    print(a)  
elif(b>=c):  
    print(b)  
else:  
    print(c)
```

OUTPUT:

```
>>10  
>>1  
>>11  
11
```

While Loop:

Syntax :

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

CODE:

```
n=int(input())  
count=1  
while n>=count:  
    print(count)  
    count=count+1;
```

OUTPUT:

```
>>5  
1  
2  
3  
4
```



5

For Loop:

Syntax:

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

CODE:

```
animals = ['cat', 'dog', 'mouse', 'lion', 'tiger']  
for animal in animals:  
    print(animal)
```

OUTPUT:

```
cat  
dog  
mouse  
lion  
tiger
```

Nested Loops:

Python programming language allows one loop inside another loop. Following section shows a few examples to illustrate the concept.

Syntax:

The syntax for a nested for loop statement in Python programming language is as follows:

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

The syntax for a nested while loop statement in Python programming language is as follows:



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

CODE:

```
for i in range(1, 6):  
    str = ""  
    for j in range(i):  
        str += "*"   
    print(str)
```

OUTPUT:

```
*  
**  
***  
****  
*****
```

CODE:

```
#checking prime in a Range  
  
n=int(input())  
k=2  
while k<=n:  
    d=2  
    flag=False  
    while d<k:  
        if (k%d ==0):  
            flag=True  
        d=d+1  
    if not flag:  
        print(k)  
    k=k+1
```

OUTPUT:



```
>>20
```

```
2
```

```
3
```

```
5
```

```
7
```

```
11
```

```
13
```

```
17
```

```
19
```

A final note on loop nesting is that we can put any type of loop inside of any other type of loop. For example a for loop can be inside a while loop or vice versa.

Break Statement:

The break statement is used to terminate the loop containing it, the control of the program will come out of that loop.

CODE:

```
i=1
while i<10:
    if i == 4:
        break
    print(i)
    i=i+1
```

OUTPUT:

```
1
```

```
2
```

```
3
```

Continue Statement:

When the program encounters a continue statement, it will skip the statements which are present after the continue statement inside the loop and proceed with the



next iterations.

CODE:

```
# even which are not divisible by 7
n=int(input())
for i in range(2,n+1,2):
    if i%7==0:
        continue
    print(i)
```

OUTPUT:

```
>>10
2
4
6
8
10
```

Pass Statement:

Pass statement is a null operation, which is used when the statement is required syntactically.

CODE:

```
i=3
if i<7:
    pass
print("end")
```

OUTPUT:

```
end
```

CONCLUSION: We learnt about the different data types available in python and



Shri Vile Parle Kelavani Mandal's

DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING

(Autonomous College Affiliated to the University of Mumbai)

NAAC Accredited with "A" Grade (CGPA : 3.18)



implemented each of them. We also learnt and implemented the different control statements in python. Overall it was an amazing learning experience and we also found out about this new tool Overleaf which is very helpful while working in groups and increases the team efficiency by a wide margin.



EXPERIMENT - 2

GROUP MEMBERS NAME:

Harshal Jain - 60004190041

Junaid Girkar - 60004190057

Khushi Chavan - 60004190061

Megh Dedhia - 60004190067

AIM: Creating functions, classes and objects using python.

FUNCTIONS:

Python Functions is a block of related statements designed to perform a computational, logical, or evaluative task. The idea is to put some commonly or repeatedly done tasks together and make a function so that instead of writing the same code again and again for different inputs, we can do the function calls to reuse code contained in it over and over again.

Functions can be both built-in or user-defined. It helps the program to be concise, non-repetitive, and organized.

SYNTAX:

```
def function_name(parameters):  
    """docstring"""  
    statement(s)  
    return expression
```

Creating a Function

We can create a Python function using the def keyword.

Example: Python Creating Function

CODE:

```
def hello(name):#function def  
    "This function says hello to the person passed in as a parameter"  
    print("Hello, " + name)
```

Calling a Function

After creating a function we can call it by using the name of the function followed by parenthesis containing parameters of that particular function.

Example: Python Calling Function

CODE:



```
hello("Jack")#function call
```

OUTPUT:

Hello, Jack

Arguments of a Function

Arguments are the values passed inside the parentheses of the function. A function can have any number of arguments separated by a comma.

Example: Python Function with arguments

In this example, we will create a simple function to check whether the number passed as an argument to the function is even or odd.

CODE:

```
def num(a):#function def
    #This function passes a number as a parameter and checks whether number is
    odd or even
    if(a%2==0):
        print(a,"is an even number")
    else:
        print(a,"is an odd number")

num(5)#function call
```

OUTPUT:

5 is an odd number

Types of Arguments

Python supports various types of arguments that can be passed at the time of the function call. Let's discuss each type in detail.

Default arguments

A default argument is a parameter that assumes a default value if a value is not provided in the function call for that argument. The following example illustrates Default arguments.

Like C++ default arguments, any number of arguments in a function can have a default value. But once we have a default argument, all the arguments to its right must also have default values.



Keyword arguments

The idea is to allow the caller to specify the argument name with values so that the caller does not need to remember the order of parameters.

CODE:

```
def full_name(fname, lname):#this function is for the full name of a person
    print("My name is : ",fname+" "+lname)
full_name(fname="Jack", lname="Fernandes")
```

OUTPUT:

```
My name is : Jack Fernandes
```

Variable-length arguments

In Python, we can pass a variable number of arguments to a function using special symbols. There are two special symbols:

- *args (Non-Keyword Arguments)
- **kwargs (Keyword Arguments)

Example 1: Variable length non-keywords argument

CODE:

```
def kid(*kids):
    print("The youngest child is " + kids[1])

kid("Jack", "Joy", "Jose")
```

OUTPUT:

```
The youngest child is Joy
```

Example 2: Variable length keyword arguments

CODE:

```
def kid(**kids):
    print("His name is " + kids["fname"])

kid(fname = "Jack", lname = "Fernandes")
```

OUTPUT:



His name is Jack

Docstring

The first string after the function is called the Document string or Docstring in short. This is used to describe the functionality of the function. The use of docstring in functions is optional but it is considered a good practice.

The below syntax can be used to print out the docstring of a function:

SYNTAX:

```
print(function_name.__doc__)
```

Example: Adding Docstring to the function

CODE:

```
def hello(name):#function def
    "This function says hello to the person passed in as a parameter"
    print("Hello, " + name)
    print(hello.__doc__)
```

OUTPUT:

This function says hello to the person passed in as a parameter

The return statement

The function return statement is used to exit from a function and go back to the function caller and return the specified value or data item to the caller.

SYNTAX:

```
return [expression_list]
```

The return statement can consist of a variable, an expression, or a constant which is returned to the end of the function execution. If none of the above is present with the return statement a None object is returned.

Example: Python Function Return Statement

CODE:

```
def square(x):#finds the square of the number
    return x*x
print("Square is ",square(5))
```



OUTPUT:

Square is 25

Is Python Function Pass by Reference or pass by value?

One important thing to note is, in Python every variable name is a reference. When we pass a variable to a function, a new reference to the object is created. Parameter passing in Python is the same as reference passing in Java.

CODE:

Example:

```
my_string = "Python"
def test(my_string):
    my_string = "PythonGuides"
    print("Inside the function:",my_string)
test(my_string)
print("Outside the function:",my_string)
```

OUTPUT:

Inside the function: PythonGuides
Outside the function: Python

Python Function within Functions

A function that is defined inside another function is known as the inner function or nested function. Nested functions are able to access variables of the enclosing scope. Inner functions are used so that they can be protected from everything happening outside the function.

CODE:

```
def f1():
    s = 'I love Python

    def f2():
        print(s)

    f2()

# Driver's code
```




```
f1()
```

OUTPUT:

```
I love Python
```

ANONYMOUS FUNCTION:

Python Lambda Functions are anonymous functions meaning that the function is without a name. As we already know that the `def` keyword is used to define a normal function in Python. Similarly, the `lambda` keyword is used to define an anonymous function in Python.

SYNTAX:

```
lambda arguments: expression
```

lambda arguments: expression

- This function can have any number of arguments but only one expression, which is evaluated and returned.
- One is free to use lambda functions wherever function objects are required.
- You need to keep in your knowledge that lambda functions are syntactically restricted to a single expression.
- It has various uses in particular fields of programming besides other types of expressions in functions.

Example: Lambda Function Example

CODE:

```
double = lambda x: x * 2  
  
print("Double is ",double(4))
```

OUTPUT:

```
Double is 8
```

Difference Between Lambda functions and def defined function

Let's look at this example and try to understand the difference between a normal `def` defined function and lambda function. This is a program that returns the cube of a given value:



Example 1: Python Lambda Function with List Comprehension

CODE:

```
choc=["5star", "silk", "milkybar"]
newchoc=[a for a in choc if "s" in a]
print("Chocolates containing letter s is",newchoc)
```

OUTPUT:

```
Chocolates containing letter s is ['5star', 'silk']
```

Example 2: Python Lambda Function with if-else

CODE:

```
eve_odd = lambda x : "even" if (x%2==0) else "odd"
print("The number is ",eve_odd(3))
```

OUTPUT:

```
The number is odd
```

Example 3: Python Lambda with Multiple statements

CODE:

```
list1=[[2,3,4],[1, 4, 16, 64],[3, 6, 9, 12]]
sort=lambda x:(sorted(y) for y in x)
get_small = lambda x, f : [y[0] for y in f(x)]
smallest=get_small(list1, sort)
print("Smallest numbers in each list are",smallest)
```

OUTPUT:

```
Smallest numbers in each list are [2, 1, 3]
```

Using lambda() Function with filter()

The filter() function in Python takes in a function and a list as arguments. This offers an elegant way to filter out all the elements of a sequence "sequence", for which the function returns True. Here is a small program that returns the even numbers from an input list:

CODE:



```
list1 = [1, 5, 4, 6, 8, 11, 3, 12]
even = list(filter(lambda x: (x%2 == 0), list1)) #only filters even numbers
print("Even numbers in the list are ", even)
```

OUTPUT:

```
Even numbers in the list are [4, 6, 8, 12]
```

Using lambda() Function with map()

The map() function in Python takes in a function and a list as an argument. The function is called with a lambda function and a list and a new list is returned which contains all the lambda modified items returned by that function for each item.

CODE:

```
list1 = [1, 5, 4, 6, 8, 11, 3, 12]
double = list(map(lambda x: (x*2), list1)) #only filters even numbers
print("doubled the numbers in the list are ", double)
```

OUTPUT:

```
doubled the numbers in the list are [2, 10, 8, 12, 16, 22, 6, 24]
```

Using lambda() Function with reduce()

The reduce() function in Python takes in a function and a list as an argument. The function is called with a lambda function and an iterable and a new reduced result is returned. This performs a repetitive operation over the pairs of the iterable. The reduce() function belongs to the functools module.

CODE:

```
from functools import reduce
list1 = [1, 2, 3, 4, 5]
sum = reduce(lambda x, y: x+y, list1)
print("Sum of number in the list is", sum)
```

OUTPUT:

```
Sum of number in the list is 15
```

RECURSION:



The term Recursion can be defined as the process of defining something in terms of itself. In simple words, it is a process in which a function calls itself directly or indirectly.

Advantages of using recursion

- A complicated function can be split down into smaller subproblems utilizing recursion.
- Sequence creation is simpler through recursion than utilizing any nested iteration.
- Recursive functions render the code look simple and effective.

Disadvantages of using recursion

- A lot of memory and time is taken through recursive calls which makes it expensive for use.
- Recursive functions are challenging to debug.
- The reasoning behind recursion can sometimes be tough to think through.

SYNTAX:

```
def func(): <--  
    |  
    | (recursive call)  
    |  
func() ----
```

CODE:

```
def factorial(k):  
    if(k > 0):  
        res=k*factorial(k-1)  
        print(res)  
    else:  
        res=1  
    return res  
  
print("Factorial till 5:")  
factorial(5)
```

OUTPUT:

```
Factorial till 5:  
1
```



2
6
24
120

OBJECTS IN PYTHON:

Python is an object-oriented programming language. Almost everything in Python is an object, with its properties and methods.

A Class is like an object constructor or a "blueprint" for creating objects.

CODE:

```
class Student :  
    pass
```

```
s1 = Student()  
s2 = Student()  
s3 = Student()
```

Class attributes

Class attributes belong to the class itself; they will be shared by all the instances.

Such attributes are defined in the class body parts usually at the top, for legibility.

CODE:

```
class Student :  
    ## Class attributes  
    totalStudents = 20  
    classTeacherName = 'Komal'
```

```
## Class Attributes belong to class  
print(Student.__dict__)
```

OUTPUT:

```
{'__module__': '__main__', 'totalStudents': 20, 'classTeacherName': 'Komal', '__dict__':  
<attribute '__dict__' of 'Student' objects>, '__weakref__': <attribute '__weakref__' of  
'Student' objects>, '__doc__': None}
```

CODE:



```
## All are referring to same class Attribute
print(Student.totalStudents)
print(s1.totalStudents)
print(s2.totalStudents)
```

OUTPUT:

```
20
20
20
```

CODE:

```
## class Attributes does not belong to particular Instance
print(s1.__dict__)
print(s2.__dict__)
```

OUTPUT:

```
{ }
{ }
```

CODE:

```
## Modifying class Attribute
Student.totalStudents = 30
```

Class method

A class method is a method that is bound to a class rather than its object. It doesn't require creation of a class instance.

CODE:

```
class Student :
    ## Class attributes
    __totalStudents = 20
    classTeacherName = 'Komal'
    ## Class Method
    @classmethod
    def getTotalStudents(cls) :
        return Student.__totalStudents
```



```
print(Student.getTotalStudents())
```

OUTPUT:

```
20
```

Instance Attributes

Unlike class attributes, instance attributes are not shared by objects. Every object has its own copy of the instance attribute (In case of class attributes all objects refer to a single copy).

CODE:

```
s1.name = "Mohit"  
s1.age = 20  
print(s1.__dict__)
```

OUTPUT:

```
{'name': 'Mohit', 'age': 20}
```

SYNTAX:

```
<object-name> = <class-name>(<arguments>)
```

The self Parameter

The self parameter is a reference to the current instance of the class, and is used to access variables that belong to the class.

It does not have to be named self, you can call it whatever you like, but it has to be the first parameter of any function in the class:

```
class Person:  
    def __init__(xyz, name, age):  
        xyz.name = name  
        xyz.age = age  
  
    def myfunc(abc):  
        print("Hello my name is " + abc.name)  
  
p1 = Person("Junaid", 20)
```



```
p1.myfunc()
```

OUTPUT:

```
Hello my name is Junaid
```

Modify Object Properties

You can modify properties on objects like this:

```
p1.age = 40
```

Delete Object Properties

You can delete properties on objects or full objects itself by using the del keyword:

```
del p1.age
```

```
del p1
```

Accessing Invalid Instance Attributes will give an error.

CODE:

```
s2 = Student()
s2.rollNumber = 101

print(s2.__dict__)
print(s2.name)
```

OUTPUT:

```
AttributeError                                Traceback (most recent call last)
<ipython-input-12-e81b27c01c56> in <module>()
----> 1 print(s2.name)
AttributeError: 'Student' object has no attribute 'name'
```

Methods:

CODE:



```
print(hasattr(s1, 'name'))
print(hasattr(s2, 'name'))
print(getattr(s1, 'name'))
print(getattr(s2, 'name', 'test'))
```

OUTPUT:

```
True
False
Mohit
test
```

CODE:

```
delattr(s1, 'name')
print(s1.__dict__)
```

OUTPUT:

```
{'age': 20}
```

Instance Method

Instance attributes are those attributes that are not shared by objects. Every object has its own copy of the instance attribute.

CODE:

```
class Student :

    ## Instance methods
    def printHello(self) :
        print("Hello")

    def print(self, str) :
        print(str)

    def printName(self) :
        print(self.name)
```

Conclusion:



Shri Vile Parle Kelavani Mandal's

DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING

(Autonomous College Affiliated to the University of Mumbai)

NAAC Accredited with "A" Grade (CGPA : 3.18)



We learnt about different types of functions, classes and objects that are available in python and implemented each of them in a python script.



EXPERIMENT - 3

Aim: Menu driven program for data structure using built in function for linked list, stack and queues

Theory:

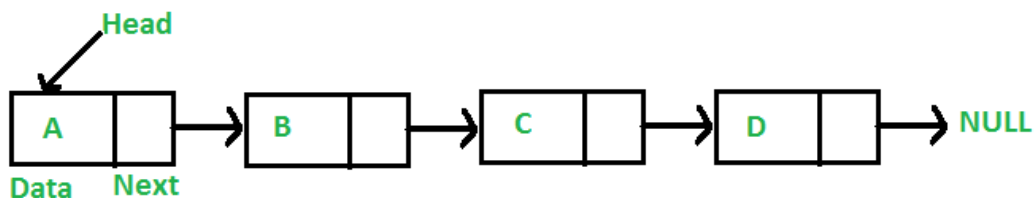
Linked List

Linked List is a sequence of links which contains items. Each link contains a connection to another link. Linked list is the second most-used data structure after array. Following are the important terms to understand the concept of Linked List.

- Link – Each link of a linked list can store a data called an element.
- Next – Each link of a linked list contains a link to the next link called Next.
- LinkedList – A Linked List contains the connection link to the first link called First.

Linked List Representation

Linked list can be visualized as a chain of nodes, where every node points to the next node.



As per the above illustration, following are the important points to be considered.

- Linked List contains a link element called first.
- Each link carries a data field(s) and a link field called next.
- Each link is linked with its next link using its next link.
- Last link carries a link as null to mark the end of the list.

Types of Linked List

Following are the various types of linked list.



- **Simple Linked List** – Item navigation is forward only.
- **Doubly Linked List** – Items can be navigated forward and backward.
- **Circular Linked List** – Last item contains link of the first element as next and the first element has a link to the last element as previous.

Basic Operations

Following are the basic operations supported by a list.

- **Insertion** – Adds an element at the beginning of the list.
- **Deletion** – Deletes an element at the beginning of the list.
- **Display** – Displays the complete list.
- **Search** – Searches an element using the given key.
- **Delete** – Deletes an element using the given key.

Code:

```
class Node:

    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedList:

    def __init__(self):
        self.start = None

    def insert_beg(self, data):
        y = Node(data)
        if self.start is None:
            self.start = y
        else:
            y.next = self.start
            self.start = y
```



```
def insert_after(self, data, val):
    y = Node(data)
    current = self.start
    while current.next is not None and current.data is not val:
        current = current.next
    if current.data is val:
        y.next = current.next
        current.next = y
    else:
        print("Value " + str(val) + " not found.")

def delete(self, data):
    current = self.start
    if self.start.data is data:
        self.start = self.start.next
        del current
        return
    while current.next is not None and current.data is not data:
        preptr = current
        current = current.next
    if current.data is data:
        preptr.next = current.next
        del current
    else:
        print("Value " + str(data) + " not found.")

def display(self):
    print("Linked List: ")
    current=self.start
    while current is not None:
        print(f"Data:{current.data}")
        current=current.next

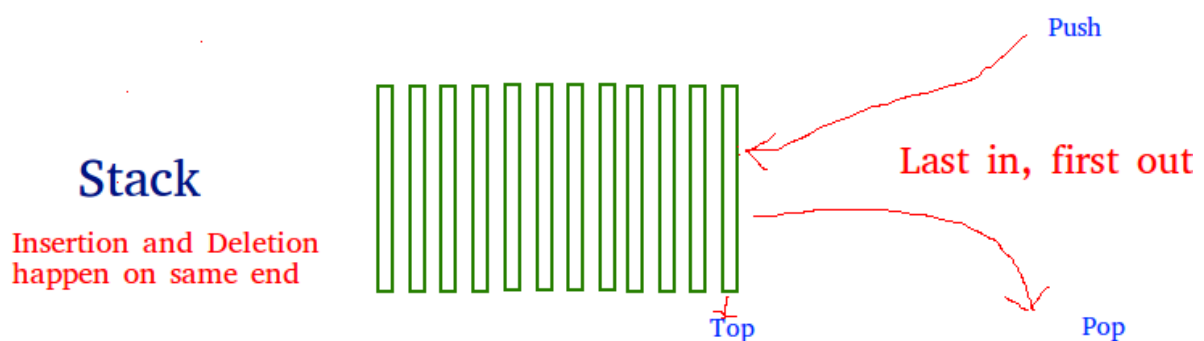
def exec():
    x = LinkedList()
    option = 0
    while option != 5:
        print("\nEnter an option:\n1)Insert at Beg\n2)Insert after a particular value")
        print("3)Delete a particular value\n4)Display\n5)Exit\n")
        option = int(input())
```



```
if option == 1:
    data = int(input("Enter the element: "))
    x.insert_beg(data)
elif option == 2:
    val = int(input("Enter the value after which data is to be inserted: "))
    data = int(input("Enter the element: "))
    x.insert_after(data, val)
elif option == 3:
    data = int(input("Enter the value to be deleted: "))
    x.delete(data)
elif option == 4:
    x.display()
elif option != 5:
    print("Invalid entry. Retry")
```

STACK:

A Stack is a linear data structure that follows the LIFO (Last-In-First-Out) principle. Stack has one end, whereas the Queue has two ends (front and rear). It contains only one pointer top pointer pointing to the topmost element of the stack. Whenever an element is added in the stack, it is added on the top of the stack, and the element can be deleted only from the stack. In other words, a stack can be defined as a container in which insertion and deletion can be done from the one end known as the top of the stack.



Standard Stack Operations

The following are some common operations implemented on the stack:



- **push()**: When we insert an element in a stack then the operation is known as a push. If the stack is full then the overflow condition occurs.
- **pop()**: When we delete an element from the stack, the operation is known as a pop. If the stack is empty means that no element exists in the stack, this state is known as an underflow state.
- **isEmpty()**: It determines whether the stack is empty or not.
- **isFull()**: It determines whether the stack is full or not.'
- **peek()**: It returns the element at the given position.
- **count()**: It returns the total number of elements available in a stack.
- **change()**: It changes the element at the given position.
- **display()**: It prints all the elements available in the stack.

Code:

```
class Node:

    def __init__(self, data):
        self.data = data
        self.next = None

class Stack:

    def __init__(self):
        self.top = None

    def push(self, data):
        y = Node(data)
        if self.top is None:
            self.top = y
        else:
            y.next = self.top
            self.top = y

    def pop(self):
        if self.top is None:
            print("Stack Underflow")
            return -1
        else:
            val = self.top.data
            cur = self.top
            self.top = self.top.next
```



```
del cur
return val

def peek(self):
    if self.top is None:
        print("Empty.")
        return
    else:
        print("Top of Stack: " + str(self.top.data))
        return

def display(self):
    print("Stack: ")
    current = self.top
    if self.top is not None:
        print(self.top.data, end=' ')
    else:
        print("Empty.")
    while current.next is not None:
        print("<- ", end=' ')
        print(current.next.data, end=' ')
        current = current.next

def exec():
    x = Stack()
    option = 0
    while option is not 5:
        print("\nEnter an option:\n1)Push\n2)Pop")
        print("3)Peek\n4)Display\n5)Exit\n")
        option = int(input())
        if option is 1:
            data = int(input("Enter the element: "))
            x.push(data)
        elif option is 2:
            y = x.pop()
            if y is not -1:
                print("Popped Element: " + str(y))
        elif option is 3:
            x.peak()
        elif option is 4:
```



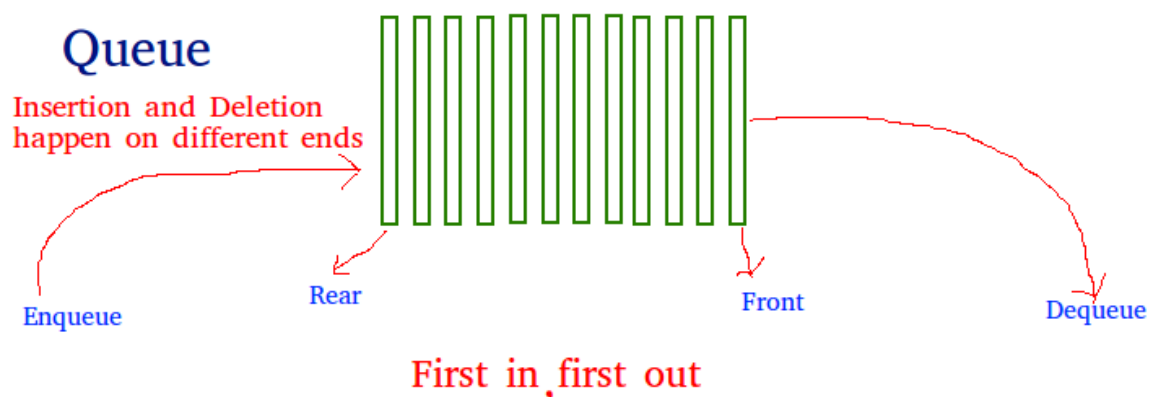
```
x.display()
elif option is not 5:
    print("Invalid entry. Retry")
```

QUEUE:

Queue is the data structure that is similar to the queue in the real world. A queue is a data structure in which whatever comes first will go out first, and it follows the FIFO (First-In-First-Out) policy. Queue can also be defined as the list or collection in which the insertion is done from one end known as the rear end or the tail of the queue, whereas the deletion is done from another end known as the front end or the head of the queue.

The real-world example of a queue is the ticket queue outside a cinema hall, where the person who enters first in the queue gets the ticket first, and the last person enters in the queue gets the ticket at last. Similar approach is followed in the queue in data structure.

The representation of the queue is shown in the below image -



Types of Queue

There are four different types of queue that are listed as follows -

- Simple Queue or Linear Queue
- Circular Queue
- Priority Queue
- Double Ended Queue (or Deque)



Operations performed on queue

The fundamental operations that can be performed on queue are listed as follows -

- **Enqueue:** The Enqueue operation is used to insert the element at the rear end of the queue. It returns void.
- **Dequeue:** It performs the deletion from the front-end of the queue. It also returns the element which has been removed from the front-end. It returns an integer value.
- **Peek:** This is the third operation that returns the element, which is pointed by the front pointer in the queue but does not delete it.
- **Queue overflow (isfull):** It shows the overflow condition when the queue is completely full.
- **Queue underflow (isempty):** It shows the underflow condition when the Queue is empty, i.e., no elements are in the Queue.

Code:

```
class Node:

    def __init__(self, data):
        self.data = data
        self.next = None

class Queue:

    def __init__(self):
        self.front = None
        self.rear = None

    def enqueue(self, data):
        y = Node(data)
        if self.rear is None:
            self.front = y
            self.rear = y
        else:
            self.rear.next = y
            self.rear = y

    def dequeue(self):
        if self.front is None:
            print("Queue Underflow")
            return -1
```



```
elif self.front is self.rear:
    val = self.front.data
    cur = self.front
    self.front = None
    self.rear = None
else:
    val = self.front.data
    cur = self.front
    self.front = self.front.next
del cur
return val

def display(self):
    print("Queue: ")
    current = self.front
    if self.front is not None:
        print(self.front.data, end=' ')
    else:
        print("Empty")
        return
    while current.next is not None:
        print("->", end=' ')
        print(current.next.data, end=' ')
        current = current.next

def exec():
    x = Queue()
    option = 0
    while option is not 4:
        print("\nEnter an option:\n1)Enqueue\n2)Dequeue")
        print("3)Display\n4)Exit\n")
        option = int(input())
        if option is 1:
            data = int(input("Enter the element: "))
            x.enqueue(data)
        elif option is 2:
            y = x.dequeue()
            if y is not -1:
                print("Dequeued Element: " + str(y))
        elif option is 3:
```



```
x.display()  
elif option is not 4:  
    print("Invalid entry. Retry")
```

OUTPUT:

LinkedList

Enter 1 for Linked List

Enter 2 for stack

Enter 3 for Queue

Enter Input: 1

Enter an option:

1)Insert at Beg

2)Insert after a particular value

3)Delete a particular value

4)Display

5)Exit

4

Linked List:

Enter an option:

1)Insert at Beg

2)Insert after a particular value

3)Delete a particular value

4)Display

5)Exit

1

Enter the element: 23

Enter an option:

1)Insert at Beg

2)Insert after a particular value

3)Delete a particular value

4)Display

5)Exit

2



Enter the value after which data is to be inserted: 23

Enter the element: 65

Enter an option:

- 1) Insert at Beg
- 2) Insert after a particular value
- 3) Delete a particular value
- 4) Display
- 5) Exit

4

Linked List:

Data: 23

Data: 65

Enter an option:

- 1) Insert at Beg
- 2) Insert after a particular value
- 3) Delete a particular value
- 4) Display
- 5) Exit

3

Enter the value to be deleted: 23

Enter an option:

- 1) Insert at Beg
- 2) Insert after a particular value
- 3) Delete a particular value
- 4) Display
- 5) Exit

4

Linked List:

Data: 65

Enter an option:

- 1) Insert at Beg
- 2) Insert after a particular value
- 3) Delete a particular value
- 4) Display
- 5) Exit



5

Stack:

Enter 1 for Linked List

Enter 2 for stack

Enter 3 for Queue

Enter Input: 2

Enter an option:

1)Push

2)Pop

3)Peek

4)Display

5)Exit

1

Enter the element: 23

Enter an option:

1)Push

2)Pop

3)Peek

4)Display

5)Exit

1

Enter the element: 45

Enter an option:

1)Push

2)Pop

3)Peek

4)Display

5)Exit

4

Stack:

45 <- 23

Enter an option:



1)Push
2)Pop
3)Peek
4)Display
5)Exit

2

Popped Element: 45

Enter an option:

1)Push
2)Pop
3)Peek
4)Display
5)Exit

3

Top of Stack: 23

Enter an option:

1)Push
2)Pop
3)Peek
4)Display
5)Exit

5

Queue:

Enter 1 for Linked List

Enter 2 for stack

Enter 3 for Queue

Enter Input: 3

Enter an option:

1)Enqueue
2)Dequeue
3)Display
4)Exit

1



Enter the element: 23

Enter an option:

- 1) Enqueue
- 2) Dequeue
- 3) Display
- 4) Exit

1

Enter the element: 46

Enter an option:

- 1) Enqueue
- 2) Dequeue
- 3) Display
- 4) Exit

1

Enter the element: 78

Enter an option:

- 1) Enqueue
- 2) Dequeue
- 3) Display
- 4) Exit

3

Queue:

23 -> 46 -> 78

Enter an option:

- 1) Enqueue
- 2) Dequeue
- 3) Display
- 4) Exit

2

Dequeued Element: 23

Enter an option:

- 1) Enqueue
- 2) Dequeue
- 3) Display



4)Exit

3

Queue:

46 -> 78

Enter an option:

1)Enqueue

2)Dequeue

3)Display

4)Exit

4

Conclusion:

We learnt about the different advanced data structures - Linked list, Stack and Queue. Then we studied every one of them in detail and implemented those structures in a python program.



EXPERIMENT - 4

GROUP MEMBERS NAME:

Harshal Jain - 60004190041

Junaid Girkar - 60004190057

Khushi Chavan - 60004190061

Megh Dedhia - 60004190067

AIM: Explore the exception functionalities in python

THEORY:

The cause of an exception is often external to the program itself. For example, an incorrect input, a malfunctioning IO device etc. Because the program abruptly terminates on encountering an exception, it may cause damage to system resources, such as files. Hence, the exceptions should be properly handled so that an abrupt termination of the program is prevented.

Python uses try and except keywords to handle exceptions. Both keywords are followed by indented blocks.

TRY EXCEPT :

The try: block contains one or more statements which are likely to encounter an exception. If the statements in this block are executed without an exception, the subsequent except: block is skipped.

If the exception does occur, the program flow is transferred to the except: block. The statements in the except: block are meant to handle the cause of the exception appropriately. For example, returning an appropriate error message. Also, You can specify the type of exception after the except keyword.

The following example will throw an exception when we try to divide an integer by a string.

CODE:

```
try:
    a=int(input("Enter a number :"))
    b='0'
    print(a/b)
except:
```



```
print('Some error occurred.')  
print("Out of try except blocks.")
```

OUTPUT:

```
Enter a number :5  
Some error occurred.  
Out of try except blocks.
```

TRY EXCEPT ELSE:

In some situations, you might want to run a certain block of code if the code block inside tries to run without any errors. For these cases, you can use the optional else keyword with the try statement. The example below tries to read and write in the file.

CODE:

```
try:  
    fh = open("testfile", "r")  
    fh.write("This is my test file for exception handling!!")  
except IOError:  
    print("Error: can't find file or read data")  
else:  
    print("Written content in the file successfully")  
    fh.close()
```

OUTPUT:

```
Error: can't find file or read data
```

TRY FINALLY

In Python, keyword 'finally' can also be used along with the try and except clauses. The finally block consists of statements which should be processed regardless of an exception occurring in the try block or not. As a consequence, the error-free try block skips the except clause and enters the finally block before going on to execute the rest of the code. If, however, there's an exception in the try block, the appropriate



except block will be processed, and then statements in the finally block will be processed before proceeding to the rest of the code.

CODE:

```
try:
    fh = open("testfile", "w")
    fh.write("This is my test file for exception handling!!")
finally:
    print("This line will be executed always")
```

OUTPUT:

```
Error: This line will be executed always
```

TRY EXCEPT FINAL :

CODE:

```
try:
    k = 5//0 # raises divide by zero exception.
    print(k)

# handles zerodivision exception
except ZeroDivisionError:
    print("Can't divide by zero")

finally:
    # this block is always executed
    # regardless of exception generation.
    print('This is always executed')
```

OUTPUT:

```
Can't divide by zero
This is always executed
```



MULTIPLE ERRORS WITH ONE EXCEPTION:

A single try block may have multiple except clauses with different exception types in it. If the type of exception doesn't match any of the except blocks, it will remain unhandled and the program will terminate.

CODE:

```
def fun(a):  
    if a < 4:  
  
        # throws ZeroDivisionError for a = 3  
        b = a/(a-3)  
  
        # throws NameError if a >= 4  
        print("Value of b = ", b)  
  
    try:  
        fun(3)  
        fun(5)  
  
    # note that braces () are necessary here for  
    # multiple exceptions  
    except ZeroDivisionError:  
        print("ZeroDivisionError Occurred and Handled")  
    except NameError:  
        print("NameError Occurred and Handled")
```

OUTPUT:

```
ZeroDivisionError Occurred and Handled
```

RAISING CUSTOM EXCEPTIONS:

Python also provides the raise keyword to be used in the context of exception handling. It causes an exception to be generated explicitly. Built-in errors are raised implicitly. However, a built-in or custom exception can be forced during execution.

CODE:



```
try:
    raise NameError("Hi there") # Raise Error
except NameError:
    print ("An exception")
    raise # To determine whether the exception was raised or not
```

OUTPUT:

An exception

```
-----
NameError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_286296\2348914011.py in <module>
      2
      3 try:
----> 4     raise NameError("Hi there") # Raise Error
      5 except NameError:
      6     print ("An exception")
```

NameError: Hi there

ASSERTION ERROR

Assertion is a programming concept used while writing a code where the user declares a condition to be true using assert statement prior to running the module. If the condition is True, the control simply moves to the next line of code. In case if it is False the program stops running and returns AssertionError Exception.

The function of assert statements is the same irrespective of the language in which it is implemented, it is a language-independent concept, only the syntax varies with the programming language.

CODE:

```
try:
    x = 1
    y = 0
    assert y != 0, "Invalid Operation"
    print(x / y)
```



```
# the error_message provided by the user gets printed
except AssertionError as msg:
    print(msg)
```

OUTPUT:

```
Invalid Operation
```

CONCLUSION

We learnt about the different errors in python and the various means of handling these errors using python's inbuilt functionalities. We have also tested out these functionalities by using them in a python script.



EXPERIMENT - 5

AIM: Python program to explore different types of modules

OS MODULE:

The OS module in Python provides functions for interacting with the operating system. OS comes under Python's standard utility modules. This module provides a portable way of using operating system-dependent functionality. The `*os*` and `*os.path*` modules include many functions to interact with the file system.

There are some functions in the OS module which are given below:

- `os.name()`
- `os.mkdir()`
- `os.getcwd()`
- `os.chdir()`
- `os.rmdir()`
- `os.error()`
- `os.popen()`
- `os.close()`
- `os.rename()`
- `os.access()`

CODE:

```
import os

def nameOfOS():
    """Name of the operating system module that it imports"""
    return os.name

def getCurrentDirectory():
    """Function to get current directory"""
    return os.getcwd()

def changeDirectory():
    """Function to change current directory to parent directory"""
```




```
print(getCurrentDirectory())
os.chdir("..")
print(getCurrentDirectory())

def makeNewDirectory():
    """Function to make a new directory."""
    return os.mkdir("d:\\newdir")

def removeDirectory():
    return os.rmdir("d:\\newdir")

def listDirectories():
    return os.listdir()

def getCurrentProcessID():
    """This function returns current process ID or PID, as it is popularly known."""
    return os.getpid()

def getTimeOfLastModificationOfPath():
    """This method returns the time of the last modification of the path."""
    return os.path.getmtime()

def getTimeOfLastAccessOfPath():
    """This method returns the time of the last access of the path."""
    return os.path.getatime()

def determineIfAPathExists():
    """This method returns True for existing paths. It returns False for broken symbolic
links."""
    return os.path.exists("/Files/Engineering")

def joinPaths():
    """Function to join various path components"""
    path = "\\Files\\Engineering\\Python\\Semester - 5"
```



```
return os.path.join(path, "python")
```

```
if __name__ == "__main__":  
    print(nameOfOS())  
    print(getCurrentDirectory())  
    print(determinelfAPathExists())  
    print(joinPaths())  
    print(getCurrentProcessID())
```

OUTPUT:

```
> python os.py  
nt  
C:\Users\junai\Downloads  
False  
\\Files\Engineering\Python\Semester - 5\python  
25456
```

DateTime Module

In Python, date and time are not a data type of their own, but a module named datetime can be imported to work with the date as well as time. Python Datetime module comes built into Python, so there is no need to install it externally.

Python Datetime module supplies classes to work with date and time. These classes provide a number of functions to deal with dates, times and time intervals. Date and datetime are an object in Python, so when you manipulate them, you are actually manipulating objects and not string or timestamps.

The DateTime module is categorized into 6 main classes –

date – An idealized naive date, assuming the current Gregorian calendar always was, and always will be, in effect. Its attributes are year, month and day.

time – An idealized time, independent of any particular day, assuming that every day has exactly 24*60*60 seconds. Its attributes are hour, minute, second, microsecond, and tzinfo.



datetime – Its a combination of date and time along with the attributes year, month, day, hour, minute, second, microsecond, and tzinfo.

timedelta – A duration expressing the difference between two date, time, or datetime instances to microsecond resolution.

tzinfo – It provides time zone information objects.

timezone – A class that implements the tzinfo abstract base class as a fixed offset from the UTC

1) **datetime.date()**

The date class is used to instantiate date objects in Python. When an object of this class is instantiated, it represents a date in the format YYYY-MM-DD. Constructor of this class needs three mandatory arguments year, month and date.

CODE:

```
import datetime
d=datetime.date(2020,11,29)
print(d)
```

OUTPUT:

```
2020-11-29
```

2) **datetime.date.today()**

To return the current local date today() function of date class is used. today() function comes with several attributes (year, month and day). These can be printed individually.

CODE:

```
import datetime
td=datetime.date.today()
print(td)
print(td.year)
print(td.month)
print(td.day)
```

OUTPUT:

```
2021-10-29
2021
10
29
```



3) **weekday(), isoweekday()** :

weekday() returns the day of the week as integer where Monday is 0 and Sunday is 6.

isoweekday() returns the day of the week as integer where Monday is 1 and Sunday is 7.

CODE:

```
import datetime
td=datetime.date.today()
print(td.weekday())
print(td.isoweekday())
```

OUTPUT:

```
4
5
```

4) **datetime.time()** :

The time class creates the time object which represents local time, independent of any day. It can take minutes, hours, seconds, milliseconds as parameters, it works without parameters as well.

CODE:

```
import datetime
t=datetime.time(10,35,55,200)
print(t)
print(t.hour)
print(t.minute)
print(t.second)
```

OUTPUT:

```
10:35:55.000200
10
35
55
```

5) **datetime.datetime()** :

The datetime class contains information on both date and time. Like a date object, datetime assumes the current Gregorian calendar extended in both directions; like a time object, datetime assumes there are exactly 3600*24 seconds every day.

CODE:



```
import datetime
dt=datetime.datetime(2021,10,29,12,30,45,1000)
print(dt)
dt_today=datetime.datetime.today()
dt_now=datetime.datetime.now()
dt_utcnow=datetime.datetime.utcnow()
print(dt_today)
print(dt_now)
print(dt_utcnow)
```

OUTPUT:

```
2021-10-29 12:30:45.001000
2021-10-29 22:51:49.223397
2021-10-29 22:51:49.223397
2021-10-29 17:21:49.223397
```

6) pytz module :

pytz brings the Olson tz database into Python and thus supports almost all time zones. This module serves the date-time conversion functionalities and helps user serving international client's base.

CODE:

```
import pytz
for tz in pytz.all_timezones:
    print(tz)
```

OUTPUT:

```
Africa/Abidjan
Africa/Accra
Africa/Addis_Ababa
Africa/Algiers
.
.
.
US/Mountain
US/Pacific
US/Samoa
UTC
Universal
W-SU
```



WET

Zulu

7) **datetime.datetime.strptime()** :

Returns a DateTime object corresponding to the date string. You cannot create datetime object from every string. The string needs to be in a certain format

CODE:

```
import datetime
dt_str = 'September 24, 2001'
dt=datetime.datetime.strptime(dt_str,'%B %d, %Y')
print(dt)
```

OUTPUT:

```
2001-09-24 00:00:00
```

8) **strftime()**

The strftime() method is defined under classes date, datetime and time. The method creates a formatted string from a given date, datetime or time object.

CODE:

```
import datetime
x = datetime.datetime.today()
print(x.strftime("%B"))
print(x.strftime("%A"))
print(x.strftime("%Y"))
print(x.strftime("%X"))
print(x.strftime("%p"))
```

OUTPUT:

```
October
Friday
2021
22:56:51
PM
```

MATH MODULE:



Python math module is defined as the most famous mathematical functions, which includes trigonometric functions, representation functions, logarithmic functions, etc. Furthermore, it also defines two mathematical constants, i.e., Pie and Euler number, etc.

Pie (π): It is a well-known mathematical constant and defined as the ratio of circumference to the diameter of a circle. Its value is 3.141592653589793.

Euler's number (e): It is defined as the base of the natural logarithmic, and its value is 2.718281828459045.

There are different math modules which are given below:

Method	Description
<code>math.acos()</code>	Returns the arc cosine of a number
<code>math.acosh()</code>	Returns the inverse hyperbolic cosine of a number
<code>math.asin()</code>	Returns the arc sine of a number
<code>math.asinh()</code>	Returns the inverse hyperbolic sine of a number
<code>math.atan()</code>	Returns the arc tangent of a number in radians
<code>math.atan2()</code>	Returns the arc tangent of y/x in radians
<code>math.atanh()</code>	Returns the inverse hyperbolic tangent of a number
<code>math.ceil()</code>	Rounds a number up to the nearest integer
<code>math.comb()</code>	Returns the number of ways to choose k items from n items without repetition and order
<code>math.copysign()</code>	Returns a float consisting of the value of the first parameter and the sign of the second parameter
<code>math.cos()</code>	Returns the cosine of a number



math.cosh()	Returns the hyperbolic cosine of a number
math.degrees()	Converts an angle from radians to degrees
math.dist()	Returns the Euclidean distance between two points (p and q), where p and q are the coordinates of that point
math.erf()	Returns the error function of a number
math.erfc()	Returns the complementary error function of a number
math.exp()	Returns E raised to the power of x
math.expm1()	Returns $E^x - 1$
math.fabs()	Returns the absolute value of a number

CODE:

```
import math
```

```
euler = f"Value of Euler e is: {math.e}"
```

```
pi = f"Value of Pi is: {math.pi}"
```

```
tau = f"Value of Tau is: {math.tau}"
```

```
inf = f"Value of Infinity is: {math.inf}"
```

```
ceil = f"Ceiling value of 5.5 is {math.ceil(5.5)}."
```

```
floor = f"Floor value of 5.5 is {math.floor(5.5)}."
```

```
fact = f"Factorial of 4 is {math.factorial(4)}"
```

```
gcd = f"GCD of 12 and 16 is {math.gcd(12, 16)}"
```

```
fabs = f"Absolute value of -25 is {math.fabs(-25)}"
```

```
fmod = f"When {7} divides {17} the remainder is: {math.fmod(17, 7)}"
```

```
expInt = f"Exponent of {3} is {math.exp(3)}."
```

```
expNegativeInteger = f"Exponent of {-3} is {math.exp(-3)}."
```

```
expFloat = f"Exponent of {2.7} is {math.exp(2.7)}."
```

```
power = f"{3} to the power of {4} is {math.pow(3, 4)}"
```

```
logWithBaseGiven = f"The log value of 2 with base 3 is: {math.log(2, 3)}"
```

```
logWithBaseTwo = f"Log value of 16 with base 2: {math.log2(16)}."
```




```

logWithBaseTen = f"Log value of 10000 with base 10: {math.log10(10000)}."
radianToDegree = f"Corresponding degree value of {math.pi/6} radians is:
{math.degrees(math.pi/6)}"
degreeToRadian = f"Corresponding radian value of 30 radians is: {math.radians(30)}"
sine = f"Sine value of {math.pi/6} is {math.sin(math.pi/6)}"
cosine = f"Cosine value of {math.pi/6} is {math.cos(math.pi/6)}"
tangent = f"Tangent value of {math.pi/6} is {math.tan(math.pi/6)}"

print(f"{euler}\n{pi}\n{tau}\n{inf}\n{ceil}\n{floor}\n{fact}\n{gcd}\n{fabs}\n{fmod}\n{explnt}\n{expNegativeInteger}\n{expFloat}\n{power}\n{logWithBaseGiven}\n{logWithBaseTwo}\n{logWithBaseTen}\n{radianToDegree}\n{degreeToRadian}\n{sine}\n{cosine}\n{tangent}")

```

OUTPUT:

```

> py math.py
Value of Euler e is: 2.718281828459045
Value of Pi is: 3.141592653589793
Value of Tau is: 6.283185307179586
Value of Infinity is: inf
Ceiling value of 5.5 is 6.
Floor value of 5.5 is 5.
Factorial of 4 is 24
GCD of 12 and 16 is 4
Absolute value of -25 is 25.0
When 7 divides 17 the remainder is: 3.0
Exponent of 3 is 20.085536923187668.
Exponent of -3 is 0.049787068367863944.
Exponent of 2.7 is 14.879731724872837.
3 to the power of 4 is 81.0
The log value of 2 with base 3 is: 0.6309297535714574
Log value of 16 with base 2: 4.0.
Log value of 10000 with base 10: 4.0.
Corresponding degree value of 0.5235987755982988 radians is: 29.999999999999996
Corresponding radian value of 30 radians is: 0.5235987755982988
Sine value of 0.5235987755982988 is 0.49999999999999994
Cosine value of 0.5235987755982988 is 0.8660254037844387
Tangent value of 0.5235987755982988 is 0.5773502691896257

```

**RANDOM MODULE:**

Python has a built-in module that you can use to make random numbers.

The random module has a set of methods:

Method	Description
seed()	Initialize the random number generator
getstate()	Returns the current internal state of the random number generator
setstate()	Restores the internal state of the random number generator
getrandbits()	Returns a number representing the random bits
randrange()	Returns a random number between the given range
randint()	Returns a random number between the given range
choice()	Returns a random element from the given sequence
choices()	Returns a list with a random selection from the given sequence
shuffle()	Takes a sequence and returns the sequence in a random order
sample()	Returns a given sample of a sequence
random()	Returns a random float number between 0 and 1
uniform()	Returns a random float number between two given parameters
triangular()	Returns a random float number between two given parameters, you can also set a



	mode parameter to specify the midpoint between the two other parameters
betavariate()	Returns a random float number between 0 and 1 based on the Beta distribution (used in statistics)
expovariate()	Returns a random float number based on the Exponential distribution (used in statistics)
gammavariate()	Returns a random float number based on the Gamma distribution (used in statistics)
gauss()	Returns a random float number based on the Gaussian distribution (used in probability theories)
lognormvariate()	Returns a random float number based on a log-normal distribution (used in probability theories)
normalvariate()	Returns a random float number based on the normal distribution (used in probability theories)

CODE:

```
import random

lst = [1, 2, 3, "Hello", "World", 19.12, -32]
randomValueFromList = f"Random value from list: {random.choice(lst)}"
random.seed(5)
print(random.random())
print(random.random())
randomNumbersInRangeOfIntegers = f"Random numbers in range of integers 5 and 15 is {random.randint(5, 15)}"
print(randomNumbersInRangeOfIntegers)
randomNumberInRangeOfNumbers = f"Random number from 50-100 is: {random.randrange(50,100)}"
print(randomNumberInRangeOfNumbers)
originalList = [1, 2, 3, 4, 5]
original = f"Original list: {originalList}"
```



```
print(original)
shuffled = f"After shuffling: {random.shuffle(originalList)}"
print(shuffled)
randomFloatInRange = f"Random float in the range 13.26 and 19.12 is
{random.uniform(13.26, 19.12)}"
print(randomFloatInRange)
```

OUTPUT:

```
> py random_Module.py
0.6229016948897019
0.7417869892607294
Random numbers in range of integers 5 and 15 is 15
Random number from 50-100 is: 83
Original list: [1, 2, 3, 4, 5]
After shuffling: None
Random float in the range 13.26 and 19.12 is 18.53927688165865
```

Conclusion : Python modules are one of the major reasons for its popularity as they help in writing programs in lesser lines of code. We learnt about the Math module, Random module, OS module and DateTime module which has many functionalities. We then implemented it in a python program.



EXPERIMENT - 6

Aim

Demonstrate File handling and Directories

- a. Python program to append data to existing file and then display the entire file.
- b. Python program to count number of lines, words and characters in a file.
- c. Python program to display file available in current directory

Theory

A. Python program to append data to existing file and then display the entire file. While reading or writing to a file, access mode governs the type of operations possible in the opened file. It refers to how the file will be used once it's 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 written in the file.

a. Append Only ('a'): Open the file for writing. The file is created if it does not exist. The handle is positioned at the end of the file. The data being written will be inserted at the end, after the existing data.

b. Append and Read ('a+'): Open the file for reading and writing. The file is created if it does not exist. The handle is positioned at the end of the file. The data being written will be inserted at the end, after the existing data.

When the file is opened in append mode, the handle is positioned at the end of the file. The data being written will be inserted at the end, after the existing data.

B. Python program to count number of lines, words and characters in a file. Counting the number of characters is important because almost all the text boxes that rely on user input, have a certain limit on the number of characters that can be inserted. For example, the character limit on a Facebook post is 63, 206 characters. Whereas, for a tweet on Twitter the character limit is 140 characters and the character limit is 80 per post for Snapchat. Counting the lines, word, and characters within a text file results in the line count, word count, and character count, which includes spaces. For instance the text file containing "Hello World\nHello Again\nGoodbye" results on lines: 3 words: 5 characters: 29. Call open(file, mode) with the pathname of a file as file and mode as "r" to open the file for reading. Use a for-loop to iterate through the file.

At each iteration, use str.strip(characters) with "\n" as characters to strip it from each line str, and use str.split() to create a list containing all the words from the line str. Also, in each iteration, add 1 to the number of lines, use len(object) with the list containing the word from the line as object to add it to the number of words, and use len(object) with the stripped line as object to add it to the number of characters.



C. Python program to display file available in current directory

The following is a list of some of the important methods/functions in Python with descriptions required to implement the program to list the directories and files in a given folder.

- `str()` – It is used to transform data value(integers, floats, list) into string.
- `abspath()` – It returns the absolute path of the file/directory name passed as an argument.
- `enumerate()` – Returns an enumerate object for the passed iterable that can be used to iterate over the items of iterable with an access to their indexes.
- `list()` – It is used to create a list by using an existing iterable(list, tuple, dictionary, set).
- `listdir()` – It is used to list the directory contents. The path of directory is passed as an argument.
- `isfile()` – It checks whether the passed parameter denotes the path to a file. If yes then returns True otherwise False
- `isdir()` – It checks whether the passed parameter denotes the path to a directory. If yes then returns True otherwise False
- `append()` – It is used to append items on a list.

Code

a. Python program to append data to existing file and then display the entire file.

```
def AppendData(fname):  
    """Python program to append data to existing file and then display the entire file. """  
    """Before appending to "file.txt" its contents of the file were as follows:"""  
    beforeAppending = open(fname)  
    print("Before appending to 'file.txt' its contents were as follows:")  
    print(beforeAppending.read())  
    beforeAppending.close()  
    with open(fname, 'a', encoding='utf-8') as f:  
        f.write("\nThis is the First Line.\n")  
        f.write("This is the Second one\n")  
        f.write("This is the final line.\n")  
        print("***80")  
    """Before appending to "file.txt" its contents were as follows:"""  
    afterAppending = open(fname)  
    print("After appending to file.txt its contents were as follows:")  
    print(afterAppending.read())
```



```
afterAppending.close()
def main():
    AppendData("file.txt")
if __name__ == '__main__':
    main()
```

OUTPUT:
Before app

OUTPUT:

Before appending to 'file.txt' its contents were as follows:

After appending to file.txt' its contents were as follows:

This is the First Line.

This is the Second one

This is the final line.

b. Python program to count number of lines, words and characters in a file.

```
def CountInFile(fname):
    """Python program to count number of lines, words and characters in a file."""
    with open(fname, 'r', encoding='utf-8') as f:
        lineCount = 0
        wordCount = 0
        characterCount = 0
        for line in f:
            line = line.strip("\n")

            words = line.split()
            lineCount += 1
            wordCount += len(words)
```



```
characterCount += len(line)
print(f"In the file {fname} we have:")
print(f"Lines: {lineCount}")
print(f"Words: {wordCount}")
print(f"Characters: {characterCount}")
def main():
    CountInFile("file.txt")

main()
```

OUTPUT:

```
In the file file.txt we have:
Lines: 4
Words: 15
Characters: 68
```

c. Python program to display files available in the current directory.

```
import os

def countDirectories(directoryList):
    """ A function that lists the directories in a folder."""
    print("Directories in the given folder: ")
    for index, dir in enumerate(directoryList):
        print(f"{str(index+1)}. {dir}")

def countFiles(fileList):
    """ A function that lists the files in a folder."""
    print("Files in the given folder: ")
    for index, file in enumerate(fileList):
        print(f"{str(index+1)}. {file}")
```




```
def displayFilesAvailableInCurrentDirectory(path="."):
```

```
    """
```

A function that calls 2 functions to separately listing out directories and files.
It takes a default argument as cwd(.). We can
pass other paths too.

```
    """
```

```
    fileList = list()
```

```
    directoryList = list()
```

```
    try:
```

```
        for f in os.listdir(path):
```

```
            if os.path.isfile(os.path.join(path, f)):
```

```
                fileList.append(f)
```

```
            else:
```

```
                if os.path.isdir(os.path.join(path, f)):
```

```
                    directoryList.append(f)
```

```
    except:
```

```
        print("\nError, please check the path")
```

```
    pass
```

```
    print(f"Given folder: {os.path.abspath(path)}")
```

```
    countFiles(fileList)
```

```
    countDirectories(directoryList)
```

```
def main():
```

```
    displayFilesAvailableInCurrentDirectory()
```

```
main()
```

OUTPUT:

Files in the given folder:

1. appendData.py
2. CountDirectories.py
3. countlines.py
4. file.txt

Directories in the given folder:

1. RandomFolder



2. RandomFolder2

Conclusion

In part a of this experiment, we learned how to append data to a pre-existing file in Python using the append mode. In part b, we learned how to count the number of lines, words and characters in a file in Python by iterating over the file line by line and counting them. In part c, we listed all the files and directories (separately) in a given folder by calling the abspath function to display the absolute path of the chosen directory, the listdir function to list the files and folders in the chosen directory from the in-built os module. We also checked if a particular item was a directory or a file by calling the isdir and isfile functions and then displaying the files and folders separately.



EXPERIMENT - 7

Aim: Make use of RE module to perform text processing

Theory:

A Regular Expressions (RegEx) is a special sequence of characters that uses a search pattern to find a string or set of strings. It can detect the presence or absence of a text by matching with a particular pattern, and also can split a pattern into one or more sub-patterns. Python provides a re module that supports the use of regex in Python. Its primary function is to offer a search, where it takes a regular expression and a string. Here, it either returns the first match or else none.

RegEx Functions

The **re** module offers a set of functions that allows us to search a string for a match:

Function	Description
findall	Returns a list containing all matches
search	Returns a Match object if there is a match anywhere in the string
split	Returns a list where the string has been split at each match
sub	Replaces one or many matches with a string



Metacharacters

Metacharacters are characters with a special meaning:

Character	Description	Example
[]	A set of characters	"[a-m]"
\	Signals a special sequence (can also be used to escape special characters)	"\d"
.	Any character (except newline character)	"he..o"
^	Starts with	"^hello"
\$	Ends with	"planet\$"
*	Zero or more occurrences	"he.*o"
+	One or more occurrences	"he.+o"
?	Zero or one occurrences	"he.?o"



{}	Exactly the specified number of occurrences	"he{2}o"
	Either or	"falls stays"
()	Capture and group	

Special Sequences

A special sequence is a `\` followed by one of the characters in the list below, and has a special meaning:

Character	Description	Example
<code>\A</code>	Returns a match if the specified characters are at the beginning of the string	<code>"\Athe"</code>
<code>\b</code>	Returns a match where the specified characters are at the beginning or at the end of a word (the "r" in the beginning is making sure that the string is being treated as a "raw string")	<code>r"\bain"</code> <code>r"ain\b"</code>
<code>\B</code>	Returns a match where the specified characters are present, but NOT at the beginning (or at the end) of a word	<code>r"\Bain"</code> <code>r"ain\B"</code>



	(the "r" in the beginning is making sure that the string is being treated as a "raw string")	
\d	Returns a match where the string contains digits (numbers from 0-9)	"\d"
\D	Returns a match where the string DOES NOT contain digits	"\D"
\s	Returns a match where the string contains a white space character	"\s"
\S	Returns a match where the string DOES NOT contain a white space character	"\S"
\w	Returns a match where the string contains any word characters (characters from a to Z, digits from 0-9, and the underscore _ character)	"\w"
\W	Returns a match where the string DOES NOT contain any word characters	"\W"
\Z	Returns a match if the specified characters are at the end of the string	"Spain\Z"

Sets



A set is a set of characters inside a pair of square brackets **[]** with a special meaning:

Set	Description
[arn]	Returns a match where one of the specified characters (a , r , or n) are present
[a-n]	Returns a match for any lower case character, alphabetically between a and n
[^arn]	Returns a match for any character EXCEPT a , r , and n
[0123]	Returns a match where any of the specified digits (0 , 1 , 2 , or 3) are present
[0-9]	Returns a match for any digit between 0 and 9
[0-5][0-9]	Returns a match for any two-digit numbers from 00 and 59
[a-zA-Z]	Returns a match for any character alphabetically between a and z , lower case OR upper case
[+]	In sets, + , * , . , , () , \$, {} has no special meaning, so [+] means: return a match for any + character in the string



Code:

```
import re
text_to_search = "abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNPOQRSTUVWXYZ
1234567890
123.456.789
123-456-789
123*456*789
cat
mat
bat
Mr. Smith
Mr David
Mrs. Riya
Mr. Ha HaHa
"

def pattern_finder(pattern, texts):
    for text in texts.split('\n'):
        matches = pattern.finditer(text)
        for match in matches:
            print(match.group())

pattern_finder(re.compile(r'abc'), text_to_search)
pattern_finder(re.compile(r'^[a-zA-Z]'), text_to_search)
pattern_finder(re.compile(r'^[b]at'), text_to_search)
pattern_finder(re.compile(r'\d\d\d'), text_to_search)
pattern_finder(re.compile(r'\d{3}.\d{3}.\d{3}'), text_to_search)
pattern_finder(re.compile(r'Mr\.'), text_to_search)
pattern_finder(re.compile(r'Mr\.[A-Z]\w*'), text_to_search)
pattern_finder(re.compile(r'M(rs|rs)\.[A-Z]\w*'), text_to_search)

# #EMAILS
emails = "khushali@gmail.com
khushali.deulkar@djsce.ac
khushali-123-deulkar@yahoo.com
"
```




```
pattern_finder(re.compile(r'[a-zA-Z0-9-].com'), emails)
pattern_finder(re.compile(r'[a-zA-Z0-9-]+@[a-zA-Z-]+\com'), emails)
pattern_finder(re.compile(r'[a-zA-Z0-9-]+@[a-zA-Z-]+\.(com|ac|net)'), emails)

# URLs
urls = ""
https://www.google.com
https://youtube.com
http://djsce.ac.in
https://www.nasa.gov
""

pattern_finder(re.compile(r'http'), urls)
pattern_finder(re.compile(r'https?'), urls)
pattern_finder(re.compile(r'https?://(www\.)?'), urls)

# Use of Group (index)
pattern=re.compile(r'https?://(www\.)?(\w+)(\.\w+)')
matches = pattern.finditer(urls)
for match in matches:
    print(match.group(0))
    print(match.group(2))
    print(match.group(3))

# Use of Sub
subbed_urls=pattern.sub(r'\1\2\3',urls)
print(subbed_urls)
```

Output:

```
abc
a
A
c
m
b
M
M
M
M
cat
```



Shri Vile Parle Kelavani Mandal's

DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING

(Autonomous College Affiliated to the University of Mumbai)

NAAC Accredited with "A" Grade (CGPA : 3.18)



mat

123

456

789

123

456

789

123

456

789

123

456

789

123.456.789

123-456-789

123*456*789

Mr.

Mr

Mr

Mr.

Mr. Smith

Mr David

Mr. Ha

Mr. Smith

Mr David

Mrs. Riya

Mr. Ha

l.com

o.com

khushali@gmail.com

khushali-123-deulkar@yahoo.com

khushali@gmail.com

khushali.deulkar@djsce.ac

khushali-123-deulkar@yahoo.com

http

http

http

http

https

https

http

https



https://www.
https://
http://
https://www.
https://www.google.com
google
.com
https://youtube.com
youtube
.com
http://djsce.ac
djsce
.ac
https://www.nasa.gov
nasa
.gov

www.google.com
youtube.com
djsce.ac.in
www.nasa.gov

Conclusion:

We learnt about regular expressions and its workings. We then looked at its applications and used it in our python program.



EXPERIMENT - 8

Aim

Creating GUI with python containing widgets such as labels, textbox, radio, checkboxes and custom dialog boxes.

Theory

Python offers multiple options for developing GUI (Graphical User Interface). Out of all the GUI methods, tkinter is the most commonly used method. It is a standard Python interface to the Tk GUI toolkit shipped with Python. Python with tkinter is the fastest and easiest way to create the GUI applications. Creating a GUI using tkinter is an easy task. To create a tkinter app:

- Importing the module – tkinter
- Create the main window (container)
- Add any number of widgets to the main window
- Apply the event Trigger on the widgets.

The basic code used to create the main window of the application is:

`m=tkinter.Tk()` where m is the name of the main window object

`mainloop()`: There is a method known by the name `mainloop()` is used when your application is ready to run. `mainloop()` is an infinite loop used to run the application, wait for an event to occur and process the event as long as the window is not closed.

`m.mainloop()`

Some widgets (like text entry widgets, radio buttons and so on) can be connected



directly to application variables by using special options: variable, textvariable, onvalue, offvalue, and value. This connection works both ways: if the variable changes for any reason, the widget it's connected to will be updated to reflect the new value. These Tkinter control variables are used like regular Python variables to keep certain values. It's not possible to hand over a regular Python variable to a widget through a variable or textvariable option. The only kinds of variables for which this works are variables that are subclassed from a class called Variable, defined in the Tkinter module. They are declared like this:

- `x = StringVar()` # Holds a string; default value ""
- `x = IntVar()` # Holds an integer; default value 0
- `x = DoubleVar()` # Holds a float; default value 0.0
- `x = BooleanVar()` # Holds a boolean, returns 0 for False and 1 for True

Tkinter imports used:

1. Label: It refers to the display box where you can put any text or image which can be updated any time as per the code.

The general syntax is:

```
w=Label(master, option=value)
```

master is the parameter used to represent the parent window.

2. RadioButton: It is used to offer multi-choice option to the user. It offers several options to the user and the user has to choose one option.

The general syntax is:

```
w = RadioButton(master, option=value)
```

3. OptionMenu: OptionMenu is basically a dropdown or popup menu that displays a



group of objects on a click or keyboard event and lets the user select one option at a time.

```
w = OptionMenu(master, option=value)
```

4. Button: To add a button in your application, this widget is used.

The general syntax is:

```
w = Button(master, option=value)
```

5. CheckButton: To select any number of options by displaying a number of options to a user as toggle buttons. The general syntax is:

```
w = CheckButton(master, option=value)
```

6. Text: To edit a multi-line text and format the way it has to be displayed.

The general syntax is:

```
w = Text(master, option=value)
```

Code

```
import tkinter as tk
from tkinter import Label, IntVar, Radiobutton, StringVar, OptionMenu, Button, Checkbutton,
Text
import tkinter.messagebox

def onClick():
    tkinter.messagebox.showinfo("Alert", "Form submitted successfully!")

root = tk.Tk()
root.geometry("500x500")
root.configure(bg="black")
```



```
root.title('Registration form')

registrationFormTitle = Label(
    root,
    text="Registration form",
    width=20,
    font=("bold", 20),
    bg="#0066CC",
).place(anchor=tk.CENTER, relx=0.5, rely=0.1)

nameLabel = Label(
    root,
    text="Name:",
    width=20,
    font=("bold", 10),
    bg="#0066CC"
).place(anchor=tk.CENTER, relx=0.3, rely=0.2)

nameEntry = Text(
    root,
    height=1.45,
    width=25,
    bg="#0066CC"
).place(anchor=tk.CENTER, relx=0.7, rely=0.2)

emailLabel = Label(
    root,
    text="Email",
    width=20,
    font=("bold", 10),
    bg="#0066CC"
).place(anchor=tk.CENTER, relx=0.3, rely=0.3)

emailEntry = Text(
    root,
    height=1.45,
    width=25,
    bg="#0066CC"
).place(anchor=tk.CENTER, relx=0.7, rely=0.3)

genderLabel = Label(
    root,
```



```
text="Gender",
width=20,
font=("bold", 10),
bg="#0066CC")
).place(anchor=tk.CENTER, relx=0.3, rely=0.4)

genderVar = IntVar()
maleButton = Radiobutton(
    root,
    text="Male",
    padx=9,
    variable=genderVar,
    value=1,
    bg="#0066CC")
).place(anchor=tk.CENTER, relx=0.6, rely=0.4)

femaleButton = Radiobutton(
    root,
    text="Female",
    padx=9,
    variable=genderVar,
    value=2,
    bg="#0066CC")
).place(anchor=tk.CENTER, relx=0.8, rely=0.4)

countryLabel = Label(
    root,
    text="Nationality",
    width=20,
    font=("bold", 10),
    bg="#0066CC")
).place(anchor=tk.CENTER, relx=0.3, rely=0.5)

listOfCountries = ['India', 'US', 'UK', 'Germany', 'Austria',
                   'Switzerland', 'Argentina', 'Egypt', 'Indonesia', 'Turkey', 'New Zealand']
c = StringVar()
dropList = OptionMenu(root, c, *listOfCountries)
dropList.config(width=20, bg="#0066CC")
c.set('Select your nationality')
dropList.place(anchor=tk.CENTER, relx=0.7, rely=0.5)

languageLabel = Label(
```




```
root,
text="Language",
width=20,
font=('bold', 10),
bg="#0066CC"
).place(anchor=tk.CENTER, relx=0.3, rely=0.6)

englishVar = IntVar()
englishButton = Checkbutton(
    root,
    text="English",
    variable=englishVar,
    bg="#0066CC"
).place(anchor=tk.CENTER, relx=0.6, rely=0.6)

germanVar = IntVar()
germanButton = Checkbutton(
    root,
    text="German",
    variable=germanVar,
    bg="#0066CC"
).place(anchor=tk.CENTER, relx=0.8, rely=0.6)

submitButton = Button(
    root,
    text='Submit',
    width=20,
    bg="#0066CC",
    command=onClick
).place(anchor=tk.CENTER, relx=0.5, rely=0.7)
root.mainloop()
```

Output

Registration form:



Shri Vile Parle Kelavani Mandal's

DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING

(Autonomous College Affiliated to the University of Mumbai)

NAAC Accredited with "A" Grade (CGPA : 3.18)



Registration form

Registration form

Name:

Email:

Gender: ☒ Male ☐ Female

Nationality:

Language: ☐ English ☐ German

A sample filled registration form:



Registration form

Registration form

Name: Junaaid Girkar

Email: junaaidgirkar@gmail.com

Gender: ☒ Male ☐ Female

Nationality: India

Language: ☒ English ☐ German

Submit

Alert (pop-up) on submission of form:

Registration form

Registration form

Name: Junaaid Girkar

Email: junaaidgirkar@gmail.com

Gender: ☒ Male ☐ Female

Nationality:

Language:

Submit

Alert

Form submitted successfully!

OK



Conclusion

Tkinter is a Python library for developing GUI applications. It is built in Python. **Tkinter** provides us with a variety of widgets such as buttons, menus and various kinds of entry fields and display areas which we can use to build our interface. In this experiment, we created a registration form using tkinter Label, Radiobutton, OptionMenu, Button, Checkbutton and Text widgets.



EXPERIMENT - 9

Here we will discuss all the CRUD operations on the SQLite3 database using Python.
CRUD contains four major operations –

C	→	Create
R	→	Read
U	→	Update
D	→	Delete

Code:

```
# Import module
import sqlite3

# Connecting to sqlite
conn = sqlite3.connect('hello.db')

# Creating a cursor object using
# the cursor() method
mycursor = conn.cursor()

mycursor.execute("create table student (stud_id INT(6) PRIMARY KEY, name
VARCHAR(20), marks INT);")

for x in mycursor:
    print(x)
mycursor.execute("alter table student add column division VARCHAR(1);")
print('(attribute, type, null, key, default, extra)')

for x in mycursor:
    print(x)
mycursor.execute("insert into student (stud_id, name, marks, division) values
(1,'Harry',98,'A');")
```



```

mycursor.execute("insert into student (stud_id, name, marks, division) values
(2,'James',37,'B');")
mycursor.execute("insert into student (stud_id, name, marks, division) values
(3,'Draco',92,'C');")
conn.commit()
mycursor.execute("select * from student;")
myresult = mycursor.fetchall()
for x in myresult:
    print(f'{x[0]}\t{x[1]}\t{x[2]}\t{x[3]}')
mycursor.execute("update student set marks = 99 where name = 'Harry';")
conn.commit()
print(mycursor.rowcount, "record(s) affected")
mycursor.execute("select * from student;")
myresult = mycursor.fetchall()
for x in myresult:
    print(f'{x[0]}\t{x[1]}\t{x[2]}\t{x[3]}')
mycursor.execute("delete from student where division = 'B';")
conn.commit()
print(mycursor.rowcount, "record(s) deleted")
mycursor.execute("select * from student;")
myresult = mycursor.fetchall()
for x in myresult:
    print(f'{x[0]}\t{x[1]}\t{x[2]}\t{x[3]}')
mycursor.execute("drop table if exists student;")
for x in mycursor:
    print(x)

```

Output:

(attribute, type, null, key, default, extra)

```

1  Harry  98  A
2  James  37  B
3  Draco  92  C
1 record(s) affected
1  Harry  99  A
2  James  37  B
3  Draco  92  C
1 record(s) deleted
1  Harry  99  A
3  Draco  92  C

```



DB Browser for SQLite - C:\Users\junai\OneDrive - Shri Vile Parle Kelavani Mandal\Desktop\DJSC\SEM 5\Python\Experime...

File Edit View Tools Help

New Database Open Database Write Changes Revert Changes Open Project Save Project Attach Database

Database Structure Browse Data Edit Pragma Execute SQL

Table: student

	stud_id	name	marks	division
1	1	Harry	99	A
2	3	Draco	92	C

Filter Filter Filter Filter

1 1

Type of data currently in cell: Text / Numeric
1 character(s)

Apply

Remote

Identity Select an identity to connect

DBHub.io Local Current Database

Name	Last modified	Size
------	---------------	------

SQL Log Plot DB Schema Remote

UTF-8

Conclusion:

We learnt about executing SQL commands in python and implemented a python program implementing CRUD functionality.



EXPERIMENT - 10

Aim: Implementation of Simple socket programming for message exchange between server and client.

Theory:

Sockets:

Sockets are the backbone of networking. They make the transfer of information possible between two different programs or devices. For example, when you open up your browser, you as a client are creating a connection to the server for the transfer of information.

In general terms, sockets are interior endpoints built for sending and receiving data. A single network will have two sockets, one for each communicating device or program. These sockets are a combination of an IP address and a Port. A single device can have 'n' number of sockets based on the port number that is being used. Different ports are available for different types of protocols. Take a look at the following image for more about some of the common port numbers and the related protocols:

Socket Programming in Python:

To achieve Socket Programming in Python, you will need to import the **socket** module or [framework](#). This module consists of built-in methods that are required for creating sockets and help them associate with each other.

Some of the important methods are as follows:

Methods	Description
<code>socket.socket()</code>	used to create sockets (required on both server as well as client ends to create sockets)



<code>socket.accept()</code>	used to accept a connection. It returns a pair of values (conn, address) where conn is a new socket object for sending or receiving data and address is the address of the socket present at the other end of the connection
<code>socket.bind()</code>	used to bind to the address that is specified as a parameter
<code>socket.close()</code>	used to mark the socket as closed
<code>socket.connect()</code>	used to connect to a remote address specified as the parameter
<code>socket.listen()</code>	enables the server to accept connections

Server

A server is either a program, a computer, or a device that is devoted to managing network resources. Servers can either be on the same device or computer or locally connected to other devices and computers or even remote. There are various types of servers such as database servers, network servers, print servers, etc.

Servers commonly make use of methods like `socket.socket()`, `socket.bind()`, `socket.listen()`, etc to establish a connection and bind to the clients.

AF_INET refers to Address from the Internet and it requires a pair of (host, port) where the host can either be a URL of some particular website or its address and the port number is an integer. SOCK_STREAM is used to create TCP Protocols.

The `bind()` method accepts two parameters as a tuple (host, port). However, it's better to use 4-digit port numbers as the lower ones are usually occupied. The



listen() method allows the server to accept connections. Here, 5 is the queue for multiple connections that come up simultaneously. The minimum value that can be specified here is 0 (If you give a lesser value, it's changed to 0). In case no parameter is specified, it takes a default suitable one.

The [while loop](#) allows accepting connections forever. 'clt' and 'adr' are the client object and address. The print statement just prints out the address and the port number of the client socket. Finally, clt.send is used to send the data in bytes.

Client

A client is either a computer or software that receives information or services from the server. In a client-server module, clients requests for services from servers. The best example is a web browser such as Google Chrome, Firefox, etc. These web browsers request web servers for the required web pages and services as directed by the user. Other examples include online games, online chats, etc.

gethostname is used when client and server are on on the same computer. (LAN – localip / WAN – publicip)

Here, the client wants to receive some information from the server and for this, you need to use the recv() method and the information is stored in another variable msg. Just keep in mind that the information being passed will be in bytes and in the client in the above program can receive up to 1024 bytes (buffer size) in a single transfer. It can be specified to any amount depending on the amount of information being transferred.

Finally, the message being transferred should be decoded and printed.

Code (Server Side):

```
import socket

# next create a socket object
s = socket.socket()
print ("Socket successfully created")

# reserve a port on your computer in our
# case it is 999 but it can be anything
port = 999
```



```
# Next bind to the port
s.bind(('', port))
print ("socket binded to %s" %(port))

# put the socket into listening mode
s.listen(5)
print ("socket is listening")

# a forever loop until we interrupt it or
# an error occurs
while True:

# Establish connection with client.
c, addr = s.accept()
print ('Got connection from', addr )

# send a thank you message to the client. encoding to send byte type.
c.send('Thank you for connecting'.encode())

# Close the connection with the client
c.close()

# Breaking once connection closed
break
```

Code (Client Side):

```
# Import socket module
import socket

# Create a socket object
s = socket.socket()

# Define the port on which you want to connect
```



```
port = 999

# connect to the server on local computer
s.connect(('127.0.0.1', port))

# receive data from the server and decoding to get the string.
print (s.recv(1024).decode())
# close the connection
s.close()
```

Output (Server Side)

```
> py server.py
Socket successfully created
socket binded to 999
socket is listening
Got connection from ('127.0.0.1', 60394)
```

Output (client side)

```
> py client.py
Thank you for connecting
```

Conclusion: There are multiple built in function in python on socket programming which can faster the speed and reduce the complexity of coding. Hence complex Socket programming can also be done using python easily.



EXPERIMENT - 11

AIM: Make use of advance modules of Python like OpenCV, Matplotlib and Numpy

Numpy:

NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays.

It is the fundamental package for scientific computing with Python. It contains various features including these important ones:

- A powerful N-dimensional array object
- Sophisticated (broadcasting) functions
- Tools for integrating C/C++ and Fortran code
- Useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data.

Arbitrary data-types can be defined using Numpy which allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

Code:

```
import numpy as np
```

```
np.zeros((2,3))
```

```
array([[0., 0., 0.],  
       [0., 0., 0.]])
```

```
np.ones((2,3))
```



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

```
np.linspace(0, 50, 11)
```

```
array([ 0.,  5., 10., 15., 20., 25., 30., 35., 40., 45., 50.])
```

```
np.arange(12).reshape(4, 3)
```

```
array([[ 0,  1,  2],  
       [ 3,  4,  5],  
       [ 6,  7,  8],  
       [ 9, 10, 11]])
```

```
np.sin(np.pi/2)
```

```
1.0
```

```
np.cos(np.pi/2)
```

```
6.123233995736766e-17
```

```
a = np.array([[1,4,5],[8,2,4]])  
b = np.array([[9,6,7],[0,3,5]])
```

```
np.multiply(a,b) # element wise
```

```
array([[ 9, 24, 35],
```



```
[ 0, 6, 20]])
```

```
np.matmul(a,b.transpose())    # matrix multiplication
```

```
array([[ 68, 37],  
       [112, 26]])
```

PANDAS

Pandas is an open-source library that is built on top of NumPy library. It is a Python package that offers various data structures and operations for manipulating numerical data and time series. It is mainly popular for importing and analyzing data much easier. Pandas is fast and it has high-performance & productivity for users.

Code:

```
import pandas as pd
```

```
data = [[23, 9.5],  
        [23, 26.5],  
        [27, 7.8],  
        [27, 17.8],  
        [39, 31.4],  
        [41, 25.9],  
        [47, 27.4],  
        [49, 27.2],  
        [50, 31.2],  
        [52, 34.6],  
        [54, 42.5],  
        [54, 28.8],  
        [56, 33.4],  
        [57, 30.2],  
        [58, 34.1],  
        [58, 32.9],  
        [60, 41.2],
```



```
[61, 35.7]]
```

```
df = pd.DataFrame(data)
df.columns = ['age', '%fat']
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 18 entries, 0 to 17
```

```
Data columns (total 2 columns):
```

```
#   Column  Non-Null Count  Dtype
```

```
---  ---
```

```
0   age    18 non-null    int64
```

```
1   %fat   18 non-null    float64
```

```
dtypes: float64(1), int64(1)
```

```
memory usage: 416.0 bytes
```

```
df.head()
```

	age	%fat
0	23	9.5
1	23	26.5
2	27	7.8
3	27	17.8
4	39	31.4

```
df.describe()
```

	age	%fat
count	18.000000	18.000000
mean	46.444444	28.783333
std	13.218624	9.254395
min	23.000000	7.800000



25%	39.500000	26.675000
50%	51.000000	30.700000
75%	56.750000	33.925000
max	61.000000	42.500000

MATPLOTLIB

Matplotlib is an amazing visualization library in Python for 2D plots of arrays. Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack. It was introduced by John Hunter in the year 2002.

One of the greatest benefits of visualization is that it allows us visual access to huge amounts of data in easily digestible visuals. Matplotlib consists of several plots like line, bar, scatter, histogram etc.

Code:

```
%matplotlib inline
```

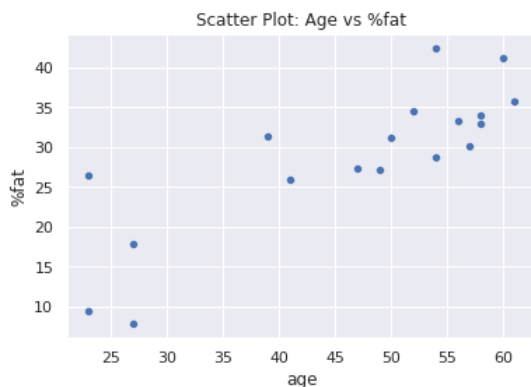
```
import seaborn as sns
```

```
import matplotlib.pyplot as plt
```

```
sns.set()
```

```
sns.scatterplot(x=df['age'], y=df['%fat'])
```

```
plt.title('Scatter Plot: Age vs %fat')
```



OpenCV

OpenCV is a huge open-source library for computer vision, machine learning, and image processing. OpenCV supports a wide variety of programming languages like Python, C++, Java, etc. It can process images and videos to identify objects, faces, or even the handwriting of a human. When it is integrated with various libraries, such as Numpy which is a highly optimized library for numerical operations, then the number of weapons increases in your Arsenal i.e whatever operations one can do in Numpy can be combined with OpenCV.

Code:

```
import cv2
```

```
img = np.ones([1000,2700,3], dtype=np.uint8)*255
```

```
# Letter E
```

```
cv2.line(img, (200, 250), (500, 250), (255, 0, 0), 20)
```

```
cv2.line(img, (200, 250), (200, 750), (255, 0, 0), 20)
```

```
cv2.line(img, (200, 750), (500, 750), (255, 0, 0), 20)
```

```
cv2.line(img, (200, 500), (500, 500), (255, 0, 0), 20)
```

```
# Letter X
```

```
cv2.line(img, (700, 250), (1000, 750), (255, 0, 0), 20)
```



```
cv2.line(img, (1000, 250), (700, 750), (255, 0, 0), 20)
```

```
# Letter P
```

```
cv2.line(img, (1200, 250), (1500, 250), (255, 0, 0), 20)
```

```
cv2.line(img, (1200, 250), (1200, 750), (255, 0, 0), 20)
```

```
cv2.line(img, (1200, 500), (1500, 500), (255, 0, 0), 20)
```

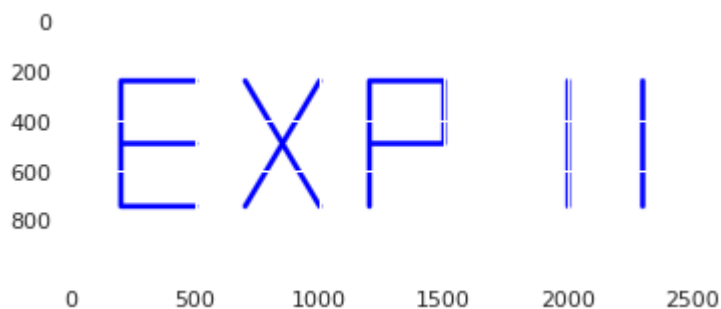
```
cv2.line(img, (1500, 250), (1500, 500), (255, 0, 0), 20)
```

```
# Number 1
```

```
cv2.line(img, (2000, 750), (2000, 250), (255, 0, 0), 20)
```

```
cv2.line(img, (2300, 750), (2300, 250), (255, 0, 0), 20)
```

```
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
```



Conclusion: We learnt about different python modules like Numpy, Pandas, OpenCV and Matplotlib. We then implemented these modules in a python program.



EXPERIMENT - 12

Github Link: <https://github.com/junaidgirkar/Django-Authentication>

THEORY:

Django is a high-level Python web framework that enables rapid development of secure and maintainable websites. Built by experienced developers, Django takes care of much of the hassle of web development, so you can focus on writing your app without needing to reinvent the wheel. It is free and open source, has a thriving and active community, great documentation, and many options for free and paid-for support.

Django helps you write software that is:

Complete

Django follows the "Batteries included" philosophy and provides almost everything developers might want to do "out of the box". Because everything you need is part of the one "product", it all works seamlessly together, follows consistent design principles, and has extensive and up-to-date documentation.

Versatile

Django can be (and has been) used to build almost any type of website – from content management systems and wikis, through to social networks and news sites. It can work with any client-side framework, and can deliver content in almost any format (including HTML, RSS feeds, JSON, XML, etc). The site you are currently reading is built with Django!

Internally, while it provides choices for almost any functionality you might want (e.g. several popular databases, templating engines, etc.), it can also be extended to use other components if needed.

Secure

Django helps developers avoid many common security mistakes by providing a framework that has been engineered to "do the right things" to protect the website automatically. For example, Django provides a secure way to manage user accounts and passwords, avoiding common mistakes like putting session information in cookies where it is vulnerable (instead cookies just contain a key, and the actual data is stored in the database) or directly storing passwords rather than a password hash.

A password hash is a fixed-length value created by sending the password through a cryptographic hash function. Django can check if an entered password is correct by running it through the hash function and comparing the output to the stored hash value. However



due to the "one-way" nature of the function, even if a stored hash value is compromised it is hard for an attacker to work out the original password.

Django enables protection against many vulnerabilities by default, including SQL injection, cross-site scripting, cross-site request forgery and clickjacking (see Website security for more details of such attacks).

Scalable

Django uses a component-based "shared-nothing" architecture (each part of the architecture is independent of the others, and can hence be replaced or changed if needed). Having a clear separation between the different parts means that it can scale for increased traffic by adding hardware at any level: caching servers, database servers, or application servers. Some of the busiest sites have successfully scaled Django to meet their demands (e.g. Instagram and Disqus, to name just two).

Maintainable

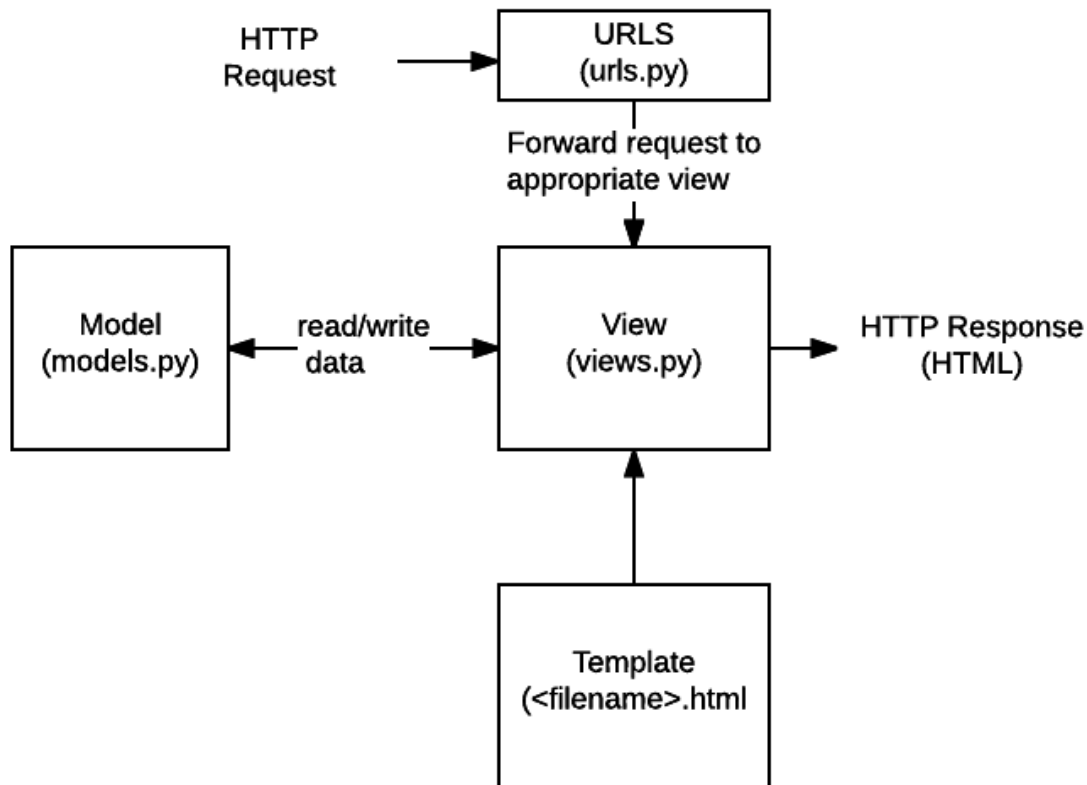
Django code is written using design principles and patterns that encourage the creation of maintainable and reusable code. In particular, it makes use of the Don't Repeat Yourself (DRY) principle so there is no unnecessary duplication, reducing the amount of code. Django also promotes the grouping of related functionality into reusable "applications" and, at a lower level, groups related code into modules (along the lines of the Model View Controller (MVC) pattern).

Portable

Django is written in Python, which runs on many platforms. That means that you are not tied to any particular server platform, and can run your applications on many flavours of Linux, Windows, and Mac OS X. Furthermore, Django is well-supported by many web hosting providers, who often provide specific infrastructure and documentation for hosting Django sites.

in a traditional data-driven website, a web application waits for HTTP requests from the web browser (or other client). When a request is received the application works out what is needed based on the URL and possibly information in POST data or GET data. Depending on what is required it may then read or write information from a database or perform other tasks required to satisfy the request. The application will then return a response to the web browser, often dynamically creating an HTML page for the browser to display by inserting the retrieved data into placeholders in an HTML template.

Django web applications typically group the code that handles each of these steps into separate files:



- **URLs:** While it is possible to process requests from every single URL via a single function, it is much more maintainable to write a separate view function to handle each resource. A URL mapper is used to redirect HTTP requests to the appropriate view based on the request URL. The URL mapper can also match particular patterns of strings or digits that appear in a URL and pass these to a view function as data.
- **View:** A view is a request handler function, which receives HTTP requests and returns HTTP responses. Views access the data needed to satisfy requests via *models*, and delegate the formatting of the response to *templates*.
- **Models:** Models are Python objects that define the structure of an application's data, and provide mechanisms to manage (add, modify, delete) and query records in the database.
- **Templates:** A template is a text file defining the structure or layout of a file (such as an HTML page), with placeholders used to represent actual content. A *view* can dynamically create an HTML page using an HTML template, populating it with data from a *model*. A template can be used to define the structure of any type of file; it doesn't have to be HTML!



Just a few of the other things provided by Django include:

- **Forms:** HTML Forms are used to collect user data for processing on the server. Django simplifies form creation, validation, and processing.
- **User authentication and permissions:** Django includes a robust user authentication and permission system that has been built with security in mind.
- **Caching:** Creating content dynamically is much more computationally intensive (and slow) than serving static content. Django provides flexible caching so that you can store all or part of a rendered page so that it doesn't get re-rendered except when necessary.
- **Administration site:** The Django administration site is included by default when you create an app using the basic skeleton. It makes it trivially easy to provide an admin page for site administrators to create, edit, and view any data models in your site.
- **Serialising data:** Django makes it easy to serialise and serve your data as XML or JSON. This can be useful when creating a web service (a website that purely serves data to be consumed by other applications or sites, and doesn't display anything itself), or when creating a website in which the client-side code handles all the rendering of data.

Implementing User Authentication in a Django Project:

CODE:

models.py

```
from django.db import models
from django.contrib.auth.models import AbstractBaseUser, BaseUserManager
from django.core.validators import RegexValidator

class UserManager(BaseUserManager):
```



```
def create_user(self, email, first_name, last_name, sap_id, password=None, **kwargs):
    if not email:
        raise ValueError("Users must have an email address")
    user = self.model(
        email=self.normalize_email(email),
        first_name=first_name,
        last_name=last_name,
        sap_id=sap_id,
        **kwargs
    )
    user.set_password(password)
    user.save(using=self._db)
    return user
```

```
def create_superuser(self, email, first_name, last_name, password):
    user = self.create_user(email, first_name, last_name, password)
    user.staff = True
    user.admin = True
    user.save(using=self._db)
    return user
```

```
sap_regex = RegexValidator(regex=r"^\d{10,12}$", message="SAP ID must be valid")
```

```
class User(AbstractBaseUser):
    email = models.EmailField(max_length=255, unique=True, default="")
    first_name = models.CharField(max_length=64, default="")
    last_name = models.CharField(max_length=64, default="")
    sap_id = models.CharField(validators=[sap_regex], max_length=12, default="")

    active = models.BooleanField(default=True)
    staff = models.BooleanField(default=False)
    admin = models.BooleanField(default=False)

    objects = UserManager()
    USERNAME_FIELD = 'email'
    REQUIRED_FIELDS = ['first_name', 'last_name']
```

```
def __str__(self):
    return self.email
def has_perm(self, perm, obj=None):
    return True
def has_module_perms(self, app_label):
```




```
return True
@property
def is_staff(self):
    return self.staff
@property
def is_admin(self):
    return self.admin
@property
def is_active(self):
    return self.active
```

views.py

```
from django.shortcuts import render, redirect
from django.contrib import messages
from django.contrib.auth import authenticate, login
from django.contrib.auth.decorators import login_required
from django.contrib.auth.forms import AuthenticationForm
from .forms import CustomUserCreationForm, LoginForm
from django.core.mail import send_mail
from django.core.mail import EmailMultiAlternatives
from django.template.loader import get_template
from django.template import Context
from django.core.exceptions import ValidationError
from django.core.validators import validate_email

##### index#####
def index(request):
    try:
        user = request.user
    except:
        user = "AnonymousUser"
    return render(request, 'index.html', {'title':'index', 'user':user})

##### register here #####
def register(request):
```



```
if request.method == 'POST':
    form = CustomUserCreationForm(request.POST)
    if form.is_valid():
        form.save()
        # username = form.cleaned_data.get('username')
        email = form.cleaned_data.get('email')
        ##### mail system
#####
        html = get_template('Email.html')
        d = { 'email': email }
        subject, from_email, to = 'welcome', 'your_email@gmail.com', email
        html_content = html.render(d)
        msg = EmailMultiAlternatives(subject, html_content, from_email, [to])
        msg.attach_alternative(html_content, "text/html")
        # msg.send()

#####
        messages.success(request, f'Your account has been created ! You are now
able to log in')
        return redirect('login')
    else:
        form = CustomUserCreationForm()
        return render(request, 'register.html', {'form': form, 'title': 'register here'})

##### login
forms#####
def Login(request):
    if request.method == 'POST':

        email = request.POST.get('email')
        try:
            validate_email(email)
        except ValidationError as e:
            print("bad email, details:", e)
        else:
            print("good email")
        password = request.POST.get('password')
        user = authenticate(request, email = email, password = password)
        if user is not None:
            form = login(request, user)
            messages.success(request, f' wecome {email} !!')
            return redirect('index')
```



```
        else:
            messages.info(request, f'account done not exit plz sign in')
    else:
        form = LoginForm()
        return render(request, 'login.html', {'form':form, 'title':'log in'})
```

urls.py

```
from .views import Login, register, index

from django.urls import include, path
from django.contrib.auth import login, logout
from django.contrib.auth import views as auth_views

urlpatterns = [
    path("", index, name='index'),

    path('signup/', register, name='register'),
    path('login/', Login, name='login'),
    path('logout/', auth_views.LogoutView.as_view(), {'next_page': 'index'}, name='logout'),
]
```

forms.py

```
from django.contrib.auth.forms import UserCreationForm, UserChangeForm
from django.forms import ModelForm

from .models import User

class CustomUserCreationForm(UserCreationForm):
```



```
class Meta:
```

```
    model = User
```

```
    fields = ('email', 'first_name', 'last_name', 'sap_id')
```

```
class CustomUserChangeForm(UserChangeForm):
```

```
    class Meta:
```

```
        model = User
```

```
        fields = ('email', 'first_name', 'last_name', 'sap_id')
```

```
class LoginForm(ModelForm):
```

```
    class Meta:
```

```
        model = User
```

```
        fields = ('email', 'password')
```

setting.py

```
.  
.   
.   
  
# Application definition  
  
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    'authentication',  
    'crispy_forms'  
]  
.
```



MyProject/urls.py

```
from django.contrib import admin
from django.urls import path,include

urlpatterns = [
    path('admin/', admin.site.urls),
    path("", include("authentication.urls")),
]
```

index.html

```
{% load static %}
{% load crispy_forms_tags %}
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <meta name="title" content="project">
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
  <meta name="language" content="English">

  <title>{{title}}</title>

<!-- bootstrap file -->
```



```
<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" />
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
<!-- bootstrap file-->

<!-- jQuery -->
<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"
integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"
crossorigin="anonymous"></script>

<link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/font-awesome@4.7.0/css/font-awesome.min.css">

<!-- main css -->
<link rel="stylesheet" type="text/css" href='{% static "index.css" %}' />

<!-- message here -->

{% if messages %}
{% for message in messages %}

<script>
    alert('{{ message }}');
</script>

{% endfor %}
{% endif %}

<!-- _____ -->

</head>

<body class="container-fluid">

<header class="row">
```



```

<!-- navbar-->
<nav class="navbar navbar-inverse navbar-fixed-top">
  <div class="container-fluid">
    <div class="navbar-header">
      <button class="navbar-toggle" data-toggle="collapse" data-target="#mainNavBar">
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
      </button>
      <a class="navbar-brand" class="styleheader" href='{% url "index" %}'>project</a>
    </div>
    <div class="collapse navbar-collapse" id="mainNavBar">
      <ul class="nav navbar-nav navbar-right">
        <li><a href="{% url 'index' %}">Home</a></li>

        {% if user.is_authenticated %}
        <li><a href="{% url 'logout' %}"><span class="glyphicon glyphicon-log-out"></span> &nbsp; Logout</a></li>
        {% else %}
        <li><a href="{% url 'register' %}"><span class="glyphicon glyphicon-user"></span> &nbsp; Sign up</a></li>
        <li><a href="{% url 'login' %}"><span class="glyphicon glyphicon-log-in"></span> &nbsp; Log in</a></li>
        {% endif %}

      </ul>
    </div>
  </div>
</nav>
</header>
<br/>
<br>
<br>
<div class="row">
  {% block start %}
  {% if user.is_authenticated %}
  <center><h1>welcome back {{user.email}}!</h1></center>
  {% else %}
  <center><h1>log in, plz . . .</h1></center>
  {% endif %}
  {% endblock %}

```



```
</div>
</body>

</html>
```

login.html

```
{% extends "index.html" %}
{% load crispy_forms_tags %}
{% block start %}

<div class="content-section col-md-8 col-md-offset-2">
<center>
<form method="POST" style="border: 1px solid black; margin: 4%; padding:10%; border-radius:1%;">
  {% csrf_token %}
  <fieldset class="form-group">
    <label for="email">Email</label><br>
    <input id="email" type="text" autocomplete="off" name="email" value="" class="form-control
input-lg" placeholder="Email" />
    <br><br>
    <label for="Password">Password</label><br>
    <input id="password" type="password" autocomplete="off" name="password" value=""
class="form-control input-lg" placeholder="Password" />

  </fieldset>
<center>
  <button style="background: black; font-size: 2rem; padding:1%;" class="btn btn-outline-info"
type="submit"><span class="glyphicon glyphicon-log-in"></span> &nbsp; login</button>
</center>
<br/>
  <sub style="text-align: left;"><a href="{% url 'register' %}" style="text-decoration: none; color: blue;
padding:2%; cursor:pointer; margin-right:2%;">don't have account,sign up</a></sub>
</form>
</center>
</div>
```




{% endblock start %}

register.html

```
{% extends "index.html" %}
{% load crispy_forms_tags %}
{% block start %}

<div class="content-section col-md-8 col-md-offset-2">
  <form method="POST" style="border: 1px solid black; margin: 4%; padding:10%; border-radius:1%;">
    {% csrf_token %}
    <fieldset class="form-group">
      {{ form|crispy}}
    </fieldset>
    <center>
      <button style="background: black; padding:2%; font-size: 2rem; color:white;" class="btn
btn-outline-info" type="submit"><span class="glyphicon glyphicon-check"></span> &nbsp; sign
up</button>
    </center>
    <br />
    <sub><a href="{% url 'login' %}" style="text-decoration: none; color: blue; padding:3%;
cursor:pointer;">Already have an account ?</a></sub>
  </form>
</div>
{% endblock start %}
```

email.html

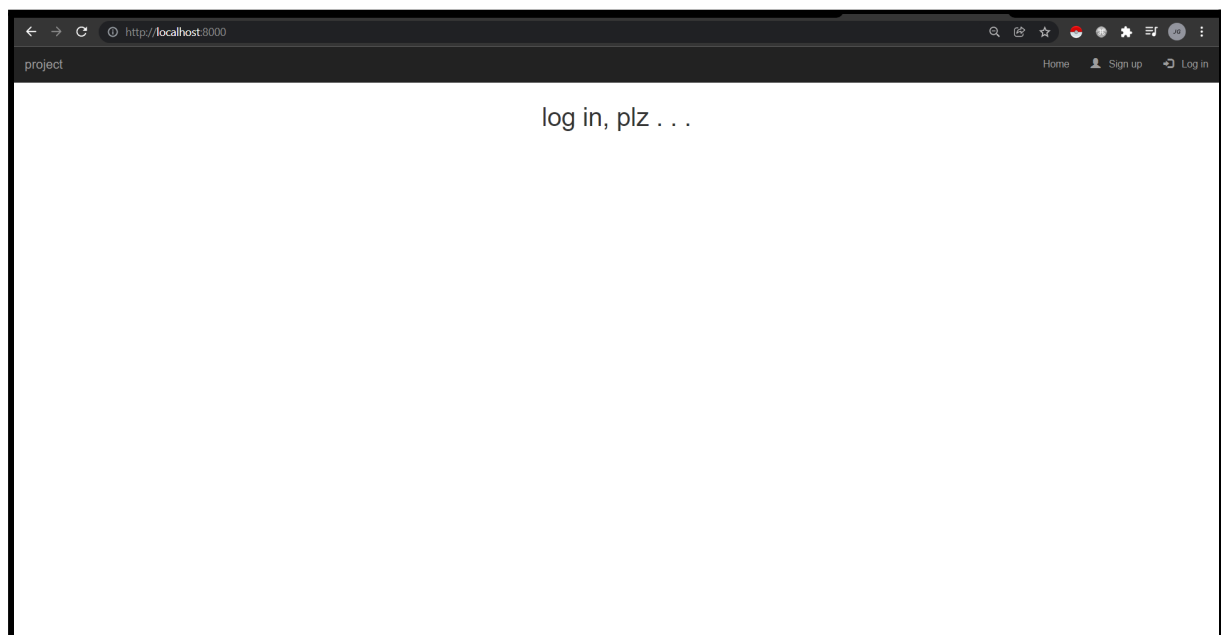
```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title></title>
```



```
<style>
    @import
url('https://fonts.googleapis.com/css?family=Roboto:400,100,300,500,700,900');
</style>
</head>
<body style="background: #f5f8fa;font-family: 'Roboto', sans-serif;">
    <div style="width: 90%;max-width:600px;margin: 20px auto;background: #ffffff;">
        <section style="margin: 0 15px;color:#425b76;">
            <h2 style="margin: 40px 0 27px 0;text-align: center;">Thank you to
registration</h2>
            <hr style="border:0;border-top: 1px solid rgba(66,91,118,0.3);max-width:
50%">
            <p style="font-size:15.5px;font-weight: bold;margin:40px 20px 15px
20px;">Hi {{username}}, we have received your details and will process it soon.</p>
        </section>
    </div>
</body>
</html>
```

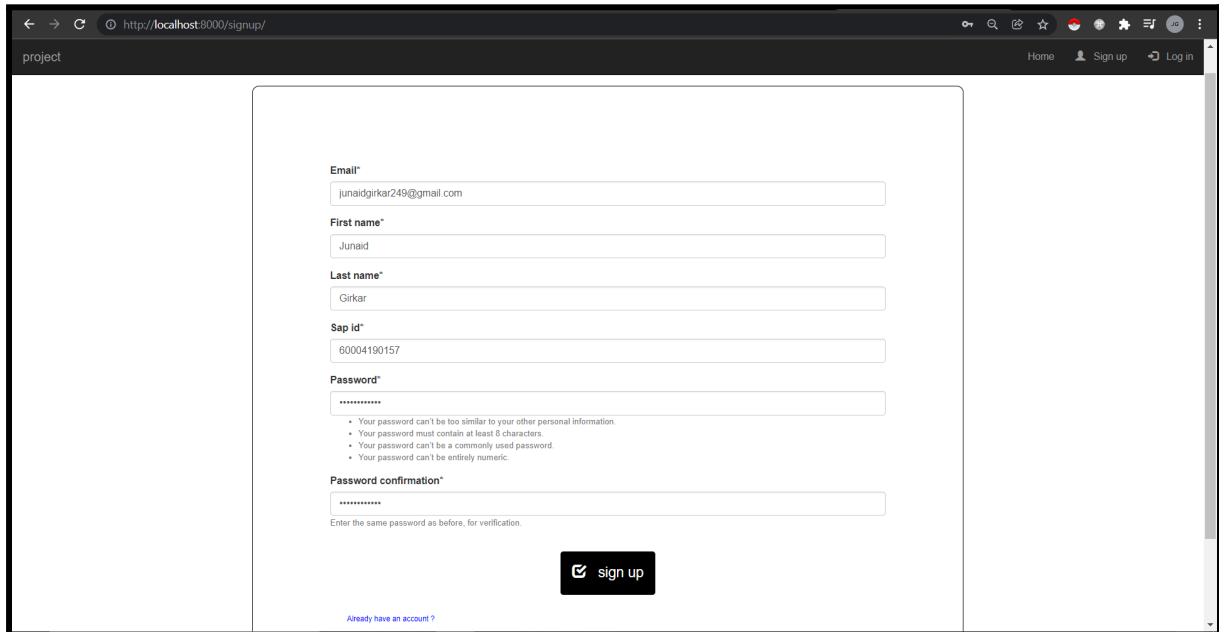
OUTPUT:

Home Page:





Register Page:



The screenshot shows a web browser window with the URL `http://localhost:8000/signup/`. The page title is "project". The registration form includes the following fields and labels:

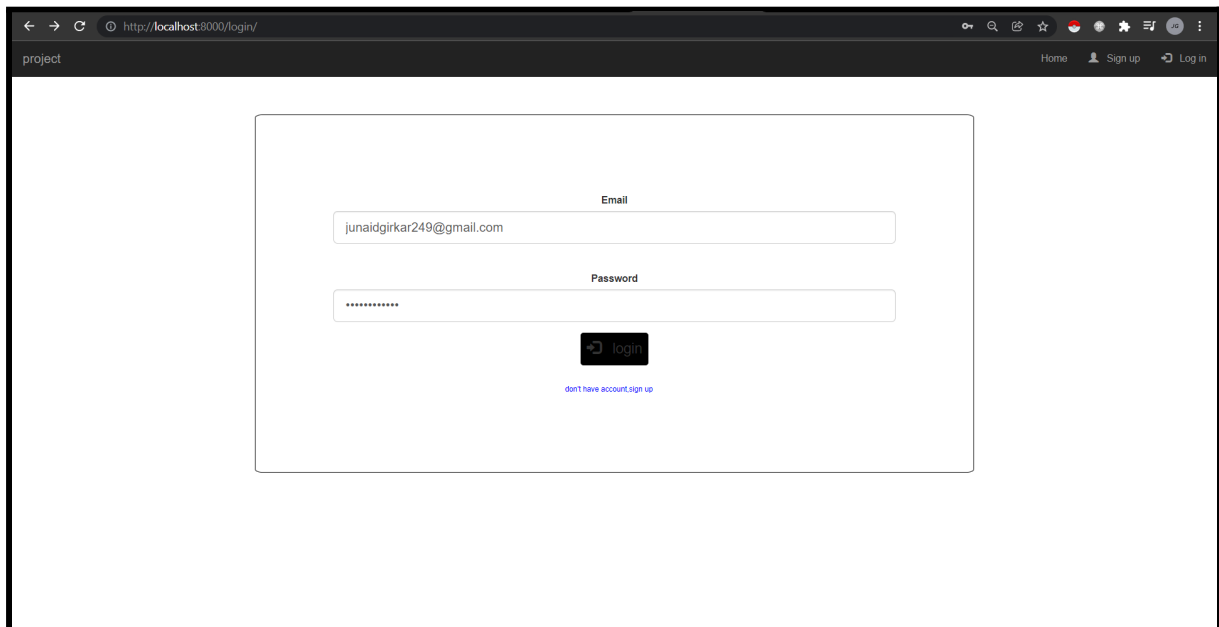
- Email***: `junaidgirkar249@gmail.com`
- First name***: `Junaid`
- Last name***: `Girkar`
- Sap id***: `60004190157`
- Password***: `*****`
- Password confirmation***: `*****`

Below the password fields, there are four bullet points:

- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

At the bottom of the form, there is a "sign up" button and a link: [Already have an account ?](#)

Login Page:



The screenshot shows a web browser window with the URL `http://localhost:8000/login/`. The page title is "project". The login form includes the following fields and labels:

- Email**: `junaidgirkar249@gmail.com`
- Password**: `*****`

Below the password field, there is a "login" button and a link: [don't have account, sign up](#)



Shri Vile Parle Kelavani Mandal's

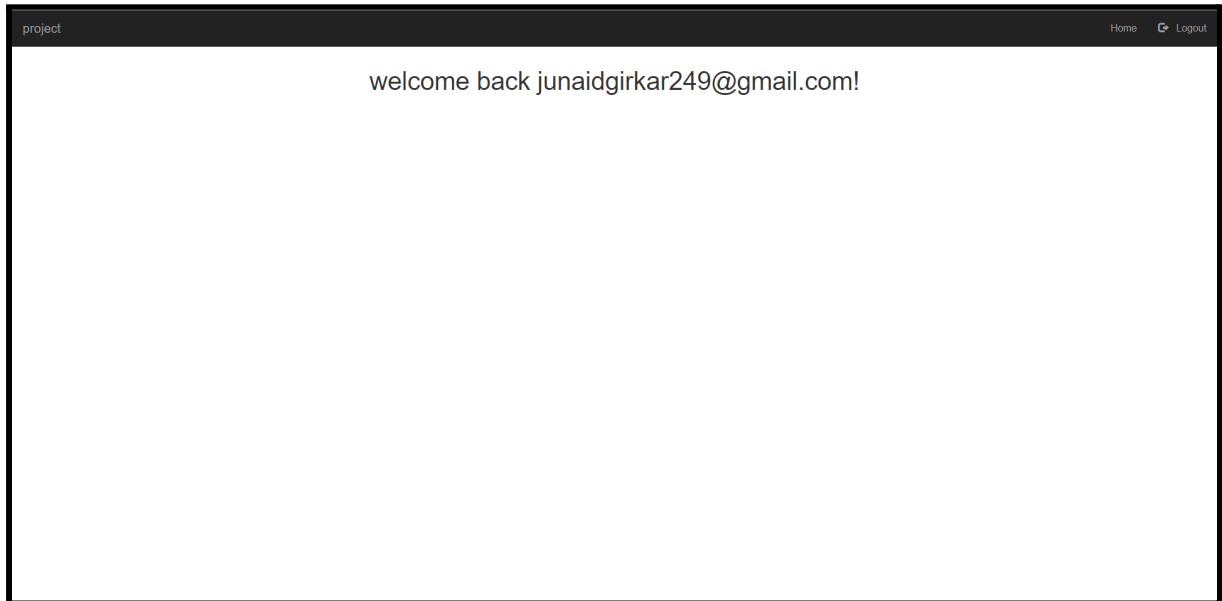
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING

(Autonomous College Affiliated to the University of Mumbai)

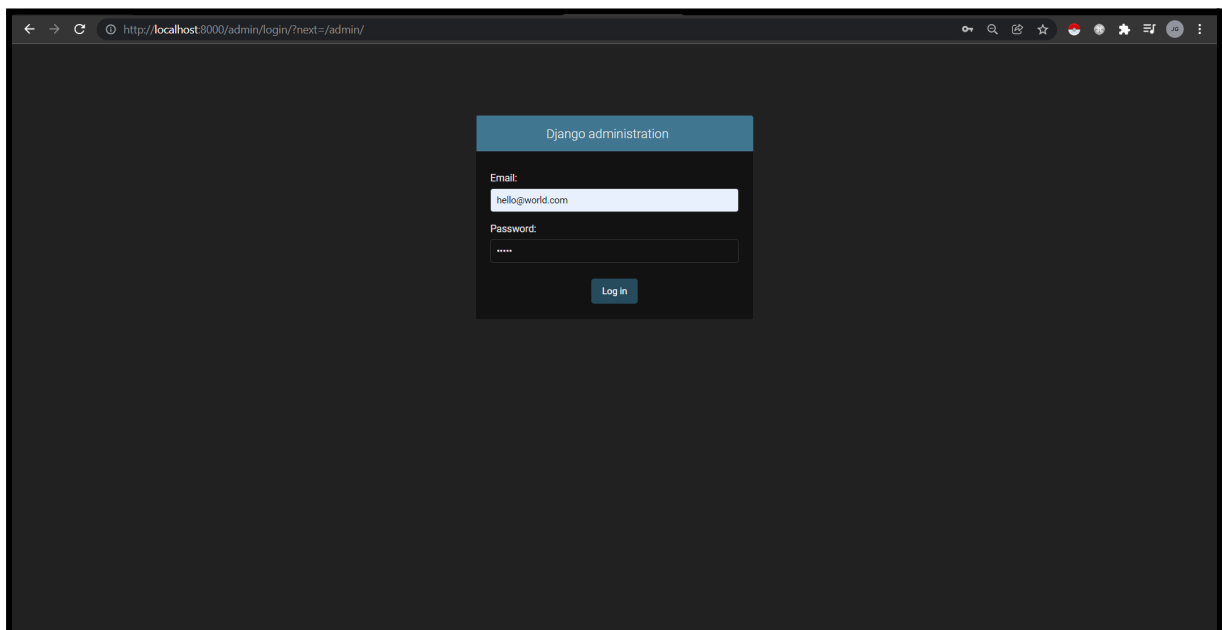
NAAC Accredited with "A" Grade (CGPA : 3.18)



Home Page (After successful login):

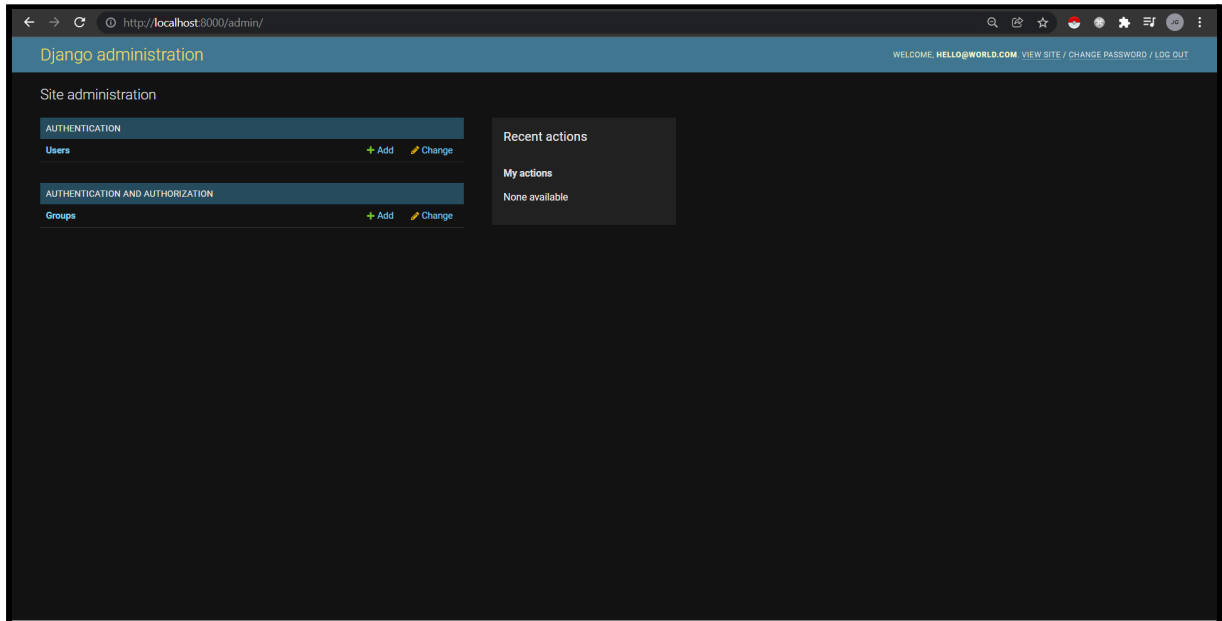


Admin Panel Login:

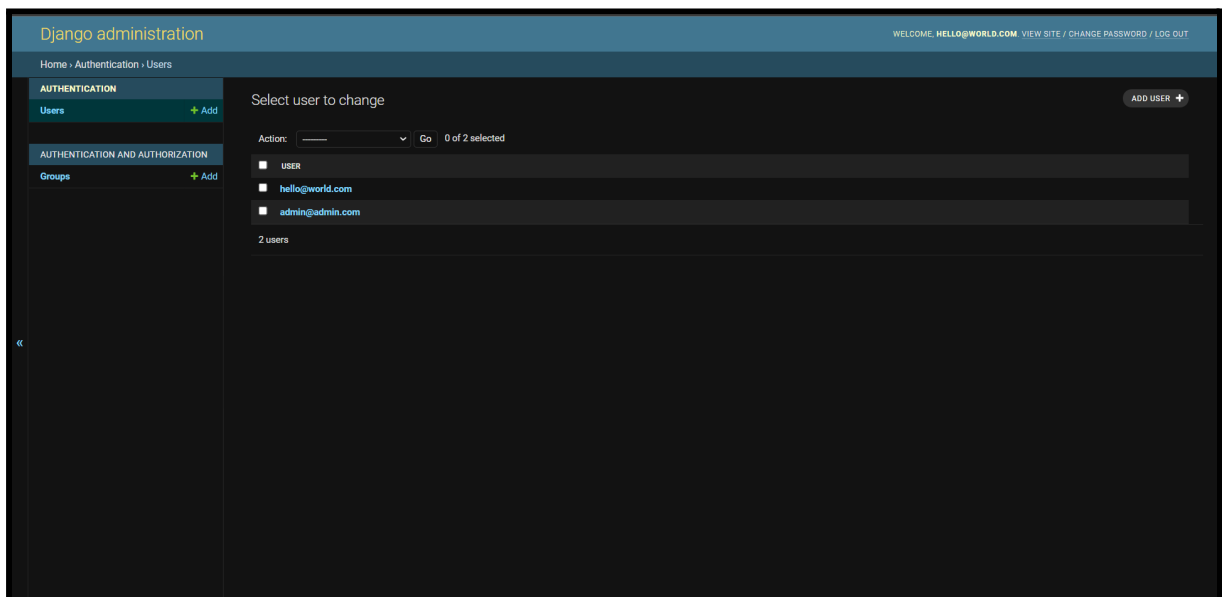




Admin Home Page:



Users List Page in Admin





User Detail Page in Admin

The screenshot displays the Django administration interface for a user management system. The browser address bar shows the URL: `http://localhost:8000/admin/authentication/user/2/change/`. The page title is "Django administration". The breadcrumb trail is "Home > Authentication > Users > hello@world.com". The left sidebar contains a menu with "AUTHENTICATION" (Users, Add) and "AUTHENTICATION AND AUTHORIZATION" (Groups, Add). The main content area is titled "Change user" and shows the details for the user "hello@world.com". The user's password is masked with a long alphanumeric string. The "Last login" information shows a date of "2021-12-29" and a time of "10:55:37". The user's email is "hello@world.com", first name is "Junaid", last name is "Girkar", and SAP ID is "60004190057". The user is marked as "Active", "Staff", and "Admin". At the bottom, there are buttons for "Delete", "Save and add another", "Save and continue editing", and "SAVE".

Conclusion:

We learnt about the Django web development framework and its functionalities. We then created a website with a user authentication system with a custom Abstract Base User model.