**PYTHON DJANGO Internship Report**

**Personal Details**

Name : Prerak Patel

College Name : Sardar Vallbhabhai patel Institute of Technology (SVIT)

Degree : BE

Semester : 7

Github URL : <https://github.com/prerak-26>

**Company Details**

Company Name : Akash Technolabs

External Guide : Akash Padhiyar

Training Duration : 26-05-2021 to 07-06-2021

**Index**

|  |  |  |
| --- | --- | --- |
| **Index** | **Task Detail** | **Page Name** |
| 1 | Introduction | 1 |
| 2 | Variables | 6 |
| 3 | Decision making | 11 |
| 4 | Functions/Loops | 13 |
| 5 | Oops | 18 |
| 6 | Django Installation | 24 |
| 7 | Django start project | 25 |
| 8 | Django files information | 25 |
| 9 | Django project | 30 |

( Day -1 ) Introduction

**Python** is a general purpose, dynamic, [high-level](https://www.javatpoint.com/classification-of-programming-languages), and interpreted programming language. It supports Object Oriented programming approach to develop applications. It is simple and easy to learn and provides lots of high-level data structures.

Python provides many useful features to the programmer. These features make it most popular and widely used language. We have listed below few-essential feature of Python.

* Easy to use and Learn
* Expressive Language
* Interpreted Language
* Object-Oriented Language
* Open Source Language
* Extensible
* Learn Standard Library
* GUI Programming Support
* Integrated
* Embeddable
* Dynamic Memory Allocation
* Wide Range of Libraries and Frameworks

Where is Python used?

Python is a general-purpose, popular programming language and it is used in almost every technical field. The various areas of Python use are given below.

* Data Science
* Date Mining
* Desktop Applications
* Console-based Applications
* Mobile Applications
* Software Development
* Artificial Intelligence
* Web Applications
* Enterprise Applications
* 3D CAD Applications
* Machine Learning

# **How to Install Python (Environment Set-up)**

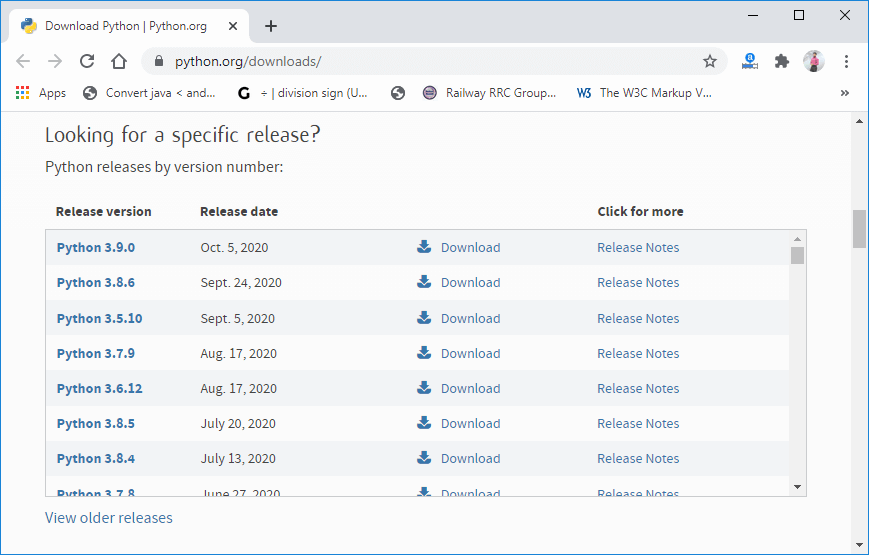
In order to become Python developer, the first step is to learn how to install or update Python on a local machine or computer. In this tutorial, we will discuss the installation of Python on various operating systems.

## Installation on Windows

Visit the link <https://www.python.org/downloads/> to download the latest release of [Python](https://www.javatpoint.com/python-tutorial). In this process, we will install Python 3.8.6 on our [Windows operating system](https://www.javatpoint.com/windows). When we click on the above link, it will bring us the following page.

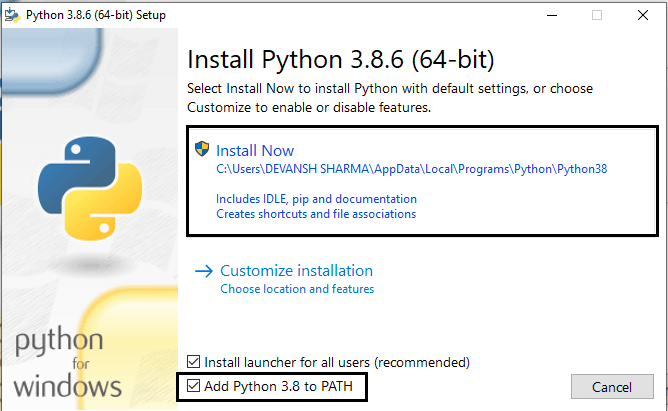
**Step - 1: Select the Python's version to download.**

Click on the download button.



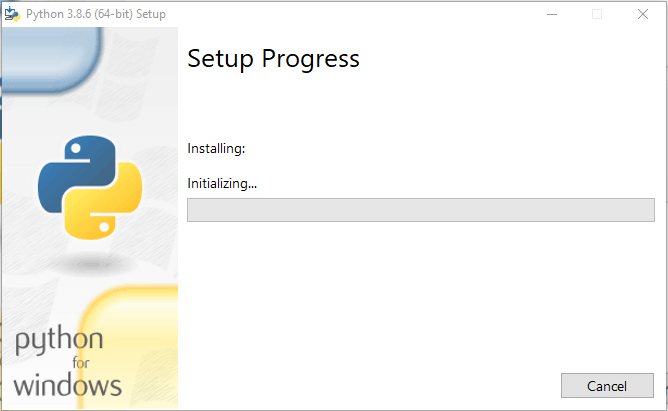
**Step - 2: Click on the Install Now**

Double-click the executable file, which is downloaded; the following window will open. Select Customize installation and proceed. Click on the Add Path check box, it will set the Python path automatically.

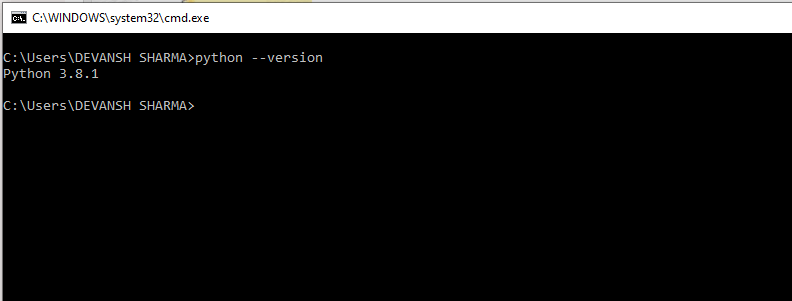


We can also click on the customize installation to choose desired location and features. Other important thing is install launcher for the all user must be checked.

**Step - 3 Installation in Process**



Now, try to run python on the command prompt. Type the command python -version in case of python3.



We are ready to work with the Python.

( Day – 2 ) Variables

Session :

* Information of variables
* Cover all data types such as float, int, string
* Cover all data structure such as list, tuple, dictionary

Variables :

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals or characters in these variables.

Multiple Assignment

Python allows you to assign a single value to several variables simultaneously. For example −

* a = b = c = 1
* a,b,c = 1,2,"john"

Data Types :

The data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters. Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.

Python has five standard data types −

* Numbers
* String
* List
* Tuple
* Dictionary

## Python Numbers

Number data types store numeric values. Number objects are created when you assign a value to them.

For example − var1 = 1

1. String

Strings in Python are identified as a contiguous set of characters represented in the quotation marks. Python allows for either pairs of single or double quotes. Subsets of strings can be taken using the slice operator ([ ] and [:] ) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.

The plus (+) sign is the string concatenation operator and the asterisk (\*) is the repetition operator.

## Python Lists

Lists are the most versatile of Python's compound data types. A list contains items separated by commas and enclosed within square brackets ([]).

The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1.

## Python Tuples

A tuple is another sequence data type that is similar to the list. A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.

The main differences between lists and tuples are: Lists are enclosed in brackets ( [ ] ) and their elements and size can be changed, while tuples are enclosed in parentheses ( ( ) ) and cannot be updated. Tuples can be thought of as **read-only** lists.

## Python Dictionary

Python's dictionaries are kind of hash table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.

Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]).

var = 1

var2 = 10.1

var3 = "Akash Techlab"

print(type(var))

print(type(var2))

print(type(var3))

str = 'Hello World!'

print(str) # Prints complete string

print(str[0]) # Prints first character of the string

print(str[2:5]) # Prints characters starting from 3rd to 5th

print(str[2:]) # Prints string starting from 3rd character

print(str \* 2) # Prints string two times

print(str + "TEST") # Prints concatenated string

list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]

tinylist = [123, 'john']

print(list) # Prints complete list

print(list[0]) # Prints first element of the list

print(list[1:3]) # Prints elements starting from 2nd till 3rd

print(list[2:]) # Prints elements starting from 3rd element

print(tinylist \* 2) # Prints list two times

print(list + tinylist) # Prints concatenated lists

tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )

tinytuple = (123, 'john')

print(tuple) # Prints the complete tuple

print(tuple[0]) # Prints first element of the tuple

print(tuple[1:3]) # Prints elements of the tuple starting from 2nd till 3rd

print(tuple[2:]) # Prints elements of the tuple starting from 3rd element

print(tinytuple \* 2) # Prints the contents of the tuple twice

print(tuple + tinytuple) # Prints concatenated tuples

dict = {}

dict['one'] = "This is one"

dict[2] = "This is two"

tinydict = {'name': 'john','code':6734, 'dept': 'sales'}

print(dict['one']) # Prints value for 'one' key

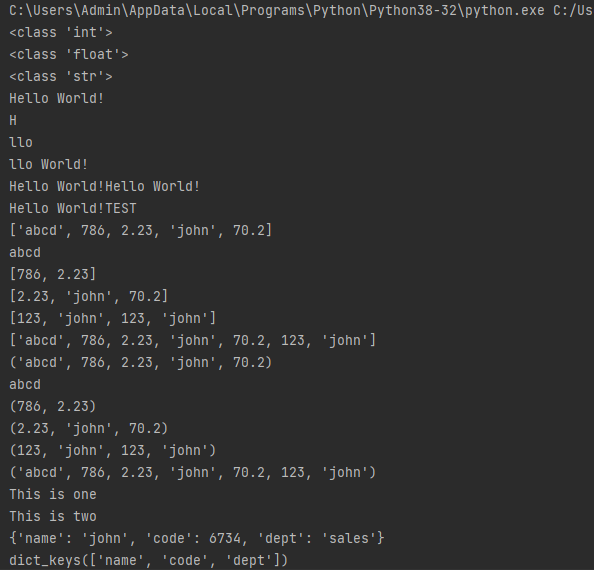
print(dict[2]) # Prints value for 2 key

print(tinydict) # Prints complete dictionary

print(tinydict.keys()) # Prints all the keys

print(tinydict.values()) # Prints all the values

output :



( Day - 3 ) Decision Making

Decision making is the most important aspect of almost all the programming languages. As the name implies, decision making allows us to run a particular block of code for a particular decision. Here, the decisions are made on the validity of the particular conditions. Condition checking is the backbone of decision making.

## The if statement

The if statement is used to test a particular condition and if the condition is true, it executes a block of code known as if-block. The condition of if statement can be any valid logical expression which can be either evaluated to true or false.

## The if-else statement

The if-else statement provides an else block combined with the if statement which is executed in the false case of the condition.

If the condition is true, then the if-block is executed. Otherwise, the else-block is executed.

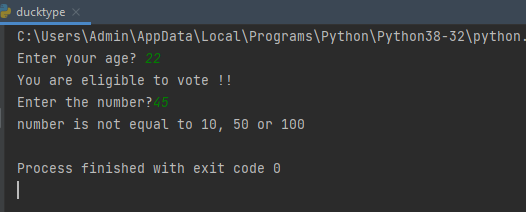
## The elif statement

The elif statement enables us to check multiple conditions and execute the specific block of statements depending upon the true condition among them. We can have any number of elif statements in our program depending upon our need. However, using elif is optional.

The elif statement works like an if-else-if ladder statement in C. It must be succeeded by an if statement.

age = int (input("Enter your age? "))  
if age>=18:  
 print("You are eligible to vote !!");  
else:  
 print("Sorry! you have to wait !!")  
  
  
number = int(input("Enter the number?"))  
if number==10:  
 print("number is equals to 10")  
elif number==50:  
 print("number is equal to 50");  
elif number==100:  
 print("number is equal to 100");  
else:  
 print("number is not equal to 10, 50 or 100")

output :



( Day – 4 ) Loops / Function

The flow of the programs written in any programming language is sequential by default. Sometimes we may need to alter the flow of the program. The execution of a specific code may need to be repeated several numbers of times.

For this purpose, The programming languages provide various types of loops which are capable of repeating some specific code several numbers of times. Consider the following diagram to understand the working of a loop statement.

For loop :

The for **loop in Python** is used to iterate the statements or a part of the program several times. It is frequently used to traverse the data structures like list, tuple, or dictionary.

While loop :

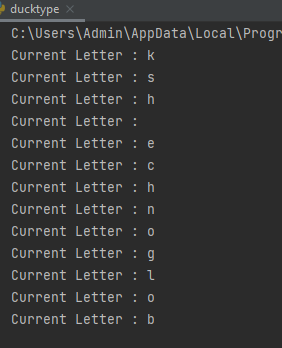
The Python while loop allows a part of the code to be executed until the given condition returns false. It is also known as a pre-tested loop.

It can be viewed as a repeating if statement. When we don't know the number of iterations then the while loop is most effective to use.

While loop :

i = 0  
str1 = 'akash technoglob'  
  
while i < len(str1):  
 if str1[i] == 'a' or str1[i] == 't':  
 i += 1  
 continue  
 print('Current Letter :', str1[i])  
 i += 1

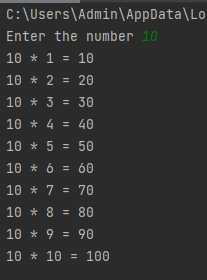
output:



For loop :

n = int(input("Enter the number "))  
for i in range(1,11):  
 c = n\*i  
 print(n,"\*",i,"=",c)

output:



**Functions :**

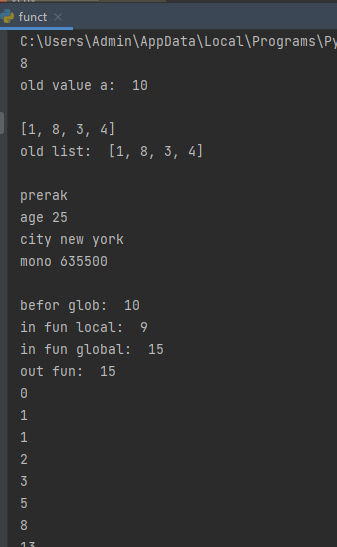
Functions are the most important aspect of an application. A function can be defined as the organized block of reusable code, which can be called whenever required.

Python allows us to divide a large program into the basic building blocks known as a function. The function contains the set of programming statements enclosed by {}. A function can be called multiple times to provide reusability and modularity to the Python program.

There are mainly two types of functions.

* User-define functions - The user-defined functions are those define by the user to perform the specific task.
* Built-in functions - The built-in functions are those functions that are pre-defined in Python.
* The def keyword, along with the function name is used to define the function.
* The identifier rule must follow the function name.
* A function accepts the parameter (argument), and they can be optional.
* The function block is started with the colon (:), and block statements must be at the same indentation.
* The return statement is used to return the value. A function can have only one return
* def update(x): #it is immmutable data type so upadated value and old value different...  
   x = 8  
   print(x)  
    
    
  a = 10  
  update(a)  
  print("old value a: ", a)  
  print()  
    
    
  def updatel(lst): #it is mmutable data type so upadated value and old value same...  
   lst[1] = 8  
   print(lst)  
    
    
  lst = [1, 2, 3, 4]  
  updatel(lst)  
  print("old list: ", lst)  
  print()  
    
    
  #unknow no of passing arrguments  
  def person(name,\*\*data):  
   print(name)  
   for i,j in data.items():  
   print(i,j)  
    
  person(name='prerak',age=25,city='new york',mono=635500)  
  print()  
    
    
    
    
  #use of globes  
    
  a = 10  
  print("befor glob: ",a)  
    
  def some():  
   a = 9  
   x = globals()['a']  
   globals()['a'] = 15  
   print("in fun local: ",a)  
   print("in fun global: ",globals()['a'])  
    
  some()  
  print("out fun: ",a)  
    
    
    
    
  def fib(n):  
   a=0  
   b=1  
   print(a)  
   print(b)  
    
   for i in range(2,n):  
   c=a+b  
   a = b  
   b = c  
    
   if c>100:  
   print(c)  
   break  
   print(c)  
    
  fib(100)  
    
    
    
  def fac(n):  
    
   if n==0:  
   return 1  
   n=n\*fac(n-1)  
   return n  
    
  v=fac(5)  
  print(v)

output :



( Day – 5 ) Oops

Python is also an object-oriented language since its beginning. It allows us to develop applications using an Object-Oriented approach. In [Python](https://www.javatpoint.com/python-tutorial), we can easily create and use classes and objects.

An object-oriented paradigm is to design the program using classes and objects. The object is related to real-word entities such as book, house, pencil, etc. The oops concept focuses on writing the reusable code. It is a widespread technique to solve the problem by creating objects.

Major principles of object-oriented programming system are given below.

* Class
* Object
* Method
* Inheritance
* Polymorphism
* Data Abstraction
* Encapsulation

## Class

The class can be defined as a collection of objects. It is a logical entity that has some specific attributes and methods. For example: if you have an employee class, then it should contain an attribute and method, i.e. an email id, name, age, salary, etc.

## Object

The object is an entity that has state and behavior. It may be any real-world object like the mouse, keyboard, chair, table, pen, etc.

## Method

The method is a function that is associated with an object. In Python, a method is not unique to class instances. Any object type can have methods.

## Inheritance

Inheritance is the most important aspect of object-oriented programming, which simulates the real-world concept of inheritance. It specifies that the child object acquires all the properties and behaviors of the parent object.

By using inheritance, we can create a class which uses all the properties and behavior of another class.

## Polymorphism

Polymorphism contains two words "poly" and "morphs". Poly means many, and morph means shape. By polymorphism, we understand that one task can be performed in different ways. For example - you have a class animal, and all animals speak. But they speak differently. Here, the "speak" behavior is polymorphic in a sense and depends on the animal. So, the abstract "animal" concept does not actually "speak", but specific animals (like dogs and cats) have a concrete implementation of the action "speak".

## Encapsulation

Encapsulation is also an essential aspect of object-oriented programming. It is used to restrict access to methods and variables. In encapsulation, code and data are wrapped together within a single unit from being modified by accident.

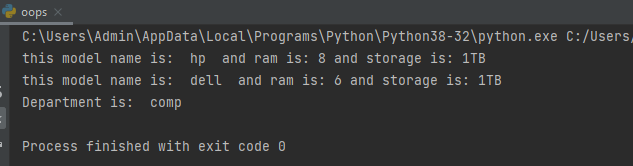
## Data Abstraction

Data abstraction and encapsulation both are often used as synonyms. Both are nearly synonyms because data abstraction is achieved through encapsulation.

Abstraction is used to hide internal details and show only functionalities. Abstracting something means to give names to things so that the name captures the core of what a function or a whole program does.

class Computer:  
  
 storage = '1TB' # static variable ( class variable )  
  
 def \_\_init\_\_(self, name, ram):  
 self.name = name # instance variable  
 self.ram = ram  
  
 def config(self):  
 print("this model name is: ",self.name," and ram is: " + str(self.ram) + " and storage is: "+str(Computer.storage))  
  
  
c1 = Computer('hp', 8)  
c2 = Computer('dell', 6)  
  
#c1.name = 'asus' # this is how to change instance variable value  
  
#Computer.storage = '2TB' # this is how to chnage class or static variable value  
  
c1.config()  
c2.config()  
  
  
  
  
# special method \_\_call\_\_  
  
  
class Eng:  
  
 def \_\_init\_\_(self,dp):  
 self.dp = dp  
  
 def \_\_call\_\_(self): # if you use call method at that time you can direct call method by using object.  
 print("Department is: ",self.dp)  
  
  
p = Eng('comp')  
p() # this is the use of call method

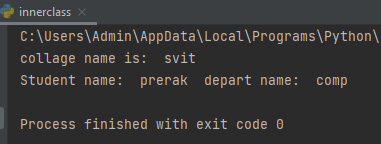
output :



Inner class :

class College:  
 def \_\_init\_\_(self, name, stname, dpname):  
 self.name = name  
 self.dp = self.Dp(stname, dpname)  
  
 def show(self):  
 print("collage name is: ", self.name)  
 self.dp.show1()  
  
 class Dp:  
 def \_\_init\_\_(self, sname, dpname):  
 self.sname = sname  
 self.dpname = dpname  
  
 def show1(self):  
 print("Student name: ", self.sname, " depart name: ", self.dpname)  
  
  
s1 = College('svit', 'prerak', 'comp')  
s1.show()

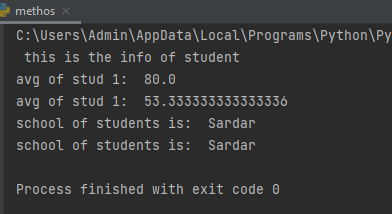
output :



Methods :

class Students:  
  
 school = 'Sardar'  
  
 def \_\_init\_\_(self, m1, m2, m3): # constructor  
 self.m1 = m1  
 self.m2 = m2  
 self.m3 = m3  
  
 def avg(self): # instance method witch use instance variable  
 return (self.m1 + self.m2 + self.m3) / 3  
  
 @classmethod  
 def getSchool(cls): # class method witch is use for doing task on class variable  
 return cls.school  
  
 @staticmethod  
 def info(): # static method witch is use for doing someother task witch is not of class work  
 print(" this is the info of student")  
  
  
Students.info() # if you want to call static method at that time you don't need to create a obje of class  
  
s1 = Students(80, 90, 70)  
s2 = Students(30, 60, 70)  
  
print("avg of stud 1: ", Students.avg(s1)) # two diff types of calling instance method  
print("avg of stud 1: ", s2.avg())  
  
print("school of students is: ", Students.getSchool()) # you can call class method by using class name as well as object refrence  
print("school of students is: ", s1.getSchool())

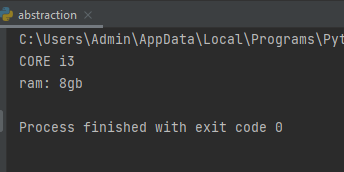
output :



Abstractions :

from abc import ABC, abstractclassmethod  
  
  
class Computer(ABC):  
  
 def intel(self):  
 print("CORE i3")  
  
 @abstractclassmethod  
 def ram(self):  
 pass  
  
  
class Comram(Computer):  
  
 def ram(self):  
 super().intel()  
 print("ram: 8gb")  
  
  
com = Comram()  
com.ram()

output :



( Day 5-10 ) Django Tutorial

1) How to install Django?

### Installing an official release with pip

This is the recommended way to install Django.

1. Install [pip](https://pip.pypa.io/). The easiest is to use the [standalone pip installer](https://pip.pypa.io/en/latest/installing/#installing-with-get-pip-py). If your distribution already has **pip** installed, you might need to update it if it’s outdated. If it’s outdated, you’ll know because installation won’t work.
2. Take a look at [venv](https://docs.python.org/3/tutorial/venv.html" \o "(in Python v3.9)). This tool provides isolated Python environments, which are more practical than installing packages systemwide. It also allows installing packages without administrator privileges. The [contributing tutorial](https://docs.djangoproject.com/en/3.2/intro/contributing/) walks through how to create a virtual environment.
3. After you’ve created and activated a virtual environment, enter the command:
4. **$** python -m pip install Django

2) Django basic:

Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it 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’s free and open source.

Ridiculously fast.

Django was designed to help developers take applications from concept to completion as quickly as possible.

Reassuringly secure.

Django takes security seriously and helps developers avoid many common security mistakes.

Exceedingly scalable.

Some of the busiest sites on the Web leverage Django’s ability to quickly and flexibly scale.

3) How to create project in Django:

**$** django-admin startproject mysite

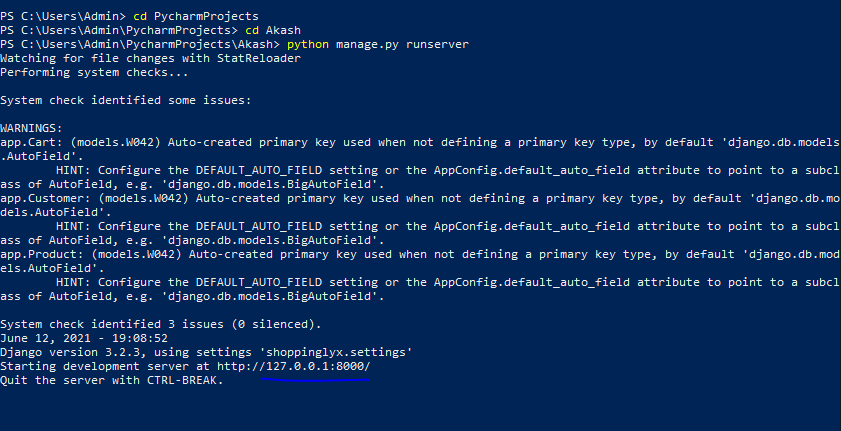
This will auto-generate some code that establishes a Django [project](https://docs.djangoproject.com/en/3.2/glossary/#term-project) – a collection of settings for an instance of Django, including database configuration, Django-specific options and application-specific settings.

This will create a **mysite** directory in your current directory.

4) How to start project:

$ python manage.py runserver

This will generate link which will be opened in browser in order to start the project.

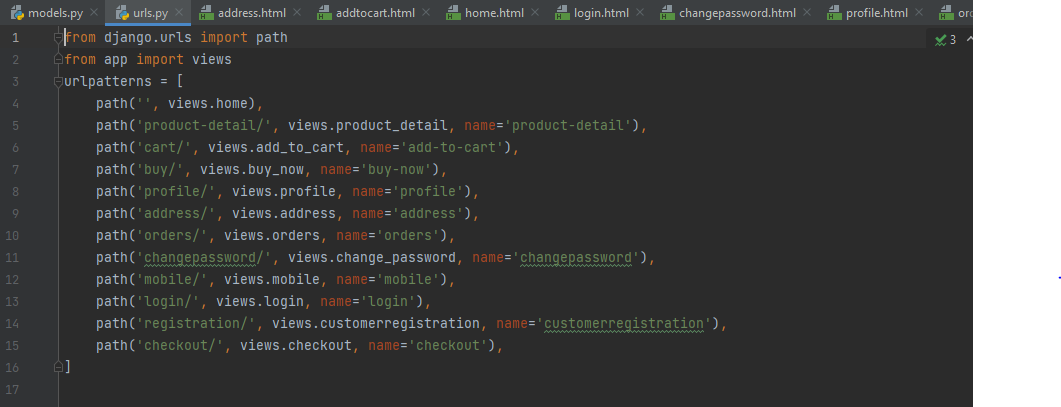


## 5) How Django processes a request

When a user requests a page from your Django-powered site, this is the algorithm the system follows to determine which Python code to execute:

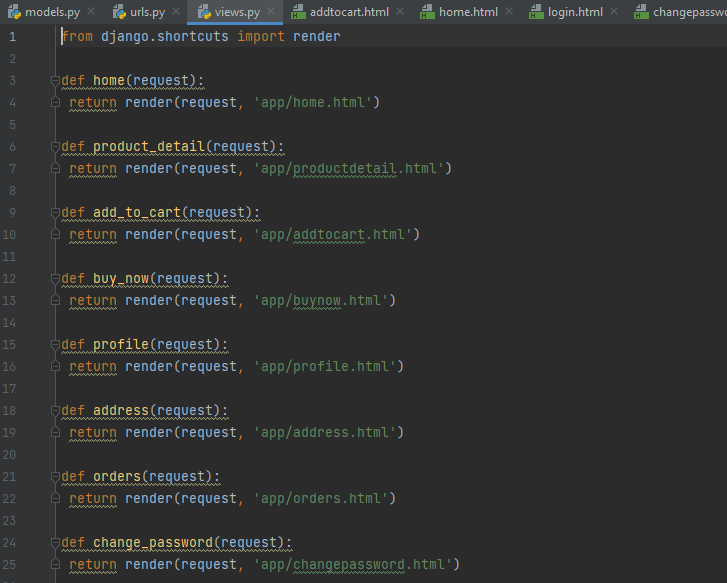
1. Django determines the root URLconf module to use. Ordinarily, this is the value of the [**ROOT\_URLCONF**](https://docs.djangoproject.com/en/3.2/ref/settings/#std:setting-ROOT_URLCONF) setting, but if the incoming **HttpRequest** object has a **[urlconf](https://docs.djangoproject.com/en/3.2/ref/request-response/" \l "django.http.HttpRequest.urlconf" \o "django.http.HttpRequest.urlconf)** attribute (set by middleware), its value will be used in place of the [**ROOT\_URLCONF**](https://docs.djangoproject.com/en/3.2/ref/settings/#std:setting-ROOT_URLCONF) setting.
2. Django loads that Python module and looks for the variable **urlpatterns**. This should be a [sequence](https://docs.python.org/3/glossary.html#term-sequence) of [**django.urls.path()**](https://docs.djangoproject.com/en/3.2/ref/urls/#django.urls.path) and/or [**django.urls.re\_path()**](https://docs.djangoproject.com/en/3.2/ref/urls/#django.urls.re_path) instances.
3. Django runs through each URL pattern, in order, and stops at the first one that matches the requested URL, matching against **[path\_info](https://docs.djangoproject.com/en/3.2/ref/request-response/" \l "django.http.HttpRequest.path_info" \o "django.http.HttpRequest.path_info)**.
4. Once one of the URL patterns matches, Django imports and calls the given view, which is a Python function (or a [class-based view](https://docs.djangoproject.com/en/3.2/topics/class-based-views/)). The view gets passed the following arguments:
   * An instance of **[HttpRequest](https://docs.djangoproject.com/en/3.2/ref/request-response/" \l "django.http.HttpRequest" \o "django.http.HttpRequest)**.
   * If the matched URL pattern contained no named groups, then the matches from the regular expression are provided as positional arguments.
   * The keyword arguments are made up of any named parts matched by the path expression that are provided, overridden by any arguments specified in the optional **kwargs** argument to **[django.urls.path()](https://docs.djangoproject.com/en/3.2/ref/urls/" \l "django.urls.path" \o "django.urls.path)** or **[django.urls.re\_path()](https://docs.djangoproject.com/en/3.2/ref/urls/" \l "django.urls.re_path" \o "django.urls.re_path)**.

If no URL pattern matches, or if an exception is raised during any point in this process, Django invokes an appropriate error-handling view.



6) views.py file:

A view function, or view for short, is a Python function that takes a Web request and returns a Web response. This response can be the HTML contents of a Web page, or a redirect, or a 404 error, or an XML document, or an image . . . or anything, really. The view itself contains whatever arbitrary logic is necessary to return that response. This code can live anywhere you want, as long as it’s on your Python path. There’s no other requirement–no “magic,” so to speak. For the sake of putting the code somewhere, the convention is to put views in a file called **views.py**, placed in your project or application directory.

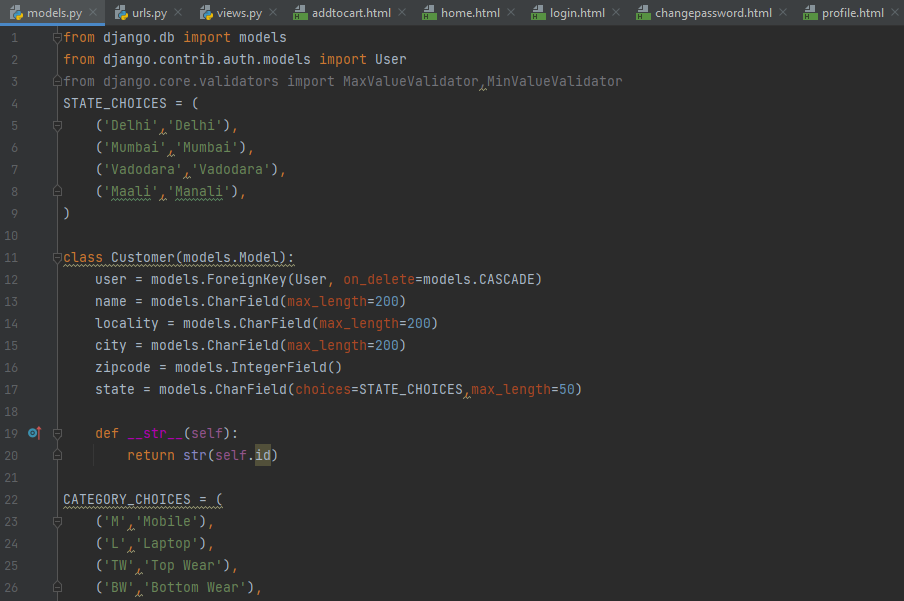


Let’s step through this code one line at a time:

* First, we import the class **[HttpResponse](https://docs.djangoproject.com/en/3.2/ref/request-response/" \l "django.http.HttpResponse" \o "django.http.HttpResponse)** from the **[django.http](https://docs.djangoproject.com/en/3.2/ref/request-response/" \l "module-django.http" \o "django.http: Classes dealing with HTTP requests and responses.)** module, along with Python’s **datetime** library.
* Next, we define a function called **current\_datetime**. This is the view function. Each view function takes an **[HttpRequest](https://docs.djangoproject.com/en/3.2/ref/request-response/" \l "django.http.HttpRequest" \o "django.http.HttpRequest)** object as its first parameter, which is typically named **request**.

Note that the name of the view function doesn’t matter; it doesn’t have to be named in a certain way in order for Django to recognize it. We’re calling it **current\_datetime** here, because that name clearly indicates what it does.

* The view returns an **[HttpResponse](https://docs.djangoproject.com/en/3.2/ref/request-response/" \l "django.http.HttpResponse" \o "django.http.HttpResponse)** object that contains the generated response. Each view function is responsible for returning an **[HttpResponse](https://docs.djangoproject.com/en/3.2/ref/request-response/" \l "django.http.HttpResponse" \o "django.http.HttpResponse)** object. (There are exceptions, but we’ll get to those later.)
* The migrations folder is where Django stores migrations, or changes to your database. There’s nothing in here you need to worry about.
* \_\_init\_\_.py tells Python that your pages app is a package.
* admin.py is where you register your models with the Django admin application.
* apps.py is a configuration file common to all Django apps.
* models.py is where the models for your app are located.
* tests.py contains test procedures that will be run when testing your app.
* And views.py is where the views for your app are located.



7) Superuser:

## Creating an admin user

First we’ll need to create a user who can login to the admin site. Run the following command:

**$** python manage.py createsuperuser

Enter your desired username and press enter.

Username: admin

You will then be prompted for your desired email address:

Email address: admin@example.com

The final step is to enter your password. You will be asked to enter your password twice, the second time as a confirmation of the first.

Password: \*\*\*\*\*\*\*\*\*\*

Password (again): \*\*\*\*\*\*\*\*\*

Superuser created successfully.

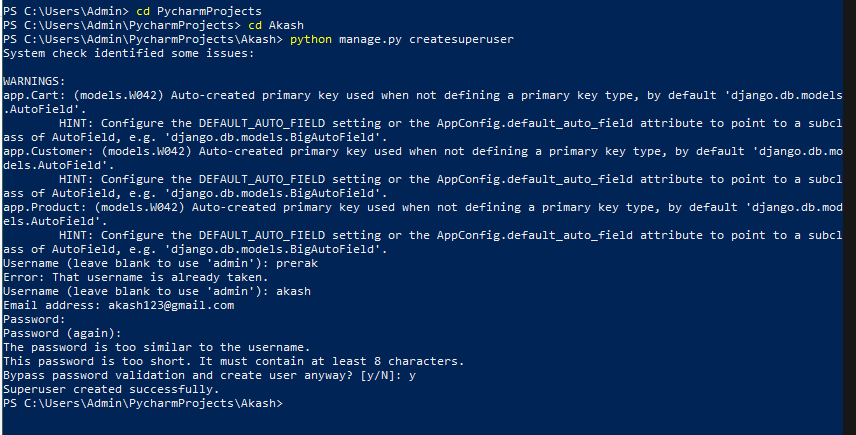
## Start the development server

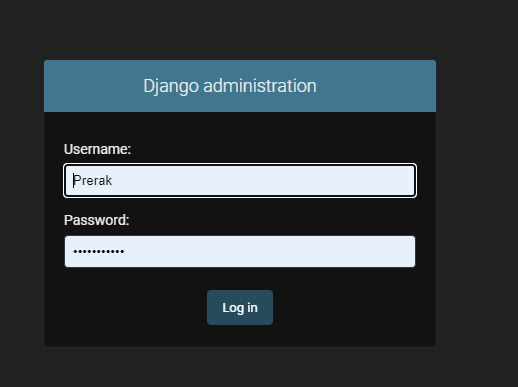
The Django admin site is activated by default. Let’s start the development server and explore it.

Recall from Tutorial 1 that you start the development server like so:

**$** python manage.py runserver

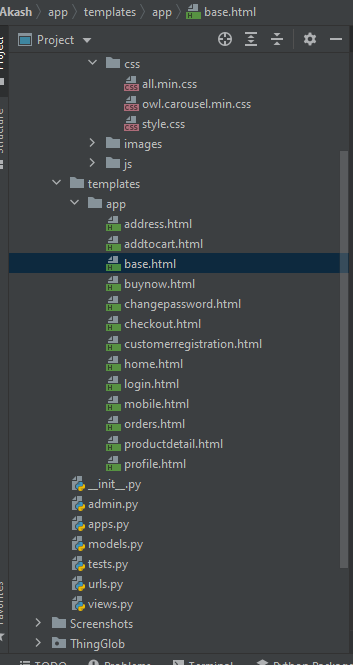
Now, open a Web browser and go to “/admin/” on your local domain – e.g., <http://127.0.0.1:8000/admin/>. You should see the admin’s login screen:



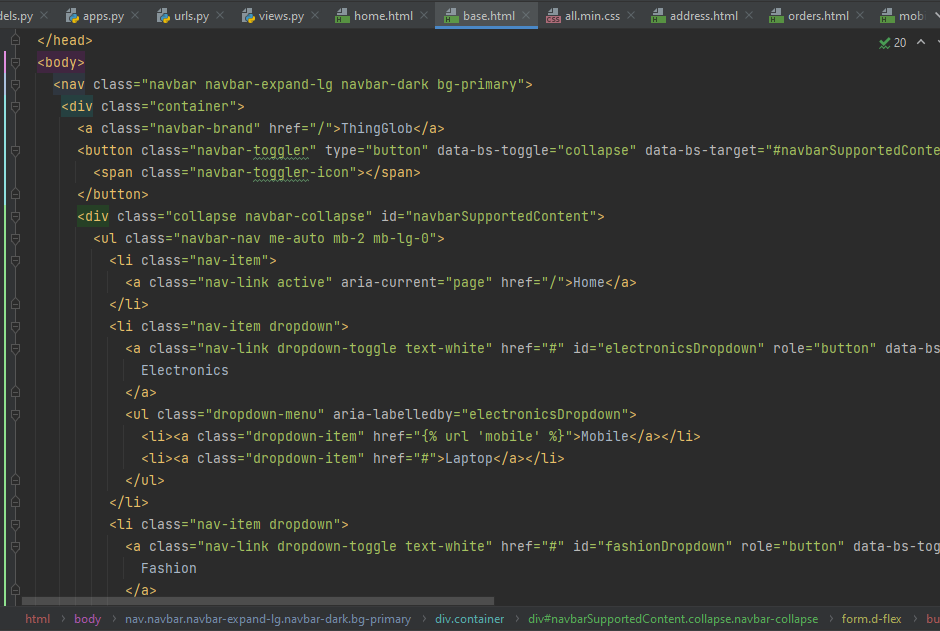


Project : ThingGlob

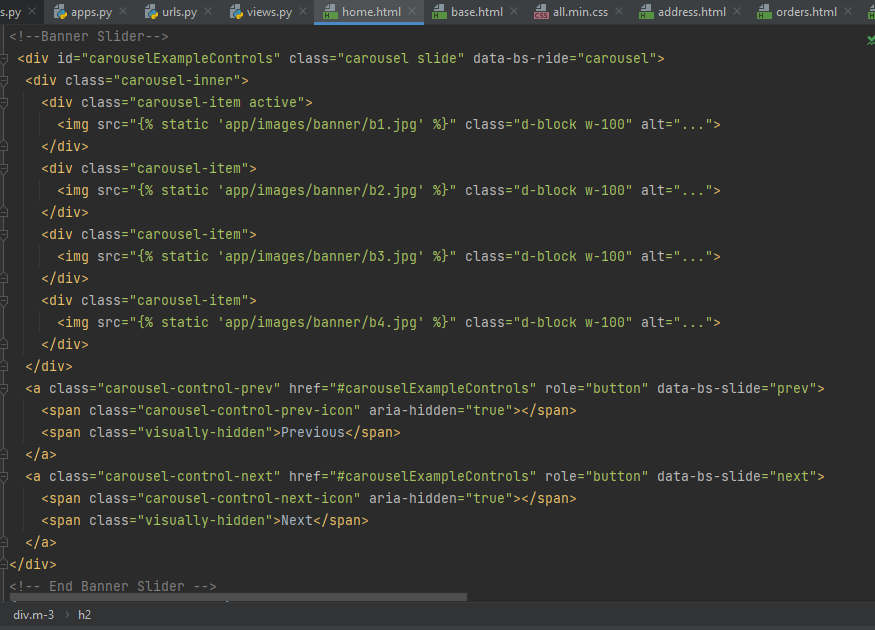
Files Formate :



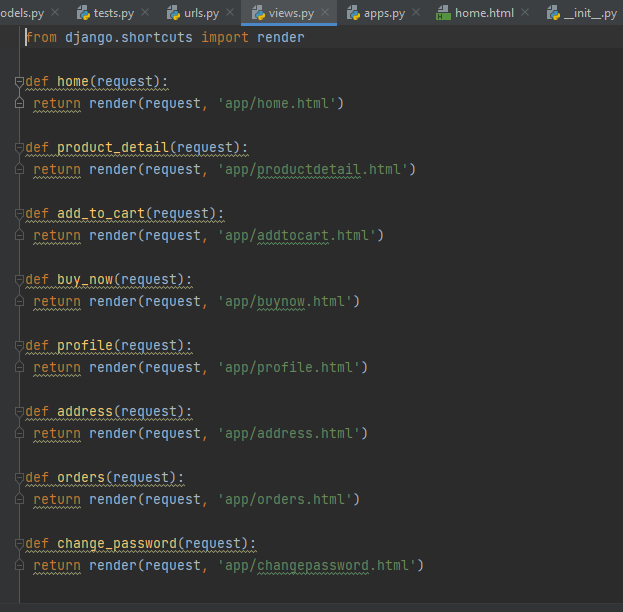
base.html :



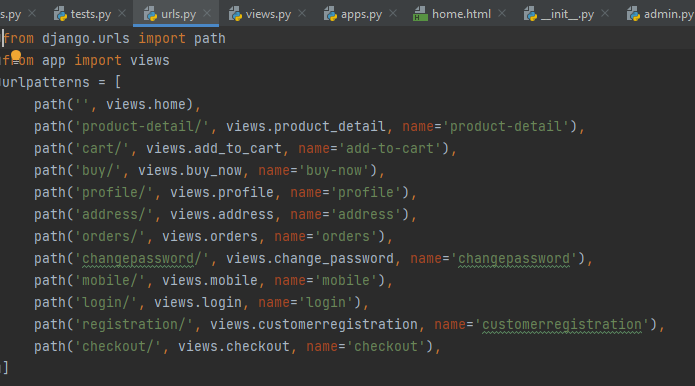
home.html :



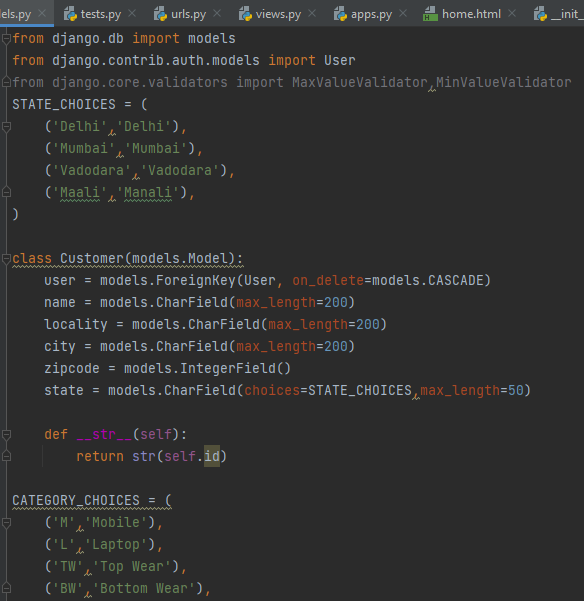
views.py :



urls.py :

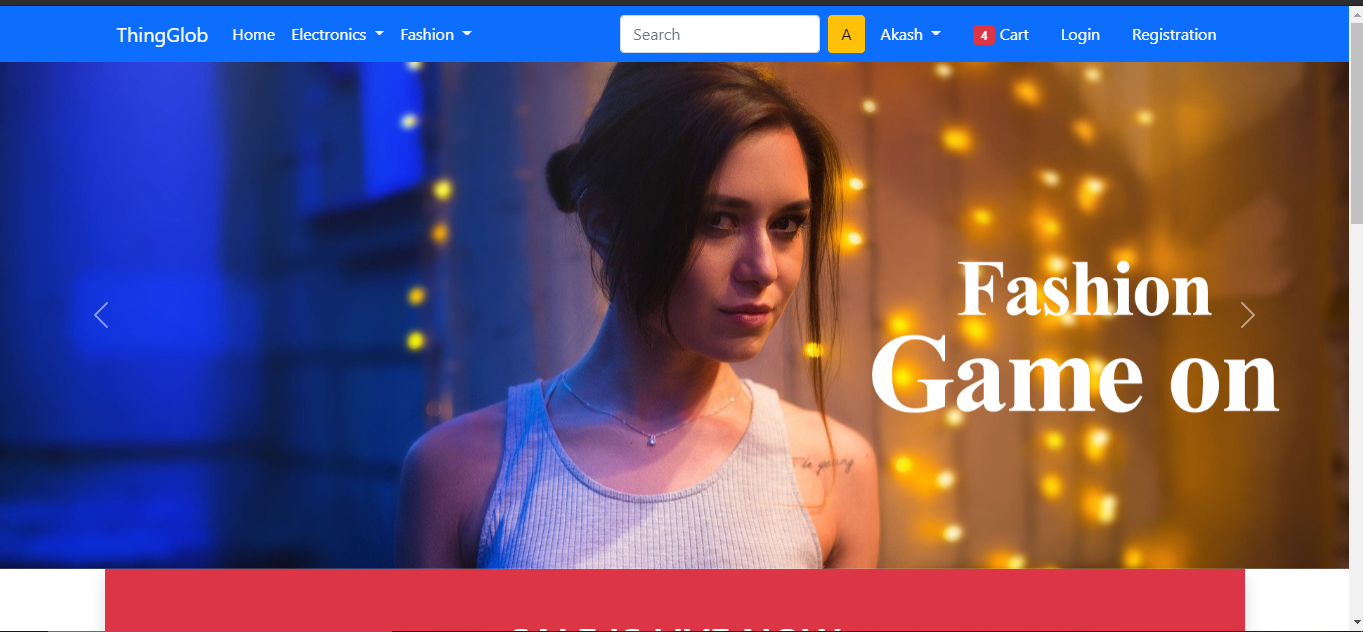


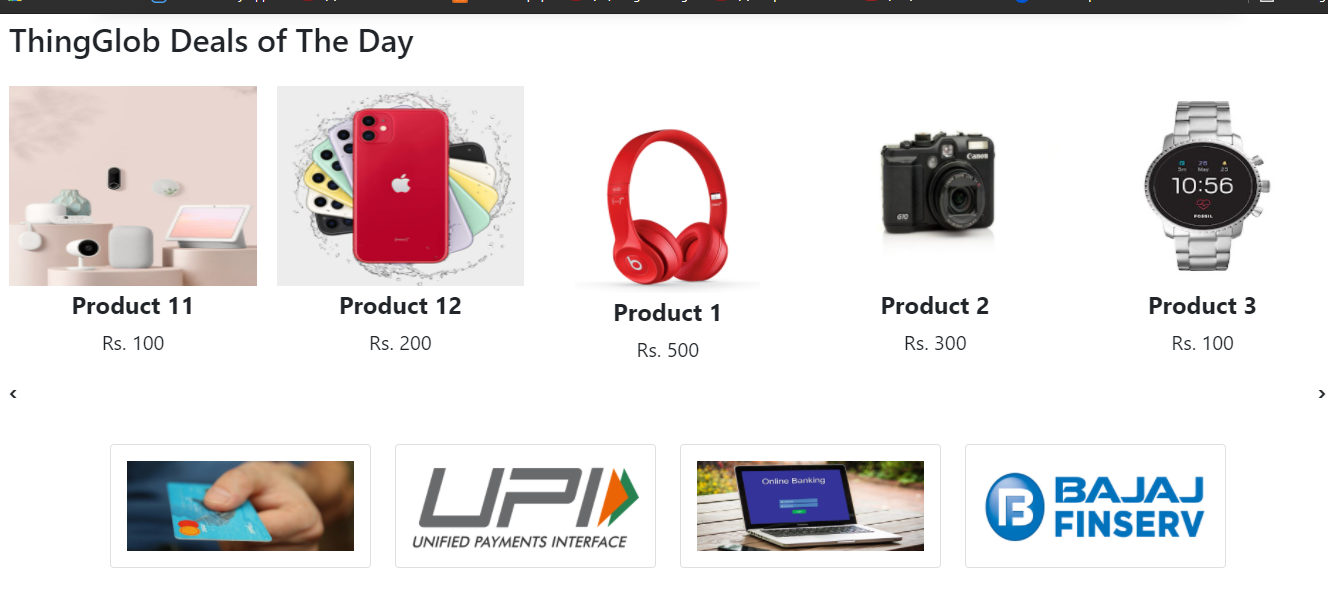
models.py :



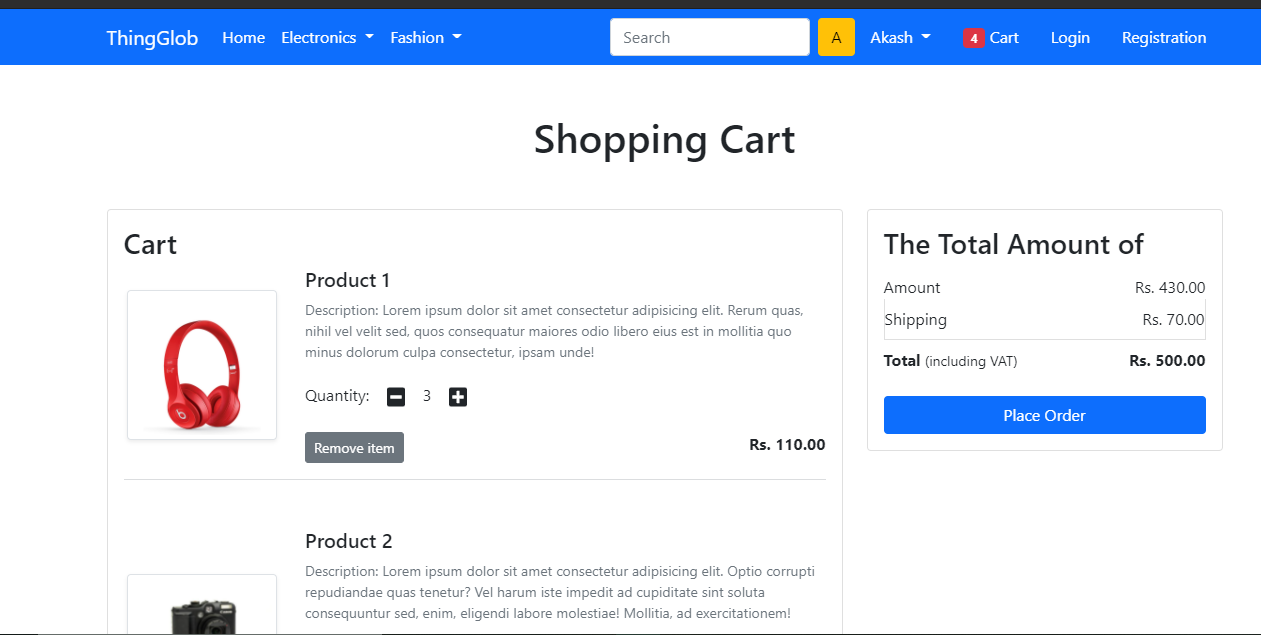
Output :

Home page :

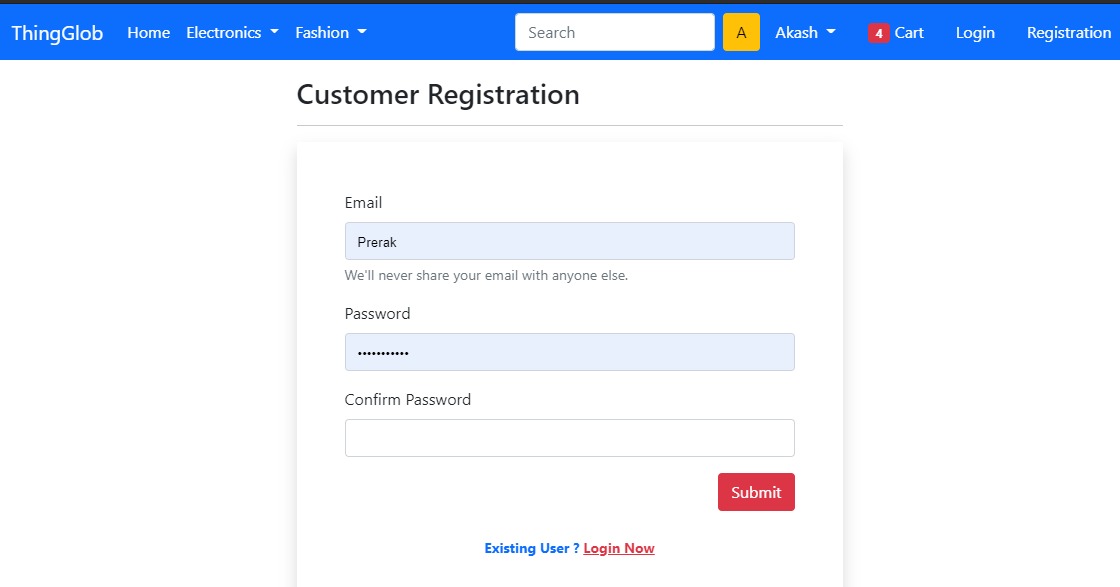




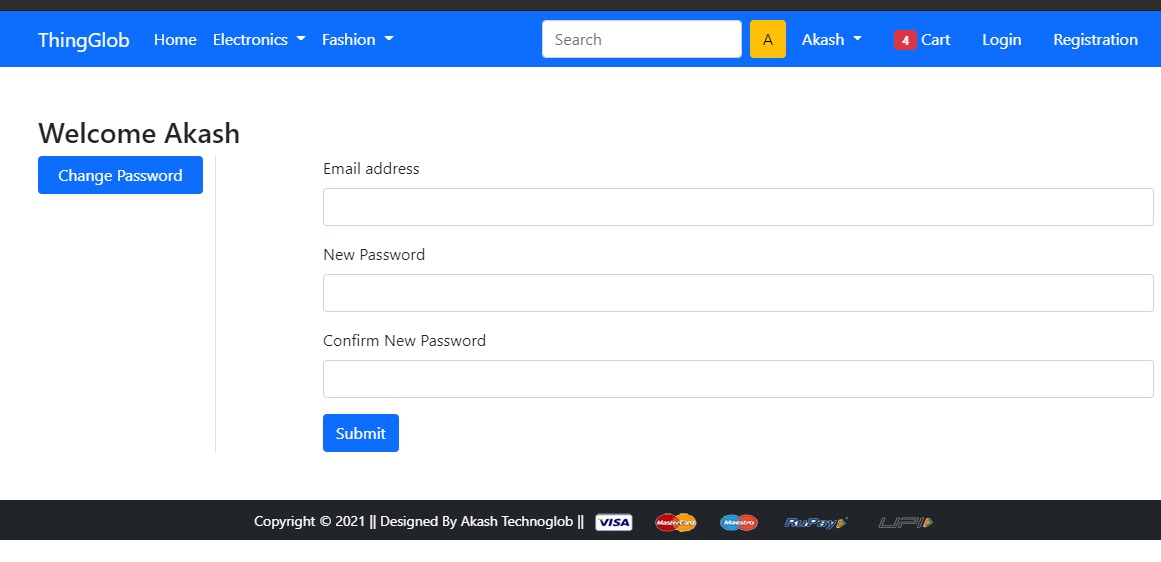
Cart page :



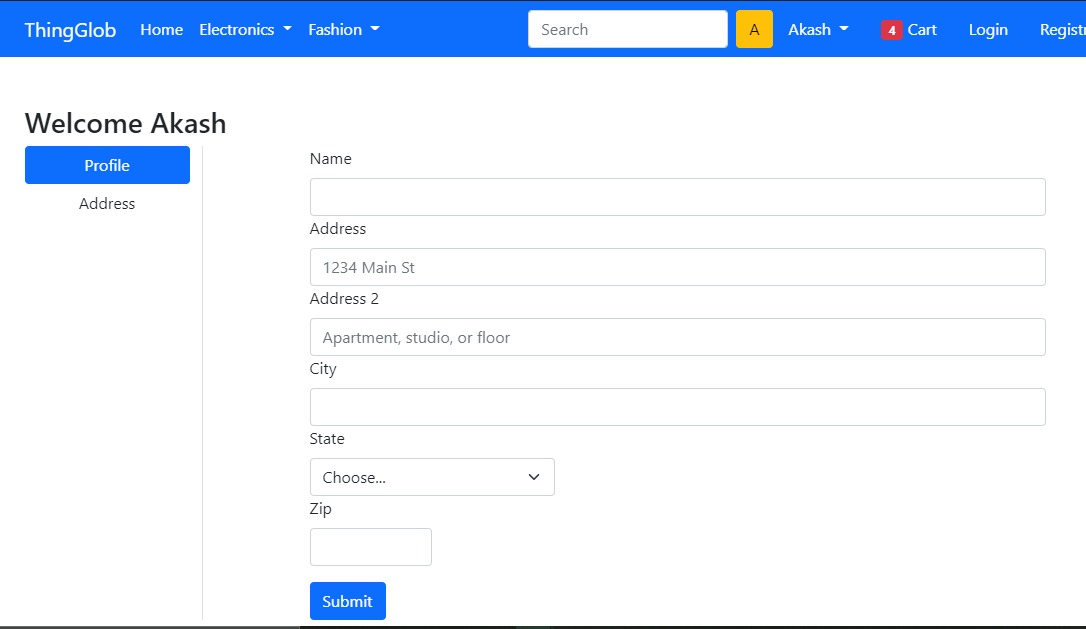
Registration Page:



Change Password:



Profile Page:



Project Github link :

<https://github.com/prerak-26/AakashT_internship>

I made this static website using Django and looking forward to make it dynamic to explore my knowledge.