**FLASK**

VIRTUAL ENVIRONMENT IN CONDA

* Needed to have a given set of dependencies for a particular project
* Create:

**conda create –name myenv**

* Create a env with a package included (ex numpy) : **conda create -n myenv numpy** or **conda create -n myenv numpy=1.14**
* Create with specific version of Python : : **conda create -n myenv python=3.5**

**conda create -n myenv pip 🡪** To create an environment with pip

* To list out all the env : **conda env list**
* Conda Virtual Environment help: <https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html>
* To activate an environment from cmd : **>** **activate <env name>**

Example : C:\Users\drsan\Downloads\original\Flask-Bootcamp-master>activate flaskenv1

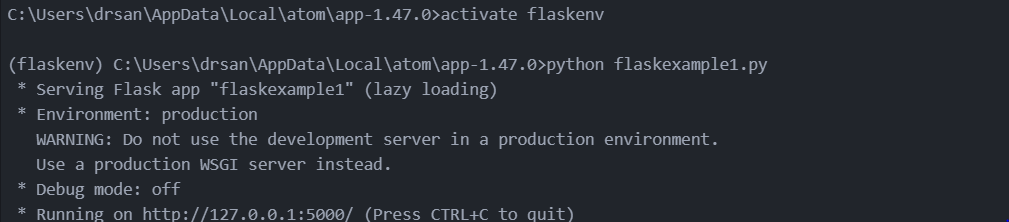
* To deactivate an environment from inside environment : **>** **deactivate**
* To install Libraries inside environment : **conda install <library> or pip install <library>**
  + Ex : conda install numpy

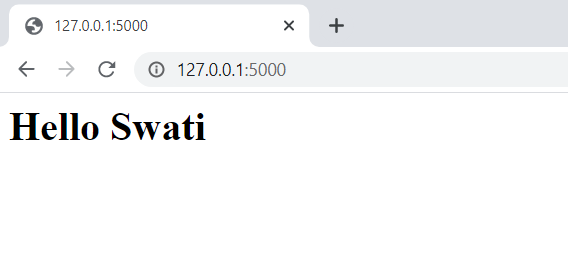
Basic Hello World Code

****

* **1.** From flask package import Flask class
* **2.** app object is created as an instance of class Flask
* **3.** Route decorate ,parameter define ,where page should be routed

# OUTPUT



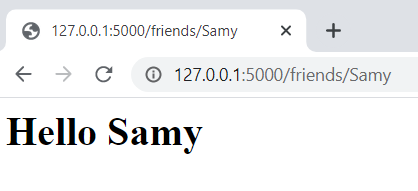


* **ADDING MORE ROUTES**

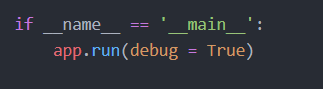
****

* **DYNAMIC ROUTING**
* We often need route extensions to be dynamic
* 2 Key Aspects :-
  + A variable in the route <variable>
  + A parameter passed to the function

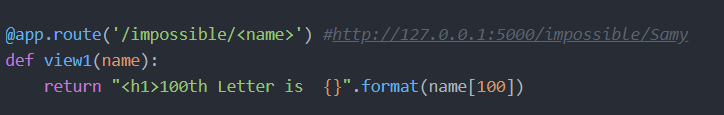




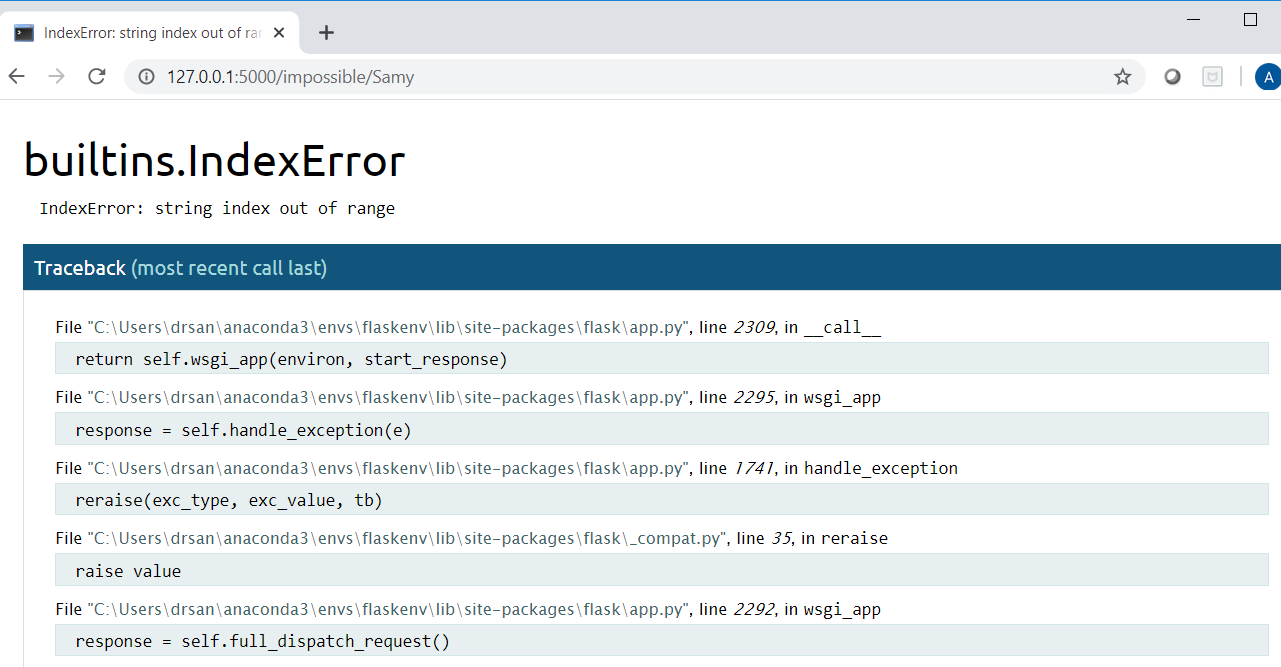
* **ENABLING DEBUG**



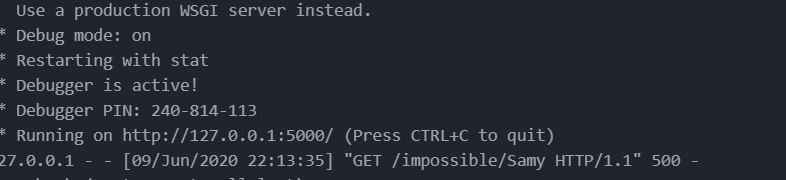
* Example:

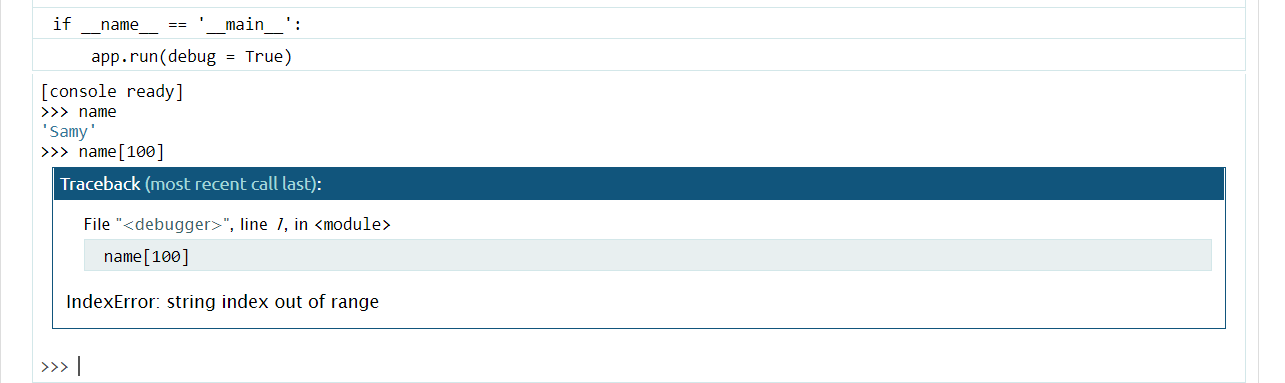


* Here if length is less than 100, without the “debug=True” argument, app will give **Internal Server error**, with debug parameter ,It will give error console with error details.



* It also provides an interactive python console but for that we need **a debugger pin** which we will get in the console/prompt used to run the app



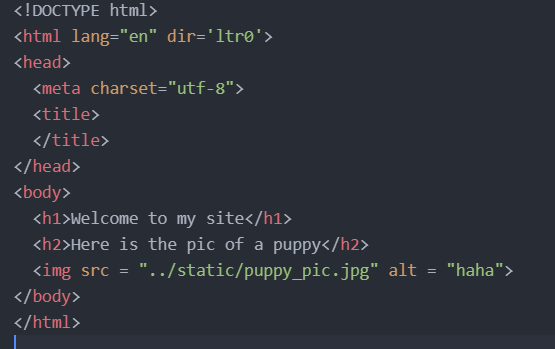


* NOTE:- “Debug = true” should only be used in Non Production Mode

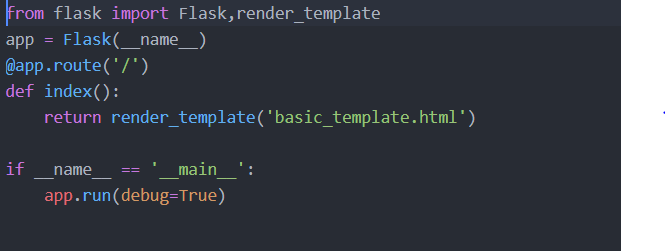
**TEMPLATE**

* We can use html files from templates folder
* Flask will automatically look for HTML Templates in the “***templates***” directory
* We can render templates simply by importing the “**render\_template**” function from flask returning an .html file from our view function
* Steps to be followed while executing templates :
  + Create a folder called ***templates*** in the working directory
  + Add the HTML file into the templates
  + Write code to execute it

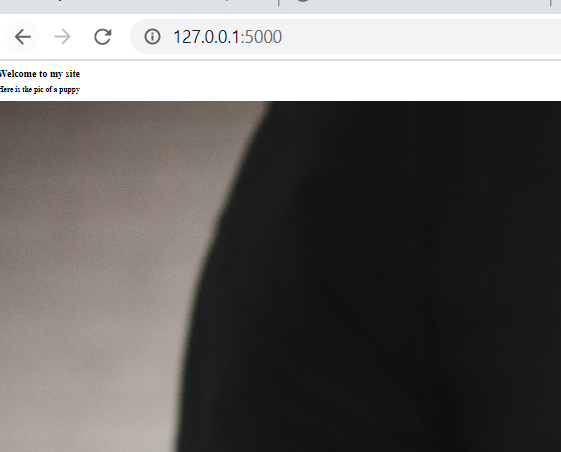
HTML:



Implementation Code:



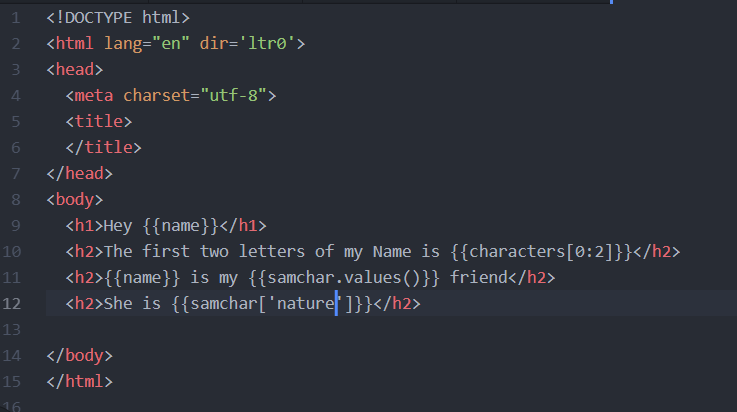
Output:



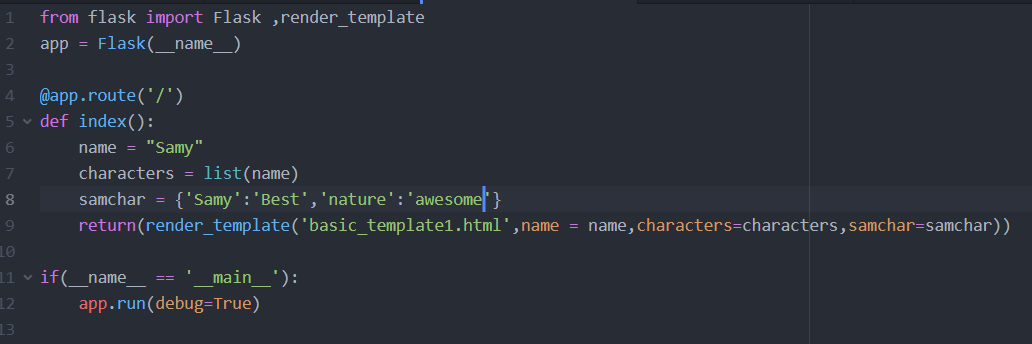
Template Variable

* ***Jinja*** template Engine is used to be able to use Python Code in our app, Changing and updating variables and logic and then send that information to the template
* Jinja template will let us directly insert variables like string ,list etcfrom our Python Code to HTML File
* Syntax for Inserting Variable:
  + {{some\_variable}}
* We set the parameter in render\_template function and then use {{ }} syntax to insert them in template

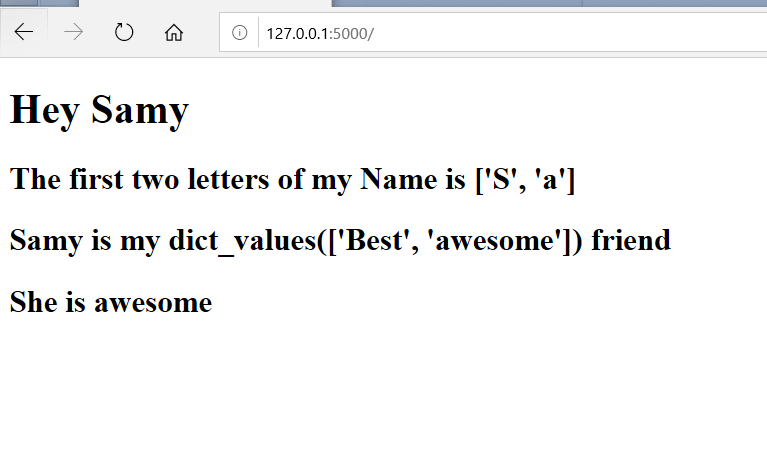
HTML:



PYTHON CODE:



OUTPUT:



Template Control Flow

* We also have access to control flow syntax in our Templates such as for loops and if
* Syntax:
  + {% %}
* Example:- Print each element of List as a bulleted html list and check if an element, ’e’ is present in it

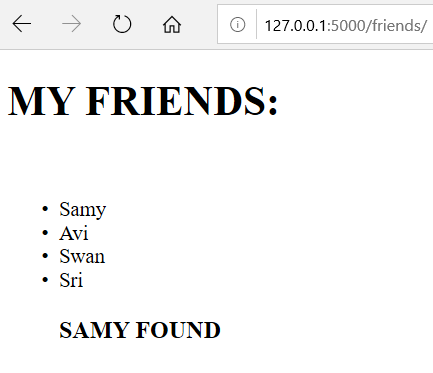
HTML:



PYTHON CODE:



OUTPUT:

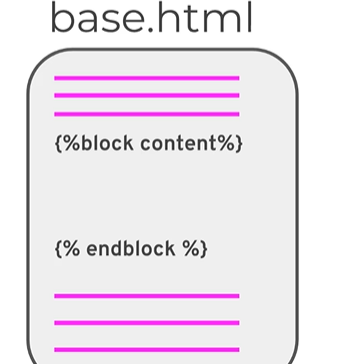


Template Inheritance

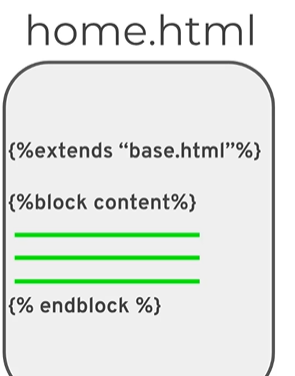
* We can have base HTML File usable in different templates
* Syntax Used:
  + **{% extend “base.html” %}**



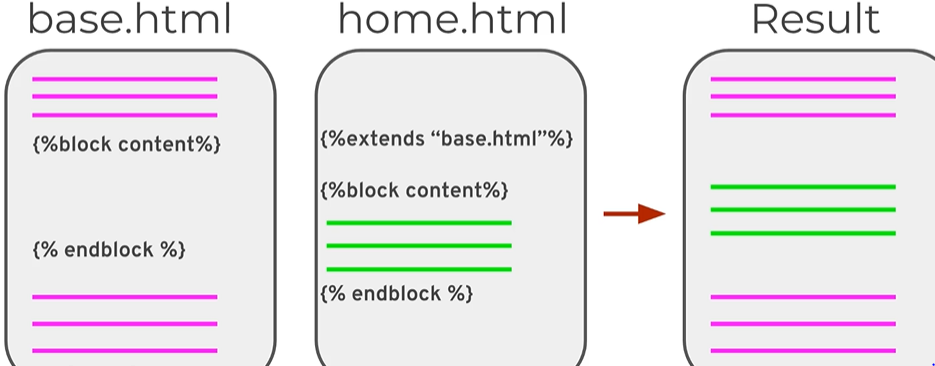
* + **{% block <blockname> %}**
* Base HTML File contains reusable aspects of the code
* Syntax:



* Implementation File:
* Syntax

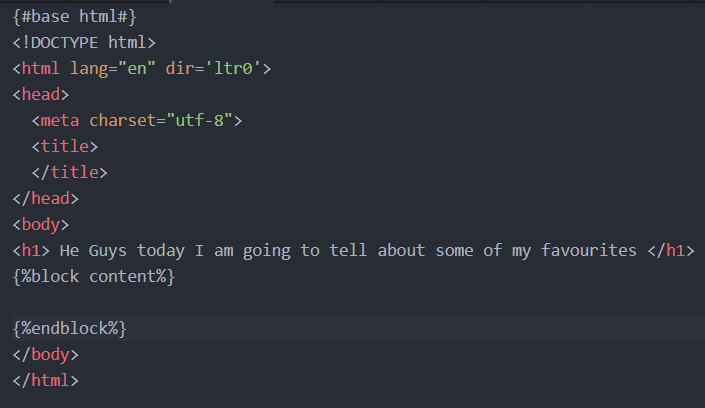


* Final Implementation:

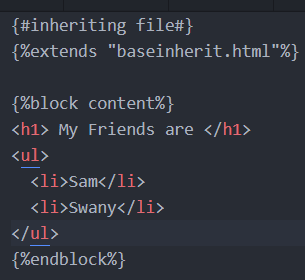
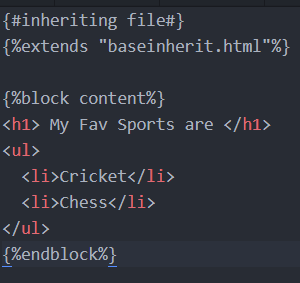


**Example:**

Baseinherit.html code:



Inherited code:

Python Code:



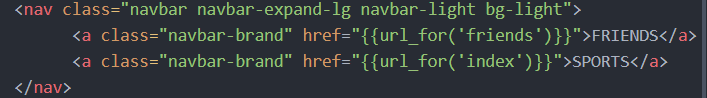
* **Filter Function:**
  1. These are a way to quickly change/edit a variable passed to a template
  2. Many a time ,it shares the same name as python method
  3. **Syntax:**
     + {{variable|filter}}
  4. **Examples:**
     + {{name}}

O/p samay

* + - {{name|capitalize}}

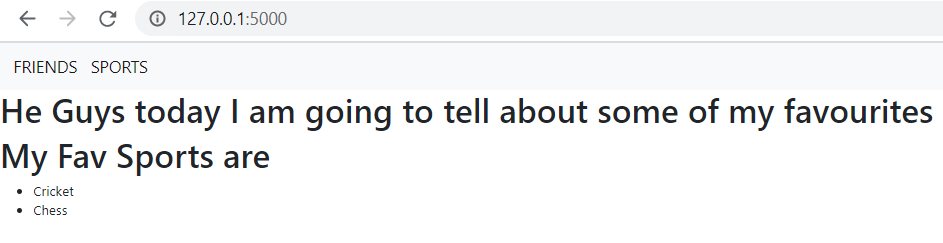
O/p Samay

* **url\_for**
  1. We can route a <a></a> link to another page using url\_for method
  2. **Syntax**
     + **{{url\_for(<method name>)}}**
  3. **Examples:**
     + {{url\_for('friends')}}



**Note:** No change in python Code

* **O/P**:



🡪Clicking on friends will route to Sports

* We can use “url\_for” to directly access a file
  1. **Syntax:**
     + **{{url\_for(<folder\_name>,filename =<file\_name>)}}**
  2. **Example**
     + **{{url\_for("static",filename ="puppy\_pic.jpg")}}**

Fetch Template Form Parameters

* We can fetch the values given in a form from “request” method
* Syntax:
  1. **request.args.get(<name>)**

**Note: HTML: for attribute is used to link label with Id**

* **Python Code:**

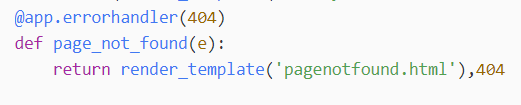


* **Form Html:**

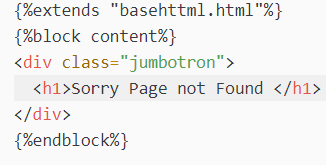
Add 404 Error Handler Page Not Found

* We create a page template for 404 Error and link it to error handler in python Code

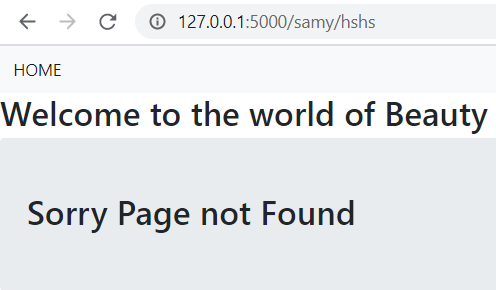
**Python Code**



**Html Code:**



**Webpage:**

* 

**FLASK FORMS**

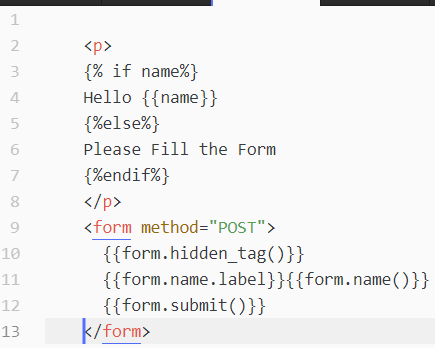
* We use flask\_wtf and wtforms packages to create form from our flask Python Scripts.
* Components to create a form:
  1. Configure a secret key for Security
     + Syntax:
       - **app.config[SECRET\_KEY] = “<secret>”**
       - This key is usually set up through Environment Variable

**Note: config is configuration dictionary of Flask**

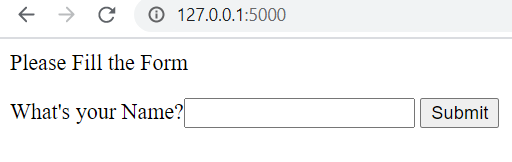
* 1. Create a wtform class.
     + Create fields for each attribute from the form
  2. Set up a view function:
     + Add methods = [‘GET’,’POST’]
     + Create an instance of Form Class
     + Handle Form Submission
* **Python Code:**

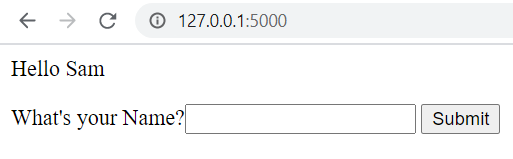


* + - form.validate\_on\_submit() 🡪This verifies if the form was submitted successfully
* **HTML CODE:**



* + - The logic used here is that if name is there in form it will return “Hello <Name>” ,if the name is not there It will return “Please Fill the Form”
    - <form> tag is used to render the form
    - {{form.hidden\_tag()}} is used for SECRET KEY
    - Similarly fields like StringField has two fields, **label and the actual field**
    - {{form.name(
* **OUTPUT:**





**Form Fields and Validators**

* Every HTML form field has a corresponding wtforms class.
* Has Validators ,which can be used to validate form Data
* We can use **session** object to grab the info provided in the form and pass it to another template
  + - session can be used to pass data into another template automatically
    - data is stored in the Server and is stored till users logs out
* **Code:**

1. from flask import Flask,render\_template,session,redirect,url\_for
2. from flask\_wtf import FlaskForm
3. from wtforms import (StringField,PasswordField,SelectField,BooleanField,

RadioField,TextAreaField,DateTimeField,SubmitField)

1. from wtforms.validators import DataRequired,Email,Length
2. app = Flask(\_\_name\_\_)
3. app.config['SECRET\_KEY'] = "Sameekshya"
4. class InfoForm(FlaskForm):
5. Name = StringField("Please Provide your Name",validaters =[DataRequired()])
6. bff = SelectField("Please Select your Best Friend" ,

choices =[("Samy","Sameekshya"),("Sri","Sriya"),("Swany","Swarna"),("Avi","Avinash")])

**#choices Syntax (<name>,<label>)**

1. gender =RadioField("Please Fill your Gender",choices =[("m","male"),("f","female")])
2. Username = StringField("Please provide the username",

validators = [Length(min=3,max=8,message="Length must be greater and 3 and less that 8"),DataRequired()])

1. Password = PasswordField("Please Enter your Password")
2. Email = StringField("Please Enter your Email",validators =[Email(),DataRequired()])
3. Bio = TextAreaField("Please Give more details about yourself")
4. Submit = SubmitField("Submit")
5. @app.route('/',methods=['GET','POST'])
6. def index():
7. form = InfoForm()
8. if form.validate\_on\_submit():

**#validate\_on\_submit is used to validate the validators**

1. session['Name'] = form.Name.data
2. session['bff'] = form.bff.data
3. session['gender'] = form.gender.data
4. session['Username'] = form.Username.data
5. session['Password'] = form.Password.data
6. session['Email']= form.Email.data
7. session['Bio'] = form.Bio.data
8. return redirect(url\_for("thankyou "))

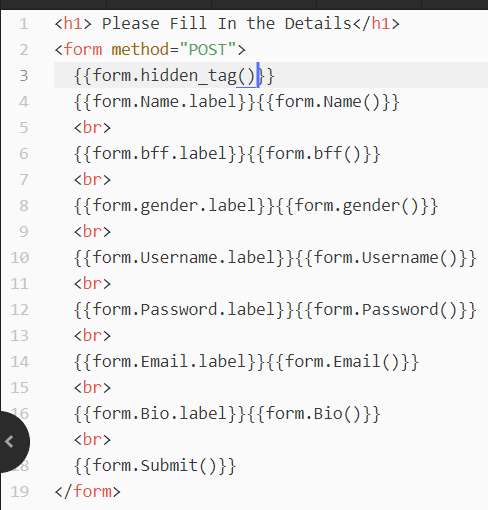
**#it will redirect to thank you page if the form is validated properly**

1. return render\_template("formfieldindex.html",form = form)
2. @app.route('/thankyou')
3. def thankyou():
4. return render\_template("thankyou.html")
5. if \_\_name\_\_ == "\_\_main\_\_":
6. app.run(debug = True)

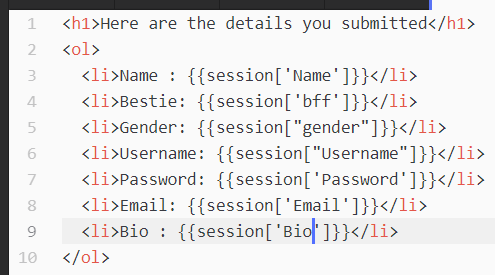
**Code Notes:**

* from wtforms.validators import DataRequired,Email,Length 🡪This is used to add validators to the fields
* validators = [Length(min=3,max=8,message="Length must be greater and 3 and less that 8"),DataRequired()] 🡪This verifies the length of user input and also verifies if data is entered
* Here the Validation will be checked ,but is will not give popup ,It will just check the validation and will not allow to submit

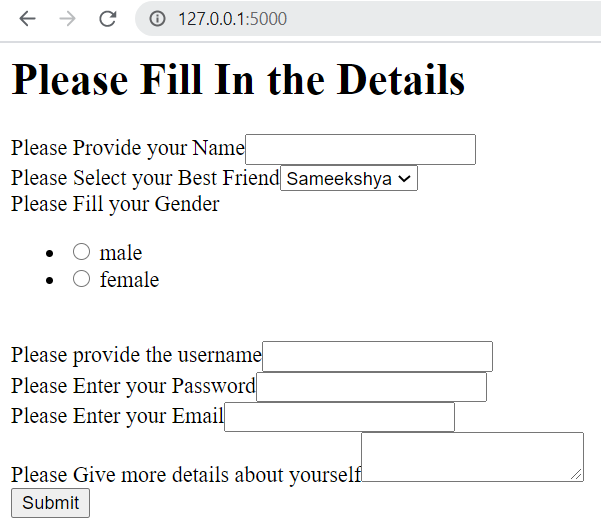
**HTML CODE(FORM):**



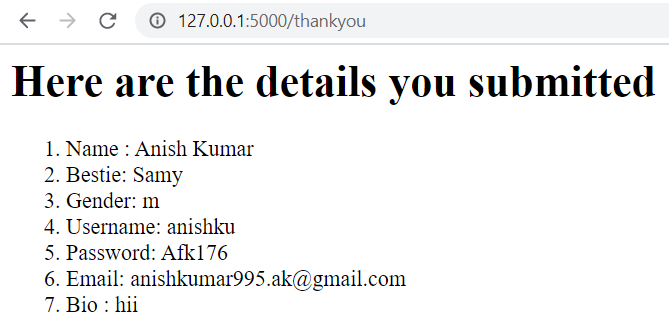
**HTML CODE(thankyou page)**



**Browser Output(Form):**



**Browser Output(thankyou page):**



**Flash Alerts**

* Flash alerts can be used when when we want to send a message to the user that we don’t need to save or fix permanently to Template Page
* Steps:
  + Import **flash** function from flask
  + Link bootstrap with it