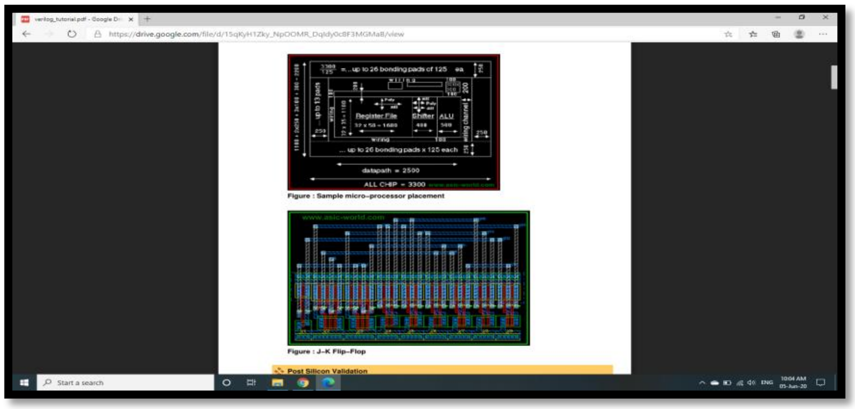
DAILY ASSESSMENT

Date:	**5/06/2020**	**Name:**	**Davis S. Patel**
Course:	**Digital Design Using HDL**	**USN:**	**4AL16EC45**
Topic:	**Verilog Tutorials and practice programs, Building/ Demo projects using FPGA , Implement a Verilog module to count number of 0's in a 16 bit number in the compiler.**	**Semester & Section:**	**8th - A**
GitHub Repository:	**Davis**		

FORENOON SESSION DETAILS

Image of session

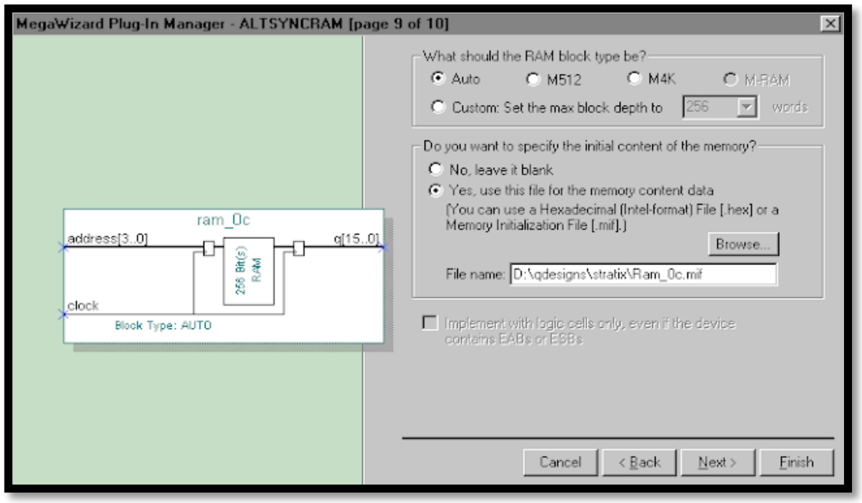


verilog_tutorial.pdf - Google Dr
https://drive.google.com/file/d/15qKyH1Zky_NpDOMR_Dqldy0c8F3MGMa8/view
...up to 26 bonding pads of 125 ea
Register File
Shifter
ALU
... up to 26 bonding pads x 125 each
datapath = 2500
ALL CHIP = 3300
www.asic-world.com
Start a search

Figure : Sample micro-processor placement

Figure : J-K Flip-Flop

Post Silicon Validation



MegaWizard Plug-In Manager - ALTSYNCRAM [page 9 of 10]

What should the RAM block type be?

[✓ Auto] M512 M4K M-RAM

Custom: Set the max block depth to 256 words

Do you want to specify the initial content of the memory?

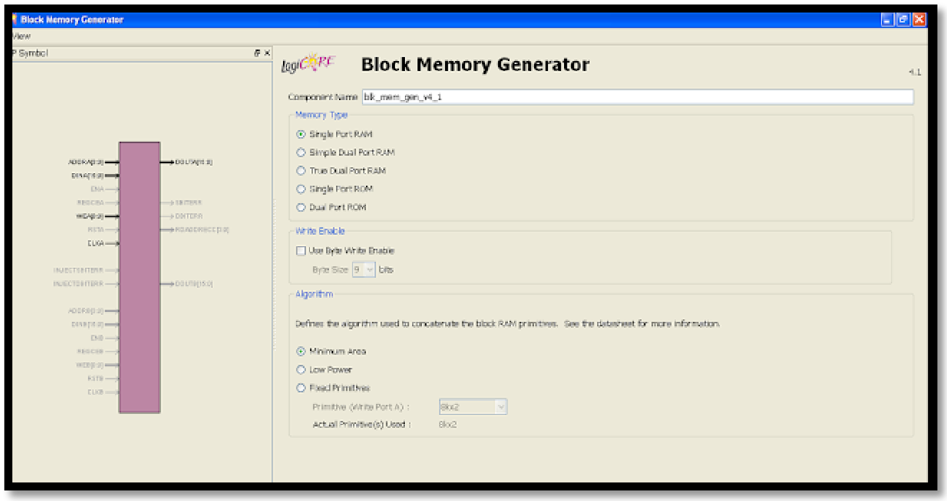
No, leave it blank

[✓ Yes, use this file for the memory content data]

(You can use a Hexadecimal (Intel-format) File [.hex] or a Memory Initialization File [.mif].)

File name: D:\qdesigns\stratix\Ram_0c.mif

Implement with logic cells only, even if the device contains EABs or ESBs



Block Memory Generator

Component Name: blk_mem_gen_v4_1

Memory Type
- [✓ Single Port RAM]
- Simple Dual Port RAM
- True Dual Port RAM
- Single Port ROM
- Dual Port ROM

Write Enable
- Use Byte Write Enable
- Byte Size: 9 bits

Algorithm

Defines the algorithm used to concatenate the block RAM primitives. See the datasheet for more information.

- [✓ Minimum Area]
- Low Power
- Fixed Primitives

Primitive (Write Port A) : 8kx2

Actual Primitive(s) Used : 8kx2

REPORT -

A hardware description Language Is a language used to describe a digital system, for example, a network switch, a microprocessor or a memory or a simple flip–flop. This just means that, by using a HDL one can describe any hardware (digital) at any level.
One can describe a simple Flip flop as that in above figure as well as one can describe a Complicated designs having 1 million gates. Verilog is one of the HDL languages available in the industry for designing the Hardware. Verilog allows us to design a Digital design at Behavior Level, Register Transfer Level (RTL), and Gate level and at switch level. Verilog allows hardware designers to express their designs with behavioral constructs, deterring the details of implementation to a later stage of design in the final design.

Design Styles:

- Top Up Design
- Bottom Up Design

Abstract Level of Verilog

- Behavioral Level

  This level describes a system by concurrent algorithms (Behavioral). Each algorithm itself is sequential, that means it consists of a set of instructions that are executed one after the other. Functions, Tasks and Always blocks are the main elements. There is no regard to the structural realization of the design.

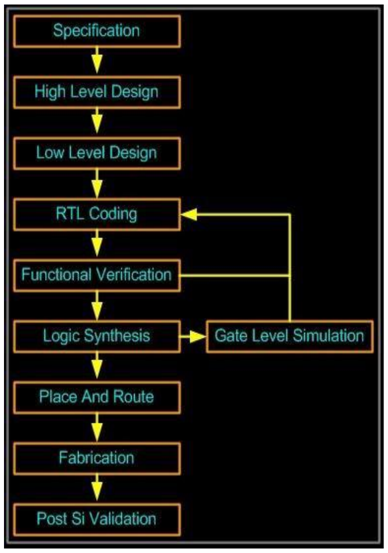
- Register Transfer Level

  Designs using the Register−Transfer Level specify the characteristics of a circuit by operations and the transfer of data between the registers. An explicit clock is used. RTL design contains exact timing possibilities, operations are scheduled to occur at certain times. Modern definition of a RTL code is "Any code that is synthesizable is called RTL code".

- Gate Level

  Within the logic level the characteristics of a system are described by logical links and their timing properties. All signals are discrete signals. They can only have definite logical values (`0', `1', `X', `Z`). The usable operations are predefined logic primitives (AND, OR, NOT etc. gates). Using gate level modeling might not be a good idea for any level of logic design. Gate level code is generated by tools like synthesis tools and this netlist is used for gate level simulation and for backend.

Typical Design Flow



Specification:

This is the stage at which we define what are the important parameters of the system/design that you are planning to design. Simple example would be, like I want to design a counter, it should be 4 bit wide, should have synchronous reset, with active high enable, When reset is active, counter output should go to "0". You can use Microsoft Word, or GNU Abiword or Open office for entering the specification.

High Level Design

This is the stage at which you define various blocks in the design and how they communicate. Let's assume that we need to design a microprocessor, High level design means splitting the design into blocks based on their function, In our case various blocks are registers, ALU, Instruction Decode, Memory Interface, etc. You can use Microsoft Word, or KWriter or Abiword or Openoffice for entering high level design.

Micro Design /Low Level Design

Low level design or Micro design is the phase in which, designer describes how each block is implemented. It contains details of State machines, counters, Mux, decoders, internal registers. For state machine entry you can use either Word, or special tools like StateCAD. It is always a good idea if waveform is drawn at various interfaces. This is the phase, where one spends a lot of time.

RTL Coding

In RTL coding, Micro Design is converted into Verilog/VHDL code, using synthesizable constructs of the language. Normally we use vim editor, but I prefer conTEXT and Nedit editor, it all depends on which editor you like. Some use Emacs.

Simulation

Simulation is the process of verifying the functional characteristics of models at any level of abstraction. We use simulators to simulate the Hardware models. To test if the RTL code meets the functional requirements of the specification, see if all the RTL blocks are functionally correct. To achieve this we need to write a testbench, which generates clk, reset and required test vectors. A sample testbench for a counter is as shown below. Normally we spend 60−70% of time in verification of design.

Synthesis

Synthesis is a process in which synthesis tools like a design compiler or Synplify takes the RTL in Verilog or VHDL, target technology, and constraints as input and maps the RTL to target technology primitives. Synthesis tool after mapping the RTL to gates, also does the minimal amount of timing analysis to see if the mapped design meets the timing requirements.

Place & Route

Gate Level netlist from the synthesis tool is taken and imported into place and route tool in Verilog netlist format. All the gates and flip–flops are places, Clock tree synthesis and reset is routed. After this each block is routed. Output of the P&R tool is GDS file, this file is used by foundry for fabricating the ASIC. Normally the P&R tool are used to output the SDF file, which is back annotated along with the gate level netlist from P&R into static analysis tool like Prime Time to do timing analysis.

```
module delay_example();

wire out1,out2,out3,out4,out5,out6;
reg b,c;

// Delay for all transitions
or #5 u_or (out1,b,c);
// Rise and fall delay
and #(1,2) u_and (out2,b,c);
// Rise, fall and turn off delay
nor #(1,2,3) u_nor (out3,b,c);
//One Delay, min, typ and max
nand #(1:2:3) u_nand (out4,b,c);
//Two delays, min,typ and max
buf #(1:4:8,4:5:6) u_buf (out5,b);
//Three delays, min, typ, and max
notif1 #(1:2:3,4:5:6,7:8:9) u_notif1 (out6,b,c);
```

//Testbench code

```
initial begin
 $monitor ( "Time = %g b = %b c=%b out1=%b out2=%b out3=%b out4=%b out5=%b out6=%b" ,
$time, b, c ,
out1, out2, out3, out4, out5, out6);
 b = 0;
 c = 0;
 #10 b = 1;
 #10 c = 1;
 #10 b = 0;
 #10 $finish;
end

endmodule
```

Verilog Operators

```
module arithmetic_operators();

initial begin
 $display ( " 5 + 10 = %d" , 5 + 10);
 $display ( " 5 − 10 = %d" , 5 − 10);
 $display ( " 10 − 5 = %d" , 10 − 5);
 $display ( " 10 * 5 = %d" , 10 * 5);
 $display ( " 10 / 5 = %d" , 10 / 5);
 $display ( " 10 / −5 = %d" , 10 / −5);
 $display ( " 10 %s 3 = %d" ,
 $display ( " +5 = %d" , +5);
 $display ( " −5 = %d" , −5);
 #10 $finish;
end

endmodule
```

Logical Operator

```
module logical_operators();

initial begin
 // Logical AND
 $display ( "1'b1 && 1'b1 = %b" , (1'b1 && 1'b1));
 $display ( "1'b1 && 1'b0 = %b" , (1'b1 && 1'b0));
 $display ( "1'b1 && 1'bx = %b" , (1'b1 && 1'bx));
 // Logical OR
 $display ( "1'b1 || 1'b0 = %b" , (1'b1 || 1'b0));
 $display ( "1'b0 || 1'b0 = %b" , (1'b0 || 1'b0));
 $display ( "1'b0 || 1'bx = %b" , (1'b0 || 1'bx));

 // Logical Negation
 $display ( "! 1'b1 = %b" , (! 1'b1));
 $display ( "! 1'b0 = %b" , (! 1'b0));
 #10 $finish;
end

endmodule
```

Bit Wise Operator

```
module bitwise_operators();

initial begin
 // Bit Wise Negation
 $display ( " ~4'b0001 = %b" , (~4'b0001));
 $display ( " ~4'bx001 = %b" , (~4'bx001));
 $display ( " ~4'bz001 = %b" , (~4'bz001));
 // Bit Wise AND
 $display ( " 4'b0001 & 4'b1001 = %b" , (4'b0001 & 4'b1001));
 $display ( " 4'b1001 & 4'bx001 = %b" , (4'b1001 & 4'bx001));
 $display ( " 4'b1001 & 4'bz001 = %b" , (4'b1001 & 4'bz001));
 // Bit Wise OR
 $display ( " 4'b0001 | 4'b1001 = %b" , (4'b0001 | 4'b1001));
 $display ( " 4'b0001 | 4'bx001 = %b" , (4'b0001 | 4'bx001));
 $display ( " 4'b0001 | 4'bz001 = %b" , (4'b0001 | 4'bz001));
 // Bit Wise XOR
 $display ( " 4'b0001 ^ 4'b1001 = %b" , (4'b0001 ^ 4'b1001));
 $display ( " 4'b0001 ^ 4'bx001 = %b" , (4'b0001 ^ 4'bx001));
 $display ( " 4'b0001 ^ 4'bz001 = %b" , (4'b0001 ^ 4'bz001));
 // Bit Wise XNOR
 $display ( " 4'b0001 ~^ 4'b1001 = %b" , (4'b0001 ~^ 4'b1001));
 $display ( " 4'b0001 ~^ 4'bx001 = %b" , (4'b0001 ~^ 4'bx001));
 $display ( " 4'b0001 ~^ 4'bz001 = %b" , (4'b0001 ~^ 4'bz001));
 #10 $finish;
end

endmodule
```

Behavioral Modeling

```
module avoid_latch_else ();


reg q;
reg enable, d;


always @ (enable or d)


if (enable) begin
 q = d;
end else begin
 q = 0;
end
```

```
initial begin
 $monitor ( " ENABLE = %b D = %b Q = %b" ,enable,d,q);
 #1 enable = 0;
 #1 d = 0;
 #1 enable = 1;
 #1 d = 1;
 #1 d = 0;
 #1 d = 1;
 #1 d = 0;
 #1 d = 1;
 #1 enable = 0;
 #1 $finish;
end
endmodule
```

Task and Function

```
module task_calling (temp_a, temp_b, temp_c, temp_d);
input [7:0] temp_a, temp_c;
output [7:0] temp_b, temp_d;
reg [7:0] temp_b, temp_d;
`include "mytask.v"
always @ (temp_a)
begin
 convert (temp_a, temp_b);
end
always @ (temp_c)
begin
 convert (temp_c, temp_d);

end
endmodule
```

Test Bench code

```
module counter_tb;
reg clk, reset, enable;
wire [3:0] count;

counter U0 (
clk (clk),
reset (reset),
enable (enable),
count (count)
);
initial
begin
 clk = 0;
 reset = 0;
 enable = 0;
end
always
#5 clk = !clk;
endmodule
```

Task 5 -Implement a verilog module to count number of 0's in a 16 bit number in compiler.

```
module num_zeros_for(
   input [15:0] A,
   output reg [4:0] ones
   );

integer i;

always@(A)
begin
   ones = 0;
   for(i=0;i<16;i=i+1)
      if(A[i] == 0'b1)
         ones = ones + 1;
```

```
end

endmodule
```

<u>output</u>

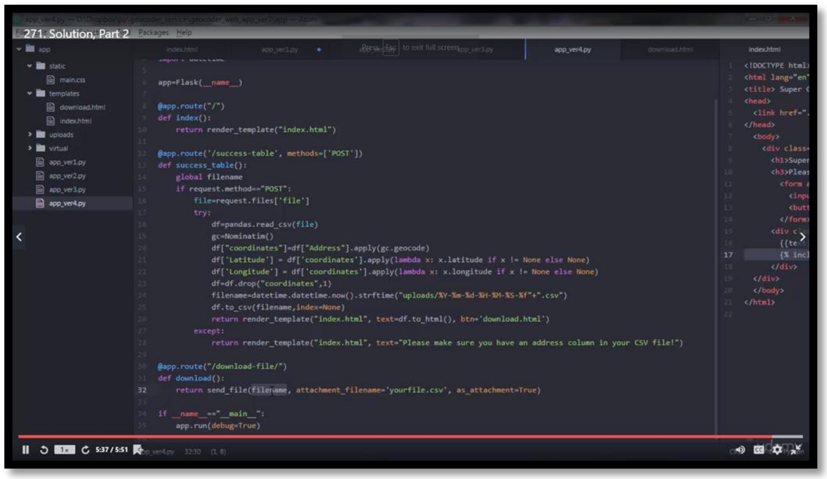
```
Input = "1010_0010_1011_0010" =>  Output = "01001" ( 9 in decimal)
Input = "0011_0110_1000_1011" =>  Output = "01000" ( 8 in decimal)
```

DAILY ASSESSMENT

Date:	**5/06/2020**	**Name:**	**Davis S. Patel**
Course:	**Python Course**	**USN:**	**4AL16EC45**
Topic:	**Application 9: Build a Data Collector Web App with PostGreSQL and Flask**	**Semester & Section:**	**8th - A**
GitHub Repository:	**Davis**		

AFTERNOON SESSION DETAILS

Image Of Session



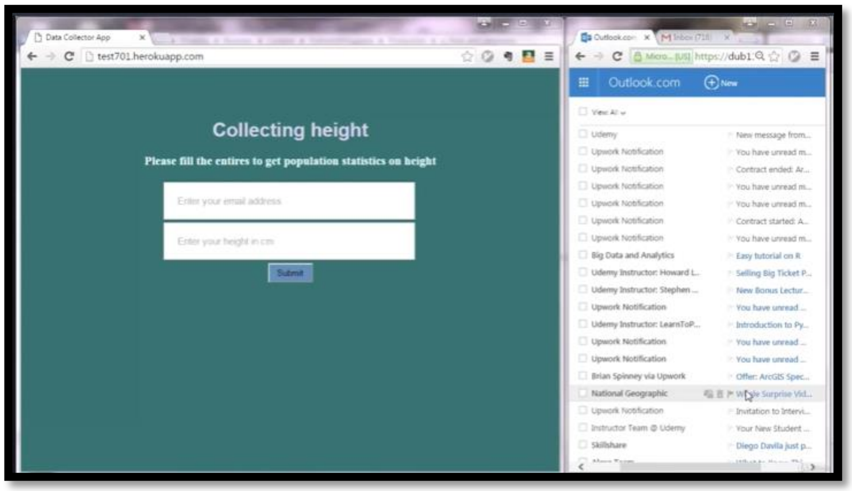
```
app=Flask(__name__)

@app.route("/")
def index():
    return render_template("index.html")

@app.route('/success-table', methods=['POST'])
def success_table():
    global filename
    if request.method=="POST":
        file=request.files['file']
        try:
            df=pandas.read_csv(file)
            gc=Nominatim()
            df["coordinates"]=df["Address"].apply(gc.geocode)
            df['Latitude'] = df['coordinates'].apply(lambda x: x.latitude if x != None else None)
            df['Longitude'] = df['coordinates'].apply(lambda x: x.longitude if x != None else None)
            df=df.drop("coordinates",1)
            filename=datetime.datetime.now().strftime("uploads/%Y-%m-%d-%H-%M-%S-%f"+".csv")
            df.to_csv(filename,index=None)
            return render_template("index.html", text=df.to_html(), btn='download.html')
        except:
            return render_template("index.html", text="Please make sure you have an address column in your CSV file!")

@app.route("/download-file/")
def download():
    return send_file(filename, attachment_filename='yourfile.csv', as_attachment=True)

if __name__=="__main__":
    app.run(debug=True)
```



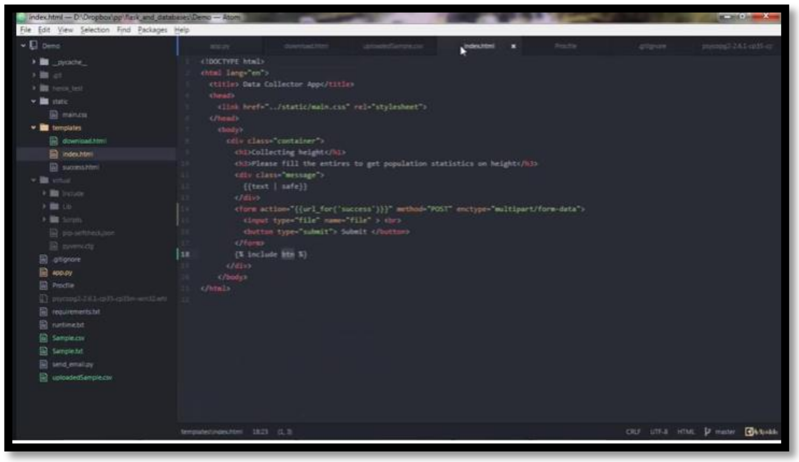
Collecting height

Please fill the entires to get population statistics on height

Enter your email address

Enter your height in cm

Submit



```
<!DOCTYPE html>
<html lang="en">
  <title> Data Collector App</title>
  <head>
    <link href="../static/main.css" rel="stylesheet">
  </head>
    <body>
      <div class="container">
        <h1>Collecting height</h1>
        <h3>Please fill the entires to get population statistics on height</h3>
        <div class="message">
          {{text | safe}}
        </div>
        <form action="{{url_for('success')}}" method="POST" enctype="multipart/form-data">
          <input type="file" name="file" > <br>
          <button type="submit"> Submit </button>
        </form>
        {% include btn %}
      </div>
    </body>
</html>
```

REPORT –

Creating an API or Web application using python has been made easy with Flask. It is a micro web framework written in Python.

Here you will create a python server using Flask, create database with PostgreSQL and deploy it on Heroku.

We will create a simple application to store details of books and get stored data to demonstrate database transactions with our python server here.

So here we use,

- python
- Flask
- PostgreSQL
- Heroku CLI
- git

Steps we follow here,

1. Install PostgreSQL to local machine
2. Install Heroku CLI
3. Create python virtual environment for the project
4. Create a sample code with Flask to check
5. Create database
6. Create configurations
7. Database migration
8. Finish the code
9. Commit changes using git and push to Heroku

1. Install PostgreSQL to local machine

Follow this step if you already haven't installed PostgreSQL on your machine.

Install PostgreSQL in Linux using the command,

```
sudo apt-get install postgresql postgresql-contrib
```

Now create a superuser for PostgreSQL

```
sudo -u postgres createuser --superuser name_of_user
```

And create a database using created user account

```
sudo -u name_of_user createdb name_of_database
```

You can access created database with created user by,

```
psql -U name_of_user -d name_of_database
```

2. Install Heroku CLI

Method 1: Heroku CLI can be installed using ***snap***.

```
sudo snap install --classic heroku
```

Method 2: You can install Heroku CLI using this shell script.

```
curl https://cli-assets.heroku.com/install.sh | sh
```

After installing Heroku CLI using any method above, Verify installation by

```
heroku --version
```

Create a Heroku account if you have not one already *here* and login to your Heroku account in CLI by

```
heroku login
```

3. Create python virtual environment for the project

Why virtual environments?
Virtual environments allow you to make isolated python environment for different projects. It is helpful when several projects need one package in different versions. With virtual environments you can manage packages on current project without considering affecting other projects or without affecting Linux operating system.

To create virtual environments we need ***virtualenv*** package. Install python virtualenv package using,

```
pip install virtualenv
```

Then create a new directory for the project. Let's say ***books_server***

```
mkdir books_server
cd books_server
```

Create a virtual environment named ***env*** inside the created directory by,

virtualenv env

This will create the virtual environment named ***env*** inside the ***books_server.***
To activate this environment use this command inside ***books_server*** directory.

source env/bin/activate

4. Create a sample code with Flask to check

For using Flask, first you need to install Flask. (Make sure that you have activated the virtual environment)

pip install Flask

Now create a file named ***app.py*** in ***books_server*** directory and put below code to test Flask before we move into real application development

to execute code run

python app.py

or

FLASK_APP=app.py flask run

5. Create database

First create the database we need here for our application named ***books_store***

sudo -u ***name_of_user*** createdb books_store

Now you can check the created database with,

psql -U ***name_of_user*** -d books_store

You should log into ***books_store*** data base if above command was success.

6. Create configurations

We need to define configurations for deploying environments. create a file named ***config.py*** with b code.

According to created configurations set ***"APP_SETTINGS"*** environment variable by running this in the terminal

```
export APP_SETTINGS="config.DevelopmentConfig"
```

Also add ***"DATABASE_URL"*** to environment variables. In this case our database URL is based on the created database. So, export the environment variable by this command in the terminal,

```
export DATABASE_URL="postgresql://localhost/books_store"
```

It should be returned when you execute echo $DATABASE_URL in terminal.
So, now our python application can get database URL for the application from the environment variable which is ***"DATABASE_URL"***

also put these 2 environment variables into a file called ***.env***