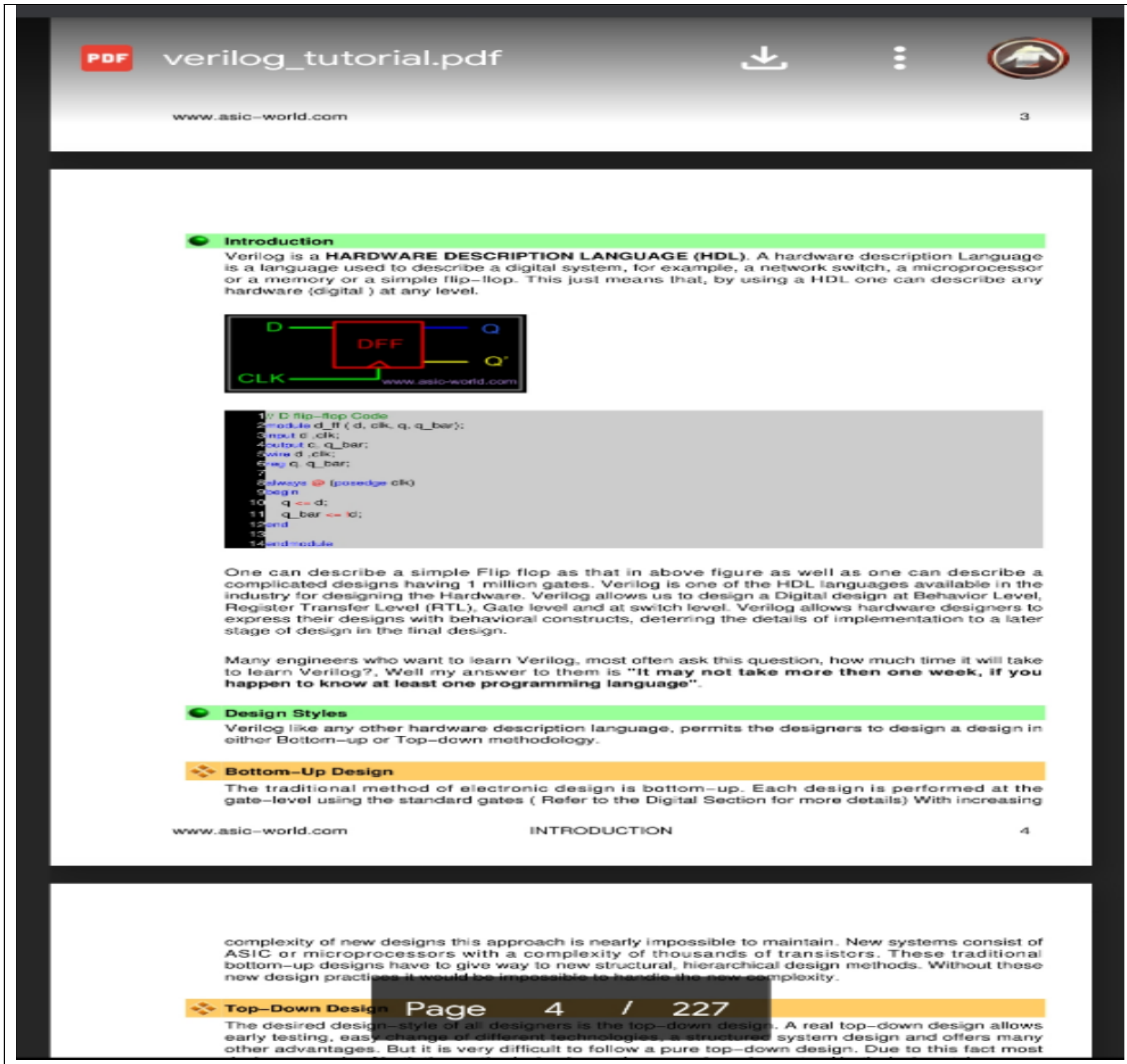
DAILY ASSESSMENT FORMAT

Date:	05-06-2020	Name:	Jagadeesha Hegde
Course:	Logic Design	USN:	4AL17EC036
Topic:	Verilog Tutorials and practice programs Building/ Demo projects using FPGA	Semester & Section:	6th A-sec
Github Repository:	Jagadeesha-036		

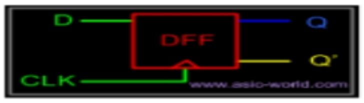
FORENOON SESSION DETAILS
Image of session



verilog_tutorial.pdf

Introduction

Verilog is a **HARDWARE DESCRIPTION LANGUAGE (HDL)**. 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.



```
// D flip–flop Code
module d_ff ( d, clk, q, q_bar);
input d ,clk;
output q, q_bar;
wire d ,clk;
reg q, q_bar;

always @ (posedge clk)
begin
  q <= d;
  q_bar <= !d;
end

endmodule
```

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), 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.

Many engineers who want to learn Verilog, most often ask this question, how much time it will take to learn Verilog?, Well my answer to them is **"It may not take more then one week, if you happen to know at least one programming language"**.

Design Styles

Verilog like any other hardware description language, permits the designers to design a design in either Bottom–up or Top–down methodology.

Bottom–Up Design

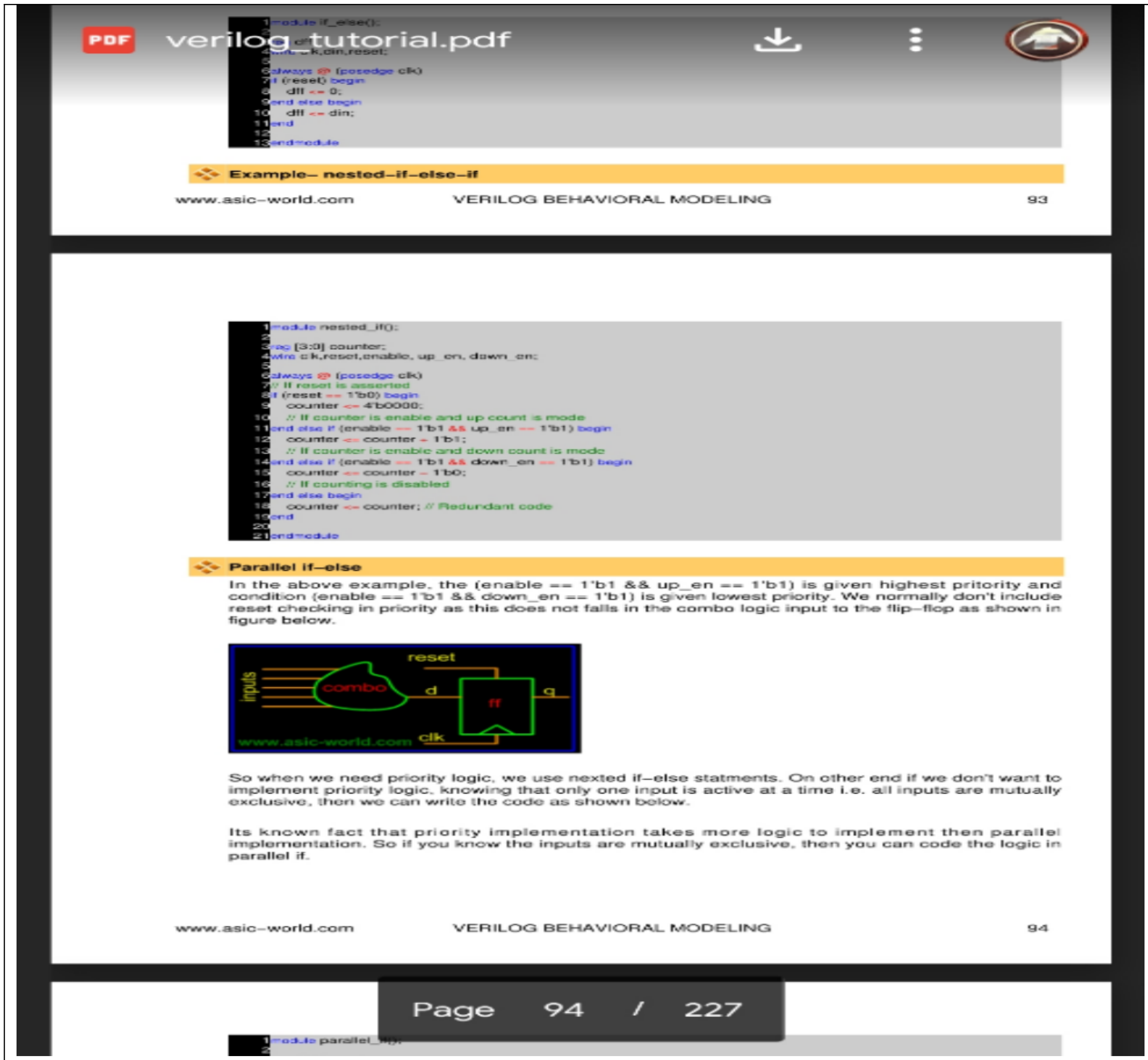
The traditional method of electronic design is bottom–up. Each design is performed at the gate–level using the standard gates (Refer to the Digital Section for more details) With increasing

complexity of new designs this approach is nearly impossible to maintain. New systems consist of ASIC or microprocessors with a complexity of thousands of transistors. These traditional bottom–up designs have to give way to new structural, hierarchical design methods. Without these new design practices it would be impossible to handle the new complexity.

Page 4 / 227

Top–Down Design

The desired design–style of all designers is the top–down design. A real top–down design allows early testing, easy change of different technologies, a structured system design and offers many other advantages. But it is very difficult to follow a pure top–down design. Due to this fact most



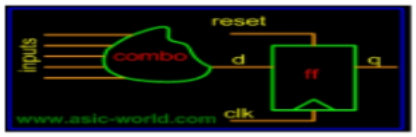
```
module if_else();

reg dff;
wire clk,din,reset;

always @ (posedge clk)
if (reset) begin
   dff <= 0;
end else begin
   dff <= din;
end

endmodule
```

Example– nested–if–else–if

```
module nested_if();

reg [3:0] counter;
wire clk,reset,enable, up_en, down_en;

always @ (posedge clk)
// If reset is asserted
if (reset == 1'b0) begin
   counter <= 4'b0000;
   // If counter is enable and up count is mode
end else if (enable == 1'b1 && up_en == 1'b1) begin
   counter <= counter + 1'b1;
   // If counter is enable and down count is mode
end else if (enable == 1'b1 && down_en == 1'b1) begin
   counter <= counter - 1'b0;
   // If counting is disabled
end else begin
   counter <= counter; // Redundant code
end

endmodule
```

Parallel if–else

In the above example, the (enable == 1'b1 && up_en == 1'b1) is given highest pritority and condition (enable == 1'b1 && down_en == 1'b1) is given lowest priority. We normally don't include reset checking in priority as this does not falls in the combo logic input to the flip–flop as shown in figure below.



So when we need priority logic, we use nexted if–else statments. On other end if we don't want to implement priority logic, knowing that only one input is active at a time i.e. all inputs are mutually exclusive, then we can write the code as shown below.

Its known fact that priority implementation takes more logic to implement then parallel implementation. So if you know the inputs are mutually exclusive, then you can code the logic in parallel if.

```
module parallel_if();
```

Report – Report can be typed or hand written for up to two pages.

Introduction

Verilog is a HARDWARE DESCRIPTION LANGUAGE (HDL). 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.

```
// D flip−flop Code

module d_ff ( d, clk, q, q_bar);

input d ,clk;

output q, q_bar;

wire d ,clk;

reg q, q_bar;

always @ (posedge clk)

begin

 q <= d;

 q_bar <= !d;

end

endmodule
```

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), 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.

Many engineers who want to learn Verilog, most often ask this question, how much time it will take to learn Verilog?, Well my answer to them is "It may not take more then one week, if you happen to know at least one programming language".

Design Styles Verilog like any other hardware description language, permits the designers to design a design in either Bottom−up or Top−down methodology.

Bottom−Up Design

The traditional method of electronic design is bottom–up. Each design is performed at the gate–level using the standard gates (Refer to the Digital Section for more details) With increasing complexity of new designs this approach is nearly impossible to maintain. New systems consist of ASIC or microprocessors with a complexity of thousands of transistors. These traditional bottom–up designs have to give way to new structural, hierarchical design methods. Without these new design practices it would be impossible to handle the new complexity.

Top–Down Design

The desired design–style of all designers is the top–down design. A real top–down design allows early testing, easy change of different technologies, a structured system design and offers many other advantages. But it is very difficult to follow a pure top–down design. Due to this fact most designs are mix of both the methods, implementing some key elements of both design styles.

Data Types

Verilog Language has two primary data types

• Nets – represents structural connections between components.

• Registers – represent variables used to store data.

Every signal has a data type associated with it:

• Explicitly declared with a declaration in your Verilog code.

Implicitly declared with no declaration but used to connect structural building blocks in your code.

•Implicit declaration is always a net type "wire" and is one bit wide.

Task code

```
module num_zero(input [15:0]A, output reg [4:0]zeros);
 integer i;
 always@(A)
  begin
```

```
   zeros=0;
   for(i=0;i<16;i=i+1)
    zeros=zeros+A[i];
  end
endmodule
```

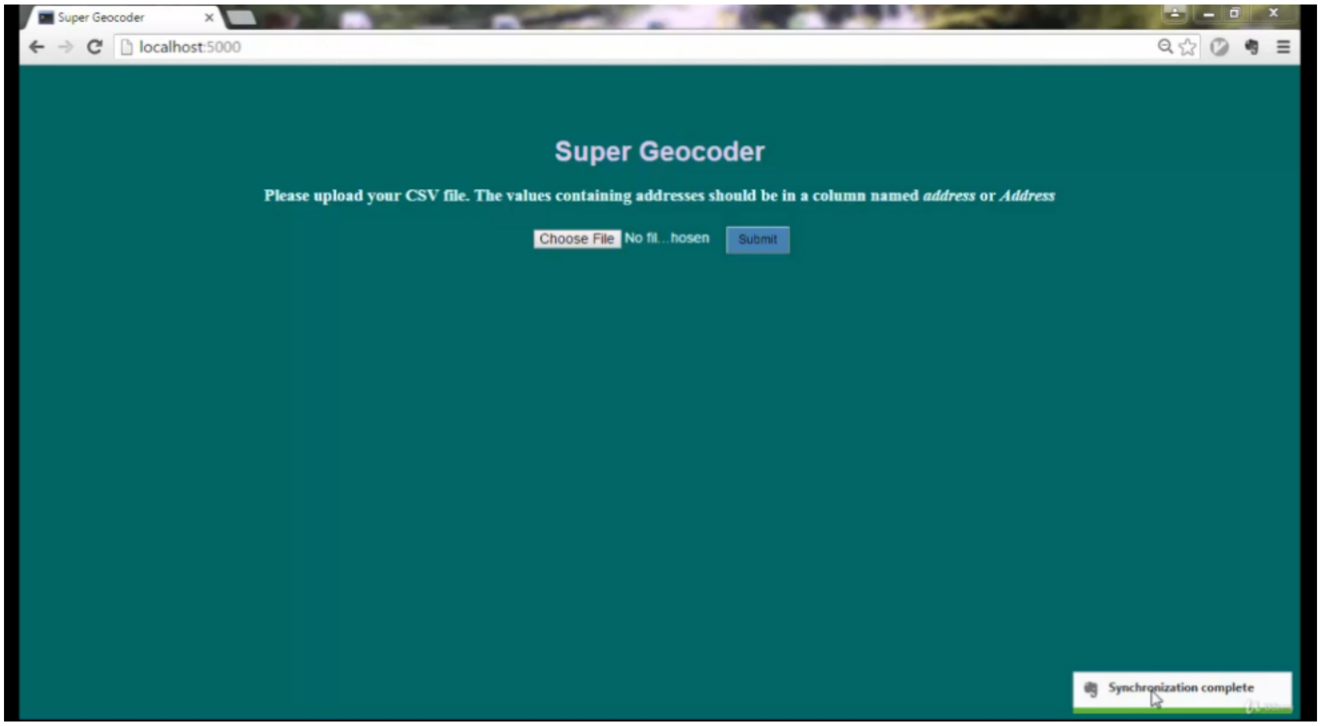
test bench code

```
module test;
 reg [15:0]A;
 wire [4:0] zeros;
 num_zero out (.A(A), .zeros(zeros));
 initial begin
  $dumpfile("dumo.vcd");
  $dumpvars(1,test);
  A=16'hFFFF; #100;
  A=16'hF56F; #100;
  A=16'h3FFF; #100;
  A=16'h0001; #100;
  A=16'hF10F; #100;
  A=16'hF822; #100;
  A=16'h7ABC; #100;
 end
endmodule
```

Date:	05-06-2020	Name:	Jagadeesha Hegde
Course:	The Python Mega Course	USN:	4AL17EC036
Topic:	Application 10: Build a Data Collector Web App with PostGreSQL and Flask	Semester & Section:	6th A-sec

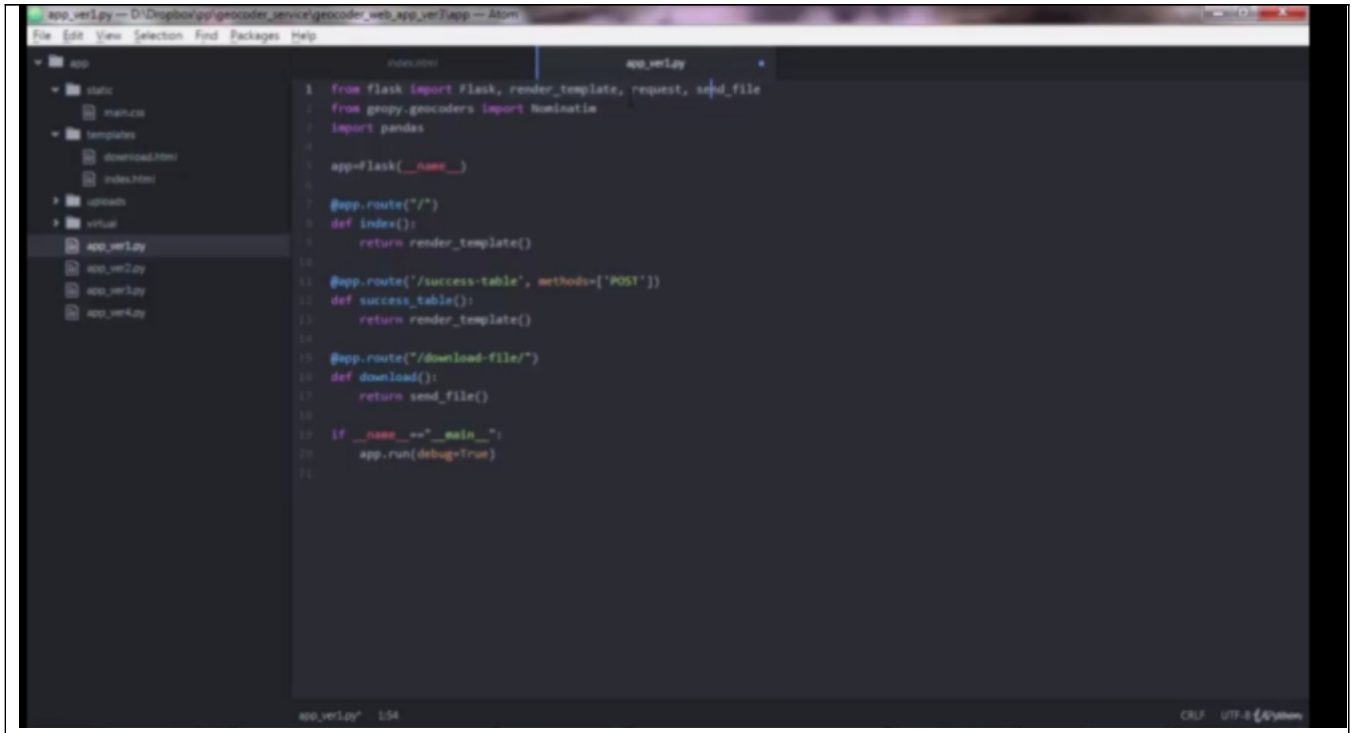
AFTERNOON SESSION DETAILS

Image of session



Super Geocoder

Please upload your CSV file. The values containing addresses should be in a column named *address* or *Address*



```python
from flask import Flask, render_template, request, send_file
from geopy.geocoders import Nominatim
import pandas

app=Flask(__name__)

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

@app.route('/success-table', methods=['POST'])
def success_table():
    return render_template()

@app.route("/download-file/")
def download():
    return send_file()

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

Report – Report can be typed or hand written for up to two pages.

Flask startup and configuration Like most widely used Python libraries, the Flask package is installable from the Python Package Index (PPI). First create a directory to work in (something like flask_todo is a fine directory name) then install the flask package. You'll also want to install flask-sqlalchemy so your Flask application has a simple way to talk to a SQL database.A good way to get moving is to turn the codebase into an installable Python distribution. At the project's root, create setup.py and a directory called todo to hold the source code. The setup.py should look something like this:

```
 requires = [
                 'flask',
                'flask-sqlalchemy',
                 'psycopg2',
              ]


setup(
          name='flask_todo',
          version='0.0',
          description='A To-Do List built with Flask',
          author='<Your actual name here>',
          author_email='<Your actual e-mail address here>',
          keywords='web flask',
         packages=find_packages(),
         include_package_data=True,
         install_requires=requires
     )
```

This way, whenever you want to install or deploy your project, you'll have all the necessary packages in the requires list. You'll also have everything you need to set up and install the package in sitepackages. For more information on how to write an installable Python distribution, check out the docs on setup.py.Within the todo directory containing your source code, create an app.py file and a blank __init__.py file. The __init__.py file allows you to import from todo as if it were an installed package. The app.py file will be the application's root. This is where all the Flask application goodness will go, and you'll create an environment variable that points to that file. If you're using

pipenv (like I am), you can locate your virtual environment with pipenv --venv and set up that environment variable in your environment's activate script.