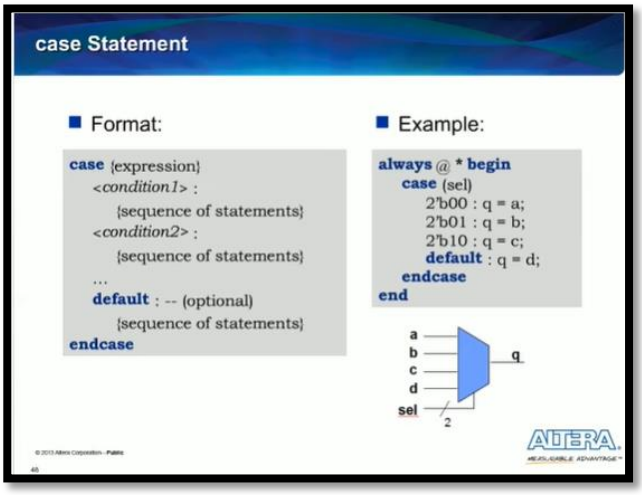
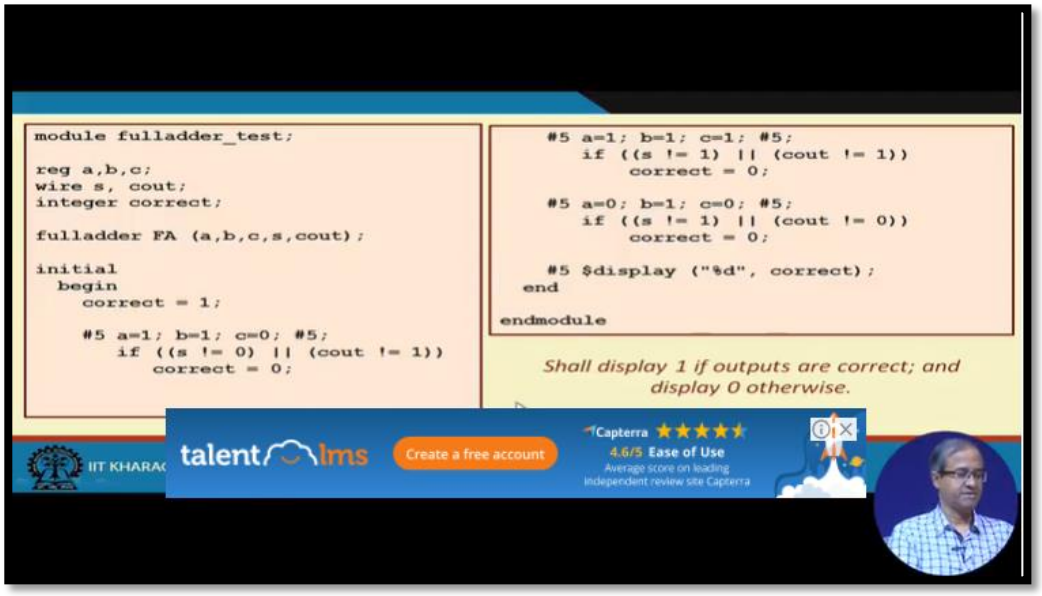
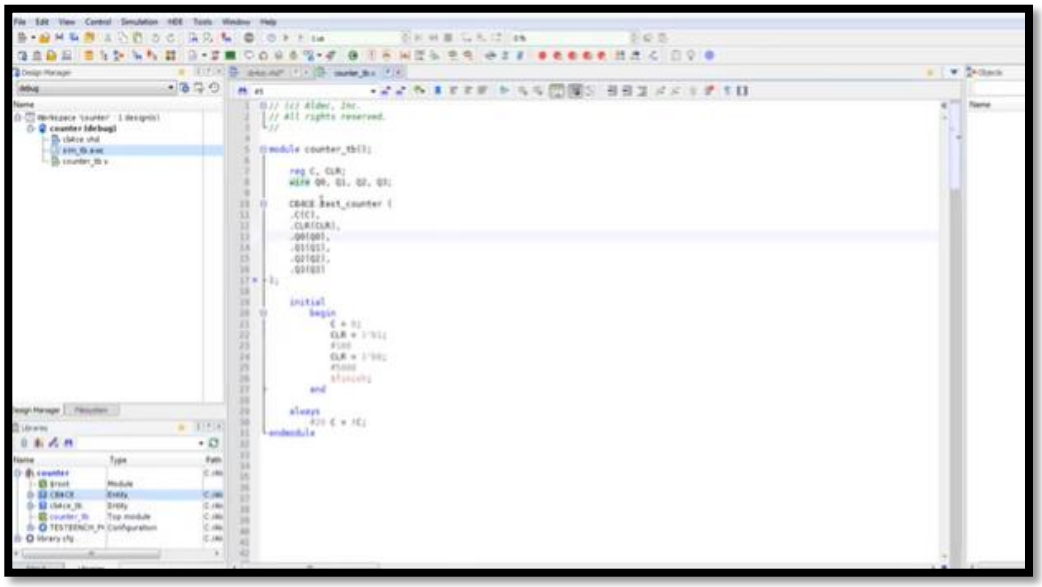
DAILY ASSESSMENT FORMAT

Date:	**2/06/2020**	**Name:**	**Davis S. Patel**
Course:	**DIGITAL DESIGN USING HDL**	**USN:**	**4AL16EC045**
Topic:	**FPGA Basics: Architecture, Applications and Uses Verilog HDL Basics by Intel Verilog Test bench code to verify the design under test (DUT)**	**Semester & Section:**	**8th - A**
GitHub Repository:	**Davis**		

FORENOON SESSION DETAILS
Image of session




module fulladder_test;
reg a,b,c;
wire s, cout;
integer correct;
fulladder FA (a,b,c,s,cout);
initial
begin
correct = 1;
#5 a=1; b=1; c=0; #5;
if ((s != 0) || (cout != 1))
correct = 0;
#5 a=1; b=1; c=1; #5;
if ((s != 1) || (cout != 1))
correct = 0;
#5 a=0; b=1; c=0; #5;
if ((s != 1) || (cout != 0))
correct = 0;
#5 $display ("%d", correct);
end
endmodule
Shall display 1 if outputs are correct; and display 0 otherwise.
talent lms
Create a free account
Capterra
4.6/5 Ease of Use
Average score on leading independent review site Capterra



REPORT –

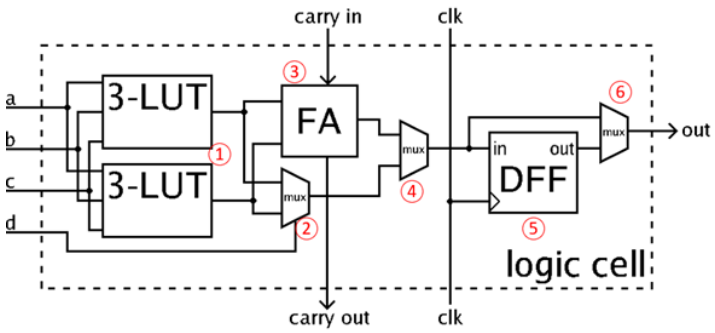
The field-programmable gate array (FPGA) is an integrated circuit that consists of internal hardware blocks with user-programmable interconnects to customize operation for a specific application. he interconnects can readily be reprogrammed, allowing an FPGA to accommodate changes to a design or even support a new application during the lifetime of the part.

The FPGA has its roots in earlier devices such as programmable read-only memories (PROMs) and programmable logic devices (PLDs). These devices could be programmed either at the factory or in the field, but they used fuse technology (hence, the expression "burning a PROM") and could not be changed once programmed. In contrast, FPGA stores its configuration information in a re-programmable medium such as static RAM (SRAM) or flash memory. FPGA manufacturers include Intel, Xilinx, Lattice Semiconductor, Microchip Technology and Micro semi.

FPGA Architecture

A basic FPGA architecture (Figure 1) consists of thousands of fundamental elements called configurable logic blocks (CLBs) surrounded by a system of programmable interconnects, called a fabric, that routes signals between CLBs. Input/output (I/O) blocks interface between the FPGA and external devices.

Depending on the manufacturer, the CLB may also be referred to as a logic block (LB), a logic element (LE) or a logic cell (LC).



The number and arrangement of components in the CLB varies by device; the simplified example in Figure 2 contains two three-input LUTs (1), an FA (3) and a D-type flip-flop (5), plus a standard mux (2) and two muxes, (4) and (6), that are configured during FPGA programming.

This simplified CLB has two modes of operation. In normal mode, the LUTs are combined with Mux 2 to form a four-input LUT; in arithmetic mode, the LUT outputs are fed as inputs to the FA together with a carry input from another CLB. Mux 4 selects between the FA output or the LUT output. Mux 6 determines whether the operation is asynchronous or synchronized to the FPGA clock via the D flip-flop.

Current-generation FPGAs include more complex CLBs capable of multiple operations with a single block; CLBs can combine for more complex operations such as multipliers, registers, counters and even digital signal processing (DSP) functions.

Originally, FPGAs included the blocks in Figure 1 and little else, but now designers can choose from products with a large range of features. Less complex devices such as simple programmable logic devices (SPLDs) and complex programmable logic devices (CPLDs) bridge the gap between discrete logic devices and entry-level FPGAs.

Entry-level FPGAs emphasize low power consumption, low logic density and low complexity per chip. Higher-function devices add functional blocks dedicated to specific functions: Examples include clock management components, phase-locked loops (PLLs), high-speed serializers and deserializers, Ethernet MACs, PCI express controllers and high-speed transceivers. These blocks can either be implemented with CLBs—termed soft IP—or designed as separate circuits; i.e., hard IP. Hard IP blocks gain performance at the expense of reconfigurability.At the high end, the FPGA product family includes complex system-on-chip (SoC) parts that integrate the FPGA architecture, hard IP and a microprocessor CPU core into a single component. Compared to separate devices, a SoC FPGA provides higher integration, lower power, smaller board size and higher-bandwidth communication between the core and other blocks.

Designers have traditionally used a hardware description language (HDL) such as VHDL (Figure 4) or Verilog to design the FPGA configuration.

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity signed_adder is
  port
  (
    aclr : in   std_logic;
    clk  : in   std_logic;
    a    : in   std_logic_vector;
    b    : in   std_logic_vector;
    q    : out  std_logic_vector
  );
end signed_adder;

architecture signed_adder_arch of signed_adder is
  signal q_s : signed(a'high+1 downto 0); -- extra bit wide

begin -- architecture
  assert(a'length >= b'length)
    report "Port A must be the longer vector if different sizes!"
    severity FAILURE;
  q <= std_logic_vector(q_s);

  adding_proc:
  process (aclr, clk)
    begin
      if (aclr = '1') then
        q_s <= (others => '0');
      elsif rising_edge(clk) then
        q_s <= ('0'&signed(a)) + ('0'&signed(b));
      end if; -- clk'd
    end process;

end signed_adder_arch;
```

FPGA Applications

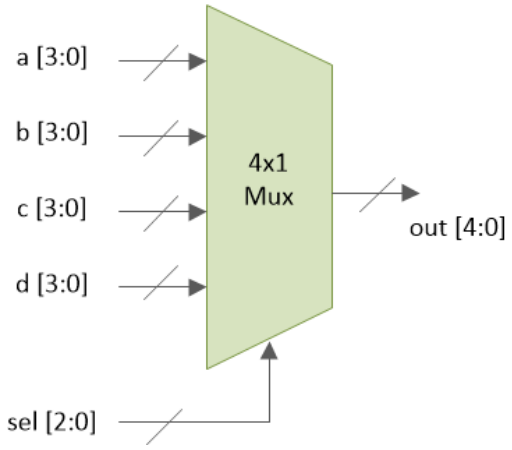
Many applications rely on the parallel execution of identical operations; the ability to configure the FPGA's CLBs into hundreds or thousands of identical processing blocks has applications in image processing, artificial intelligence (AI), data center hardware accelerators, enterprise networking and automotive advanced driver assistance systems (ADAS).

Many of these application areas are changing very quickly as requirements evolve and new protocols and standards are adopted. FPGAs enable manufacturers to implement systems that can be updated when necessary.

A good example of FPGA use is high-speed search: Microsoft is using FPGAs in its data centers to run Bing search algorithms. The FPGA can change to support new algorithms as they are created. If needs change, the design can be repurposed to run simulation or modeling routines in an HPC application. This flexibility is difficult or impossible to achieve with an ASIC.

Other FPGA uses include aerospace and defense, medical electronics, digital television, consumer electronics, industrial motor control, scientific instruments, cybersecurity systems and wireless communications.

Task 2 - Implement a 4:1 MUX and write the test bench code to verify the module.



Using Assign Statement –

```
module mux_4to1_assign ( input [3:0] a, input [3:0] b, input [3:0] c,input [3:0] d, input [1:0]sel, output [3:0] out);

 assign out = sel[1] ? (sel [0] ? d : c) : (sel[0] ? b : a);

 endmodule
```

Test bench code –

```
 module tb_4to1_mux;
  reg [3:0] a;
  reg [3:0] b;
  reg [3:0] c;
  reg [3:0] d;
  wire [3:0] out;
  reg [1:0] sel;
  integer i;
mux_4to1_case  mux0 (   .a (a),
                        .b (b),
                        .c (c),
                        .d (d),
                        .sel (sel),
                        .out (out));
initial begin
$monitor ("[%0t] sel=0x%0h a=0x%0h b=0x%0h c=0x%0h d=0x%0h out=0x%0h", $time,
sel, a, b, c, d, out);
sel <= 0;
     a <= $random;
     b <= $random;
     c <= $random;
     d <= $random;
```

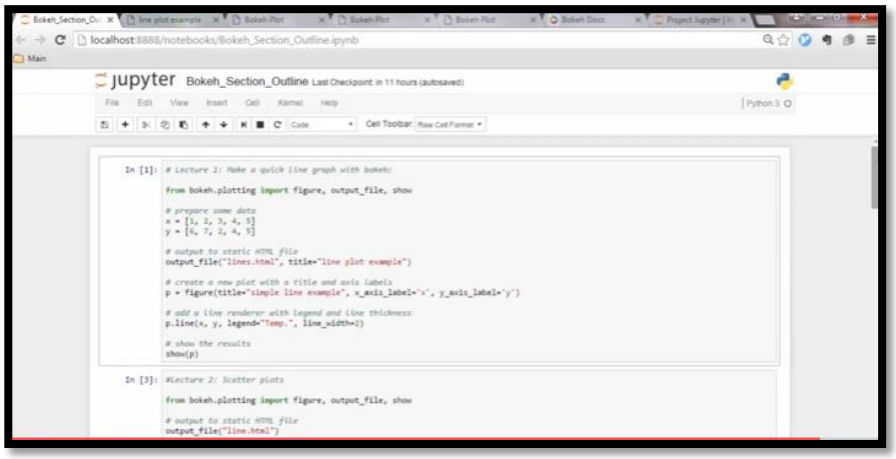
```
 for (i = 1; i < 4; i=i+1) begin
#5 sel <= i;
 end
#5 $finish;
end
endmodule
```

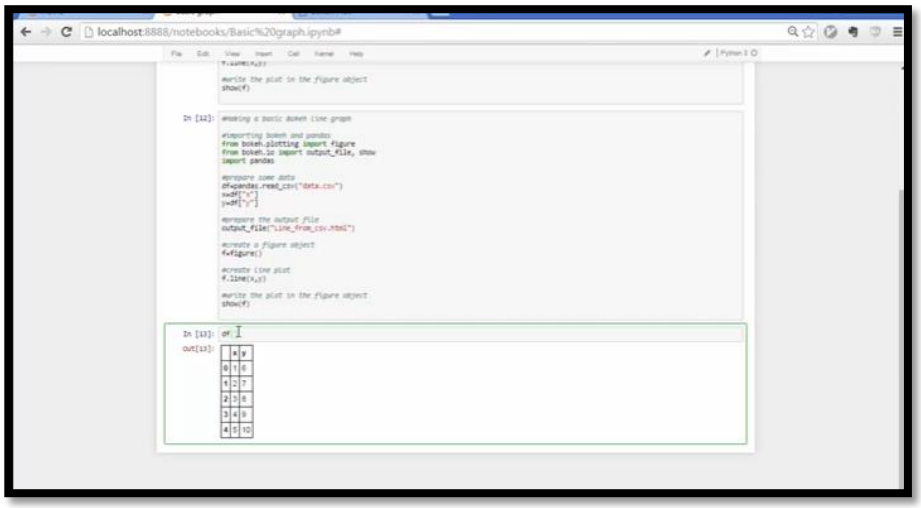
DAILY ASSESSMENT FORMAT

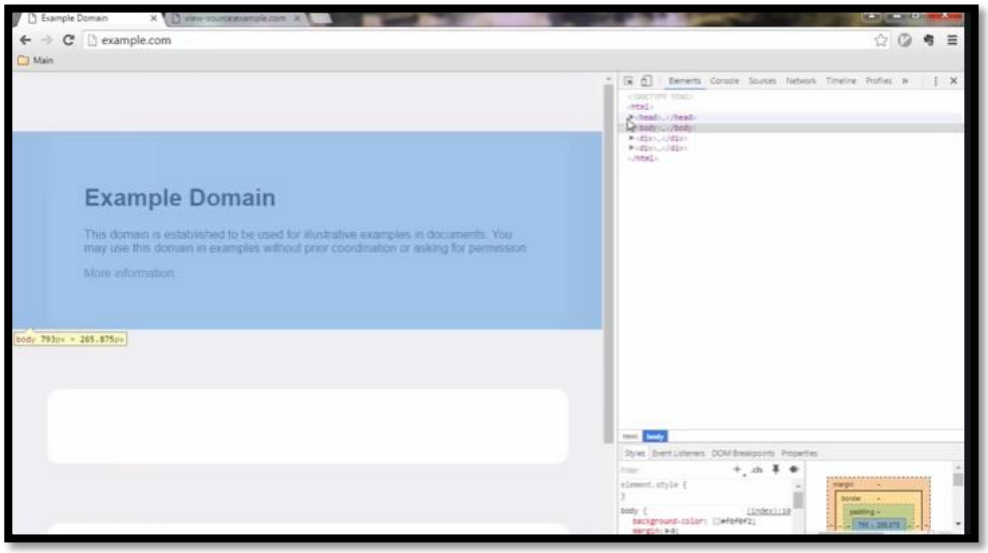
Date:	**2/06/2020**	**Name:**	**Davis S. Patel**
Course:	**Python Course**	**USN:**	**4AL16EC045**
Topic:	**Interactive Data Visualization with Bokeh Webscraping with Python Beautiful Soup**	**Semester & Section:**	**8th - A**
GitHub Repository:	**Davis**		

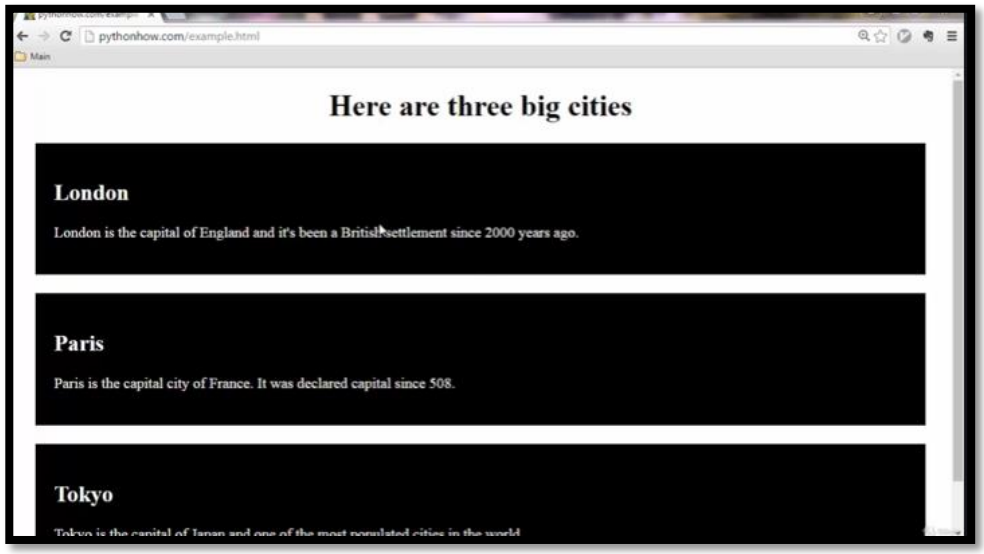
AFTERNOON SESSION DETAILS

Image of Session






Example Domain
example.com
Main
Example Domain
This domain is established to be used for illustrative examples in documents. You may use this domain in examples without prior coordination or asking for permission.
More information...


pythonhow.com/example.html
Main
Here are three big cities
London
London is the capital of England and it's been a British settlement since 2000 years ago.
Paris
Paris is the capital city of France. It was declared capital since 508.
Tokyo

REPORT –

Bokeh prides itself on being a library for *interactive* data visualization. Unlike popular counterparts in the Python visualization space, like Matplotlib and Seaborn, Bokeh renders its graphics using HTML and JavaScript. This makes it a great candidate for building web-based dashboards and applications. However, it's an equally powerful tool for exploring and understanding your data or creating beautiful custom charts for a project or report.

Bokeh is a library for creating interactive data visualizations in a web browser. It offers a concise, human-readable syntax, which allows for rapidly presenting data in an aesthetically pleasing manner. If you've worked with visualization in Python before, it's likely that you have used matplotlib. It's worth briefly mentioning how Bokeh differs from matplotlib, and when one might be preferred to the other.

Matplotlib has existed since 2002 and has long been a standard of Python data visualization. Bokeh emerged in 2013. This difference in age means that Matplotlib matured long before Bokeh was released; however, in a short period of time, Bokeh has reached a high level of maturity.

The intended uses of matplotlib and Bokeh are quite different. Matplotlib creates static graphics that are useful for quick and simple visualizations, or for creating publication quality images. Bokeh creates visualizations for display on the web (whether locally or embedded in a webpage) and most importantly, the visualizations are meant to be highly interactive. Matplotlib does not offer either of these features.

If would you like to visually interact with your data in an exploratory manner or you would like to distribute interactive visual data to a web audience, Bokeh is the library for you! If your main interest is producing finalized visualizations for publication, matplotlib may be better, although Bokeh does offer a way to create static graphics.

With this differences in mind, as we work through the lesson, I'll emphasize the interactive aspects that make Bokeh useful for exploring and disseminating historical data and that set it apart from other libraries like matplotlib.

To implement and use Bokeh, we first import some basics that we need from the bokeh.plotting module.

figure is the core object that we will use to create plots. figure handles the styling of plots, including title, labels, axes, and grids, and it exposes methods for adding data to the plot. The output_file function defines how the visualization will be rendered (namely to an html file) and the show function will be invoked when the plot is ready for output. show tells Bokeh that all of the data has been added to the plot and it is time to render it.

Installing Bokeh

If you haven't installed Bokeh yet, you can easily install it with pip from the terminal:

```
pip install bokeh
```

Or you use pip3:

```
pip3 install bokeh
```

#Snippet producing the triangle based plot

```
#Making a basic Bokeh line graph


#importing Bokeh
from bokeh.plotting import figure
from bokeh.io import output_file, show


#prepare some data
x=[3,7.5,10]
y=[3,6,9]


#prepare the output file
```

```
output_file("Line.html")

#create a figure object
f=figure()

#create line plot
f.triangle(x,y)

#write the plot in the figure object
show(f)
```

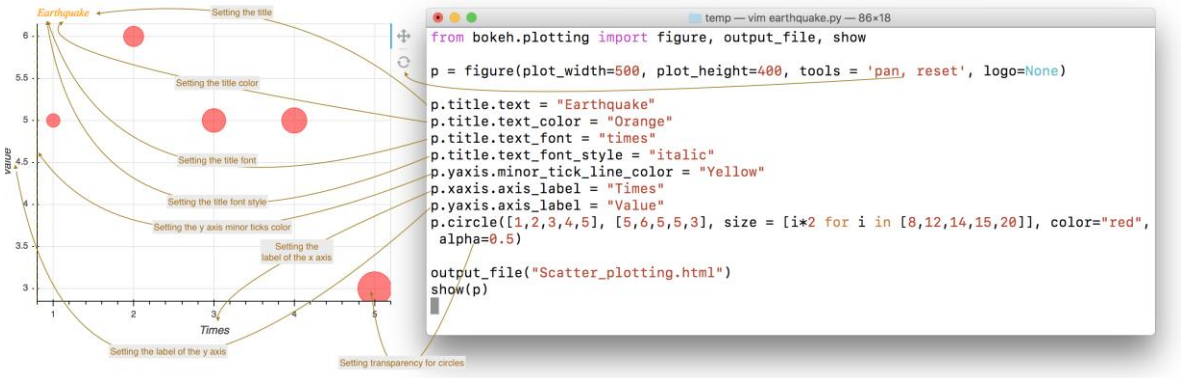
#Snippet producing the circle based plot

```
#Making a basic Bokeh line graph

#importing Bokeh
from bokeh.plotting import figure
from bokeh.io import output_file, show

#prepare some data
x=[3,7.5,10]
y=[3,6,9]

#prepare the output file
output_file("Line.html")
```

```
#create a figure object

f=figure()


#create line plot

f.circle(x,y)

#write the plot in the figure object

show(f)
```

Visual Attributes

Once we have built a basic plot, we can customize its visual attributes including changing the `title` color and font, adding labels for `xaxis` and `yaxis`, changing the color of the axis ticks, etc. All these properties are illustrated in the diagram below:



And here is the code:

```
from bokeh.plotting import figure, output_file, show

p = figure(plot_width=500, plot_height=400, tools = 'pan, reset')

p.title.text = "Earthquakes"

p.title.text_color = "Orange"

p.title.text_font = "times"
```

```
p.title.text_font_style = "italic"

p.yaxis.minor_tick_line_color = "Yellow"

p.xaxis.axis_label = "Times"

p.yaxis.axis_label = "Value"

p.circle([1,2,3,4,5], [5,6,5,5,3], size = [i*2 for i in [8,12,14,15,20]], color="red", alpha=0.5)

output_file("Scatter_plotting.html")

show(p)
```