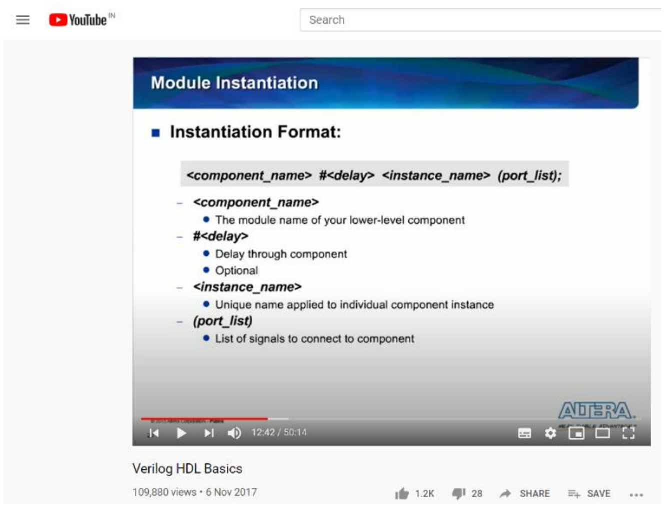
# DAILY ASSESSMENT FORMAT

| | | | |
|---|---|---|---|
| **Date:** | **02 JUN 2020** | **Name:** | **PAVITHRAN S** |
| **Course:** | **DIGITAL DESIGN USING HDL** | **USN:** | **4AL17EC068** |
| **Topic:** | • FPGA Basics: Architecture, Applications and Uses<br>• Verilog HDL Basics by Intel<br>• Verilog Testbench code to verify the design under test (DUT) | **Semester & Section:** | **6TH B** |
| **Github Repository:** | **Pavithran** | | |

| **FORENOON SESSION DETAILS** |
|---|
| **Image of session** |



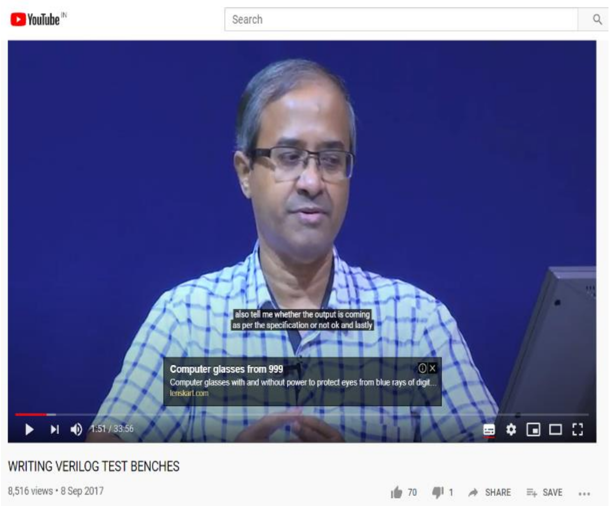
Module Instantiation

- Instantiation Format:

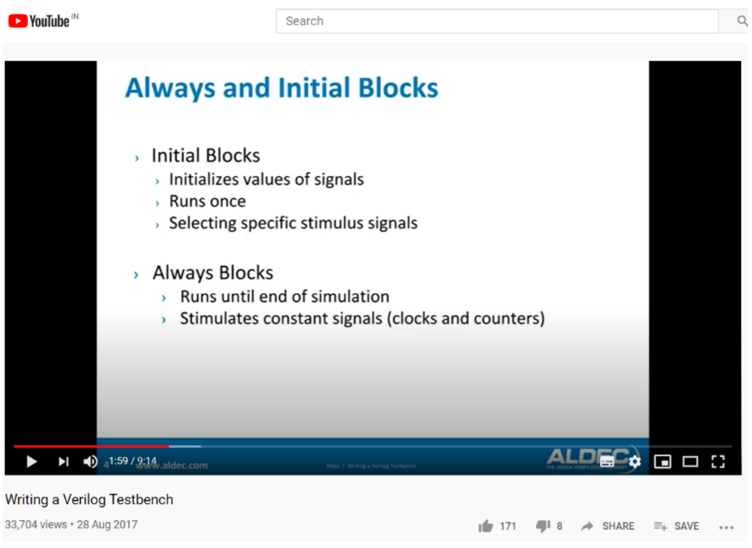
  <component_name> #<delay> <instance_name> (port_list);

  - <component_name>
    - The module name of your lower-level component
  - #<delay>
    - Delay through component
    - Optional
  - <instance_name>
    - Unique name applied to individual component instance
  - (port_list)
    - List of signals to connect to component

Verilog HDL Basics
109,880 views • 6 Nov 2017


YouTube
Search
also tell me whether the output is coming as per the specification or not ok and lastly
Computer glasses from 999
Computer glasses with and without power to protect eyes from blue rays of digit...
lenskart.com
1:51 / 33:56
70
1
SHARE
SAVE

WRITING VERILOG TEST BENCHES
8,516 views • 8 Sep 2017



Always and Initial Blocks

- Initial Blocks
  - Initializes values of signals
  - Runs once
  - Selecting specific stimulus signals
- Always Blocks
  - Runs until end of simulation
  - Stimulates constant signals (clocks and counters)

Writing a Verilog Testbench
33,704 views • 28 Aug 2017

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

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.

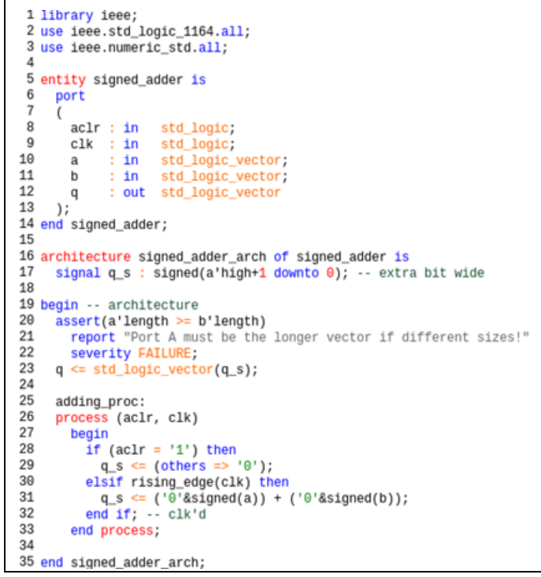
What is FPGA?

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. The 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 Microsemi.

How do we transform this collection of thousands of hardware blocks into the correct configuration to execute the application? An FPGA-based design begins by defining the required computing tasks in the development tool, then compiling them into a configuration file that contains information on how to hook up the CLBs and other modules. The process is similar to a software development cycle except that the goal is to architect the hardware itself rather than a set of instructions to run on a predefined hardware platform.

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



```
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 Uses: An Attractive Choice for Certain Applications

The ability to configure the hardware of the FPGA, reconfigure it when needed and optimize it for a particular set of functions makes the FPGA an attractive option in many applications.

FPGAs are often used to provide a custom solution in situations in which developing an ASIC would be too expensive or time-consuming. An FPGA application can be configured in hours or days instead of months. Of course, the flexibility of the FPGA comes at a price: An FPGA is likely to be slower, require more PCB area and consume more power than an equivalent ASIC.

Even when an ASIC will be designed for high-volume production, FPGAs are widely used for system validation, including pre-silicon validation, post-silicon validation and firmware development. This allows manufacturers to validate their design before the chip is produced in the factory.

## 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.

## FPGA History: What Comes Next?

With these emerging applications, the FPGA market is growing at a healthy clip: It was valued at $5.34 billion in 2016 and is expected to grow to $9.50 billion in 2023, according to industry researchers MarketsandMarkets. That's a compound annual growth rate (CAGR) of 8.5 percent, compared to a CAGR of about 2 percent for the much larger ($74 billion) general microprocessor market.

The exponential growth of data, and the emergence of fast-changing fields such as AI, machine learning, HPC and genomics, require architectures that are fast, flexible and adaptable. FPGAs are well-positioned to take advantage of these new opportunities.

## HDL:

Verilog is a HARDWARE DESCRIPTION LANGUAGE (HDL). It is a language used for describing a digital system like a network switch or a microprocessor or a memory or a flip−flop. It means, by using a HDL we can describe any digital hardware at any level. Designs, which are described in HDL are independent of technology, very easy for designing and debugging, and are normally more useful than schematics, particularly for large circuits.

Verilog supports a design at many levels of abstraction. The major three are −

- Behavioral level
- Register-transfer level
- Gate level

### Behavioral level

This level describes a system by concurrent algorithms (Behavioural). Every algorithm is sequential, which means it consists of a set of instructions that are executed one by one. Functions, tasks and blocks are the main elements. There is no regard to the structural realization of the design.

### Register−Transfer Level

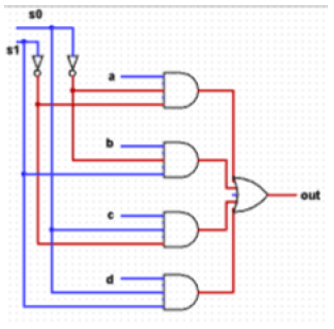
Designs using the Register−Transfer Level specify the characteristics of a circuit using operations and the transfer of data between the registers. Modern definition of an RTL code is "Any code that is synthesizable is called RTL code".

### Gate Level

Within the logical 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 (basic gates). Gate level modelling may not be a right idea for logic design. Gate level code is generated using tools like synthesis tools and his netlist is used for gate level simulation and for backend.

**TODAY’S TASK**

**Implement a 4:1 MUX and write the test bench code to verify the module**



Logic diagram 4:1 mux

STRUCTURAL:

```
module and_gate(output a, input b, c, d);
assign a = b & c & d;
endmodule
module not_gate(output f, input e);
assign e = ~ f;
endmodule
module or_gate(output l, input m, n, o, p);
assign l = m | n | o | p;
endmodule
module m41(out, a, b, c, d, s0, s1);
output out;
input a, b, c, d, s0, s1;
wire s0bar, s1bar, T1, T2, T3;
not_gate u1(s1bar, s1);
not_gate u2(s0bar, s0);
and_gate u3(T1, a, s0bar, s1bar);
and_gate u4(T2, b, s0, s1bar);
and_gate u5(T3, c, s0bar, s1);
and_gate u6(T4, d, s0, s1);
or_gate u7(out, T1, T2, T3, T4);
endmodul
```

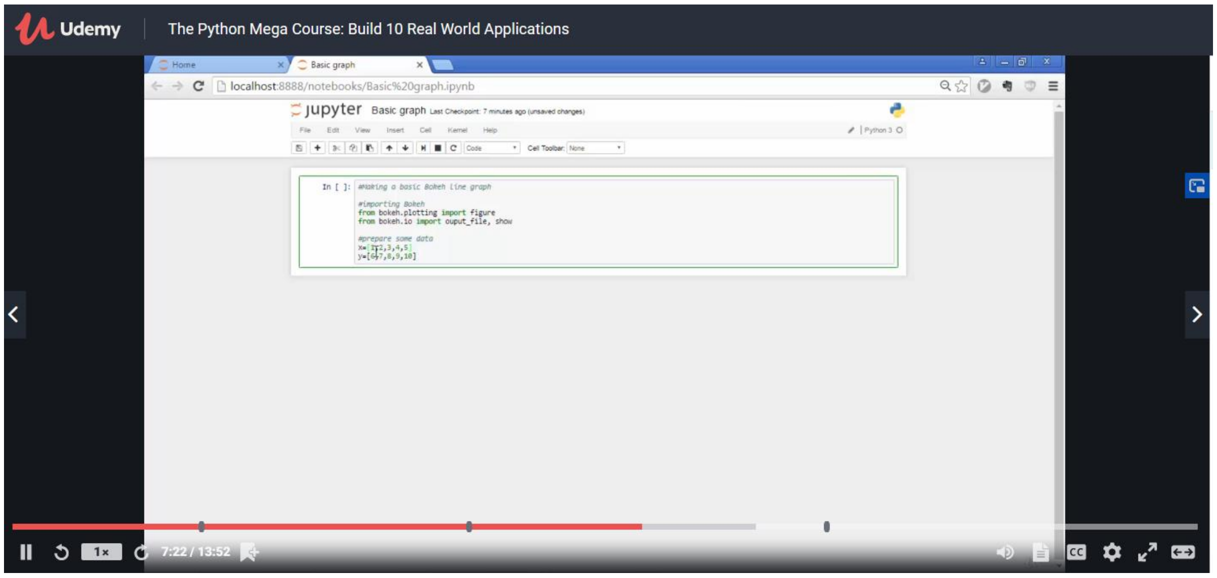
TESTBENCH:

```
module top;
wire  out;
reg  a;
reg  b;
reg  c;
reg  d;
reg s0, s1;
m41 name(.out(out), .a(a), .b(b), .c(c), .d(d), .s0(s0), .s1(s1));
 initial
 begin
 a=1'b0; b=1'b0; c=1'b0; d=1'b0;
 s0=1'b0; s1=1'b0;
 #500 $finish;
end
always #40 a=~a;
always #20 b=~b;
always #10 c=~c;
always #5 d=~d;
always #80 s0=~s0;
always #160 s1=~s1;
always@(a or b or c or d or s0 or s1)
$monitor("At time = %t, Output = %d", $time, out);
endmodule;
```

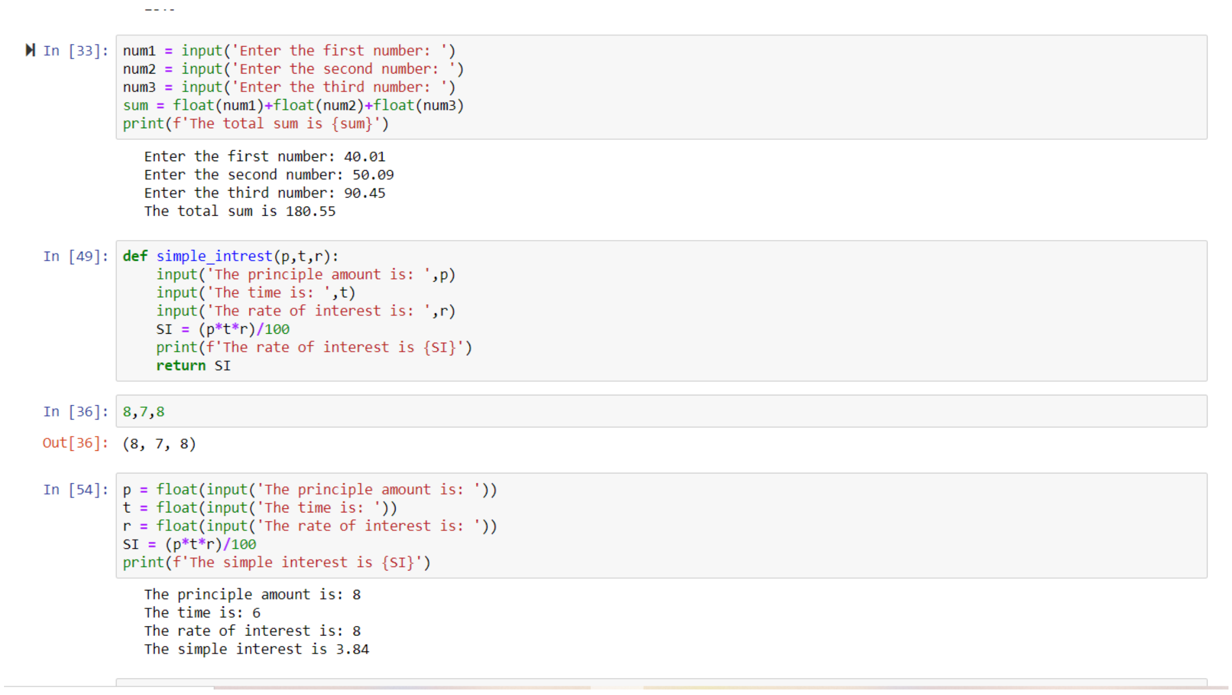
|  |  |  |  |
|---|---|---|---|
| **Date:** | **02 JUN 2020** | **Name:** | **PAVITHRAN S** |
| **Course:** | **PYTHON** | **USN:** | **4AL17EC068** |
| **Topic:** | **FOR LOOP IN PYTHON, WHILE LOOP IN PYTHON, USEFUL OPERATOR IN PYTHON** | **Semester & Section:** | **6TH B** |
| **Github Repository:** | **Pavithran** |  |  |

## AFTERNOON SESSION DETAILS

**Image of session**

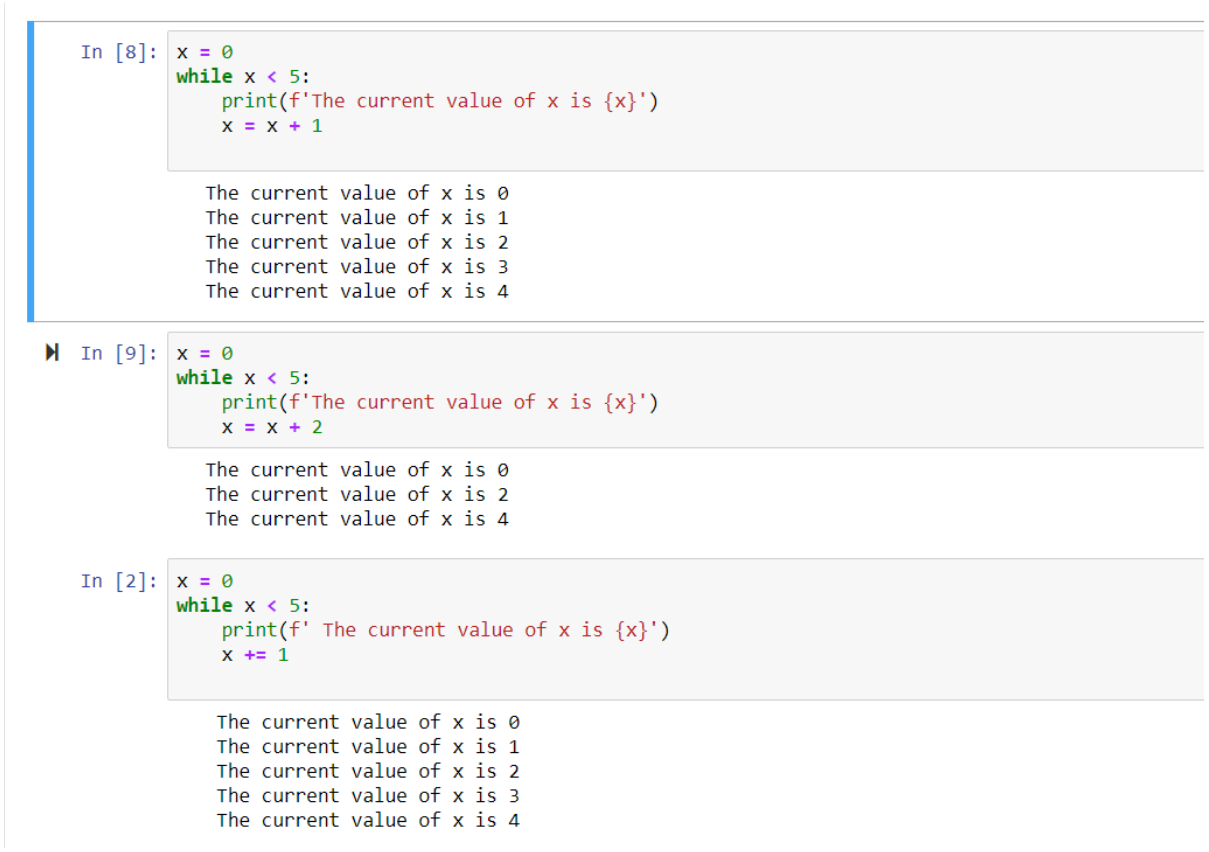

Udemy
The Python Mega Course: Build 10 Real World Applications
localhost:8888/notebooks/Basic%20graph.ipynb
jupyter Basic graph Last Checkpoint: 7 minutes ago (unsaved changes)
7:22 / 13:52

```
In [ ]: #Making a basic Bokeh line graph

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

        #prepare some data
        x=[1,2,3,4,5]
        y=[6,7,8,9,10]
```



```
In [33]: num1 = input('Enter the first number: ')
         num2 = input('Enter the second number: ')
         num3 = input('Enter the third number: ')
         sum = float(num1)+float(num2)+float(num3)
         print(f'The total sum is {sum}')

         Enter the first number: 40.01
         Enter the second number: 50.09
         Enter the third number: 90.45
         The total sum is 180.55

In [49]: def simple_intrest(p,t,r):
             input('The principle amount is: ',p)
             input('The time is: ',t)
             input('The rate of interest is: ',r)
             SI = (p*t*r)/100
             print(f'The rate of interest is {SI}')
             return SI

In [36]: 8,7,8

Out[36]: (8, 7, 8)

In [54]: p = float(input('The principle amount is: '))
         t = float(input('The time is: '))
         r = float(input('The rate of interest is: '))
         SI = (p*t*r)/100
         print(f'The simple interest is {SI}')

         The principle amount is: 8
         The time is: 6
         The rate of interest is: 8
         The simple interest is 3.84
```

**WHILE:**



```
In [8]: x = 0
        while x < 5:
            print(f'The current value of x is {x}')
            x = x + 1
```

```
The current value of x is 0
The current value of x is 1
The current value of x is 2
The current value of x is 3
The current value of x is 4
```

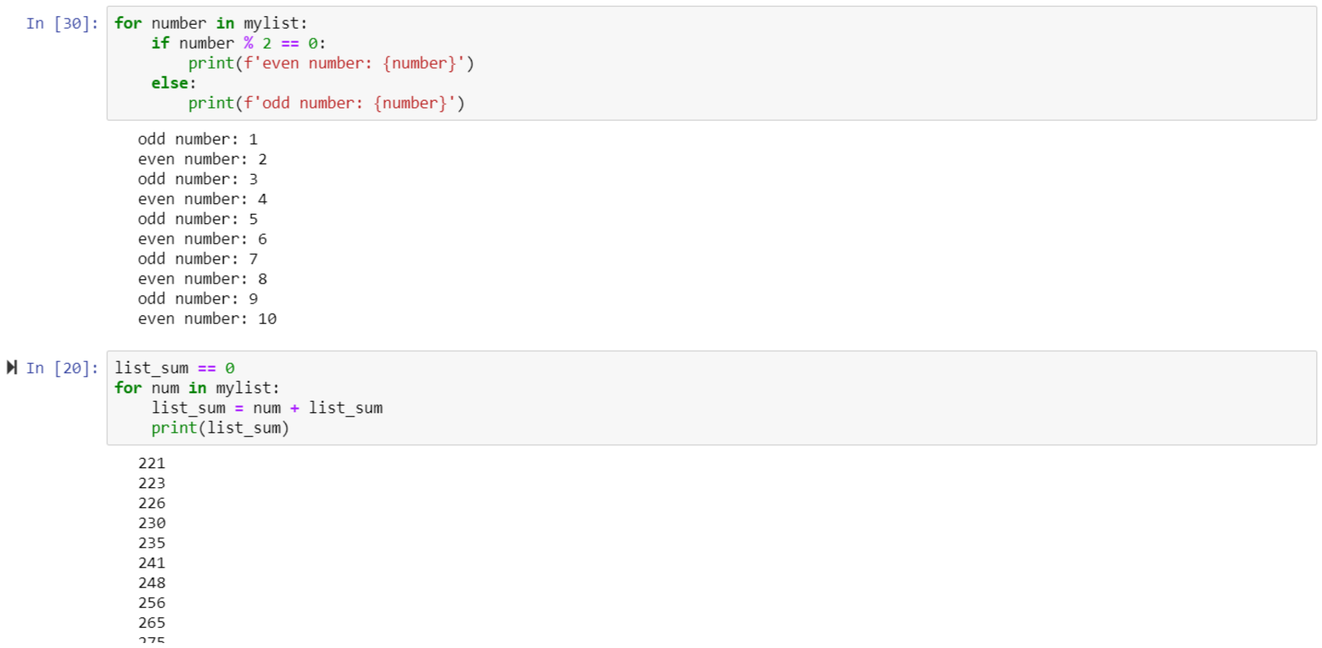
```
In [9]: x = 0
        while x < 5:
            print(f'The current value of x is {x}')
            x = x + 2
```

```
The current value of x is 0
The current value of x is 2
The current value of x is 4
```

```
In [2]: x = 0
        while x < 5:
            print(f' The current value of x is {x}')
            x += 1
```

```
 The current value of x is 0
 The current value of x is 1
 The current value of x is 2
 The current value of x is 3
 The current value of x is 4
```

**FOR:**



```
In [30]: for number in mylist:
             if number % 2 == 0:
                 print(f'even number: {number}')
             else:
                 print(f'odd number: {number}')
```

```
odd number: 1
even number: 2
odd number: 3
even number: 4
odd number: 5
even number: 6
odd number: 7
even number: 8
odd number: 9
even number: 10
```

```
In [20]: list_sum == 0
         for num in mylist:
             list_sum = num + list_sum
             print(list_sum)
```

```
221
223
226
230
235
241
248
256
265
```

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

**How to use "While Loop"**

While loop does the exactly same thing what "if statement" does, but instead of running the code block once, they jump back to the point where it began the code and repeats the whole process again.

Syntax

```
while expression
 Statement
```

**How to use "For Loop"**

In Python, "for loops" are called **iterators.**Just like while loop, "For Loop" is also used to repeat the program.

But unlike while loop which depends on condition true or false. "For Loop" depends on the elements it has to iterate.

**How to use break statements in For Loop**

Breakpoint is a unique function in For Loop that allows you to break or terminate the execution of the for loop

**Example**:

```
#use a for loop over a collection
		#Months = ["Jan","Feb","Mar","April","May","June"]
		#for m in Months:
				#print m

# use the break and continue statements
for x in range (10,20):
						if (x == 15): break
						#if (x % 2 == 0) : continue
						print(x)
```

Output

```
10
11
12
13
14
```

In this example, we declared the numbers from 10-20, but we want that our for loop to terminate at number 15 and stop executing further. For that, we declare break function by defining (x==15): break, so as soon as the code calls the number 15 it terminates the program Code Line 10 declare variable x between range (10, 20)

- Code Line 11 declare the condition for breakpoint at x==15,
- Code Line 12 checks and repeats the steps until it reaches number 15
- Code Line 13 Print the result in output

**How to use "continue statement" in For Loop**

Continue function, as the name indicates, will terminate the current iteration of the for loop BUT will continue execution of the remaining iterations.

**Example**

```
#use a for loop over a collection
        #Months = ["Jan","Feb","Mar","April","May","June"]
        #for m in Months:
                #print m

# use the break and continue statements
for x in range (10,20):
                        #if (x == 15): break
                        if (x % 5 == 0) : continue
                        print(x)
```

Output

```
11
12
13
14
16
17
18
19
```