Python Turtle - Lesson 6

Page Content

> **In this lesson you will learn:**
>
> - about Boolean logic and how it is used in Python
> - about Boolean comparisons and how to use then
> - about Boolean operators and how to use them
> - how to use Boolean comparison and operators to make complex conditional statements

Part 1: Boolean logic

Video link

Boolean Introduction

In programming Boolean is all about `True` and `False` values:

- Boolean variables only contain either `True` or `False`
- Comparison operators (`==`, `!=`, `>`, `< >=` or `<=`) return either `True` or `False`
- Boolean operators (we'll learn about these later) return either `True` or `False`

The values `True` and `False` are special values. If you type them into your IDE the syntax highlighting will indicate that they are special.

In Python testing if something is `True` or `False` is called testing the **turthiness**. When you compare two values, you are testing it's truthiness.

Comparison operators

The conditions in our `if` and `while` statements test truthiness using comparison operators. Let's refresh those.

There are six comparison operators you can use. Create a new file called `lesson_6_pt_1.py` and enter the code below.

```
1 print("jeff" == "jeff")  # equal to
2 print(1 != 1)  # not equal to
3 print(500 > 300)  # greater than
4 print(100 >= 250)  # greater than or equal to
5 print("a" < "q")  # less than
6 print(-30 <= 3)  # less than or equal to
```

PRIMM:

- **Predict** the six values the **Shell** will display (hint, they will be either `True` or `False`).
- **Run** the code and see if your predictions are correct.

It doesn't matter if the values are literals (magic numbers) or if they are stores in a variable. Change your code to the code below.

```
1 score = 10
2 print(score > 5)
```

PRIMM:

- **Predict** if the code will print `True` or `False`
- **Run** the code and see if your prediction was correct.

Boolean Operations

You can also complete operations on Boolean values using Boolean operators. Boolean operations are like preforming a calculation, but only with Boolean values (ie. `True` and `False`). Like all things Boolean, they return a single `True` or `False` value. They are useful for creating complex condition tests.

There are three Boolean operators:

- `and`
- `or`
- `not`

The `not` operator

The simplest operator to understand is the `not` operator. It reverses the Boolean value:

- `not True` returns `False`
- `not False` returns `True`

Change the code in your program to the code below:

```
1 print("not True is:", not True)
2 print("not False:", not False)
```

PRIMM:

- **Predict** what you think will be written to the **Shell** when your run this code.
- **Run** the code and check your predictions.

The `and` operator

The `and` operator and the `or` operator are a little bit more complicated.

The `and` operators will return `True` if **all** the values in the operation are `True`.

Again, change your code so it reflects the code below:

```
1 print("True and True is:", True and True)
2 print("True and False is:", True and False)
3 print("False and True is:", False and True)
4 print("False and False is:", False and False)
5 print("True and True and True is:", True and True and True)
6 print("True and True and False is:", True and True and False)
```

PRIMM:

- **Predict** what you think will be written to the **Shell** when your run this code.
- **Run** the code and check your predictions.
- Let's **Investigate** that code

Code breakdown:

- `Line 1`: `print("True and True is:", True and True)`

    - `True and True` → all values are `True` → returns `True`
    - `True and True is: True` is printed
- `Line 2`: `print("True and False is:", True and False)`
    - `True and False` → not all values are `True` → returns `False`
    - `True and False is: False` is printed
- `Line 3`: `print("False and True is:", True and False)`
    - `False and True` → not all values are `True` → returns `False`
    - `False and True is: False` is printed
- `Line 4`: `print("False and False is:", True and False)`
    - `False and False` → not all values are `True` → returns `False`
    - `False and False is: False` is printed
- `Line 5`: `print("True and True and True is:", True and True and True)`
    - `True and True and True` → all values are `True` → returns `True`
    - `True and True and True: True` is printed
- `Line 6`: `print("True and True and False is:", True and True and False)`
    - `True and True and False` → not all values are `True` → returns `False`
    - `True and True and False is: False` is printed

The `or` operator

The `or` operator is the inverse of the `and` operator.

The `or` operator will return `True` if **any one** of the values in the operation is `True`.

Change your code so it reflects the code below:

```
1 print("True or True is:", True or True)
2 print("True or False is:", True or False)
3 print("False or True is:", False or True)
4 print("False or False is:", False or False)
5 print("True or True or True is:", True or True or True)
6 print("True or False or False is:", True or False or False)
```

PRIMM:

- **Predict** what you think will be written to the **Shell** when your run this code.
- **Run** the code and check your predictions.
- Let's **Investigate** that code

Code breakdown:

- `Line 1`: `print("True or True is:", True or True)`
    - `True or True` → at least one value is `True` → returns `True`
    - `True or True is: True` is printed
- `Line 2`: `print("True or False is:", True or False)`
    - `True or False` → at least one value is `True` → returns `True`
    - `True or False is: False` is printed
- `Line 3`: `print("False or True is:", True or False)`
    - `False or True` → at least one value is `True` → returns `True`
    - `False or True is: False` is printed
- `Line 4`: `print("False or False is:", True or False)`
    - `False or False` → no values are `True` → returns `False`
    - `False or False is: False` is printed
- `Line 5`: `print("True or True or True is:", True or True or True)`
    - `True or True or True` → at least one value is `True` → returns `True`
    - `True or True or True: True` is printed
- `Line 6`: `print("True or True or False is:", True or True or False)`
    - `True or True or False` → at least one value is `True` → returns `True`
    - `True or False or False is: True` is printed

Using Boolean operators

So far, we have been returning `True` or `False` from other values of `True` and `False`. This isn't that useful but remember comparison operators return Boolean values. Boolean operators can create conditions with multiple comparison operators. This provides complex conditions for your `if` and `while` statements.

Consider the following code:

```
1 print(7 < 8 and "a" < "o")
```

PRIMM:

- **Predict** what you think will be written to the **Shell** when your run this code.
- **Run** the code and check your predictions.
- Let's **Investigate** that code

Code breakdown:

- `Line 1`: `print(7 < 8 and "a" < "o")`
  - first Python will complete the comparison operations from left to right
    - `7 < 8` returns `True`
    - `"a" < "o"` returns `True`
  - the code is now: `print(True and True)`
    - `True and True` returns `True`
  - Python prints `True` to the **Shell**

> **Combining multiple comparison operations**
>
> Conditions with multiple comparisons need comparisons on both sides of the Boolean operator.
>
> `10 > 5 and 10 > 13` is **not** the same as `10 > 5 and 13`.

Part 2: Mouse input in Turtle

To reinforce our understanding of Boolean logic, we are going to do something new with Turtle. So far, we have only accepted user input via the **Shell**, but Turtle can also use mouse input (and keys as well).

We are going to use the code below for our Boolean exercise, but we will have to explore it first.

Download **`lesson_6_pt_2.py`** file and save it to your lesson folder.

```python
import turtle

## Prepare the windows and turtle ##
def set_scene():
    turtle.setup(800, 600)

    ## Respond to mouse click (signal) ##
    turtle.onscreenclick(draw_dot)

    ## Set up the grid ##
    my_ttl.speed(0)
    for i in range(4):
        my_ttl.forward(400)
        my_ttl.back(400)
        my_ttl.right(90)
    my_ttl.penup()


## Reaction to signal (slot) ##
def draw_dot(x, y):
    print(x, y)
    color = "orange"
    size = 10
    my_ttl.goto(x, y)
    my_ttl.dot(size, color)


## Main Program
my_ttl = turtle.Turtle()
set_scene()
my_ttl.hideturtle()
```

- **Predict** what you think will be written to the **Shell** when your run this code.
- **Run** the code and check your predictions.
- Let's **Investigate** that code.

We'll do the code breakdown in three sections in the order they are executed:

- `Lines 29` to `31`: the main program
  - `Line 29`: `my_ttl = turtle.Turtle()` → create a Turtle object and names it `my_ttl`
  - `Line 30`: `set_scene()` calls the `set_scene()` function
  - `Line 31`: `my_ttl.hideturtle()` make the turtle invisible
- `Lines 4` to `16`: the `set_scene` function
  - `Line 4`: `def set_scene()` → defines the `set_scene` function without any arguments
  - `Line 5`: `turtle.setup(800, 600)` → creates a `800` x `600` window
  - `Line 8`: `turtle.onscreenclick(draw_dot)` → this is **new**
    - if a mouse click is detected:
      - calls the `draw_dot` function
      - passes to the `draw_dot` function the `x` and `y` coordinates of where the mouse clicked
  - `Line 11`: `my_ttl.speed(0)` → a turtle speed of `0` means you don't see the turtle move
  - `Lines 12` to `15`: draws four lines from `(0, 0)` making the four quadrants
  - `Line 16`: `penup` prevents the turtle from drawing a line to the mouse click coordinates (try commenting it out and see what happens)
- `Lines 20` to `25`: the `draw_dot` function
  - `Line 20`: `def draw_dot(x, y):`
    - defines the `draw_dot` function
    - accepts the two arguments `x` and `y` which are passed from `line 8`
    - `turtle.onscreenclick()` always passes the `x` and `y` coordinates as arguments
  - `Line 21`: prints the `x` and `y` coordinates to the **Shell** (to help you plan your code)
  - `Line 22`: assigns `"orange"` to the variable `color`
  - `Line 23`: assigns `10` to the variable `size`
  - `Line 24`: sends the turtle to the `x` and `y` coordinates
  - `Line 25`: `my_ttl.dot(size, color)` draws a dot at the turtle position of size `size` and colour `color`

Exercises

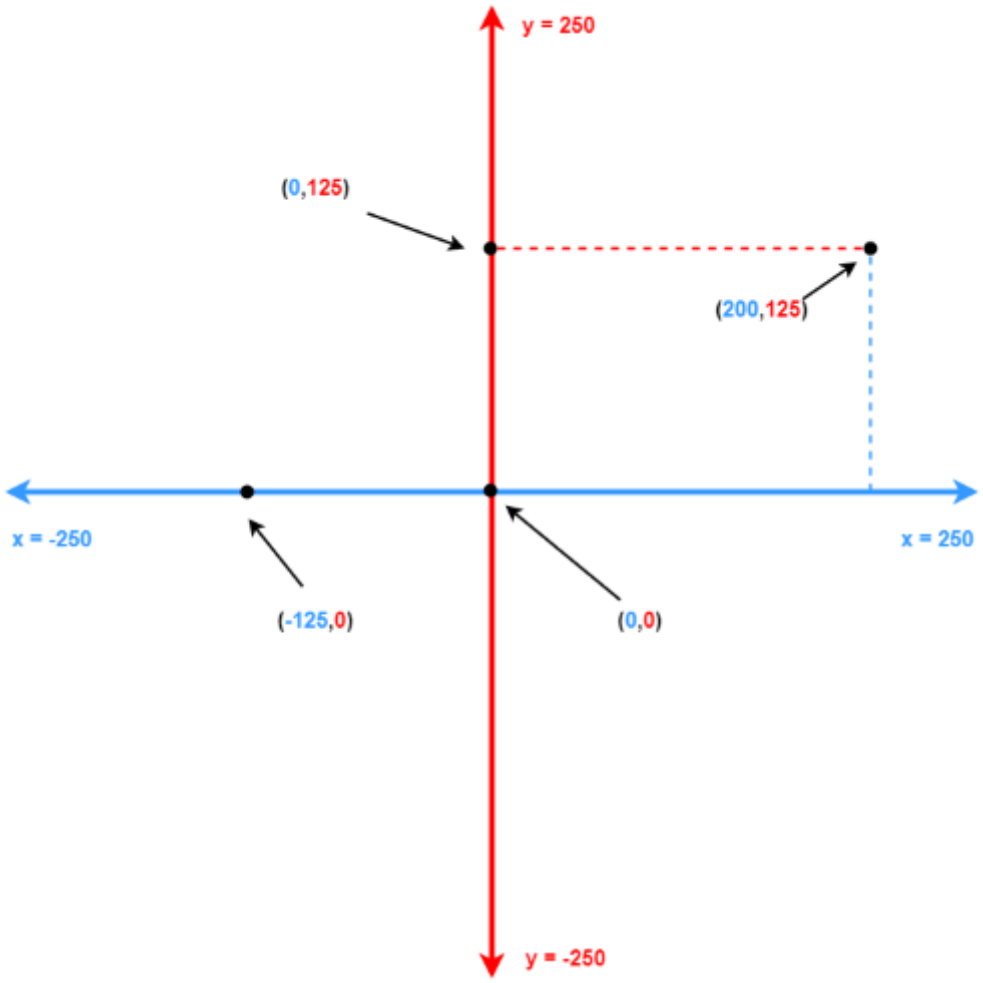
In this course, the exercises are the **make** component of the PRIMM model. Work through the following exercises and make your own code.

So far, the dot colour is always orange. In these exercises the quadrant of the mouse click will determine the dot colour.

To do this your will need to use:

- `if` ... `elif` ... `else` statements
- Boolean comparisons
- Boolean operations

You will also need to remember how coordinates work in Turtle.



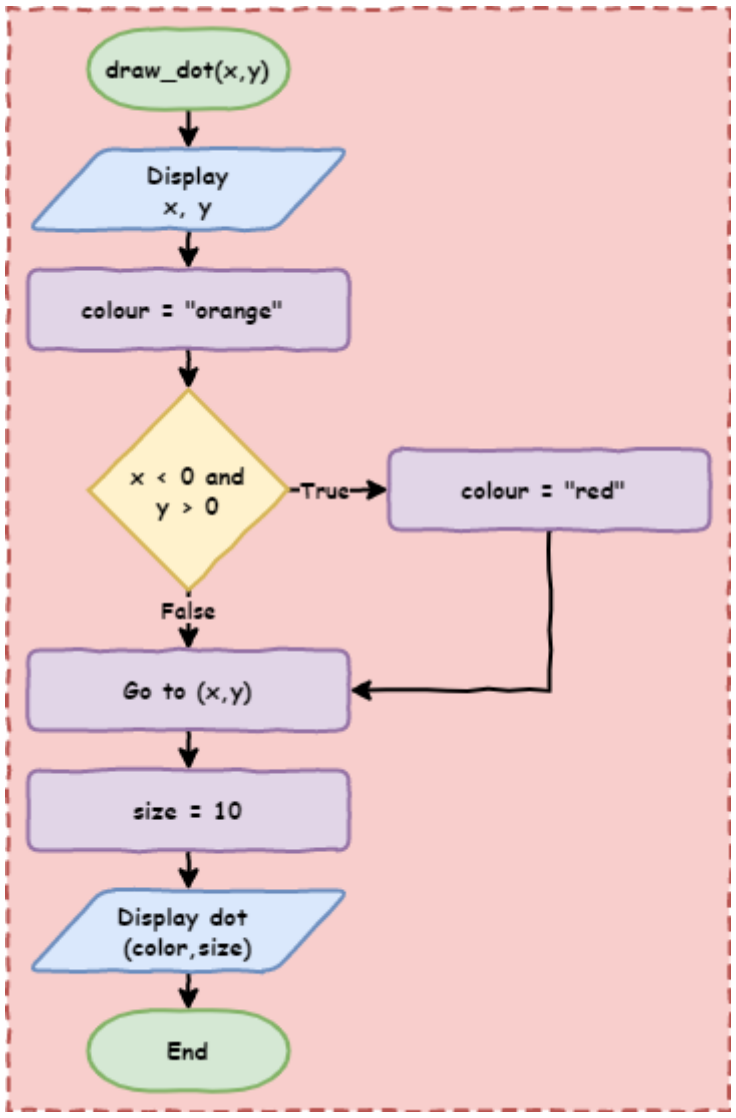
Exercise 1

Download `lesson_6_ex_1.py` file and save it to your lesson folder. Below is its code.

```python
import turtle

## Prepare the windows and turtle ##
def set_scene():
    turtle.setup(800, 600)

    ## Respond to mouse click (signal) ##
    turtle.onscreenclick(draw_dot)

    ## Set up the grid ##
    my_ttl.speed(0)
    for i in range(4):
        my_ttl.forward(400)
        my_ttl.back(400)
        my_ttl.right(90)
    my_ttl.penup()


## Reaction to signal (slot) ##
def draw_dot(x, y):
    print(x, y)
    color = "orange"

    ###################################
    ######## Answer goes here ########
    ###################################
    """ Part A
    Use an 'if' statement to set the dot color to red
    when the mouse clicks in the top right quadrant

    You can determine the position using the variables
    x and y

    To change the colour of the dot to red, run the command

    color = 'red'

    """

    ###################################
    ###################################
    ###################################

    my_ttl.goto(x, y)
    size = 10
    my_ttl.dot(size, color)


my_ttl = turtle.Turtle()
set_scene()
my_ttl.hideturtle()
```

Follow the instructions in the comments from `line 24` to `line 42`.

To help, here is the flowchart for the `draw_dot` function:



Exercise 2

Download [lesson_6_ex_2.py](#) file and save it to your lesson folder. Below is its code.

```
1 import turtle
2
3 ## Prepare the windows and turtle ##
4 def set_scene():
5     turtle.setup(800, 600)
6
7     ## Respond to mouse click (signal) ##
8     turtle.onscreenclick(draw_dot)
9
10    ## Set up the grid ##
11    my_ttl.speed(0)
12    for i in range(4):
13        my_ttl.forward(400)
14        my_ttl.back(400)
15        my_ttl.right(90)
16    my_ttl.penup()
17
18
19 ## Reaction to signal (slot) ##
20 def draw_dot(x, y):
21     print(x, y)
22     color = "orange"
23
24     ###################################
25     ######## Answer goes here ########
26     ###################################
27     """ Part B
28     Use both 'if' and 'else' to set the dot color to red
29     if the mouse is clicked in the top right quadrant and
30     green if clicked anywhere else
31     """
32
33     ###################################
34     ###################################
35     ###################################
36
37     my_ttl.goto(x, y)
38     size = 10
39     my_ttl.dot(size, color)
40
41
42 my_ttl = turtle.Turtle()
43 set_scene()
44 my_ttl.hideturtle()
```

Follow the instructions in the comments from `line 24` to `line 35`.

Exercise 3

Download [lesson_6_ex_3.py](#) file and save it to your lesson folder. Below is its code.

```
1 import turtle
2
3 ## Prepare the windows and turtle ##
4 def set_scene():
5     turtle.setup(800, 600)
6
7     ## Respond to mouse click (signal) ##
8     turtle.onscreenclick(draw_dot)
9
10    ## Set up the grid ##
11    my_ttl.speed(0)
12    for i in range(4):
13        my_ttl.forward(400)
14        my_ttl.back(400)
15        my_ttl.right(90)
16    my_ttl.penup()
17
18
19 ## Reaction to signal (slot) ##
20 def draw_dot(x, y):
21     print(x, y)
22     color = "orange"
23
24     ###################################
25     ######## Answer goes here ########
26     ###################################
27     """ Part C
28     Use 'if', 'elif' and 'else' keywords to set the dot color to
29     red when the mouse is clicked in the top right quadrant,
30     blue in the top left quadrant, yellow in the bottom left quadrant
31     and green in the bottom right quadrant
32     """
33
34     ###################################
35     ###################################
36     ###################################
37
38     my_ttl.goto(x, y)
39     size = 10
40     my_ttl.dot(size, color)
41
42
43 my_ttl = turtle.Turtle()
44 set_scene()
45 my_ttl.hideturtle()
```

Follow the instructions in the comments from `line 24` to `line 36`.

By Damien Murtagh