2. Loops and logic

Ash Suresh

September 11, 2024

Understanding loops and conditional statements using turtle module

- We will demonstrate the crucial concepts of loop and conditional/logical statements "visually", using the turtle module in python.

```
>>> import turtle as tl
>>> tl.forward(100)
```

- This will move the turtle/cursor 100 pixels to the right.

- If you can't install and use turtle on your system, you may run it online in this website: https://pythonsandbox.com/turtle

Understanding loops and logical statements using turtle module

```
>>> import turtle as tl
>>> tl.forward(100)
>>> tl.right(90)
```

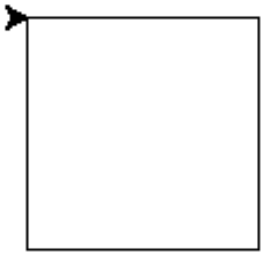
- This will tilt the turtle/cursor 90 degrees to the right.

What will this code block do?

```
>>> import turtle as tl
>>> tl.forward(100)
>>> tl.right(90)
>>> tl.forward(100)
>>> tl.right(90)
>>> tl.forward(100)
>>> tl.right(90)
>>> tl.forward(100)
>>> tl.right(90)
```

What will this code block do?

```
>>> import turtle as tl
>>> tl.forward(100)
>>> tl.right(90)
>>> tl.forward(100)
>>> tl.right(90)
>>> tl.forward(100)
>>> tl.right(90)
>>> tl.forward(100)
>>> tl.right(90)
```



What will this code block do?

- These two statements were repeated 4 times to draw a square.

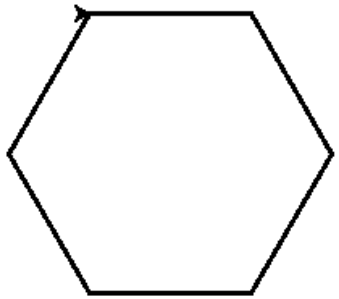
```
>>> tl.forward(100)
>>> tl.right(90)
```

- Repetitive tasks are performed using loops in programming.
- Let's try the for loop in python.

```
>>> import turtle as tl
>>> for i in [1,2,3,4]:
...     tl.forward(100)
...     tl.right(90)
>>>
```

- The list [1,2,3,4] is an iterable, a string of length 4 or a tuple (or any iterable) can be used for this purpose.

Exercise 1: Write a code to draw a hexagon.



```
>>> import turtle as tl
>>> tl.reset()
>>> for i in [1,2,3,4,5,6]:
...     tl.forward(100)
...     tl.right(60)
>>>
```

Try this code

```
>>> import turtle as tl
>>> tl.reset()
>>> n = 8
>>> for i in range(n):
...     tl.forward(100)
...     tl.right(360.0/n)
>>>
```

Conditional statments

Run this code block, what will be the output?

```
>>> for i in range(6):
...     if i%2 == 0:
...         print(f'{i} is even')
...     else:
...         print(f'{i} is odd')
>>>
```

Conditional statments

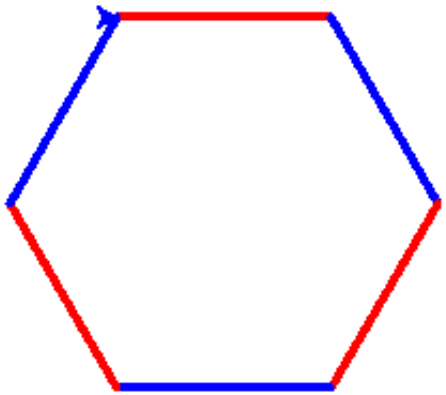
Run this code block, what will be the output?

```
>>> for i in range(6):
...     if i%2 == 0:
...         tl.color('red')
...     else:
...         tl.color('blue')
...     tl.forward(100)
...     tl.right(360.0/6)
>>>
```

Conditional statments

Run this code block, what will be the output?

```
>>> for i in range(6):
...     if i%2 == 0:
...         tl.color('red')
...     else:
...         tl.color('blue')
...     tl.forward(100)
...     tl.right(360.0/6)
>>>
```

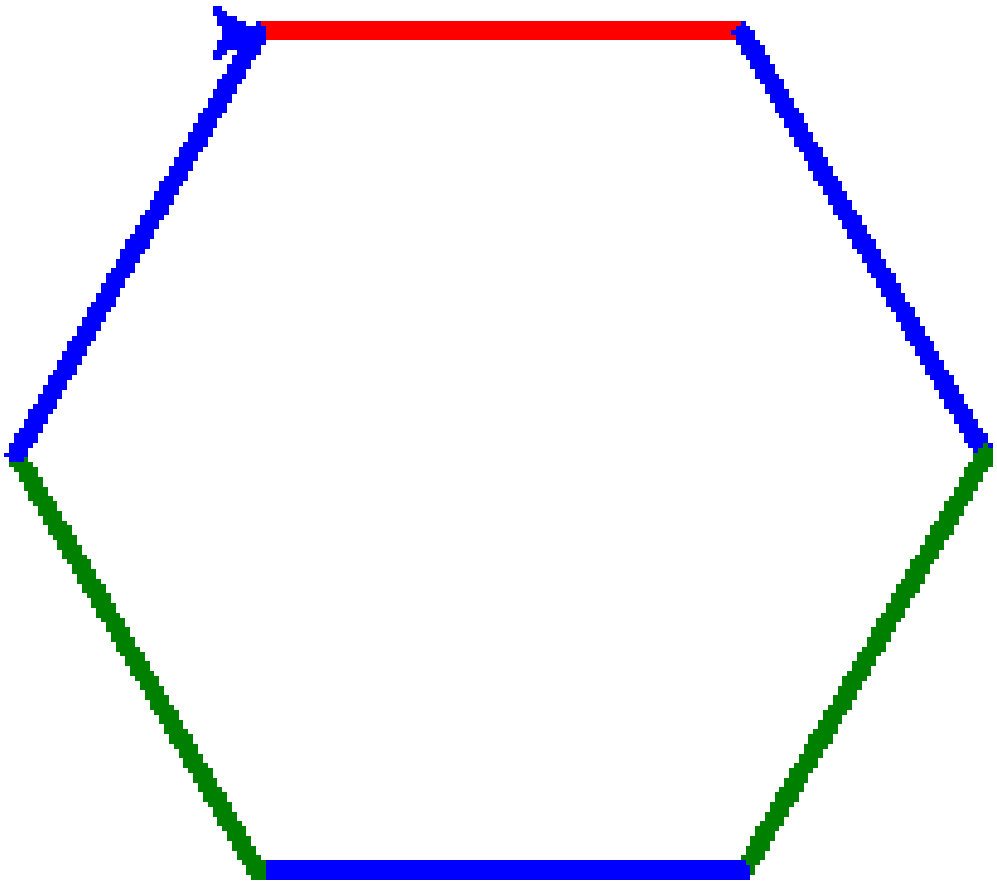


Conditional statments: More than two conditions

Now run this code.

```
>>>import turtle as tl
>>>tl.reset()
>>>tl.width(4)
>>>clr = 'red'
>>>for i in range(6):
>>>    tl.color(clr)
>>>    tl.forward(100)
>>>    tl.right(60)
>>>    if clr == 'red':
>>>            clr = 'blue'
>>>    elif clr == 'blue':
>>>            clr = 'green'
>>>    else:
>>>            clr = 'blue'
```

Conditional statments: More than two conditions



Putting loops and conditionals in user defined functions

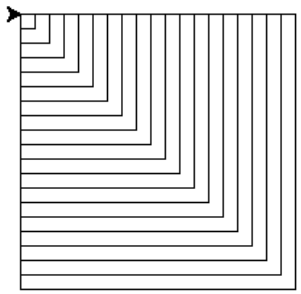
- Instead of writing the code again and again, we can convert blocks of codes to user defined functions for re-usability as well as to maintain the modularity or "blockness" of the code.

```
>>>import turtle as tl
>>>tl.reset()
>>>tl.width(4)
>>>def square():
...     for i in range(4):
...         tl.forward(100)
...         tl.right(90)
>>>square()
```

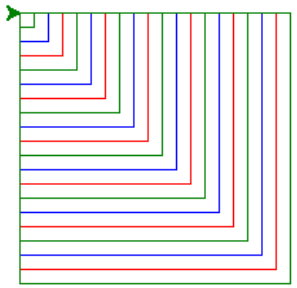
- This function doesn't take any arguments.

Exercise 2: Write a user defined function which takes the side length of the square as input. Use default argument(s).

1. Use this function and a for loop to draw this figure:

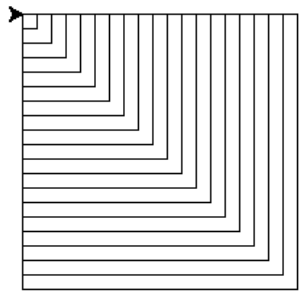


2. Now modify the code to generate this figure:

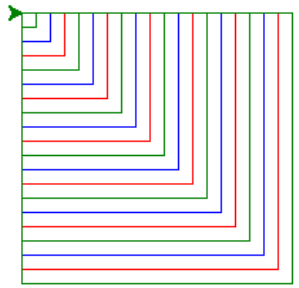


Exercise 2: Write a user defined function which takes the side length of the square as input. Use default argument(s).

1. Use this function and a for loop to draw this figure:



2. Now modify the code to generate this figure:



Hint: make a list of strings naming the colors, use tl.color(...) in the for loop, what are the 3 possible remainders when you divide an number by 3?

Solution: Exercise 2 a,b

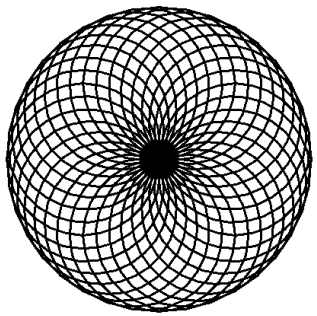
```
>>> import turtle as tl
>>> tl.reset()
>>> tl.width(4)
>>> def square(l=100):
...     for i in range(4):
...             tl.forward(l)
...             tl.right(90)
...
>>> colors = ['red','green','blue']
>>> for l in range(0,200,10):
...     tl.color(colors[l%3])
...     square(l)
...
>>>
```

Run this code, just for dopamine.

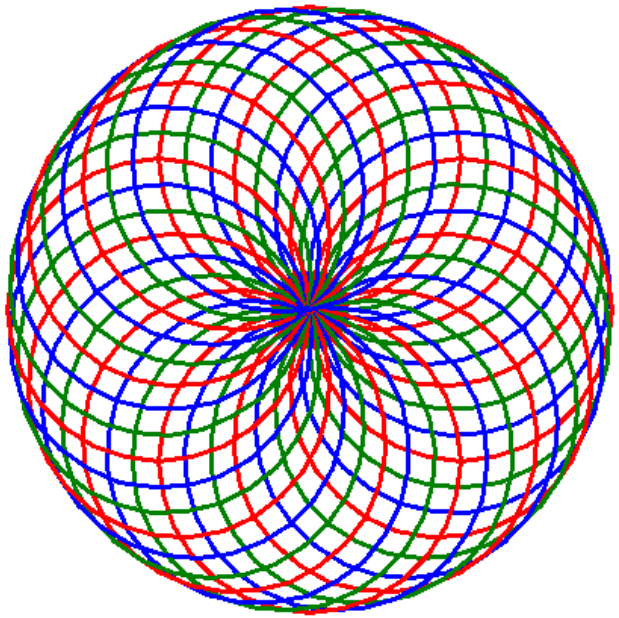
```
>>> import turtle as tl
>>> tl.reset()
>>> tl.speed('fastest')
>>> tl.width(3)
>>> for i in range(36):
...     tl.circle(100)
...     tl.right(360/36)
...
>>>
```

Run this code, just for dopamine.

```
>>> import turtle as tl
>>> tl.reset()
>>> tl.speed('fastest')
>>> tl.width(3)
>>> for i in range(36):
...     tl.circle(100)
...     tl.right(360/36)
...
>>>
```

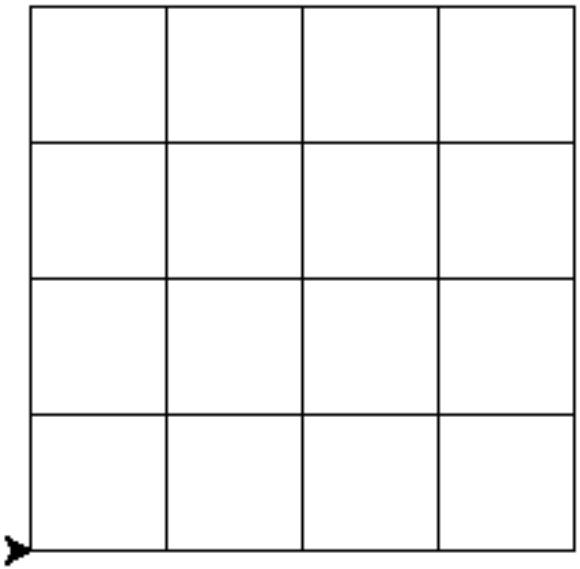


Generate a figure like this.



Nested for loops

How to generate this pattern?



Nested for loops

```
>>> import turtle as tl
>>> tl.reset()
>>> tl.speed('fastest')
>>> def draw_grid(rows, columns, tile_size):
...     """Function to draw a grid of square tiles"""
...     for row in range(rows):
...         for col in range(columns):
...             square(tile_size)
...             tl.penup()
...             tl.forward(tile_size)
...             tl.pendown()
...         tl.penup()
...         tl.backward(tile_size * columns)  # Move
    back to the start of the row
...         tl.right(90)
...         tl.forward(tile_size) #Move to the next row
...         tl.left(90)
...         tl.pendown()
...
```

Nested for loops

```
>>> def draw_grid(rows, columns, tile_size):
...     """Function to draw a grid of square tiles"""
...     for row in range(rows):
...         for col in range(columns):
...             square(tile_size)
...             tl.penup()
...             tl.forward(tile_size)
...             tl.pendown()
...         tl.penup()
...         tl.backward(tile_size * columns)  # Move
    back to the start of the row
...         tl.right(90)
...         tl.forward(tile_size) #Move to the next row
...         tl.left(90)
...         tl.pendown()
...
>>> # Draw a grid with 4 tiles along rows and columns,
    tile size of 50 units
>>> draw_grid(4, 4, 50)
```

Another solution

```
def grid(nr,nc,l=100):
    tl.penup()
    tl.goto(0,0)
    for i in range(nr+1):
        tl.penup()
        tl.goto(0,i*l)
        tl.pendown()
        tl.forward((nc)*l)
    tl.left(90)
    tl.penup()
    tl.goto(0,0)
    for i in range(nc+1):
        tl.penup()
        tl.goto(i*l,0)
        tl.pendown()
        tl.forward((nr)*l)
```