

# Introduction to Programming using Python



Susan Ibach | Technical Evangelist  
Christopher Harrison | Content Developer

# Meet Susan Ibach| @hockeygeekgirl

## Technical Evangelist

Helping developers understand Visual Studio, app building

Microsoft Certified Trainer

My first program was written in basic on a computer with 64K of memory

## Will not admit how many years coding experience

Basic, Fortran, COBOL, VB, C#, HTML, Python

Frequent blogger and presenter

marathoner, wife, and mother of two awesome boys!



# Meet Christopher Harrison | @geektrainer



## Content Developer

Focused on ASP.NET and Office 365 development

Microsoft Certified Trainer

Still misses his Commodore 64

## Long time geek

Regular presenter at TechEd

Periodic blogger

Certification advocate

Marathoner, husband, father of one four legged child

# Course Topics

## Introduction to Programming using Python - Day One

01 | Getting started

02 | Displaying text

03 | String variables

04 | Storing numbers

05 | Working with dates and times

06 | Making decisions with code

07 | Complex decisions with code

# Course Topics

## Introduction to Programming using Python - Day Two

08 | Repeating events

12 | Reading from files

09 | Repeating events until done

13 | Functions

10 | Remembering lists

14 | Handling errors

11 | How to save information in files

# Setting Expectations

- Target Audience
  - People new to programming
  - Students
  - Career changers
  - IT Pros
  - Anyone with an interest in learning to code
- If you want to follow along...
  - Install Visual Studio Express
  - Install the Python tools
    - Instructions coming soon...

# Join the MVA Community!

- Microsoft Virtual Academy
  - Free online learning tailored for IT Pros and Developers
  - Over 2M registered users
  - Up-to-date, relevant training on variety of Microsoft products
- “Earn while you learn!”
  - Get 50 MVA Points for this event!
  - Visit <http://aka.ms/MVA-Voucher>
  - Enter this code: IntProgPython (expires 27 Oct 14)

# Repeating events

## for loops

Susan Ibach | Technical Evangelist  
Christopher Harrison | Content Developer



Sometimes we need to perform an action more than once

- Pour a cup of coffee for each guest
- Wash the dishes until they are all clean
- Make a name card for each guest attending a party

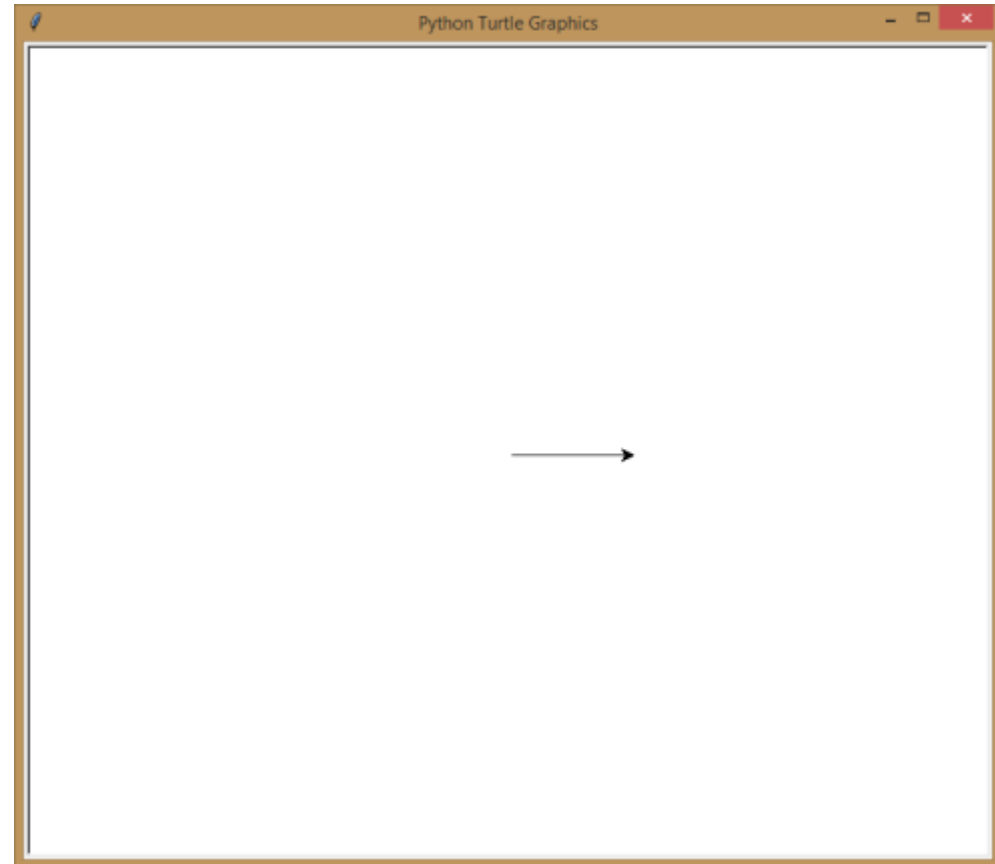
# In code, we use loops to repeat a task

- We are going to have some fun in this module by drawing objects
- We will use loops to draw some of our objects

# Hello, turtle

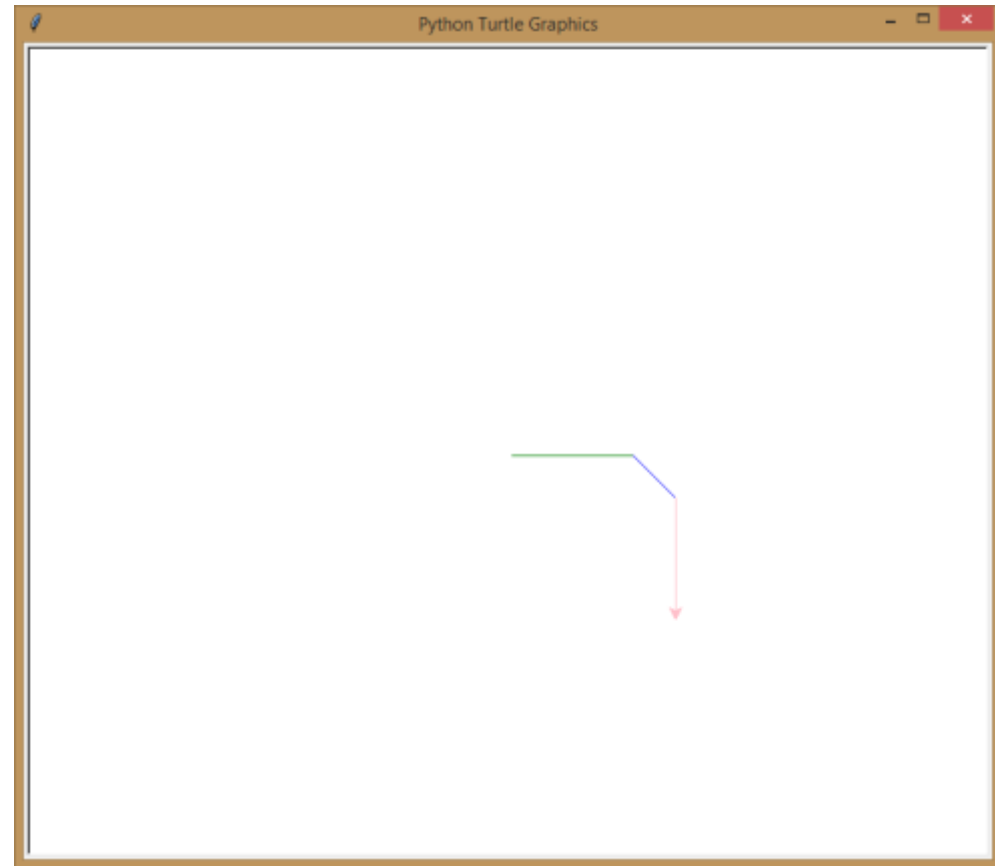
# Did you know Python can draw?

```
import turtle  
turtle.forward(100)
```



turtle is a library that lets you draw

```
import turtle
turtle.color('green')
turtle.forward(100)
turtle.right(45)
turtle.color('blue')
turtle.forward(50)
turtle.right(45)
turtle.color('pink')
turtle.forward(100)
```



# DEMO

---

Drawing with turtle

You can probably guess what some of the turtle commands do

Command	Action
<code>right(x)</code>	Rotate right x degrees
<code>left(x)</code>	Rotate left x degrees
<code>color('x')</code>	Change pen color to x
<code>forward(x)</code>	Move forward x
<code>backward(x)</code>	Move backward x

# For loops



# How would we get turtle do draw a square?

```
import turtle  
turtle.forward(100)  
turtle.right(90)  
turtle.forward(100)  
turtle.right(90)  
turtle.forward(100)  
turtle.right(90)  
turtle.forward(100)
```

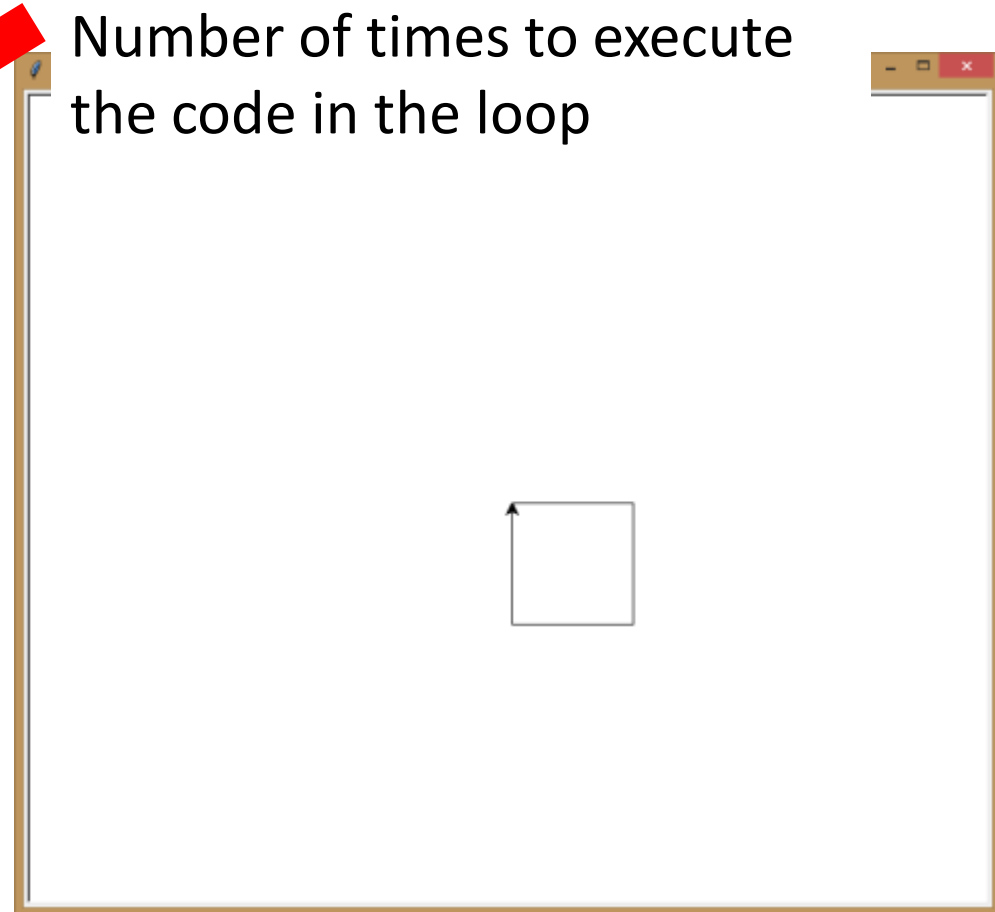
We are just repeating  
the same two lines of  
code



# Loops allow us to repeat the same line of code as often as we want

```
import turtle
for steps in range(4):
    turtle.forward(100)
    turtle.right(90)
```

Number of times to execute  
the code in the loop

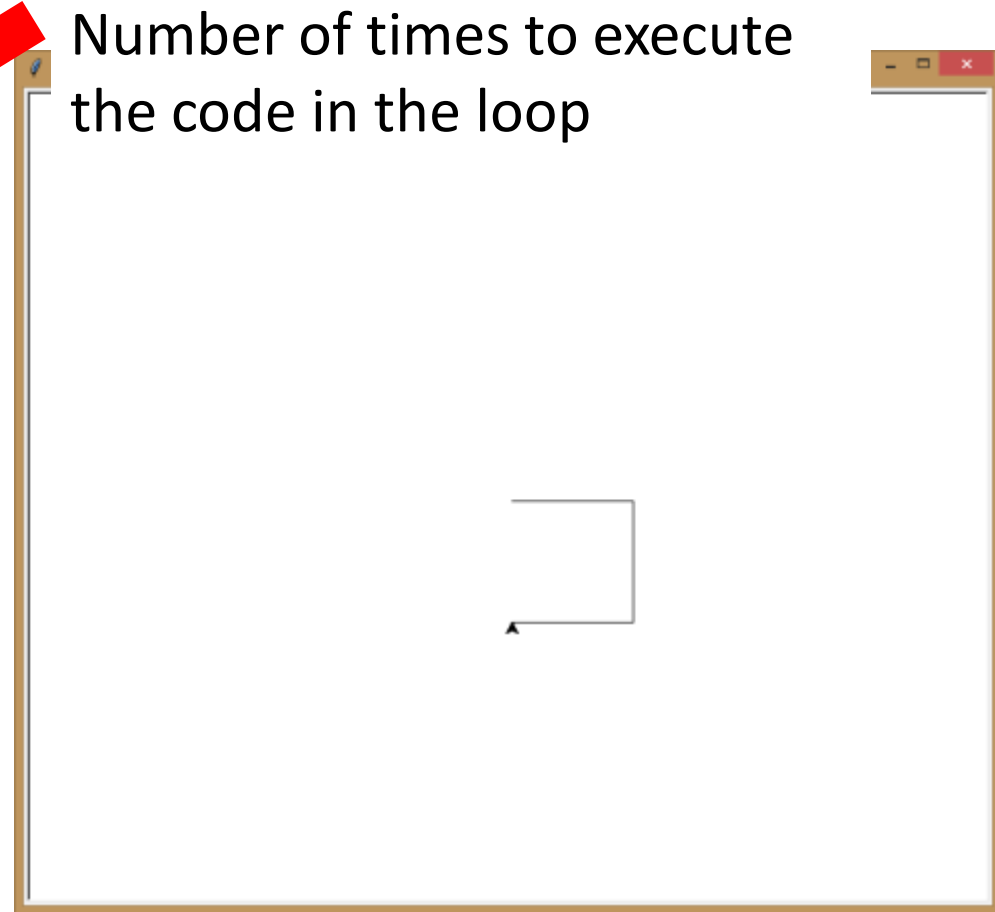


You MUST indent the code  
you want repeated

When you change the range, you change the number of times the code executes

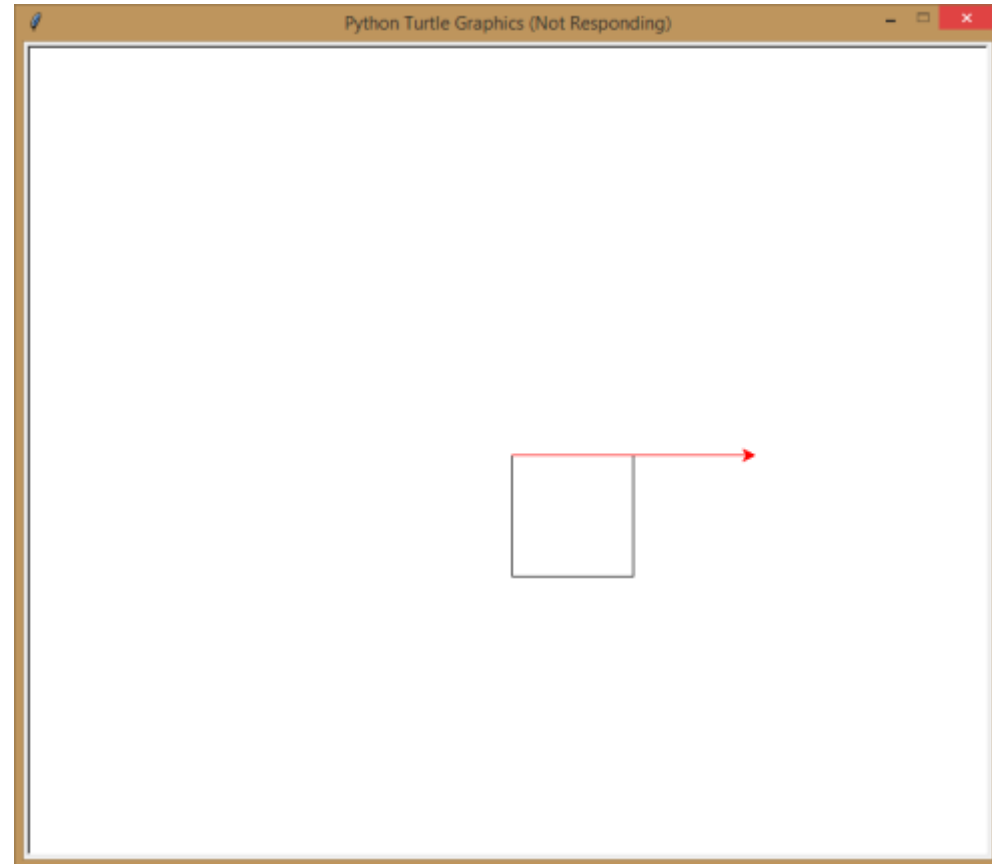
```
import turtle  
for steps in range(3):  
    turtle.forward(100)  
    turtle.right(90)
```

Number of times to execute  
the code in the loop



# Only the indented code is repeated!

```
import turtle
for steps in range(4):
    turtle.forward(100)
    turtle.right(90)
turtle.color('red')
turtle.forward(200)
```



# DEMO

---

Using loops to draw shapes

# Now we can make new typing mistakes!

Can you find three mistakes in this code?

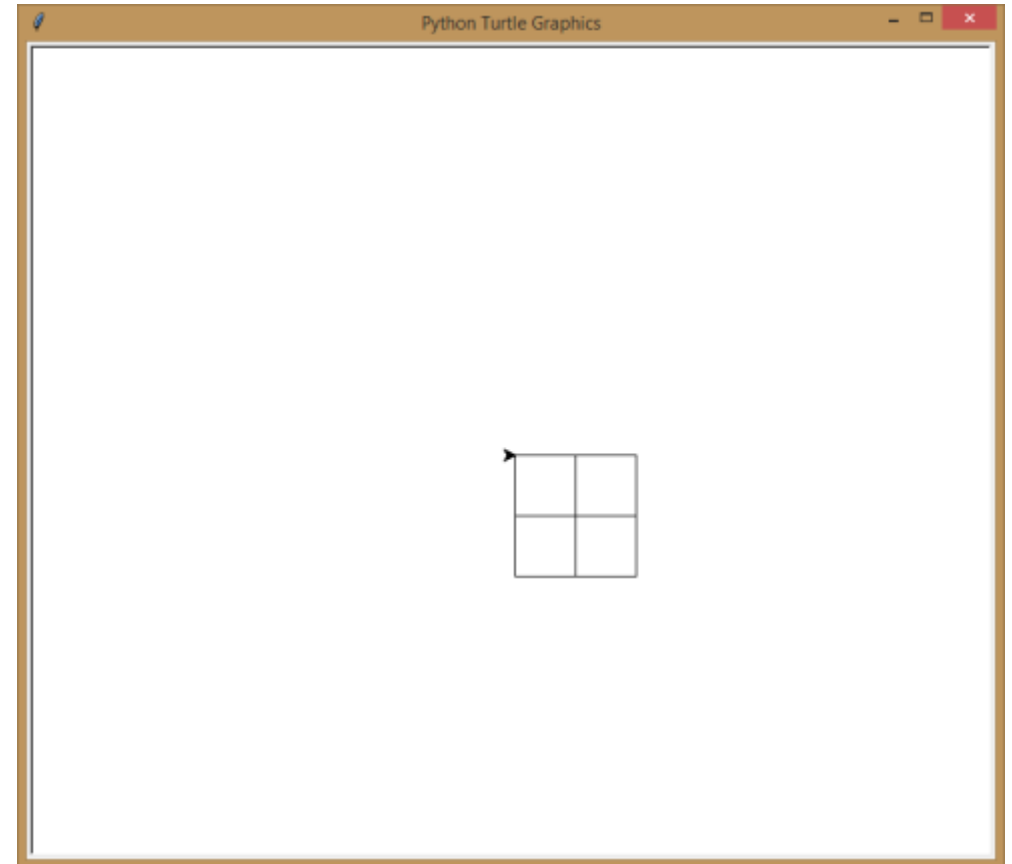
```
improt turtle
for steps in range(4)
    turtle.forward(100)
turtle.right(90)
```

```
import turtle
for steps in range(4):
    turtle.forward(100)
    turtle.right(90)
```

# Nested loops

You can have lots of fun when you put a loop inside another loop!

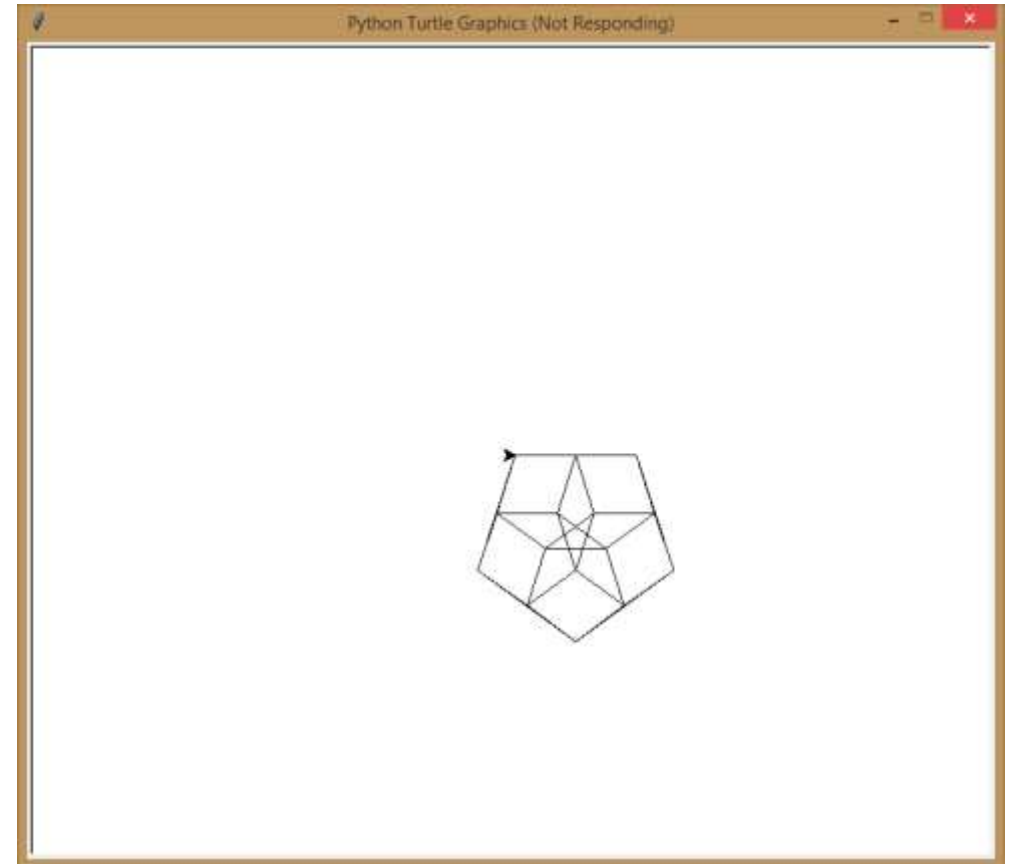
```
import turtle
for steps in range(4):
    turtle.forward(100)
    turtle.right(90)
    for moresteps in range(4):
        turtle.forward(50)
        turtle.right(90)
```





# Just for fun

```
import turtle
for steps in range(5):
    turtle.forward(100)
    turtle.right(360/5)
    for moresteps in range(5):
        turtle.forward(50)
        turtle.right(360/5)
```



# DEMO

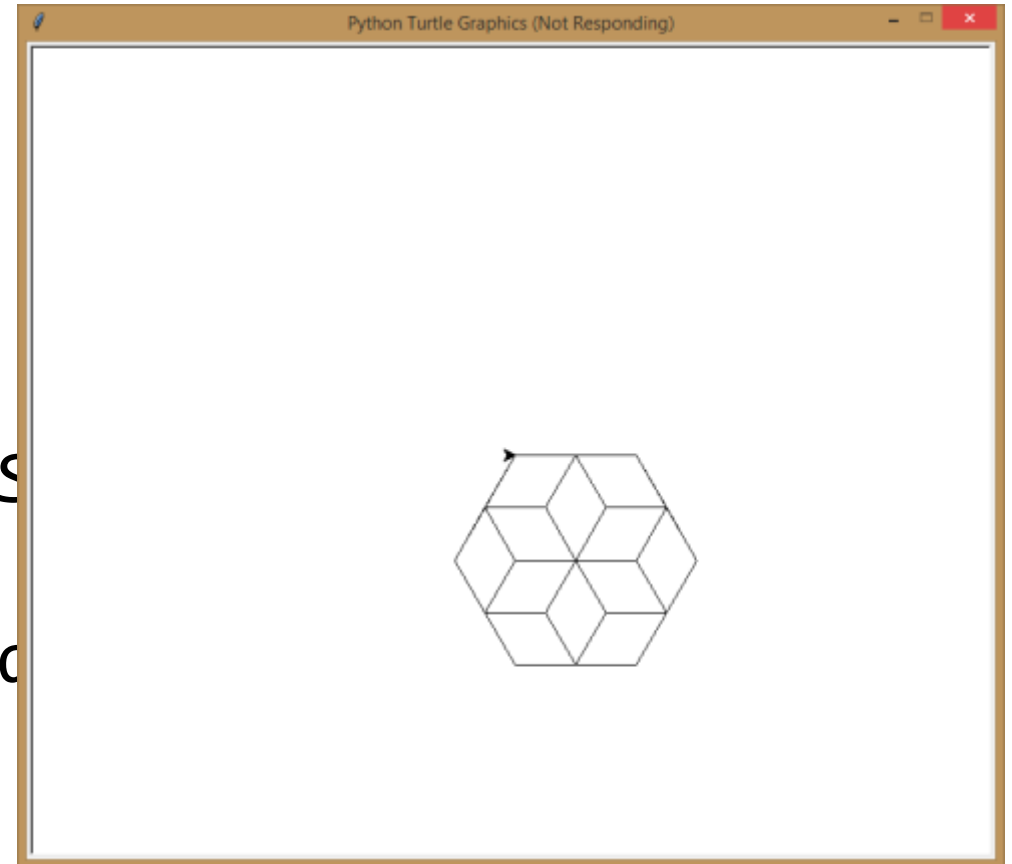
---

Nested loops

# Variables in loops

We could use a variable to decide the number of sides our object will have

```
import turtle
nbrSides = 6
for steps in range(nbrSides):
    turtle.forward(100)
    turtle.right(360/nbrSides)
    for moresteps in range(nbrSides):
        turtle.forward(50)
        turtle.right(360/nbrSides)
```



What's the advantage of using a variable here instead of just typing in the number?

```
import turtle
nbrSides = 6
for steps in range(nbrSides):
    turtle.forward(100)
    turtle.right(360/nbrSides)
    for moresteps in range(nbrSides):
        turtle.forward(50)
        turtle.right(360/nbrSides)
```

When we use a variable and we want to change a value that appears in many places, we only have to update one line of code!

```
import turtle
nbrSides = 6
for steps in range(nbrSides):
    turtle.forward(100)
    turtle.right(360/nbrSides)
    for moresteps in range(nbrSides):
        turtle.forward(50)
        turtle.right(360/nbrSides)
```

Now when we have to change our code, we are less likely to make a mistake by forgetting to change one of the values

# DEMO

---

Using a variable in our loop

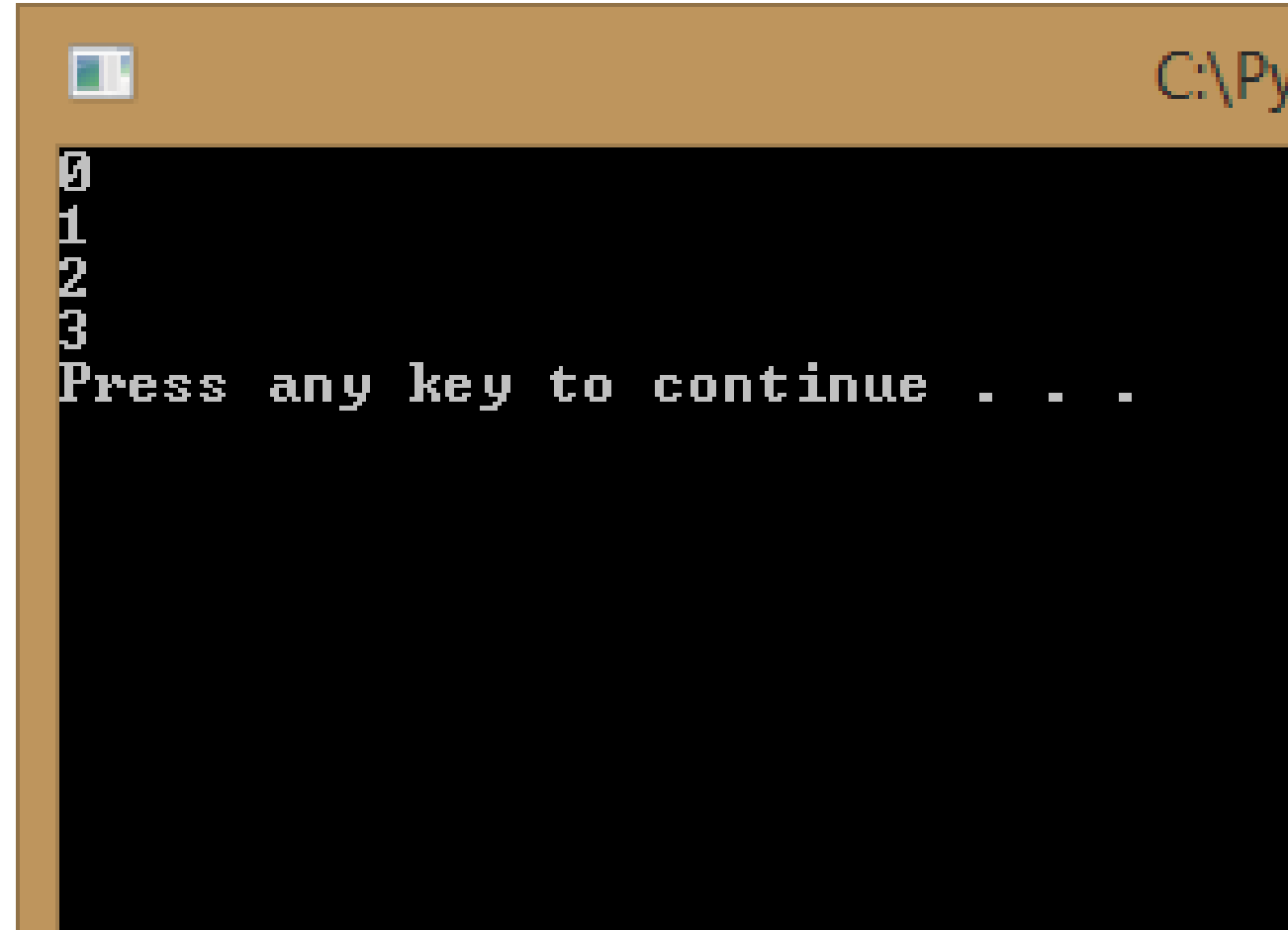
# Accessing the loop value



# You can look at the loop values within the loop

```
for steps in range(4) :  
    print(steps)
```

Yes, counting starts at zero in for loops, that's pretty common in programming



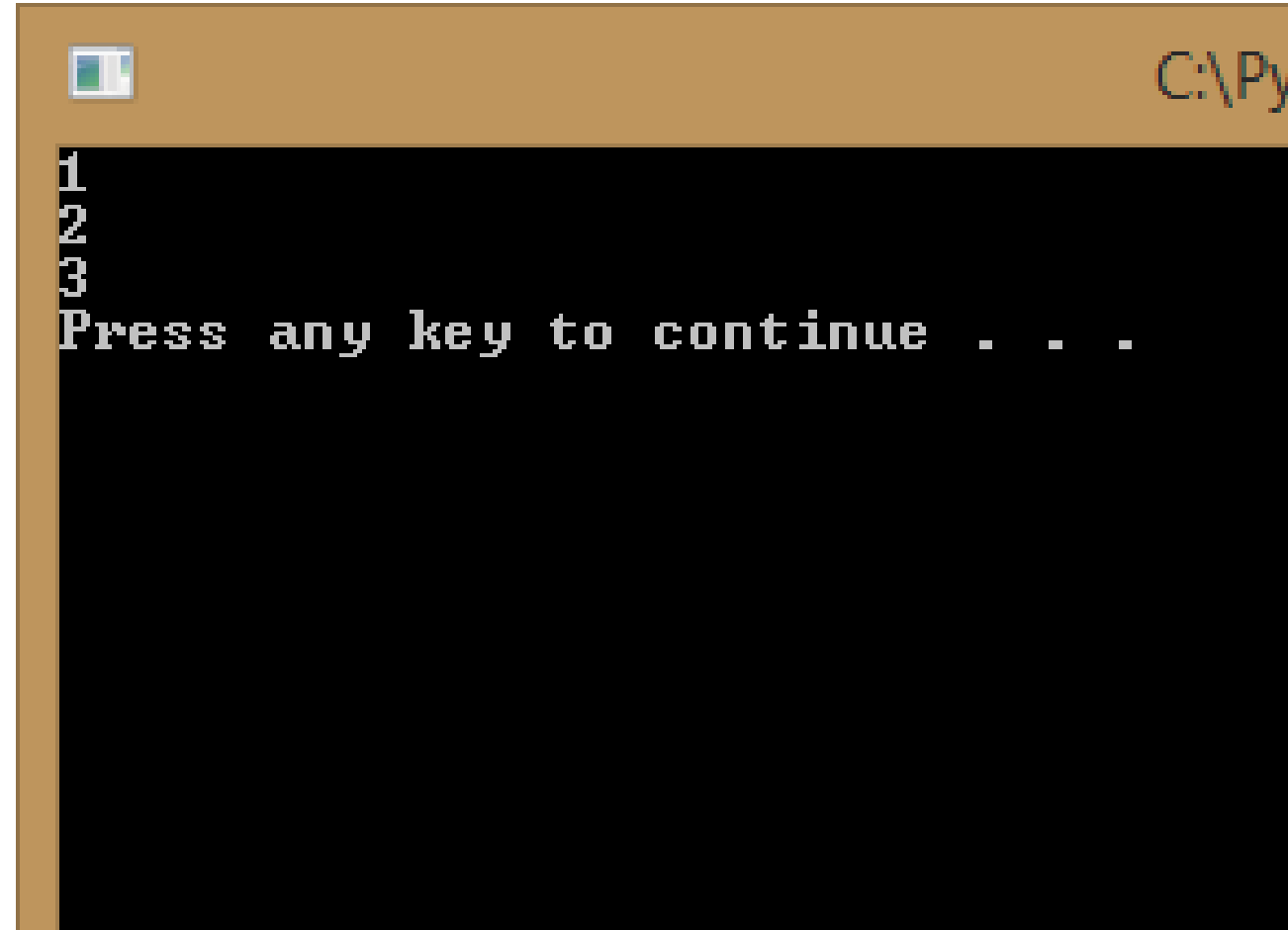
A screenshot of a Python shell window. The window has a title bar with a small icon on the left and the path 'C:\P...' on the right. The main area is black with white text. It shows the output of the code: '0', '1', '2', '3' on separate lines, followed by 'Press any key to continue . . .' on the next line.

```
C:\P...  
0  
1  
2  
3  
Press any key to continue . . .
```

If you need to start counting from "1" you can specify numbers to count to and from

```
for steps in range(1,4) :  
    print(steps)
```

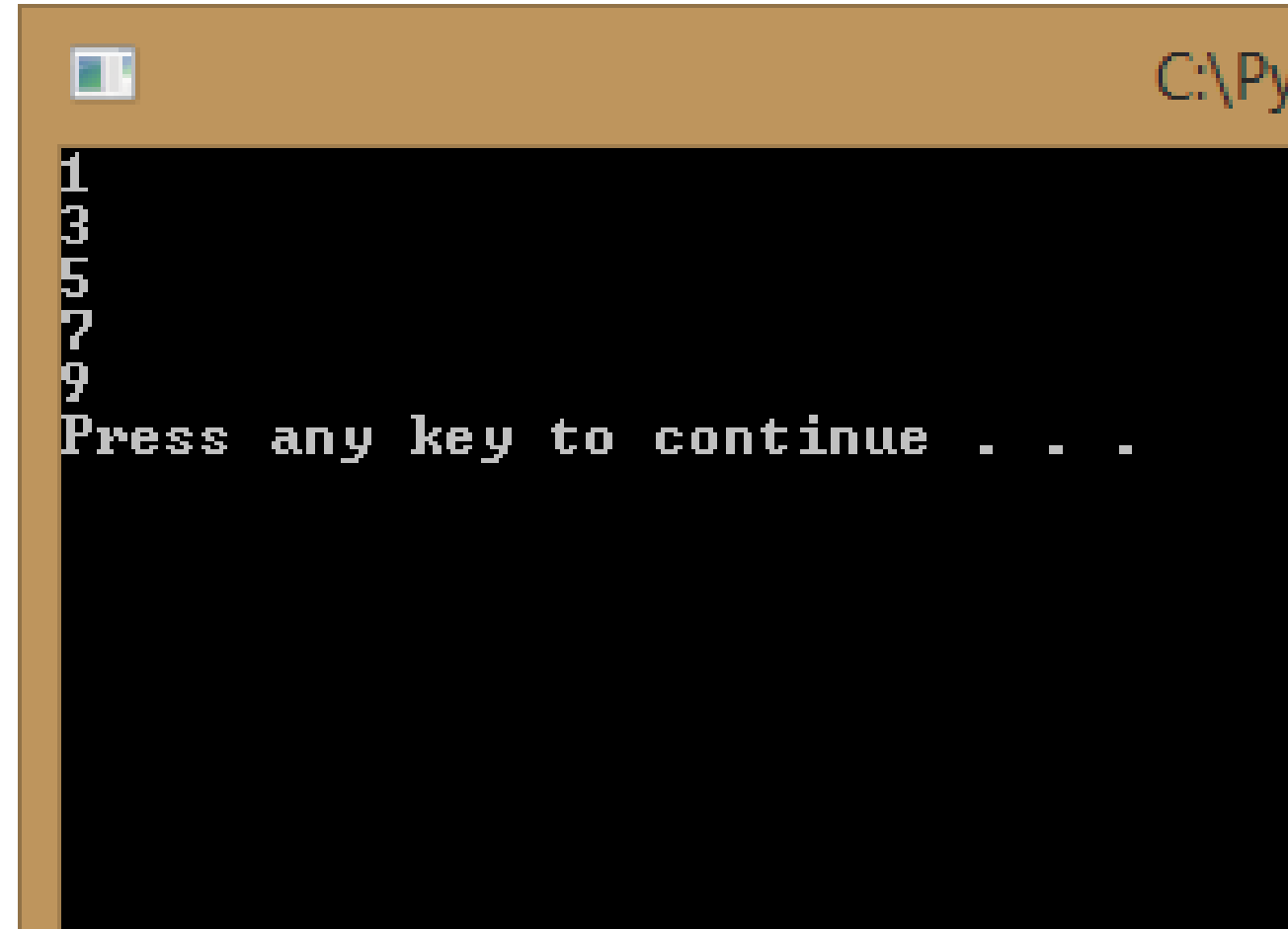
Did you notice this time the loop only executed three times?



```
C:\P>  
1  
2  
3  
Press any key to continue . . .
```

You can also tell the loop to skip values by specifying a step

```
for steps in range(1,10,2) :  
    print(steps)
```



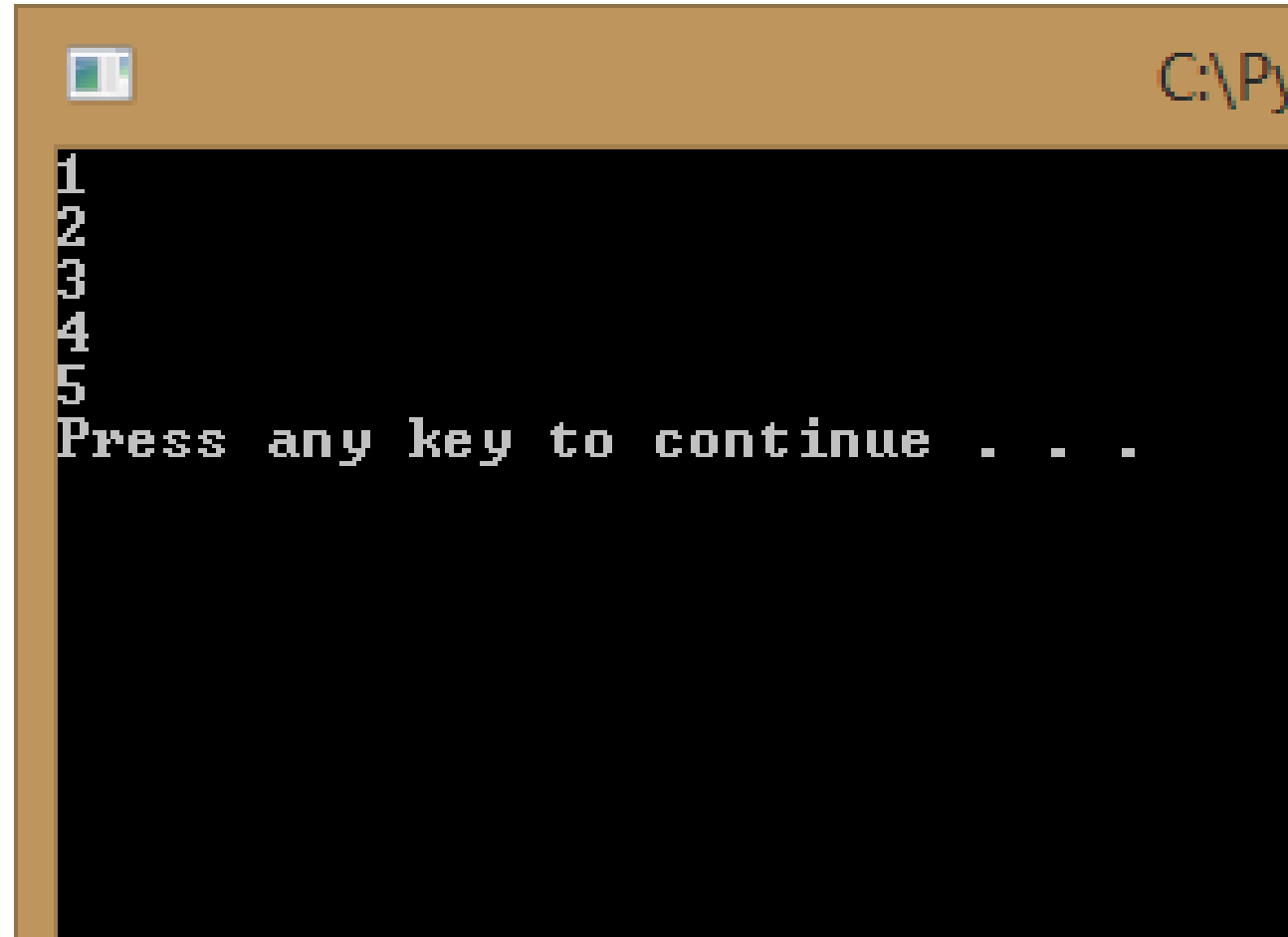
The screenshot shows a Python interpreter window with a brown title bar. The window title is partially visible as 'C:\P...'. The command prompt shows the execution of the code from the previous block. The output is a vertical list of odd numbers from 1 to 9. Below the output, the prompt 'Press any key to continue . . .' is displayed.

```
C:\P...  
1  
3  
5  
7  
9  
Press any key to continue . . .
```

One of the cool things about Python is the way you can tell it exactly what values you want to use in the loop

```
for steps in [1,2,3,4,5]:  
    print(steps)
```

This requires using [ ]  
brackets instead of ( ) and  
you don't use the "range"  
keyword

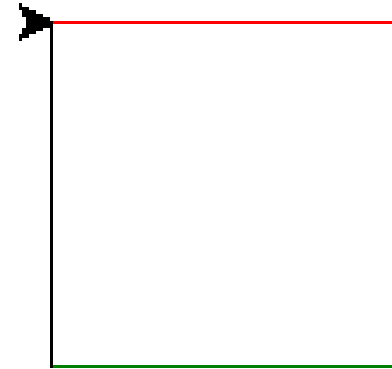
A screenshot of a Python interpreter window. The window has a title bar with a small icon on the left and the path 'C:\P...' on the right. The main area is a black terminal with white text. It shows the numbers 1, 2, 3, 4, and 5 printed on separate lines. Below these numbers, it says 'Press any key to continue . . .' with three dots.

```
1  
2  
3  
4  
5  
Press any key to continue . . .
```

# And you don't have to use numbers!

```
import turtle
for steps in ['red', 'blue', 'green', 'black'] :
    turtle.color(steps)
    turtle.forward(100)
    turtle.right(90)
```

What do you think this code will do?



# DEMO

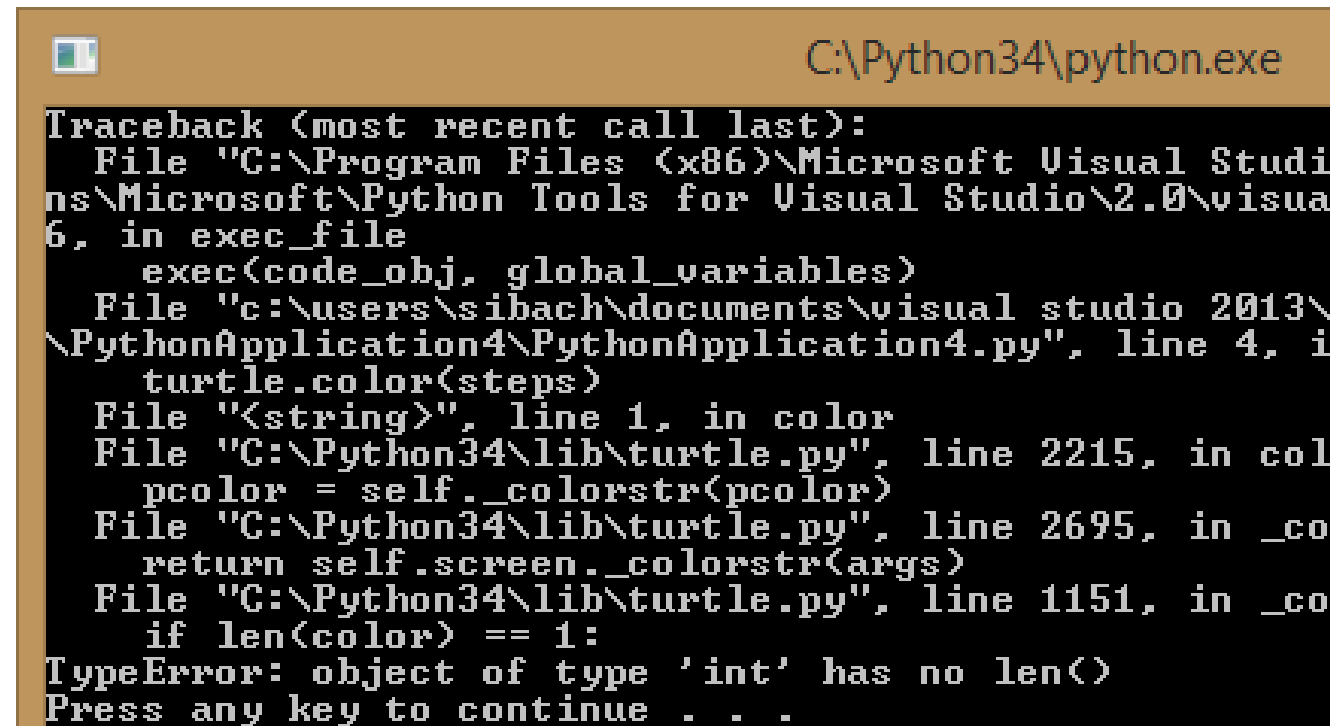
---

Using an explicit list of values in your loop

You can even mix up different datatypes (e.g. numbers and strings) but...

```
import turtle
for steps in ['red', 'blue', 'green', 'black', 8] :
    turtle.color(steps)
    turtle.forward(100)
    turtle.right(90)
```

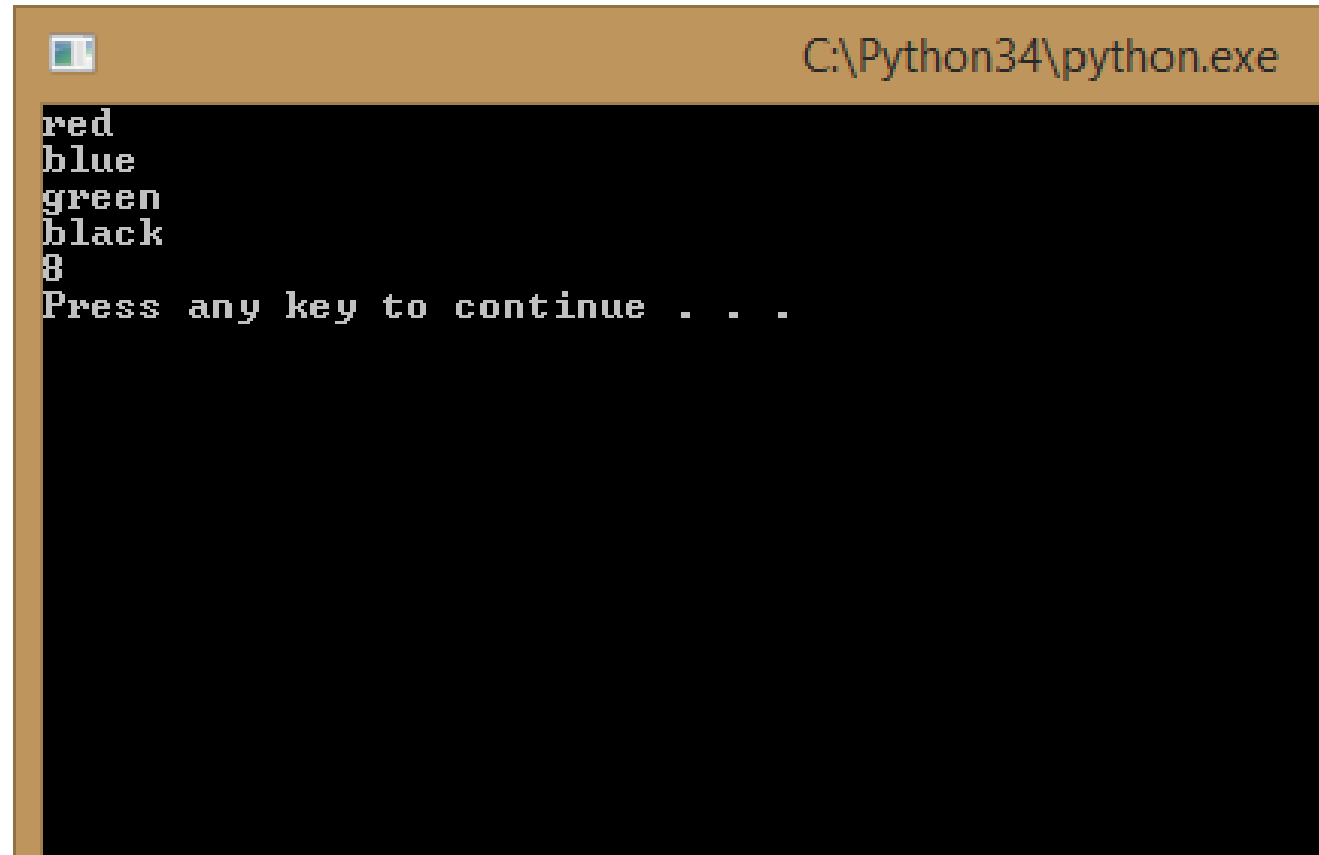
You had better make sure any code using that value can handle the different datatypes!

A screenshot of a Windows command prompt window with a black background and white text. The title bar at the top shows the file path 'C:\Python34\python.exe'. The text in the window is a Python traceback showing an error. It starts with 'Traceback (most recent call last):' followed by several lines of file and line numbers. The final line of the traceback is 'TypeError: object of type 'int' has no len()'. Below this, it says 'Press any key to continue . . .'.

```
C:\Python34\python.exe
Traceback (most recent call last):
  File "C:\Program Files (x86)\Microsoft Visual Studi
ns\Microsoft\Python Tools for Visual Studio\2.0\visua
6, in exec_file
    exec(code_obj, global_variables)
  File "c:\users\sibach\documents\visual studio 2013\
\PythonApplication4\PythonApplication4.py", line 4, i
    turtle.color(steps)
  File "<string>", line 1, in color
  File "C:\Python34\lib\turtle.py", line 2215, in col
    pcolor = self._colorstr(pcolor)
  File "C:\Python34\lib\turtle.py", line 2695, in _co
    return self.screen._colorstr(args)
  File "C:\Python34\lib\turtle.py", line 1151, in _co
    if len(color) == 1:
TypeError: object of type 'int' has no len()
Press any key to continue . . .
```

You can't set the color to a number so the code crashed, but print can accept strings or numbers

```
for steps in ['red', 'blue', 'green', 'black', 8] :  
    print (steps)
```



A screenshot of a Windows command prompt window titled "C:\Python34\python.exe". The window has a black background with white text. It displays the output of a Python script: "red", "blue", "green", "black", and "8" on separate lines. Below these, it says "Press any key to continue . . .". The window's title bar is brown and contains a small icon on the left and the file path on the right.

```
C:\Python34\python.exe  
red  
blue  
green  
black  
8  
Press any key to continue . . .
```



# Your challenge

- Get turtle to draw an octagon
- Hint: to calculate the angle, you take 360 degrees and divide it by the number of sides of the shape you are drawing
- Extra challenge: Let the user specify how many sides the object will have and draw whatever they ask
- Double bonus challenge, add a nested loop to draw a smaller version of the object inside!

# Congratulations

- You can manage problems which require repeating the same task over and over a fixed number of times





# Microsoft

©2013 Microsoft Corporation. All rights reserved. Microsoft, Windows, Office, Azure, System Center, Dynamics and other product names are or may be registered trademarks and/or trademarks in the U.S. and/or other countries. The information herein is for informational purposes only and represents the current view of Microsoft Corporation as of the date of this presentation. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information provided after the date of this presentation. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS PRESENTATION.