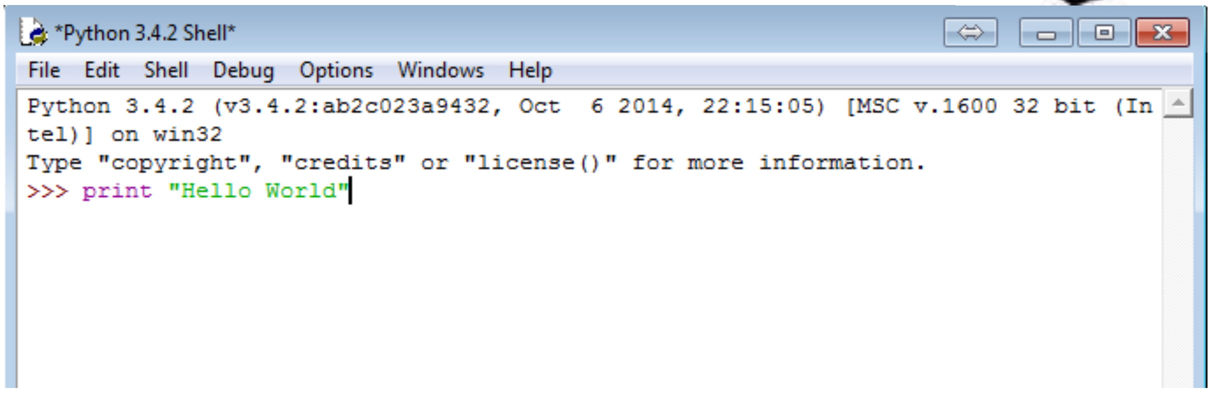
Python Tutorial 1 Basics

Getting the Computer to write to us (Printing and Strings)

For now let's start of really easy and make something called a <u>Print statement</u>

A print statement is a way the computer can talk to us through text.

In this example were going to get the computer to say "Hello World"



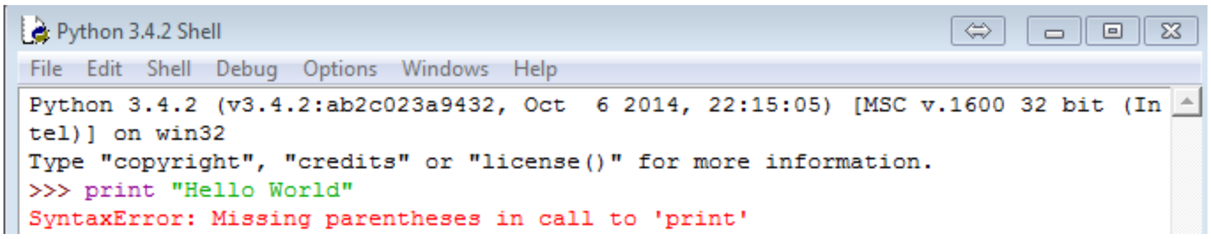
```
Python 3.4.2 (v3.4.2:ab2c023a9432, Oct  6 2014, 22:15:05) [MSC v.1600 32 bit (In
tel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print "Hello World"
```

Code:

print "Hello World"

Press Enter to get the computer to say "Hello World"

<u>Did it work? It shouldn't have!</u>



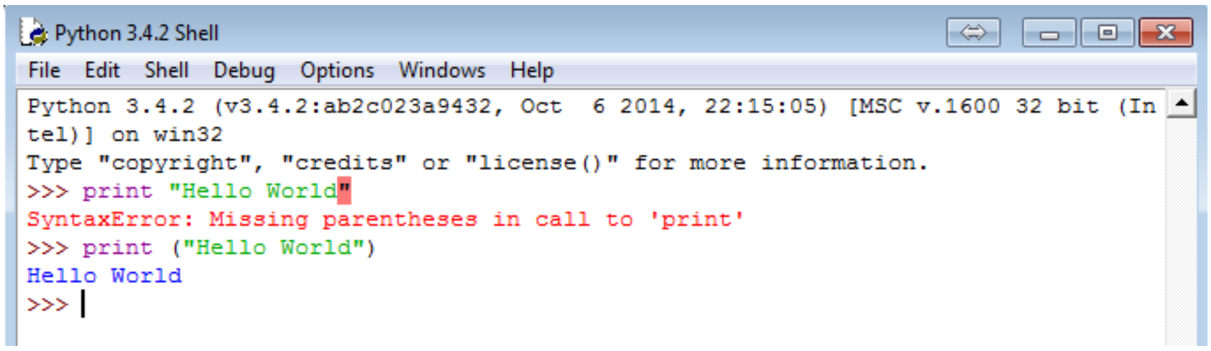
```
Python 3.4.2 (v3.4.2:ab2c023a9432, Oct  6 2014, 22:15:05) [MSC v.1600 32 bit (In
tel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print "Hello World"
SyntaxError: Missing parentheses in call to 'print'
```

In our statement we are missing the parentheses see if you can fix the mistake.

<u>If you don't know what a parentheses is then go to the next page or ask a mentor.</u>

Go to the next page for the answer

Hopefully you managed to solve the small mistake. This mistake may seem simple now but in 1000 lines of code it could be really hard to find. So make sure you check your code and test it as you write it!



```
Python 3.4.2 (v3.4.2:ab2c023a9432, Oct  6 2014, 22:15:05) [MSC v.1600 32 bit (In
tel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print "Hello World"
SyntaxError: Missing parentheses in call to 'print'
>>> print ("Hello World")
Hello World
>>> 
```

Code:

print ("Hello World")

Advanced description! ☺

You basically got the computer to print out something called a string

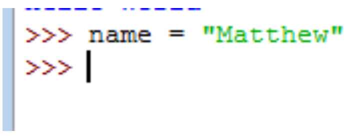
A string is basically a sequence of characters (e.g. a . @ 4 $ Fred) there are other data types such as floats, ints, tuples etc but we will cover those later on.

Variables

Imagine a variable as being a chest. You can place things inside the chest, check what is inside the chest and take things out of the chest. We can also name our chest too! ☺

Here is an example of creating our chest in python. In this example this chest will be called name and you will put your own name inside it.

Here is my chest:



```
>>> name = "Matthew"
>>> 
```

Code:

name = "Your Name Here "

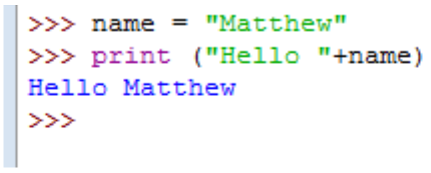
Press Enter

Don't worry if nothing appears. This is normal ☺

We are now going to look into our chest and get the PC to tell us what's inside by using the print command we learnt previously. I've also added a "Hello " at the start so our computer is polite ☺.

Code:

print ("Hello "+name)



```
>>> name = "Matthew"
>>> print ("Hello "+name)
Hello Matthew
>>>
```

Advanced Description!

A variable is a storage location paired with a symbolic name or identifier which may contain a known or unknown piece of information typically referred to as a value. If you don't understand don't worry it will make sense once you go through the exercise.

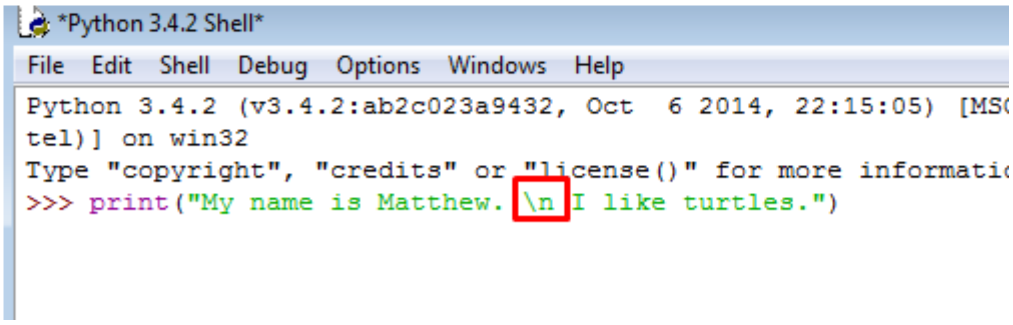
When we add our variables to strings it's known as **concatenation**. Also notice that I added an extra space after "Hello ". See what happens when you remove the space.

Extra String operations and Line Art ♪┏(°.°)┛ ┗(°.°)┓ ┗(°.°)┛ ┏(°.°)┓ ♪

New Line Command

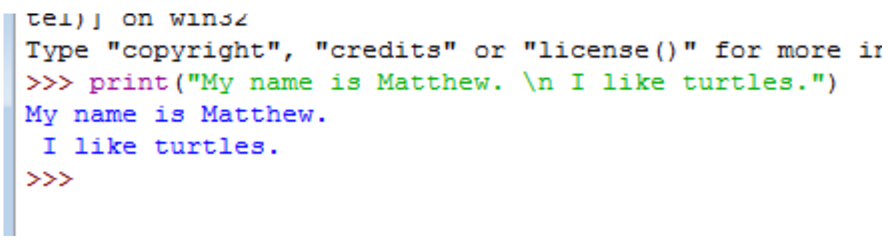
By now I'm sure you realise that YOU control what the computer writes back.

We can create new lines in python without having to call the print function more than once by including a new line "\n".



```
Python 3.4.2 (v3.4.2:ab2c023a9432, Oct  6 2014, 22:15:05) [MS
tel)] on win32
Type "copyright", "credits" or "license()" for more informati
>>> print("My name is Matthew. \n I like turtles.")
```

Result:

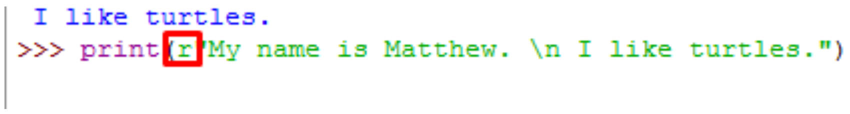


```
tel)] on win32
Type "copyright", "credits" or "license()" for more in
>>> print("My name is Matthew. \n I like turtles.")
My name is Matthew.
 I like turtles.
>>>
```

Code:

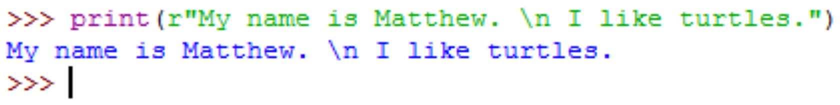
print("My name is Matthew. \n I like turtles.")

We can also let python ignore the “\n” command and just print it out normally by typing r before the string.



```
 I like turtles.
>>> print(r"My name is Matthew. \n I like turtles.")
```

Output:



```
>>> print(r"My name is Matthew. \n I like turtles.")
My name is Matthew. \n I like turtles.
>>> 
```

Code:

print(r"My name is Matthew. \n I like turtles.")

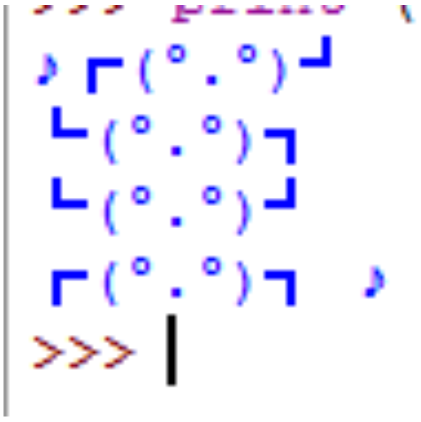
Small challenge

With the skills you've learnt can you get these guys to dance on individual lines?

You should be able to copy and paste these guys into python without an issue

♪┏(°.°)┛ ┗(°.°)┓ ┗(°.°)┛ ┏(°.°)┓ ♪

Like this?



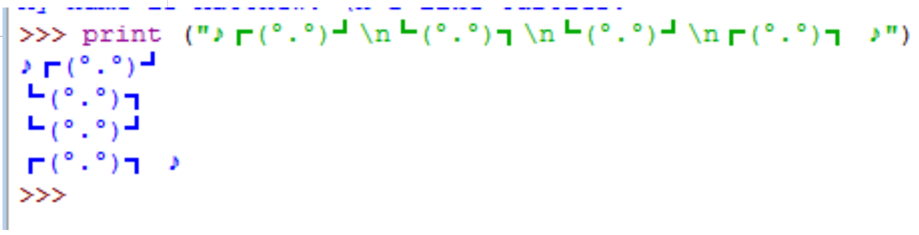
```
♪┏(°.°)┛
 ┗(°.°)┓
 ┗(°.°)┛

 ┏(°.°)┓   ♪
>>> 
```

Answer on the next page ┏(°.°)┛

Why not try using other forms of Line art and see what you can come up with!

(•_•) (•_•)>⌐■-■ (⌐■_■)

Answer



```
>>> print ("♪ ┏(°.°)┛ \n ┗(°.°)┓ \n ┗(°.°)┛ \n ┏(°.°)┓  ♪")
♪ ┏(°.°)┛
 ┗(°.°)┓
 ┗(°.°)┛
 ┏(°.°)┓  ♪
>>>
```

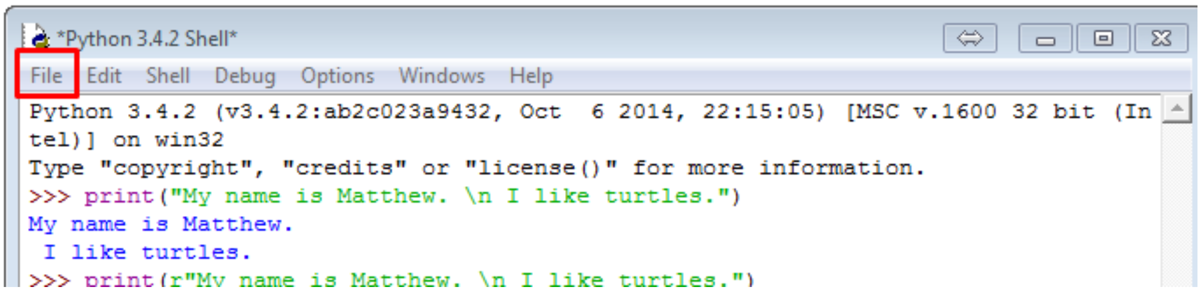
Code:

print ("♪┏(°.°)┛ \n ┗(°.°)┓ \n ┗(°.°)┛ \n ┏(°.°)┓ ♪")

Creating a Python Script file

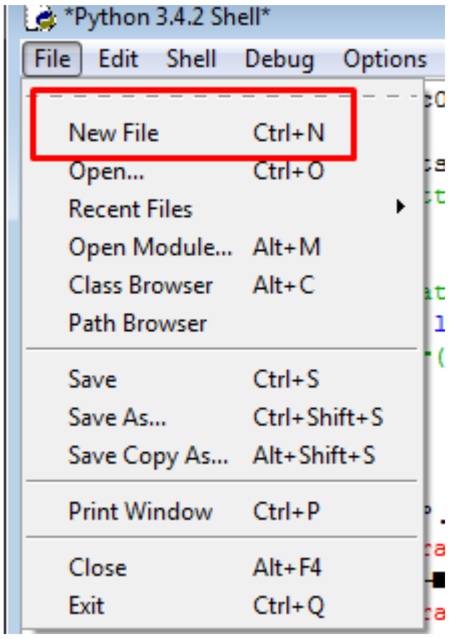
If you want to save all your code without having to re write it all over again then a script file is what you need!

Click File

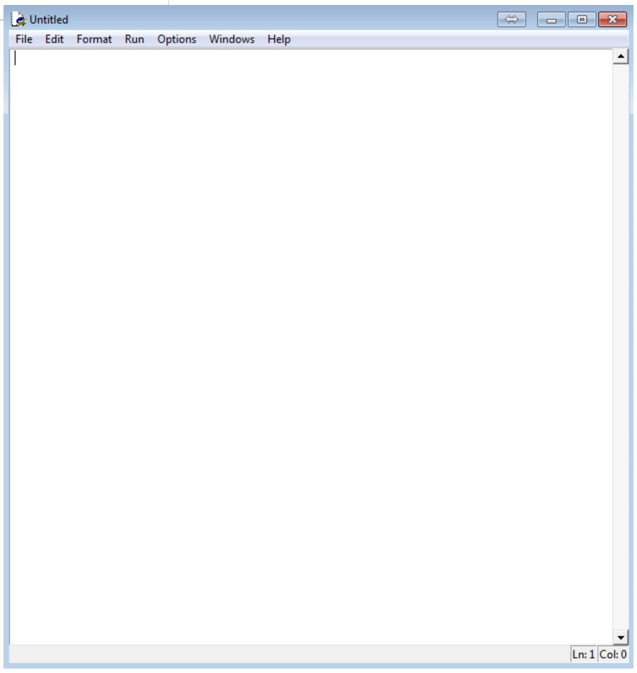


```
Python 3.4.2 (v3.4.2:ab2c023a9432, Oct  6 2014, 22:15:05) [MSC v.1600 32 bit (In
tel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print("My name is Matthew. \n I like turtles.")
My name is Matthew.
 I like turtles.
>>> print(r"My name is Matthew. \n I like turtles.")
```

Then Click on New File or simply press Ctrl+N ༼つ ◕_◕ ༽つ (It should be Command + N for Apple users, Python should tell you what the keyboard shortcut is on the side).

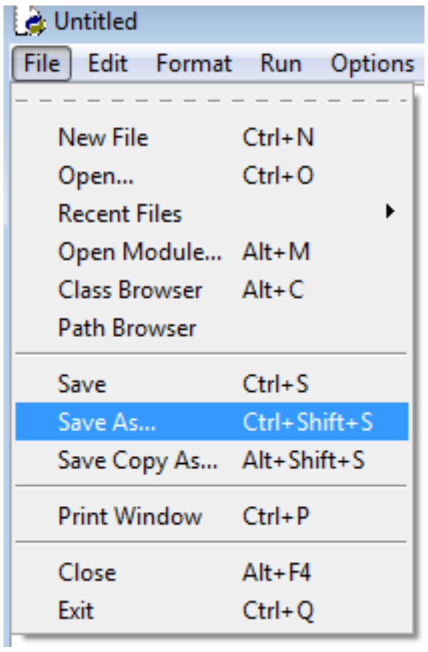


This should pop up

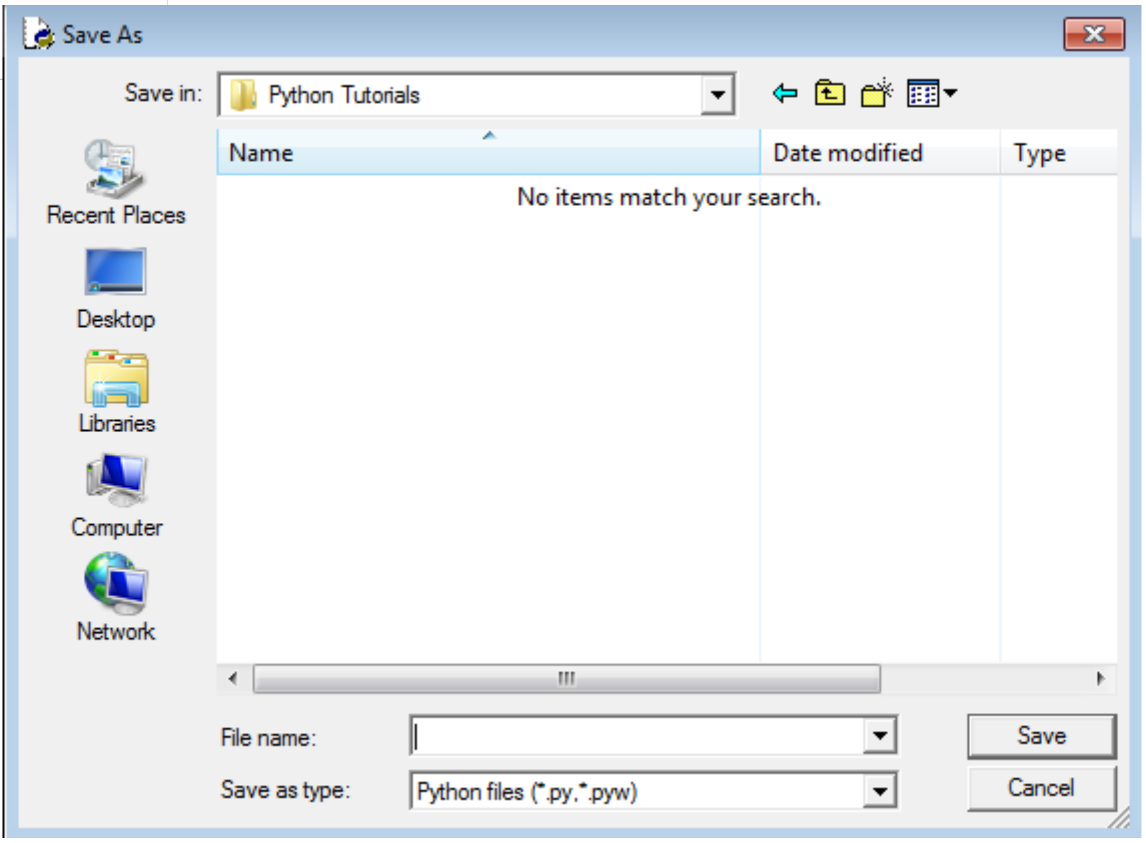


Before we start adding our code we should save it first.

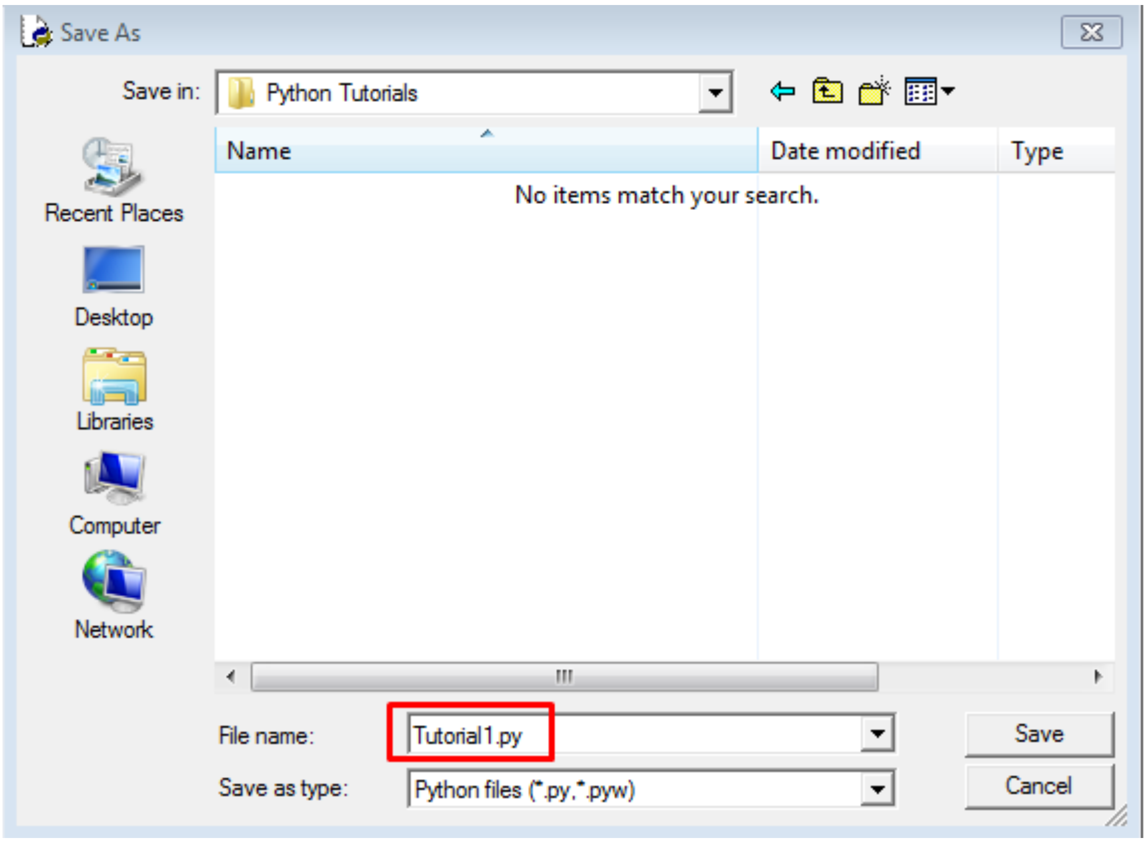
In the new window click file and then Save As



Navigate to a suitable location for you script



IMPORTANT! When creating the file name make sure you add .py at the end!

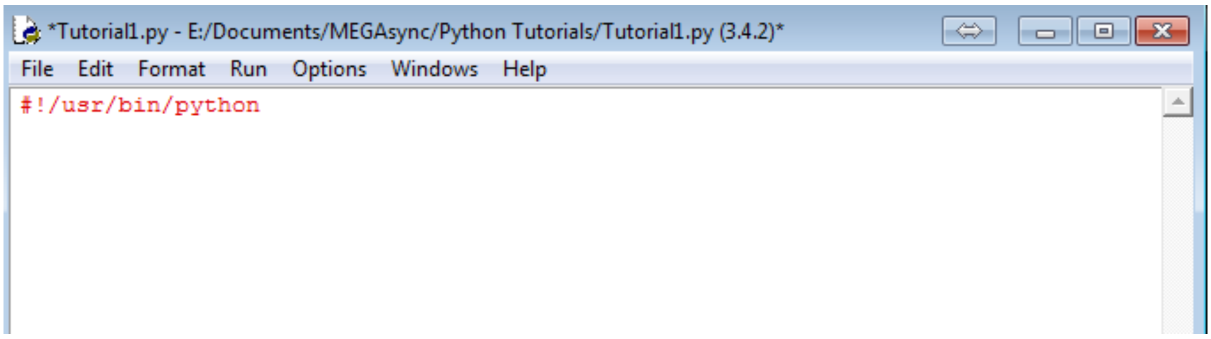


Once you have done that you can start adding your code here without losing it in IDLE.

Make sure you add this to the first line of all the scripts you write:

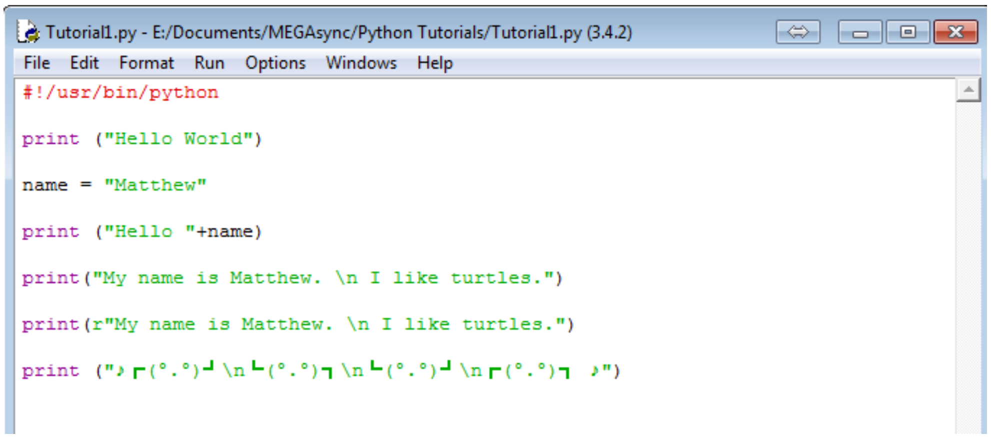
#!/usr/bin/python

For now I won't explain what this line does but feel free to ask a mentor if you would like to know more.


Tutorial1.py - E:/Documents/MEGAsync/Python Tutorials/Tutorial1.py (3.4.2)
File Edit Format Run Options Windows Help

```
#!/usr/bin/python
```

Let's add all the examples to this one script and run it all together!

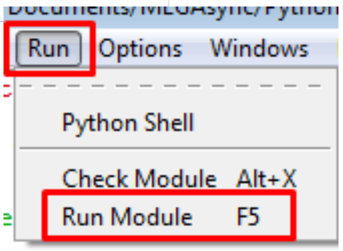

Tutorial1.py - E:/Documents/MEGAsync/Python Tutorials/Tutorial1.py (3.4.2)
File Edit Format Run Options Windows Help

```
#!/usr/bin/python

print ("Hello World")

name = "Matthew"

print ("Hello "+name)

print("My name is Matthew. \n I like turtles.")

print(r"My name is Matthew. \n I like turtles.")

print ("♪┏(°.°)┛ \n┗(°.°)┓ \n┗(°.°)┛ \n┏(°.°)┓ ♪")
```

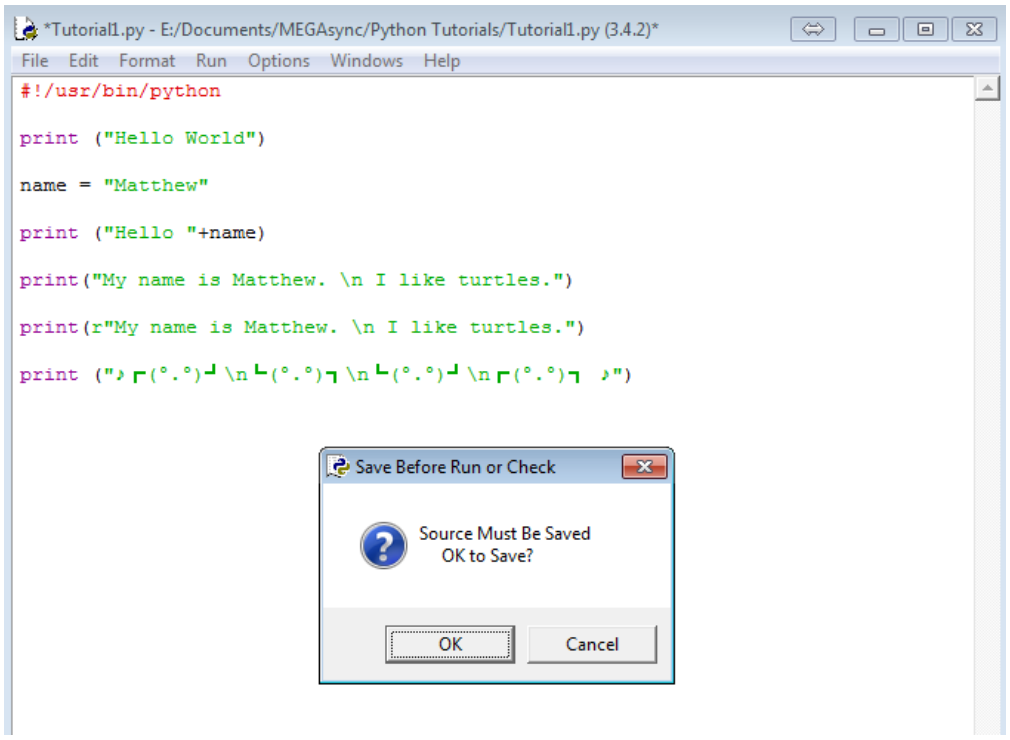
Code:

```
#!/usr/bin/python

print ("Hello World")

name = "Matthew"

print ("Hello "+name)

print("My name is Matthew. \n I like turtles.")

print(r"My name is Matthew. \n I like turtles.")

print ("♪┏(°.°)┛ \n ┗(°.°)┓ \n ┗(°.°)┛ \n ┏(°.°)┓ ♪")
```

Then press F5 to execute or click Run then Run Module

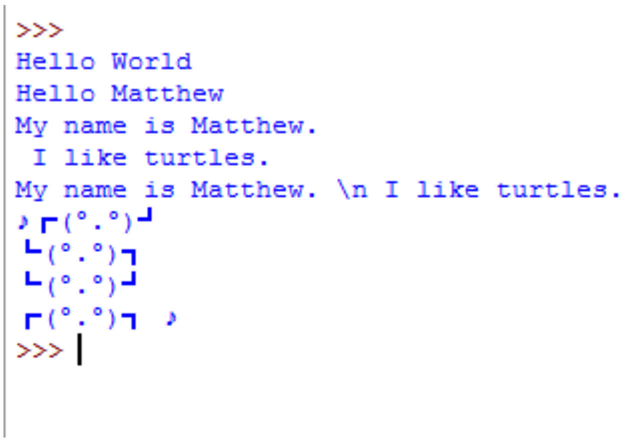


Click Ok to save the code


Tutorial1.py - E:/Documents/MEGAsync/Python Tutorials/Tutorial1.py (3.4.2)
File Edit Format Run Options Windows Help
Save Before Run or Check
Source Must Be Saved
OK to Save?
OK
Cancel

```python
#!/usr/bin/python

print ("Hello World")

name = "Matthew"

print ("Hello "+name)

print("My name is Matthew. \n I like turtles.")

print(r"My name is Matthew. \n I like turtles.")

print ("♪┏(°.°)┛ \n┗(°.°)┓ \n┗(°.°)┛ \n┏(°.°)┓ ♪")
```

And here is all the code examples ran all together!



```
>>>
Hello World
Hello Matthew
My name is Matthew.
 I like turtles.
My name is Matthew. \n I like turtles.
♪┏(°.°)┛
┗(°.°)┓
┗(°.°)┛
┏(°.°)┓ ♪
>>> 
```

Python Tutorial 2 If and Else Conditions

Getting the Computer to make a decision

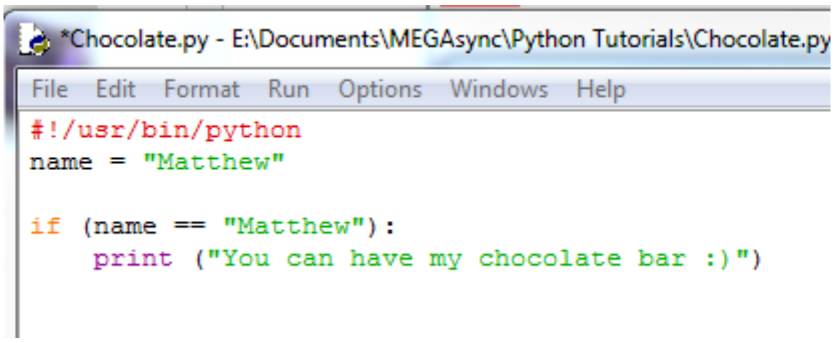
If:

Let’s pretend the Computer has a chocolate bar and it will only give it to someone called Matthew. How could we do this?

Well! We can use something called an If Condition don’t worry its very simple and easy to understand☺.

From looking at the previous tutorial create your own python file and name it
Chocolate.py

Add the code below and run it!


*Chocolate.py - E:\Documents\MEGAsync\Python Tutorials\Chocolate.py
File
Edit
Format
Run
Options
Windows
Help

```
#!/usr/bin/python
name = "Matthew"

if (name == "Matthew"):
    print ("You can have my chocolate bar :)")
```

Code:

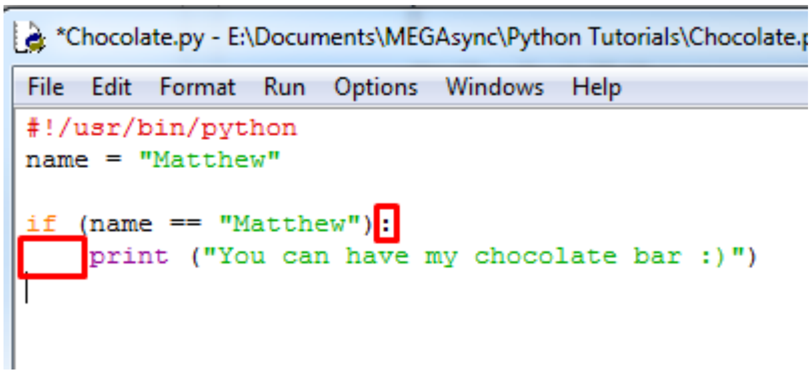
#!/usr/bin/python

name = "Matthew"

if (name == "Matthew"):

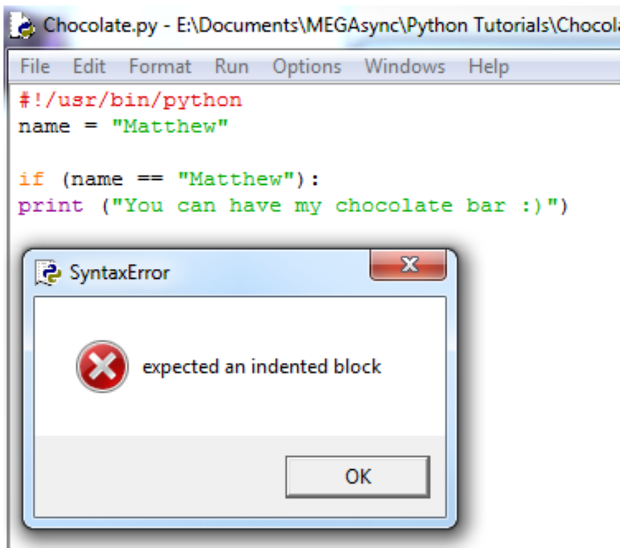
    print ("You can have my chocolate bar :)")

DO NOT remove the space before the word print. By typing the : and pressing enter Python will automatically create the space.


*Chocolate.py - E:\Documents\MEGAsync\Python Tutorials\Chocolate.p
File
Edit
Format
Run
Options
Windows
Help

```
#!/usr/bin/python
name = "Matthew"

if (name == "Matthew"):
    print ("You can have my chocolate bar :)")
```

The following error will appear if you remove the space.

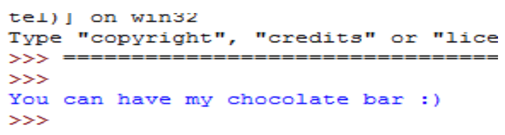

Chocolate.py - E:\Documents\MEGAsync\Python Tutorials\Chocol
File Edit Format Run Options Windows Help
SyntaxError
OK

```
#!/usr/bin/python
name = "Matthew"

if (name == "Matthew"):
print ("You can have my chocolate bar :)")
```

expected an indented block

To fix this error just press tab before the word print.

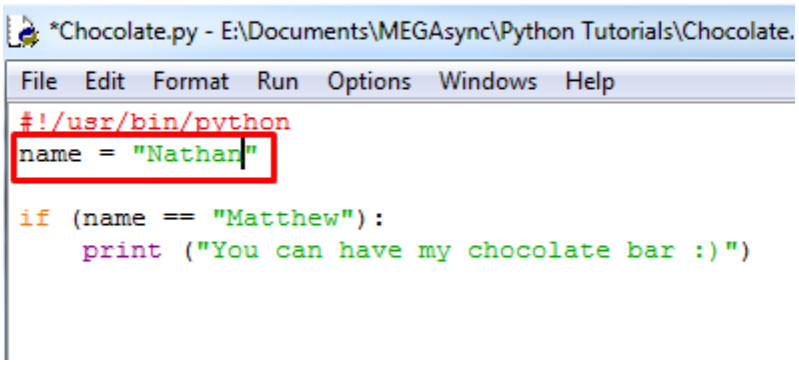
When you have fixed any errors Python should say:



```
tel)] on win32
Type "copyright", "credits" or "lice
>>> ================================
>>>
You can have my chocolate bar :)
>>>
```

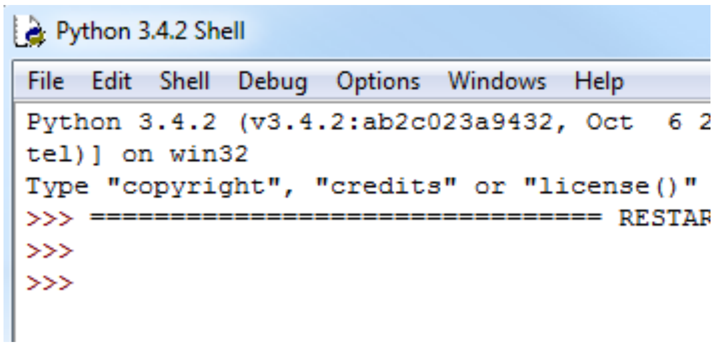
BUT what if the person's name isn't Matthew???

Well let's see what our code does when we change the name inside our "name" variable. (Check the first tutorial if you don't know what a variable is).


*Chocolate.py - E:\Documents\MEGAsync\Python Tutorials\Chocolate.
File Edit Format Run Options Windows Help

```
#!/usr/bin/python
name = "Nathan"

if (name == "Matthew"):
    print ("You can have my chocolate bar :)")
```

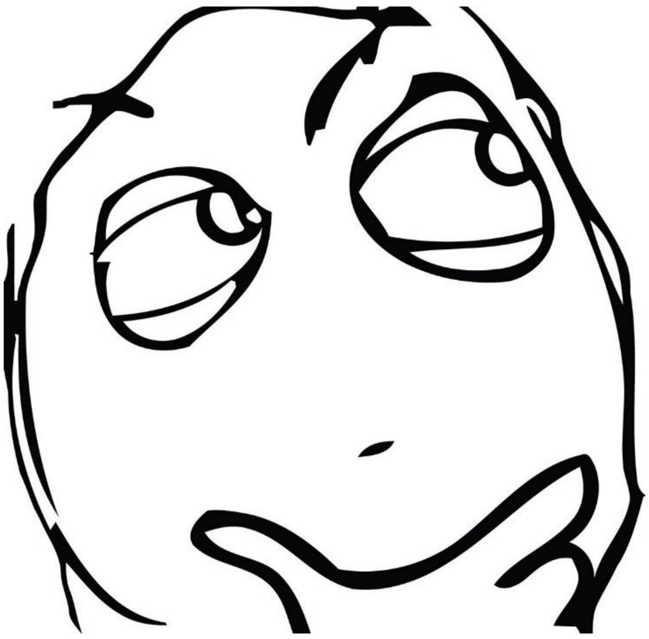
I changed the name from Matthew to Nathan but I don't want Nathan to have the chocolate because he's not cool enough. Let's see what happens:



```
Python 3.4.2 (v3.4.2:ab2c023a9432, Oct  6 2
tel)] on win32
Type "copyright", "credits" or "license()"
>>> ================================ RESTAR
>>>
>>>
```

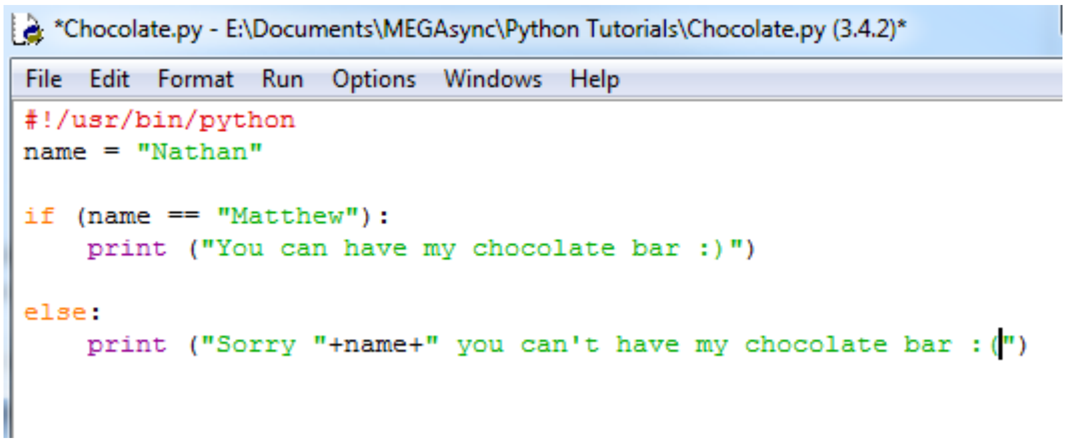
Oh dear! Python doesn't say anything. Let's fix that!

By using something called an Else statement ☺



Think of else as the word “Otherwise” or “then”. So if the name is not Matthew “then” we should do this instead.

Try out the following code below:


Chocolate.py - E:\Documents\MEGAsync\Python Tutorials\Chocolate.py (3.4.2)
File Edit Format Run Options Windows Help

```
#!/usr/bin/python
name = "Nathan"

if (name == "Matthew"):
    print ("You can have my chocolate bar :)")

else:
    print ("Sorry "+name+" you can't have my chocolate bar :(")
```

Code:

#!/usr/bin/python

name = "Nathan"

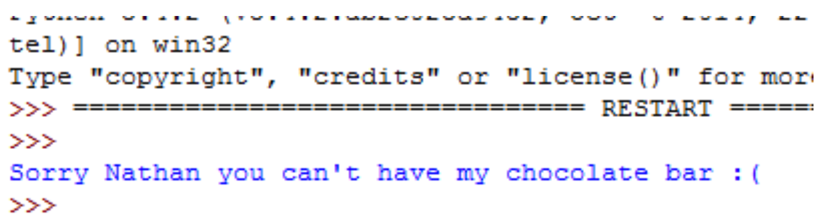
if (name == "Matthew"):

    print ("You can have my chocolate bar :)")

else:

    print ("Sorry "+name+" you can't have my chocolate bar :(")

Let’s see what happens :D



```
tel)] on win32
Type "copyright", "credits" or "license()" for mor
>>> ================================ RESTART =====
>>>
Sorry Nathan you can't have my chocolate bar :(
>>>
```

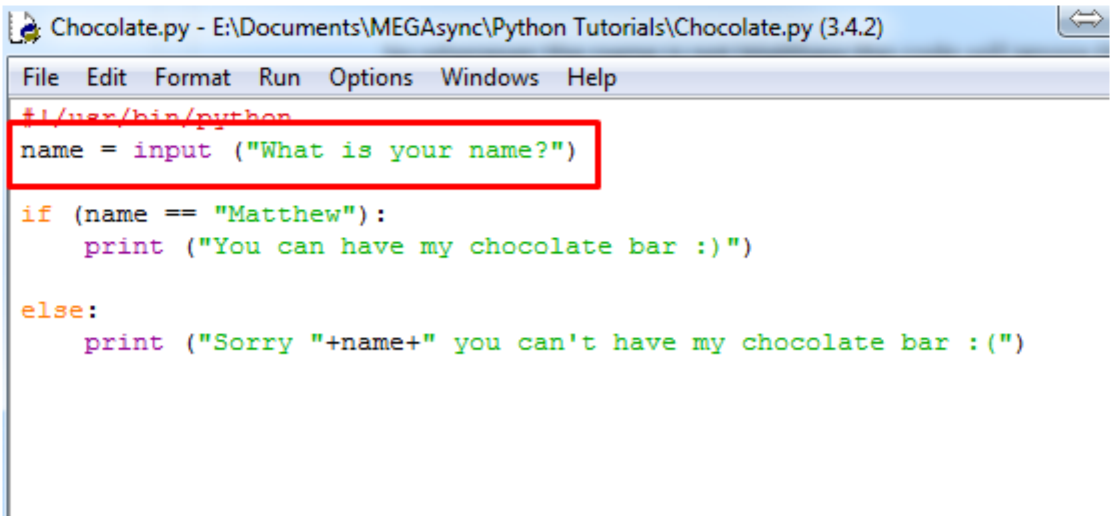
So whenever the name is not Matthew the code will ignore the if statement and go to the else statement instead.

If you’re still confused feel free to ask a mentor to help you understand ☺

Let’s enhance the code a little bit with an input statement so it’s easier to change the names around.

Input:

Here I have added an input statement to our variable.


Chocolate.py - E:\Documents\MEGAsync\Python Tutorials\Chocolate.py (3.4.2)
File Edit Format Run Options Windows Help

```
#!/usr/bin/python
name = input ("What is your name?")

if (name == "Matthew"):
    print ("You can have my chocolate bar :)")

else:
    print ("Sorry "+name+" you can't have my chocolate bar :(")
```

Code:

#!/usr/bin/python

name = input ("What is your name?")

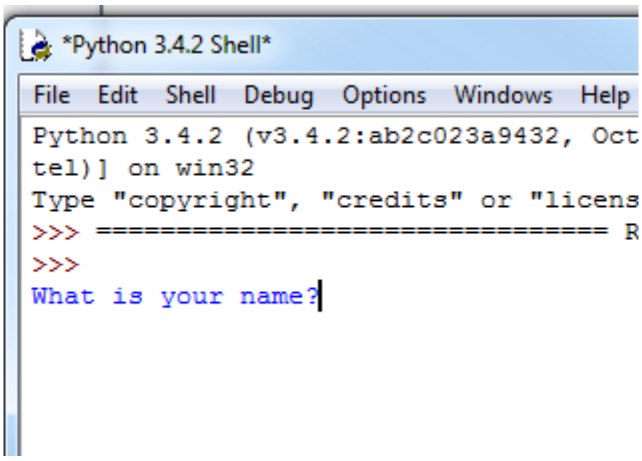
if (name == "Matthew"):

    print ("You can have my chocolate bar :)")

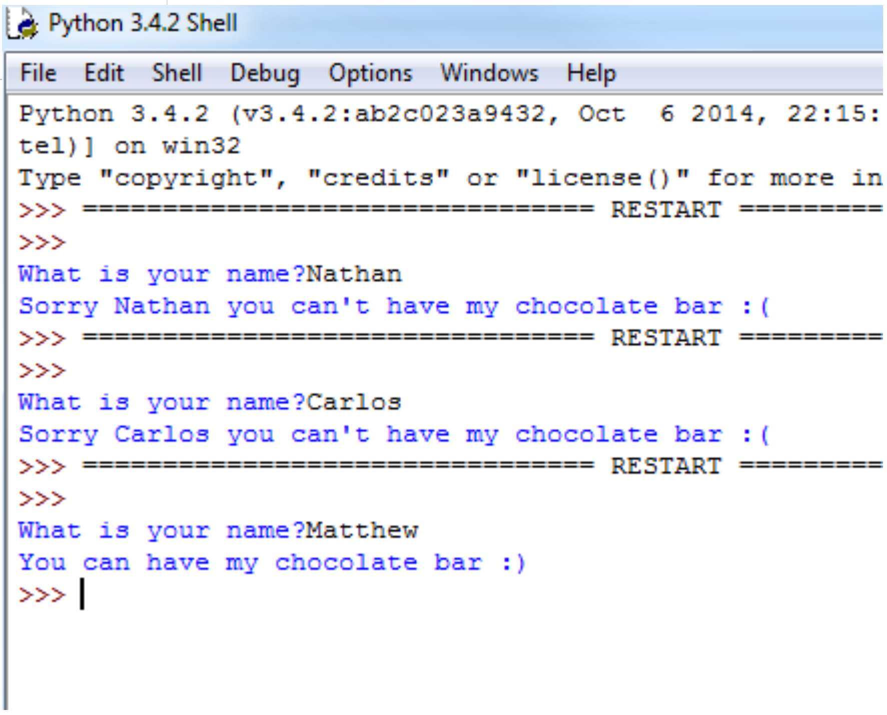
else:

    print ("Sorry "+name+" you can't have my chocolate bar :(")

When you first run the code this will pop up. Feel free to write anything inside.



```
Python 3.4.2 (v3.4.2:ab2c023a9432, Oct
tel)] on win32
Type "copyright", "credits" or "licens
>>> ================================ R
>>>
What is your name?
```



```
Python 3.4.2 (v3.4.2:ab2c023a9432, Oct  6 2014, 22:15:
tel)] on win32
Type "copyright", "credits" or "license()" for more in
>>> ================================ RESTART =========
>>>
What is your name?Nathan
Sorry Nathan you can't have my chocolate bar :(
>>> ================================ RESTART =========
>>>
What is your name?Carlos
Sorry Carlos you can't have my chocolate bar :(
>>> ================================ RESTART =========
>>>
What is your name?Matthew
You can have my chocolate bar :)
>>>
```

Now we can give the computer all sorts of names and get it to respond back :D

Loops

The great thing about Computers is that it's very easy for them to do the same thing over and over again. Without us having to write lots of lines of code. They are known as Loops, there are two loops used in programming "For" and "While" loops.

The idea behind a loop is to keep doing the same thing over and over until a criteria is met.

For example the dog will not stop chasing his tail until he catches it

However when the dog catches the tail the dog will stop chasing it because he has the tail (criteria is met). So the loop ends ☺. Feel free to ask the mentors if this still seems confusing.

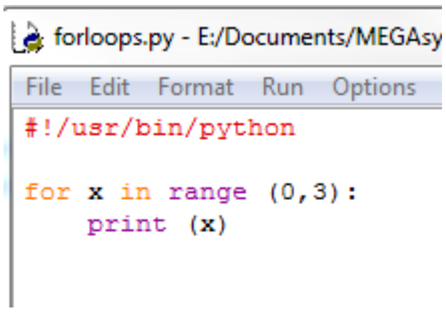
This idea applies to For and While loops ☺

For loops

So let's create a for loop that counts from 0 to 2 and then stops.

Create a new python file and call it forloops.py

Put in the following code:


forloops.py - E:/Documents/MEGAsy
File Edit Format Run Options

```
#!/usr/bin/python

for x in range (0,3):
    print (x)
```

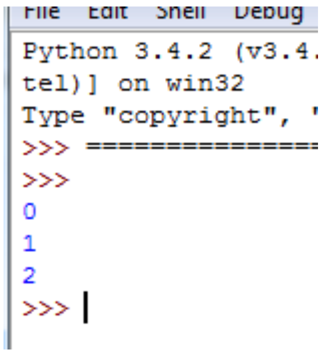
Code:

#!/usr/bin/python

for x in range (0,3):

  print (x)

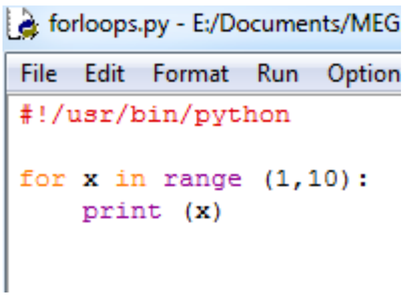
Result:



```
Python 3.4.2 (v3.4.
tel)] on win32
Type "copyright", '
>>> ===============
>>>
0
1
2
>>>
```

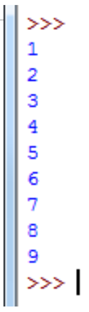
We can increase the ranges too, and make it count up to bigger numbers or make it start from a different a number besides zero by changing what's in the brackets.

Can you guess what numbers python will print out in this example without running the code?



```
#!/usr/bin/python

for x in range (1,10):
    print (x)
```

If you guessed that it would count from 1 to 9 then well done!



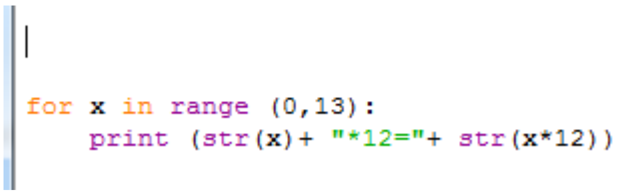
```
>>>
1
2
3
4
5
6
7
8
9
>>>
```

Have you ever wanted to do your timetables homework within seconds?

Well now that you know for loops it's very easy for you to figure it out when you forget to do it ☺.

Here is an example where Python does the 12 times table with just 2 lines!!!!

Don't worry about the "str" word we will cover that later on, feel free to ask a mentor about it

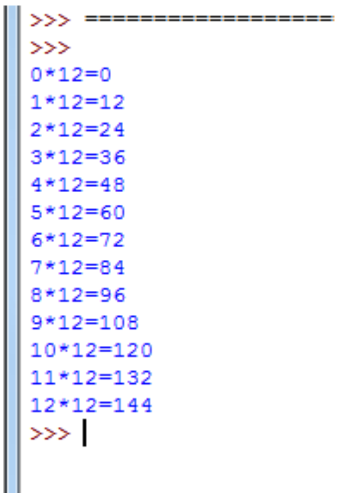


```
for x in range (0,13):
    print (str(x)+ "*12="+ str(x*12))
```

Code:

for x in range (0,13):

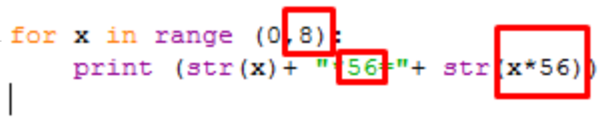
  print (str(x)+ "*12="+ str(x*12))



```
>>> ==================
>>>
0*12=0
1*12=12
2*12=24
3*12=36
4*12=48
5*12=60
6*12=72
7*12=84
8*12=96
9*12=108
10*12=120
11*12=132
12*12=144
>>>
```

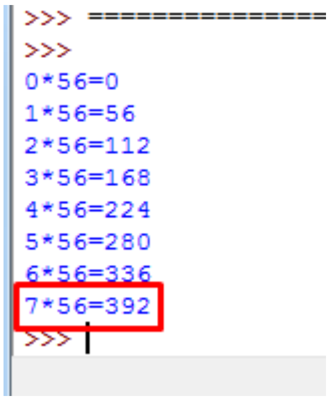
We can give python bigger numbers to calculate too! You could do the 56 times table… All within seconds. Use python to work out what 7*56= is? But using only for loops ☺

Answer is on the next page →



```
for x in range (0,8):
    print (str(x)+ "*56="+ str(x*56))
```

I've highlighted the changes I made. The first red box makes the loop stop at 7, the second box just prints out 56= and the last box works out the maths. X is a variable that increments by one (adds 1 every time it loops).



```
>>> ================
>>>
0*56=0
1*56=56
2*56=112
3*56=168
4*56=224
5*56=280
6*56=336
7*56=392
>>>
```

So 7*56=392

Enjoy doing your maths homework within minutes ☺

Code:

```
for x in range (0,8):
    print (str(x)+ "*56="+ str(x*56))
```

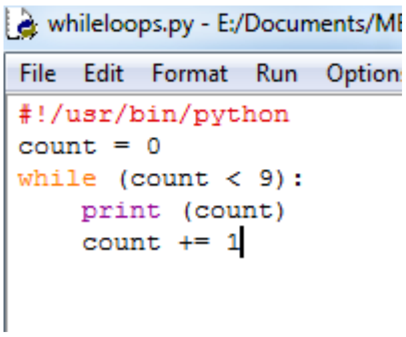
Now onto while loops ☺

While loops:

Unlike for loops, while loops behave in a different way. They are not restricted by two numbers and can end up getting into things called infinite loops which I'll show you further on ;P.

So let's make a python file called whileloops.py

We are going to make a simple while loop the counts from 0 to 8



```
#!/usr/bin/python
count = 0
while (count < 9):
    print (count)
    count += 1
```

Code:

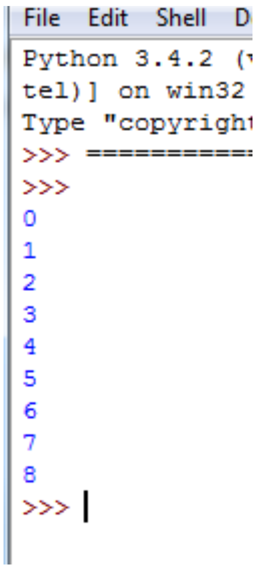
#!/usr/bin/python

count = 0

while (count < 9):

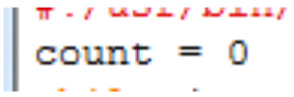
    print (count)

    count += 1



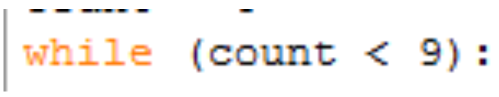
```
Python 3.4.2 (
tel)] on win32
Type "copyrigh
>>> ==========
>>>
0
1
2
3
4
5
6
7
8
>>>
```

So let’s break this code down line by line so you understand what’s going on.



```
count = 0
```

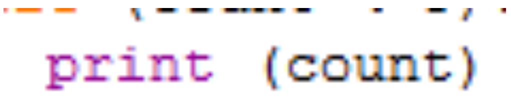
In case you forgot, this is just a variable containing the value 0



```
while (count < 9):
```

Here we create our while loop and give it a rule. Which is:

“If the variable count is is less than 9 then keep looping”



```
  print (count)
```

This line simply displays what value is inside count

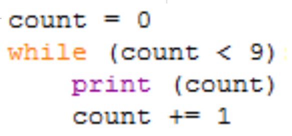
```
count += 1
```

This important line stops us from getting into an infinite loop. It basically adds 1 to the value inside count.

So 0+1 = 1, 1+1=2, 2+1=3 etc…..

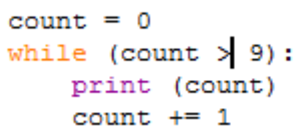
However we will look at infinite loops in a moment.

Let’s play around with some while conditions and see what Python does.



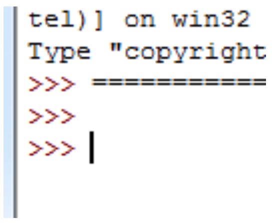
```
count = 0
while (count < 9)
    print (count)
    count += 1
```

We’ve seen what < does but what about > (more than)



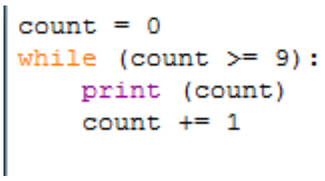
```
count = 0
while (count > 9):
    print (count)
    count += 1
```

In this scenario > will do nothing because the condition is met (the dog caught his tail)

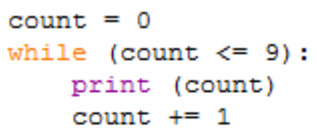


```
tel)] on win32
Type "copyright
>>> ===========
>>>
>>> 
```

Next we have >= which mean greater than or equal to. The opposite of this being less than or equal too <=



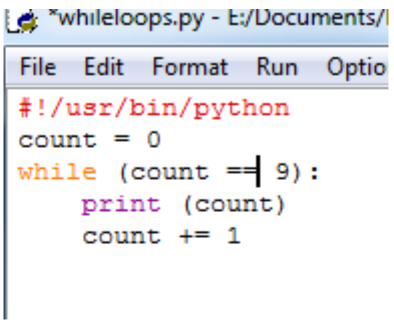
```
count = 0
while (count >= 9):
    print (count)
    count += 1
```



```
count = 0
while (count <= 9):
    print (count)
    count += 1
```

Test out these two and see what they do.

Another condition is == which means equals we used this in the previous tutorial for if statements



*whileloops.py - E:/Documents/

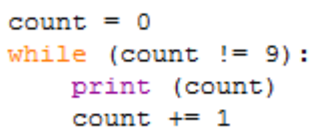
File Edit Format Run Optio

```
#!/usr/bin/python
count = 0
while (count == 9):
    print (count)
    count += 1
```

Again the code will do nothing because count does not contain 9 since its set to 0.

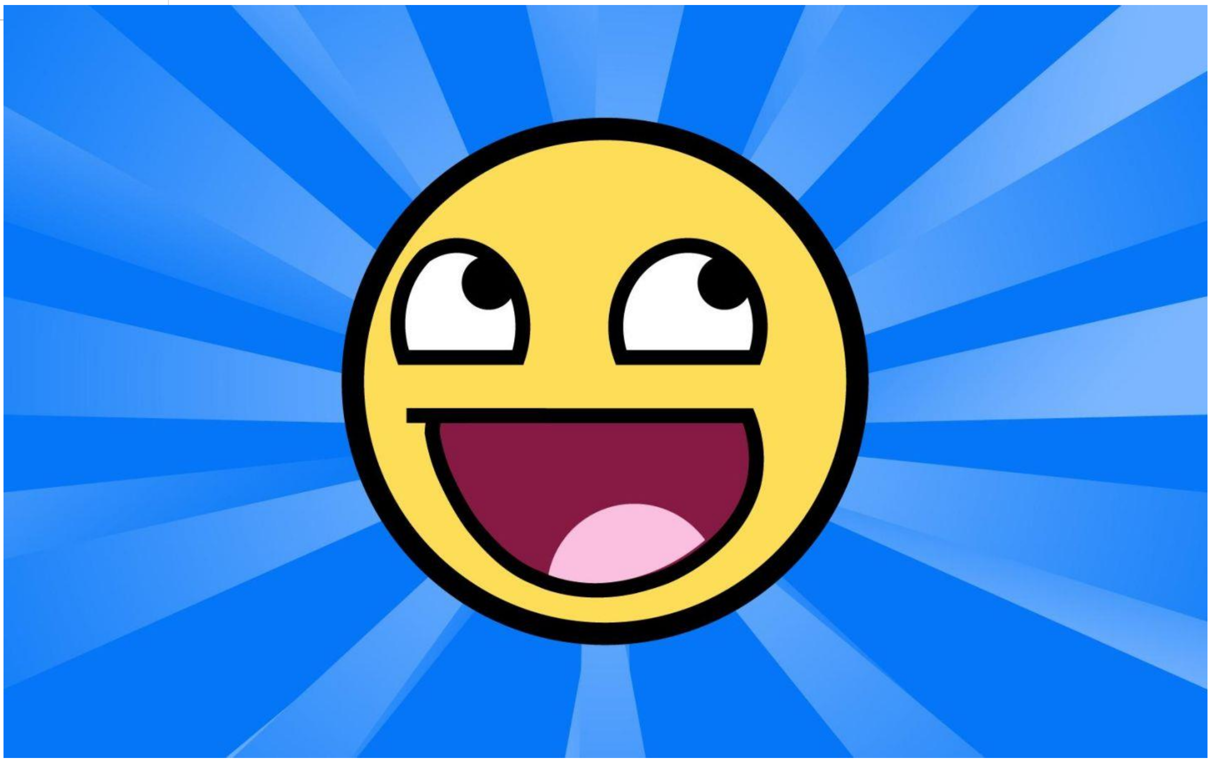
!= means not equal too so

“if count is not equal to 9 then run the loop”



```
count = 0
while (count != 9):
    print (count)
    count += 1
```

Let’s make an infinite loop >:)



READ ME: Before we start remember this keyboard combo to get you out of the loop

Ctrl + Shift + C

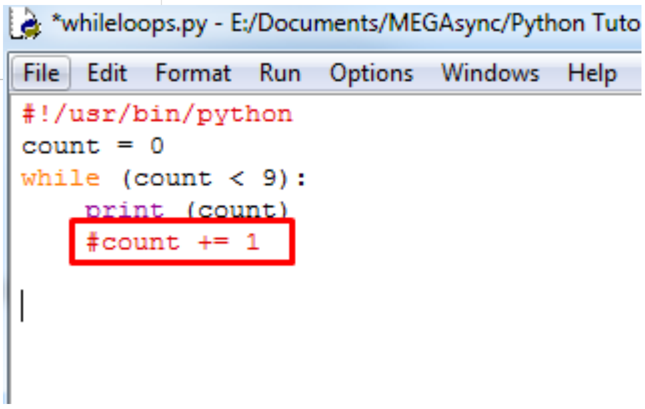
We are just going to comment out some code with a special button on your keyboard

#

This symbol will allow you to add comments into your code to help you understand your code when you forget what a line does.

Anything with a #in front will be ignored by python.

The code will turn red to indicate the affected lines.

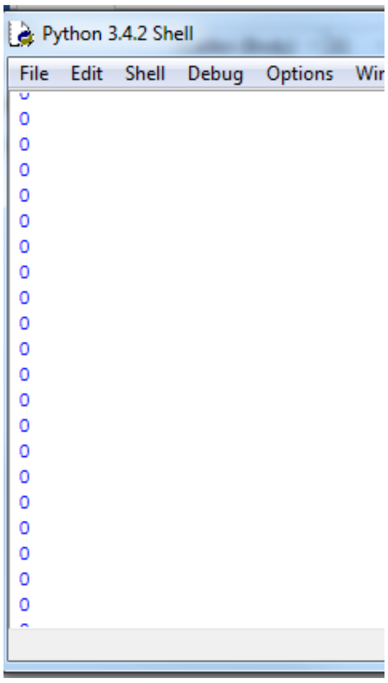

*whileloops.py - E:/Documents/MEGAsync/Python Tuto
File Edit Format Run Options Windows Help

```
#!/usr/bin/python
count = 0
while (count < 9):
        print (count)
        #count += 1
```

Your code is ready for the infinite loop now >:)

Code:

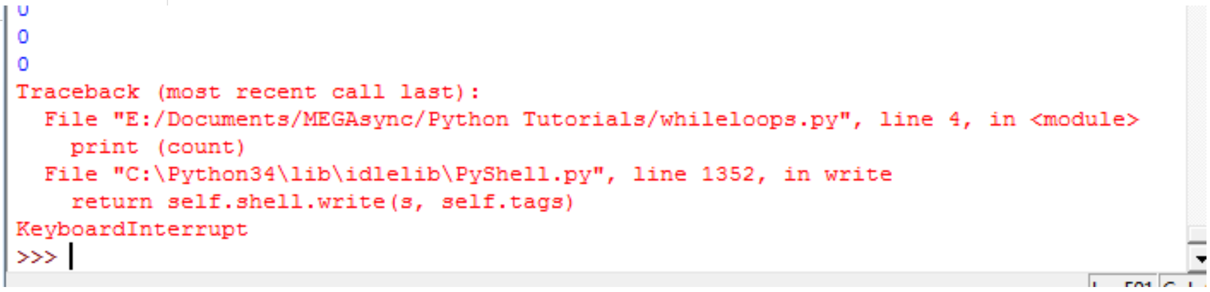
```
#!/usr/bin/python

count = 0

while (count < 9):

    print (count)

    #count += 1
```



```
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
```

Python will keep making zeros forever and will never stop unless you press the keyboard combo I told you about or if you shut down your computer….

But I'm sure pressing Ctrl + Shift + C is a better alternative ☺

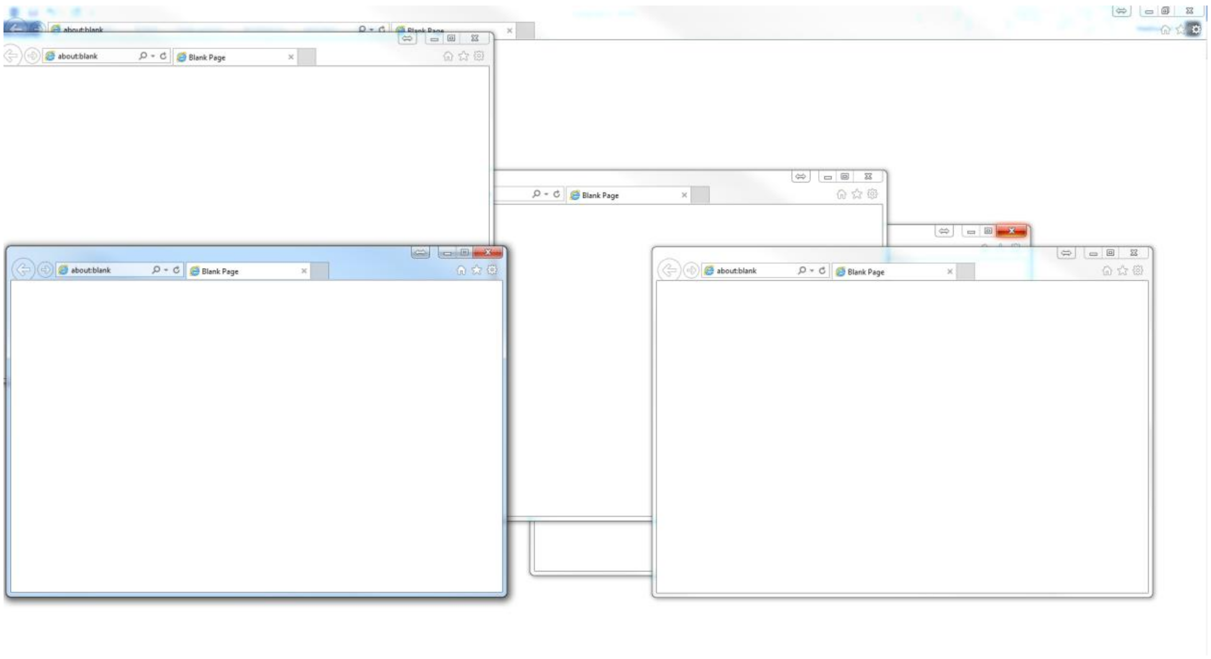


```
0
0
0
Traceback (most recent call last):
  File "E:/Documents/MEGAsync/Python Tutorials/whileloops.py", line 4, in <module>
    print (count)
  File "C:\Python34\lib\idlelib\PyShell.py", line 1352, in write
    return self.shell.write(s, self.tags)
KeyboardInterrupt
>>> |
```

Python will just break out of the loop and show the following message when you end the loop

Depending on how the code is written infinite loops can be used in a good ways or really bad ways.

For example imagine a python script that just opened Internet Explorer over and over…. forever… Well until the Computer crashes due to lack of memory.



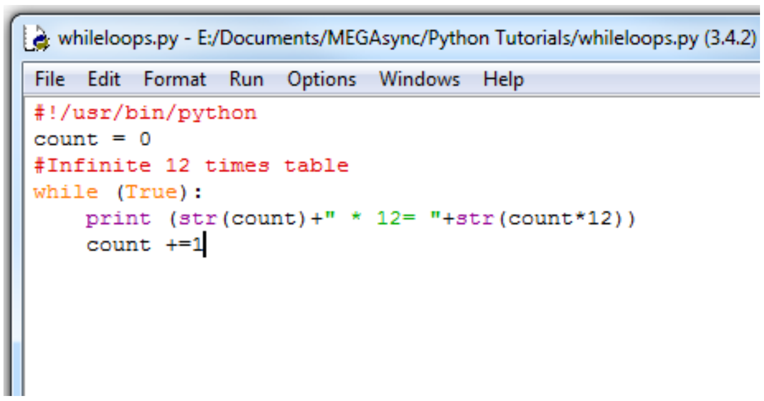
You get the idea right? The PC will crash and shut down.

A good use for an infinite loop is when you activate your webcam. The loop will allow the camera to keep catching frames until the loop is forced to stop.

Let's make an infinite times table loop :D

So this example will run the 12 times table until you tell it to stop with the keyboard interrupt. Feel free to make a new python file

ENJOY!!



whileloops.py - E:/Documents/MEGAsync/Python Tutorials/whileloops.py (3.4.2)

```
#!/usr/bin/python
count = 0
#Infinite 12 times table
while (True):
    print (str(count)+" * 12= "+str(count*12))
    count +=1
```

Code:

#!/usr/bin/python

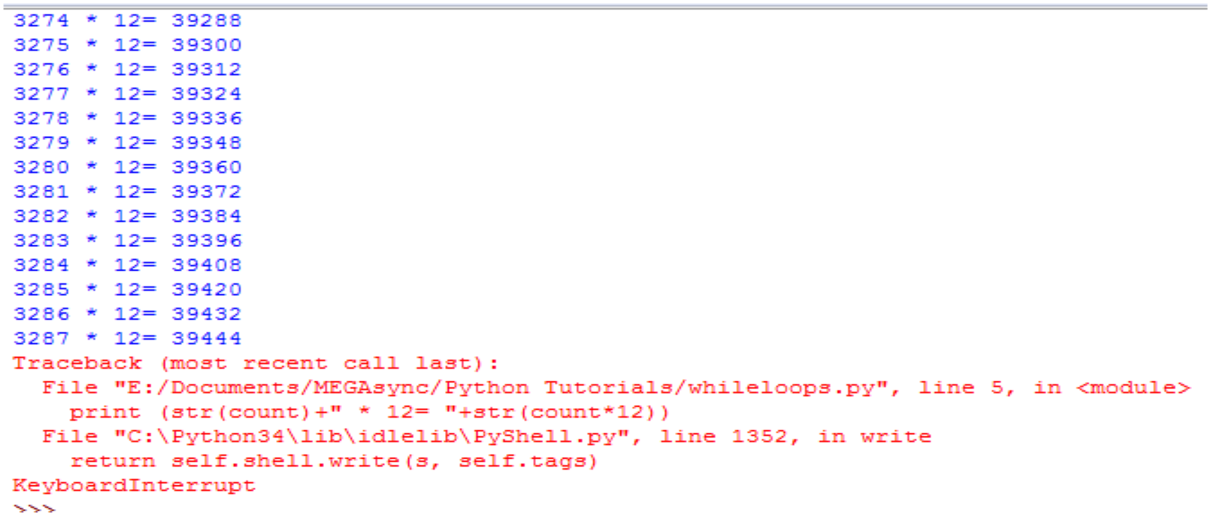
count = 0

#Infinite 12 times table

while (True):

  print (str(count)+" * 12= "+str(count*12))

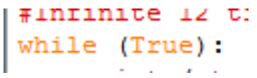
  count +=1



```
3274 * 12= 39288
3275 * 12= 39300
3276 * 12= 39312
3277 * 12= 39324
3278 * 12= 39336
3279 * 12= 39348
3280 * 12= 39360
3281 * 12= 39372
3282 * 12= 39384
3283 * 12= 39396
3284 * 12= 39408
3285 * 12= 39420
3286 * 12= 39432
3287 * 12= 39444
Traceback (most recent call last):
  File "E:/Documents/MEGAsync/Python Tutorials/whileloops.py", line 5, in <module>
    print (str(count)+" * 12= "+str(count*12))
  File "C:\Python34\lib\idlelib\PyShell.py", line 1352, in write
    return self.shell.write(s, self.tags)
KeyboardInterrupt
>>>
```

Feel free to stop it whenever you want. However if you leave it for too long python may crash due to the numbers being too big!!! If Python reaches 2^32= 4294967296 then it should crash unless you are in 64 bit then it will crash when you reach 2^64 = 18446744073709551616!!!

Feel free to ask a mentor about this line:



```
#Infinite 12 t:
while (True):
```

Functions

These tutorials may be difficult to grasp so ask a mentor if you are having issues!

Functions or methods are a great way to simplify your code. With some creativity we can make lots of print statements without it being tedious. Method can organise your code so that you can call specific lines lots of times.

For example:

```
def say(phrase):
    print (phrase)

say("Hi")
```

Is exactly the same as:

```
print("Hi")
```

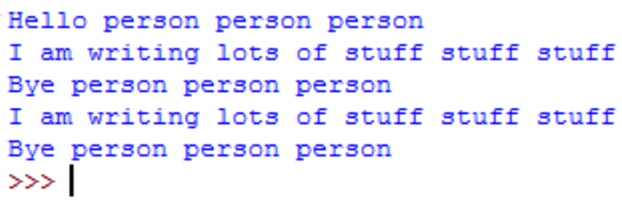
Try it out here's the code:

```
def say(phrase):
   print (phrase)
say("Hi")
```

We can make functions more useful like this. So instead of writing "I am writing lots of stuff stuff stuff" over and over. We can just write stuff() lots of times.

```
def greet():
    print("Hello person person person")

def stuff():
    print("I am writing lots of stuff stuff stuff")

def bye():
    print ("Bye person person person")

greet()

stuff()

bye()

stuff()

bye()
```

Output:



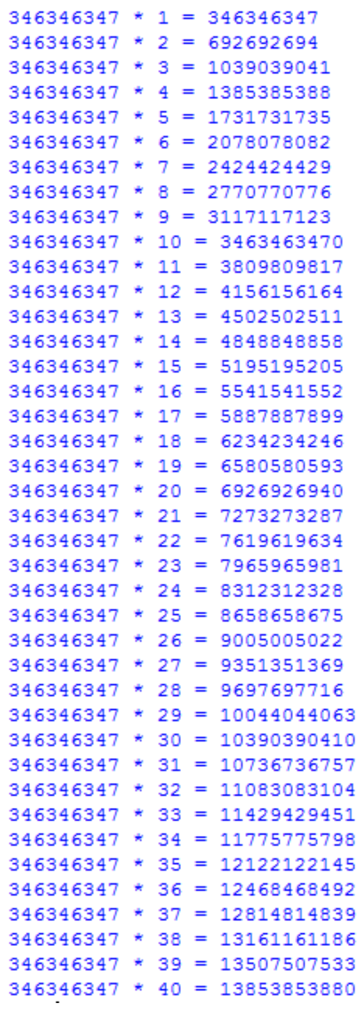
```
Hello person person person
I am writing lots of stuff stuff stuff
Bye person person person
I am writing lots of stuff stuff stuff
Bye person person person
>>> 
```

Code:

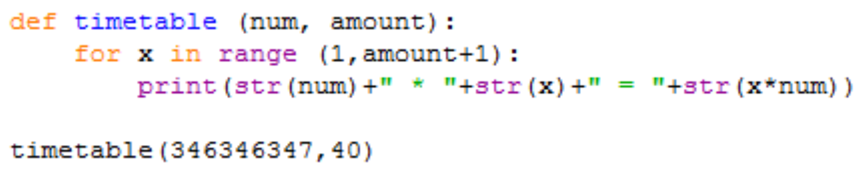
```
def greet():

   print("Hello person person person")


def stuff():

   print("I am writing lots of stuff stuff stuff")


def bye():

   print ("Bye person person person")


greet()


stuff()


bye()


stuff()


bye()
```

Timetable Function

With functions we can write a timetable program without having to write lots of lines. So here's the 346346347 times table up to 40 or more!! With only 4 lines of code!!!!!



```
346346347 * 1 = 346346347
346346347 * 2 = 692692694
346346347 * 3 = 1039039041
346346347 * 4 = 1385385388
346346347 * 5 = 1731731735
346346347 * 6 = 2078078082
346346347 * 7 = 2424424429
346346347 * 8 = 2770770776
346346347 * 9 = 3117117123
346346347 * 10 = 3463463470
346346347 * 11 = 3809809817
346346347 * 12 = 4156156164
346346347 * 13 = 4502502511
346346347 * 14 = 4848848858
346346347 * 15 = 5195195205
346346347 * 16 = 5541541552
346346347 * 17 = 5887887899
346346347 * 18 = 6234234246
346346347 * 19 = 6580580593
346346347 * 20 = 6926926940
346346347 * 21 = 7273273287
346346347 * 22 = 7619619634
346346347 * 23 = 7965965981
346346347 * 24 = 8312312328
346346347 * 25 = 8658658675
346346347 * 26 = 9005005022
346346347 * 27 = 9351351369
346346347 * 28 = 9697697716
346346347 * 29 = 10044044063
346346347 * 30 = 10390390410
346346347 * 31 = 10736736757
346346347 * 32 = 11083083104
346346347 * 33 = 11429429451
346346347 * 34 = 11775775798
346346347 * 35 = 12122122145
346346347 * 36 = 12468468492
346346347 * 37 = 12814814839
346346347 * 38 = 13161161186
346346347 * 39 = 13507507533
346346347 * 40 = 13853853880
```



```
def timetable (num, amount):
    for x in range (1,amount+1):
        print(str(num)+" * "+str(x)+" = "+str(x*num))

timetable(346346347,40)
```

Code:

```
def timetable (num, amount):
    for x in range (1,amount+1):
        print(str(num)+" * "+str(x)+" = "+str(x*num))


timetable(346346347,40)
```

Challenge

1. Write a program that will print out “Hello, Matthew” and “Hello, Carlos” using methods.

2. Write a program that uses a method to add (concatenate) “Codedojo Rocks!!!” to any sentence sent to the method.

If you are stuck ask a mentor for help!

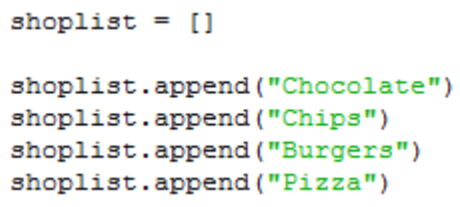
Lists, Tuples and Dictionary’s

Let’s jump straight in and make a shopping list!

In python we can easily do this with a list.

First let’s create our shopping list:

Feel free to add more items into the list!



```
shoplist = []

shoplist.append("Chocolate")
shoplist.append("Chips")
shoplist.append("Burgers")
shoplist.append("Pizza")
```

Code:

shoplist = []

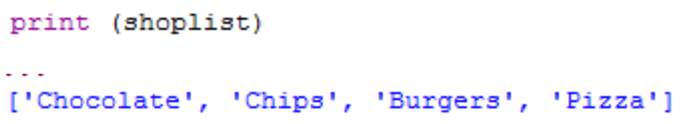
shoplist.append("Chocolate")

shoplist.append("Chips")

shoplist.append("Burgers")

shoplist.append("Pizza")

To view what is inside the list we can simply print our list:



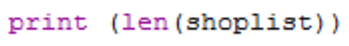
```
print (shoplist)

...
['Chocolate', 'Chips', 'Burgers', 'Pizza']
```

Code:

print (shoplist)

```
4
```

We can also count the number of items in our list (len means length):

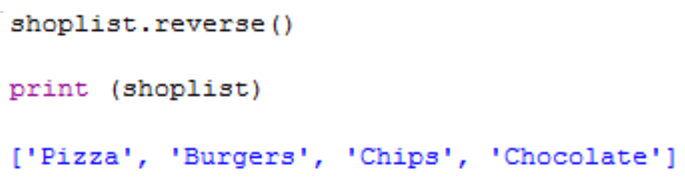


```
print (len(shoplist))
```

Code:

print (len(shoplist))

Maybe we want to reverse the order of our list that’s easy too!



```
shoplist.reverse()

print (shoplist)

['Pizza', 'Burgers', 'Chips', 'Chocolate']
```

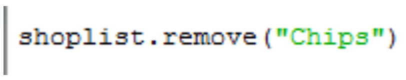
Code:

shoplist.reverse()

print (shoplist)

We can remove specific items too!

Let’s remove Chips from our list



```
shoplist.remove("Chips")
```

Code:

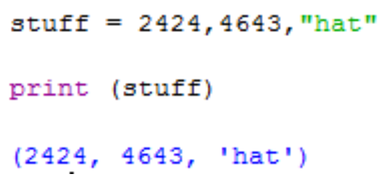
shoplist.remove("Chips")

Print out the list again and see if chips has been removed.

Tuples

Tuples are similar to lists however once they are created they cannot be changed.

Let’s make a tuple:



```
stuff = 2424,4643,"hat"

print (stuff)

(2424, 4643, 'hat')
```

Code:

```
stuff = 2424,4643,"hat"


print (stuff)
```

Dictionary

Dictionaries are another form of a list but they use something called keys. To retrieve data from the Dictionary we use the associated key. Think of the key as a word and the data as the definition.

So let's create a dictionary with people's names and their ID numbers.

The names will be the keys and the numbers are the information for the keys.

So here is our dictionary

```
ID= {'Matt': 4567, 'Carlos': 4364, 'Nathan': 3564}
```

We can easily print out the contents like this

```
print(ID)
```

```
{'Matt': 4567, 'Nathan': 3564, 'Carlos': 4364}
```

We can add more to our dictionary too! Here we are adding Sam to the dictionary.

```
ID['Sam']=4365
```

```
{'Carlos': 4364, 'Sam': 4365, 'Matt': 4567, 'Nathan': 3564}
```

We can print out individual data such as Carlos

```
print(ID['Carlos'])
```

```
4364
```

We can delete individuals such as Nathan because he smells.

```
del ID['Nathan']
```

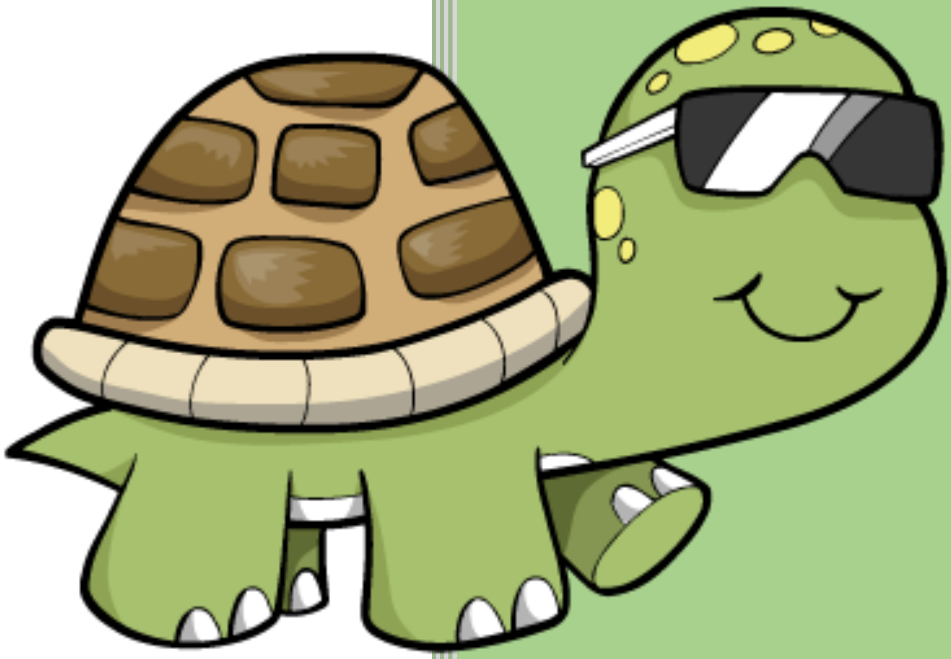
```
{'Carlos': 4364, 'Sam': 4365, 'Matt': 4567}
```

Lastly we can print out what keys are in use without the data attached

```
print(ID.keys())
```

```
dict_keys(['Matt', 'Sam', 'Carlos'])
```

Turtle with Python



Matthew Hart

Coder-Dojo Whitechapel

9/24/2015

Contents

Welcome to Turtle ☺

Moving forward

Let's get started by creating an empty file. **DO NOT SAVE YOUR FILE AS turtle**

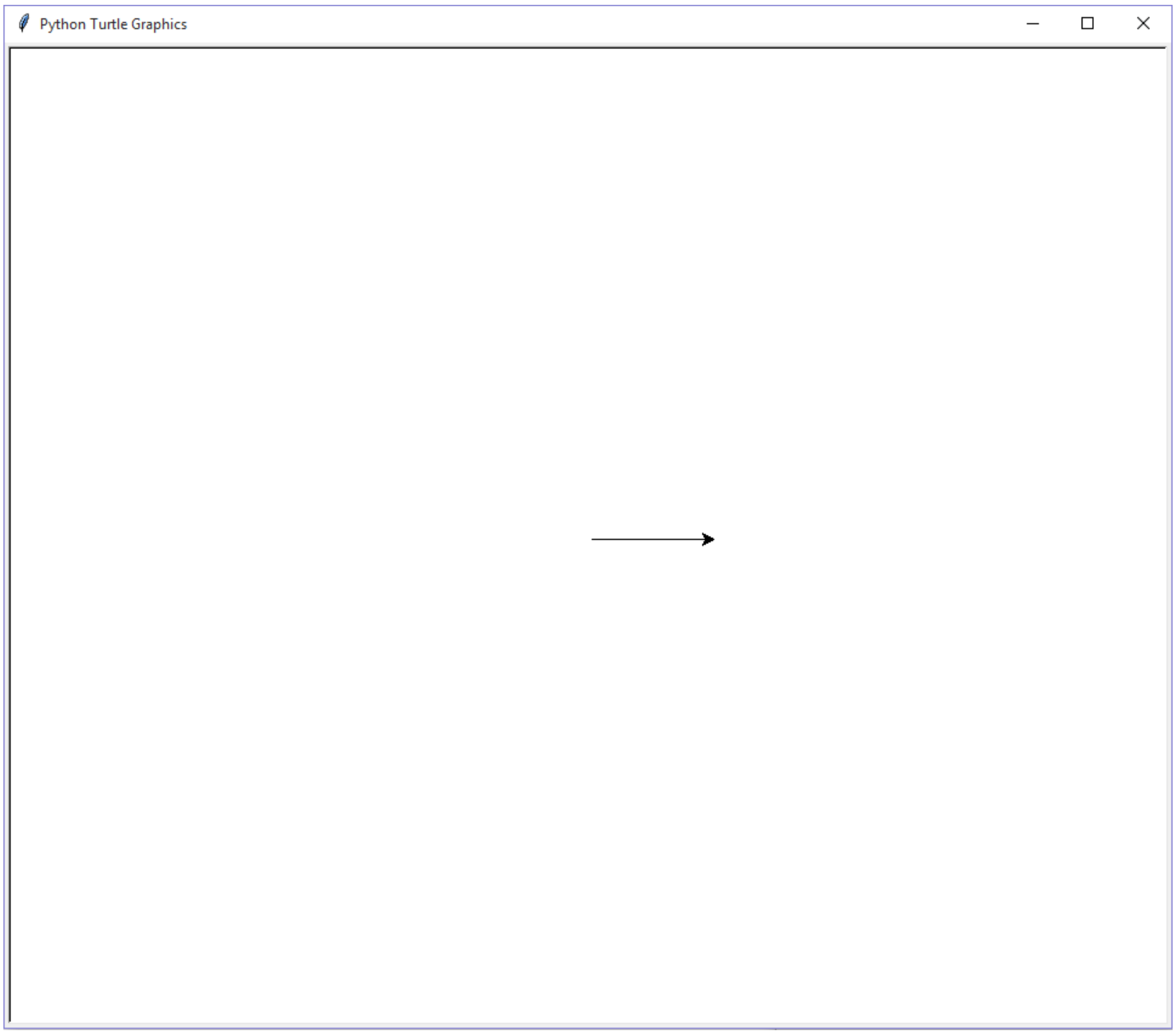
Add the following:

```
from turtle import*

fd(100)
```

fd means forward

bk means backward

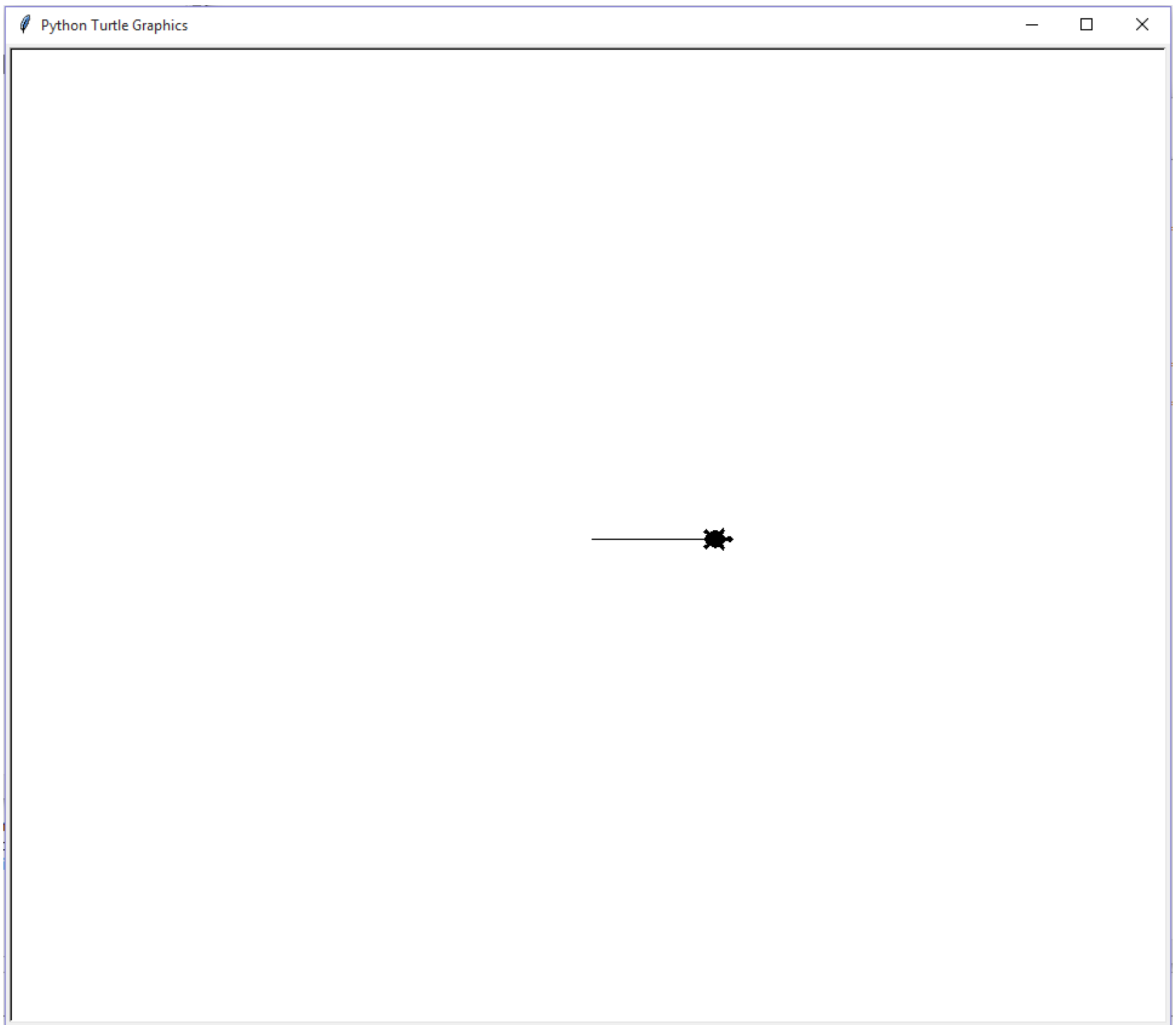
A popup will display with your turtle moving forward:



Hang on a second! That's not a turtle…. Let's fix that ☺

Changing shape

```
from turtle import*
shape("turtle")
fd(100)
```



That’s better!

If you don’t like this shape fear not, there are others!

“arrow”, “turtle”, “circle”, “square”, “triangle”, “classic”.

Feel free to use any of these ☺

Turning left and right

With our current code let's make a square

So let's make the first corner!

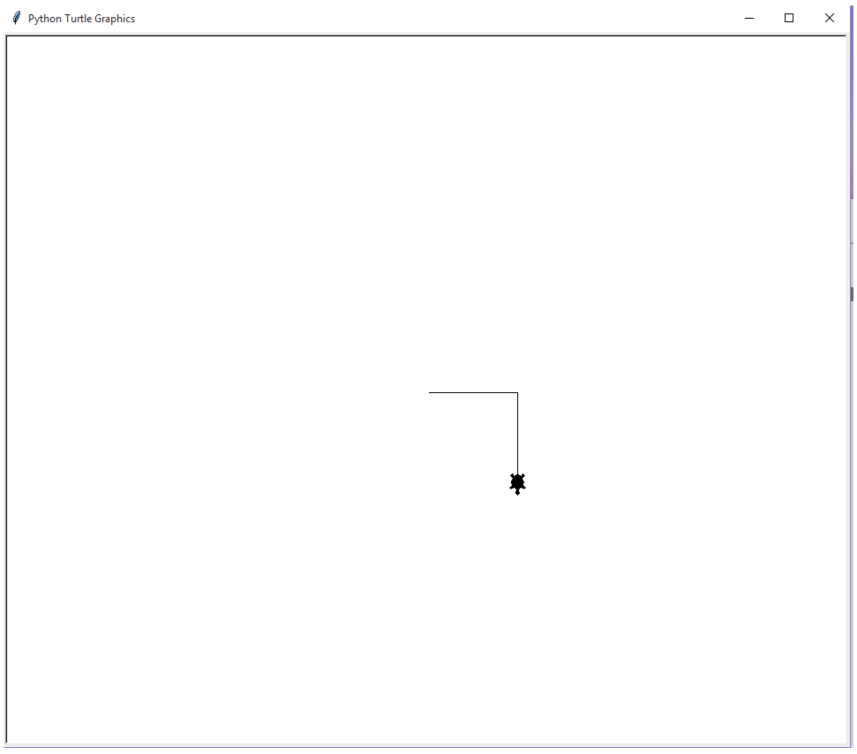
rt means right

lt means left

The number next in the brackets tells the turtle how many degrees to turn

```
from turtle import*
shape("turtle")
fd(100)
rt(90)
fd(100)
```

You should get the following



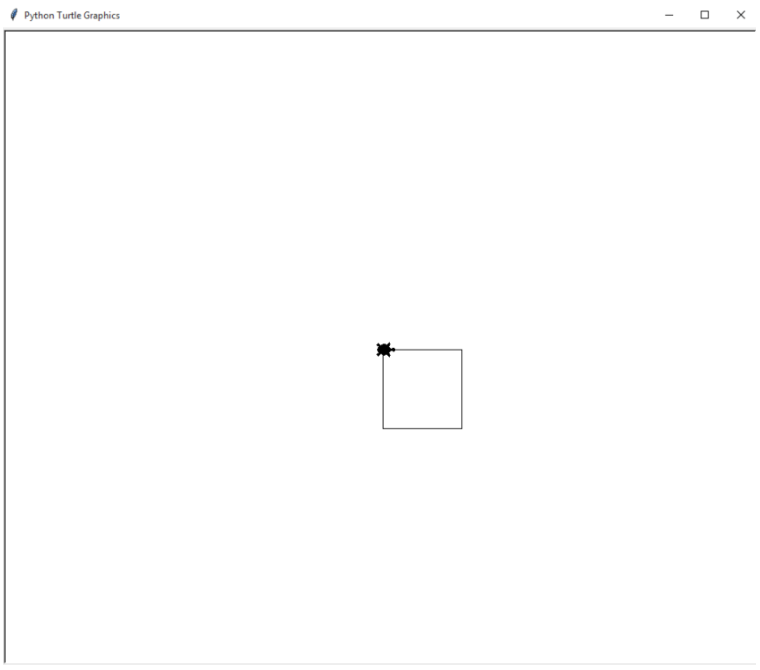
Now we could be **lazy** by copying and pasting the same thing another three times to make a square like this:

```
from turtle import*
shape("turtle")
fd(100)
rt(90)
fd(100)
rt(90)
fd(100)
rt(90)
fd(100)
```

OR!!!!!

We could try using a for loop to accomplish this ☺ and be even **more lazy** :D

```
from turtle import*
shape("turtle")
for x in range(0, 4):
    fd(100)
    rt(90)
```



Look at how smaller the code is now. For loops are the way forward :D

How to make any shape you want!

So you are probably wondering how can we make other shapes?

Well if you know your maths then it should be easy ☺

So let's start off with an equilateral triangle:

How many degrees are in a triangle?

Correct 180 ☺

How many sides are on a triangle?

There are 3 sides so let's put that into our for loop

```
for x in range(0, 3):
    fd(100)
    rt()
```

So this is our code so far. Next we need to add the value rt.

Feel free to change fd to any value as this will affect the size of the shape.

So to get the value of rt we need to figure out the angle of each corner:

Degrees of shape divided by the number of sides

180 divided by 3 = 60

```
for x in range(0, 3):
    fd(100)
    rt(60)
```

Try out the code

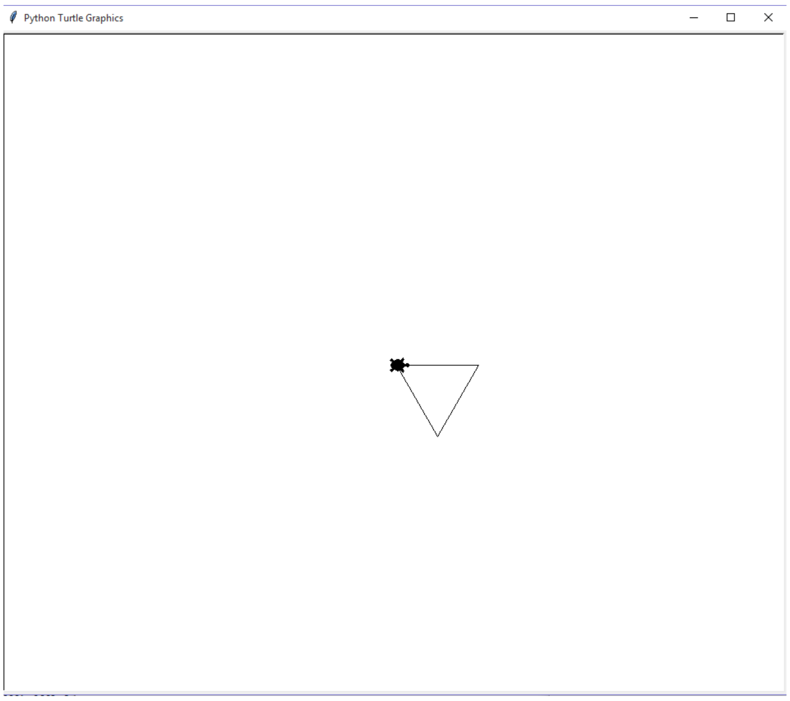
Did it work? No? Let's see why.

The problem is that you are not turning enough. Try this

Degrees of shape takeaway the value we got previously

180-60 = 120

```
for x in range(0, 3):
    fd(100)
    rt(120)
```



Success!

Feel free to ask a mentor to explain why its 120 and not 60

Do you think you could make a circle? Or a hexagon or an octagon?

A clue for a circle is to give it 360 sides ;)

Go crazy and enter random values

Here are some others that me and other mentors have made (Carlos)

I forgot to mention!

To change the speed of the turtle

Type

speed() and put a number from 1-11 inside.

11 is the turtles max speed

```
from turtle import*


shape("turtle")
speed(11)
count = 0
while count < 1000:
    forward(2)
    right(0+count)
    backward(2)
    forward(1)
    left(0+count)
    count+=1
    if count == 50 or count == 100 or count ==
150 or count == 1000:
        left(90)


done()
```

```
from turtle import*


shape("turtle")
speed(11)
count = 0
while count < 1000:
    forward(50-count)
    right(0+count)
    backward(25-count)
    count+=1
```

Object Oriented Programming (OOP)

Matthew Hart

Coder-Dojo Whitechapel

11/27/2015

Contents

Welcome to OOP ☺

What is it?

OOP is a way of making an object with code. Such as a dog or maybe us mentors. At first this will be very confusing but don't worry we are here to help if you get stuck!!

Right so let's get started:

Creating a dog class

Let's write the first line which is naming our object. In this example we will be calling it Dog

```
Class Dog:
```

Press Enter as the next lines need to be indented just like a method

Adding a constructor

So what is a constructor? Think of it as the builder of our dog. The constructor will create our dog for us.

Add the following code (This should be indented)

```
    def __init__(self, name, age, bark):
        self.name = name
        self.age = age
        self.bark = bark
```

Okay lets break this down:

- Def __init__ is the constructor which takes the following arguments self, name, age and bark
- Self refers to the actual object (in this case dog). Lets say we create a dog called Freddie self is Freddie.
- Name, age and bark are going to be the YOU Guessed it. The name, age and bark style of our dog.
- The last 3 lines are just assigning what's in the brackets to our Dog we create.

It will start to make more sense as we continue. If you still seem confused ask a mentor! ☺

Create a speak method for our dog

If you remember our method defining from previous tutorials this should be easy to understand.

```
def speak(self):
        print (self.name +" goes "+ self.bark)
```

We call our Dog called Freddie aka the self part and get Freddie's name and his bark sound.

Again just keep going if you seem confused

Create Freddie

So let's create Freddie

```
Dog1 = Dog("Freddie","1 years old", "woof")
```

Here we are creating a variable called Dog1 with our Dog object inside. The name of this dog object is Freddie. He is 1 years old and goes woof.

If we try to run the code he won't speak so let's get him to speak ☺

Command Freddie to speak

```
Dog1.speak()
```

Now try to run the code! Freddie should speak

Add more dogs

```
Dog2 = Dog("Rufus","4 years old", "ruff")
Dog2.speak()


Dog3 = Dog("Bella","3 years old", "blerrgghhh")
Dog3.speak()
```

Feel free to add your own methods or create other classes! Maybe a mentor class :D. Where can create us and get us to say things.

Final code listing

You can view all the code together on trinket or down below!

https://trinket.io/python/576b44dd76

```
class Dog:
    def __init__(self, name, age, bark):
        self.name = name
        self.age = age
        self.bark = bark

    def speak(self):
        print (self.name +" goes "+ self.bark)





Dog1 = Dog("Freddie","1 years old", "woof")
Dog1.speak()

Dog2 = Dog("Rufus","4 years old", "ruff")
Dog2.speak()

Dog3 = Dog("Bella","3 years old", "blerrgghhh")
Dog3.speak()
```