Python

Methodologies for Data Science

Lars Sorensen - 16:137:603
Fall 2015 - biglars@cs.rutgers.edu
Unit Two

Python Methodologies for Data Sciences

Take a breath. You've done a lot in the last two weeks…

Loading Python on your machine - Figuring out how to run IDLE - Using the Python interactive command window - Creating a programs -

Print function - Variables - Assignment - Arithmetic - Modulus - Strings - Integers - Floats - Casting - Input function

Booleans - Arithmetic conditionals (>, <, ==, !=, <=, >=) - Boolean operators (And, Or, Not) - if - elif - else

That's a lot of stuff….

Python Methodologies for Data Sciences

This is why I bring it up though. Learning how to program is like building a pyramid. What you have learned the last two weeks will be used by you from now until you stop programming. They are the basics and you need to practice them again, and again and again.

We are going to move on and talk about sequences now. They are just the way that Python organizes lists of things. There are just three of them: strings, lists, and files (and we're not worrying about files until Unit 3). After that we are going to discuss one of the most important concepts in programming, iteration.

Even though we are moving on do not think that what you've done the last few weeks can be put aside. You need to code. You need to practice. You'll see what I mean as we continue down the Python road. Right now we're going to talk about strings. Then we'll look at Lists and after a break, we'll look at iteration. This is when things get fun and the usefulness of your code is going to skyrocket, trust me. Computers are dumb, they only do what we tell them to, but man they do it fast and we're going to see just how fast when we do iteration and loops.

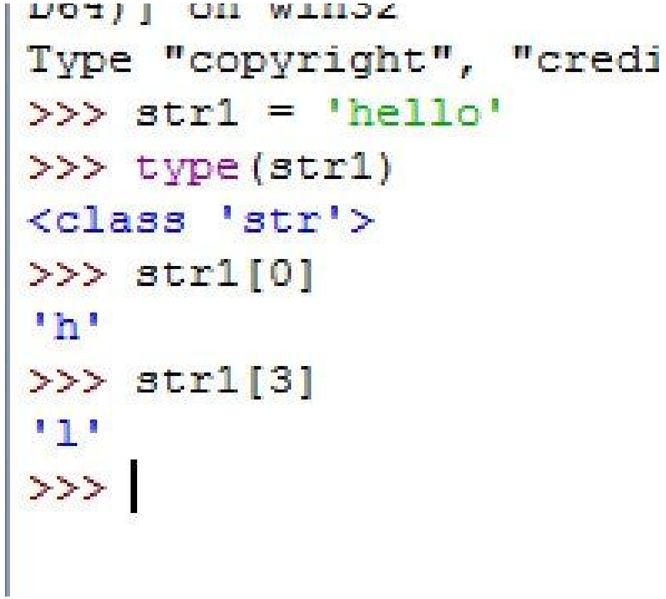
Strings

Python Methodologies for Data Sciences

A string is just a “sequence” of characters. We will define the variable str1 and assign it the string ‘hello’

Now, look at the third command in the window to the right, str1[0]. This “index” is how we can refer to individual characters in a string. Here, the zero refers to the first character in the string.

In Computer Science we start counting at 0. You are a programmer now, so get used to it.

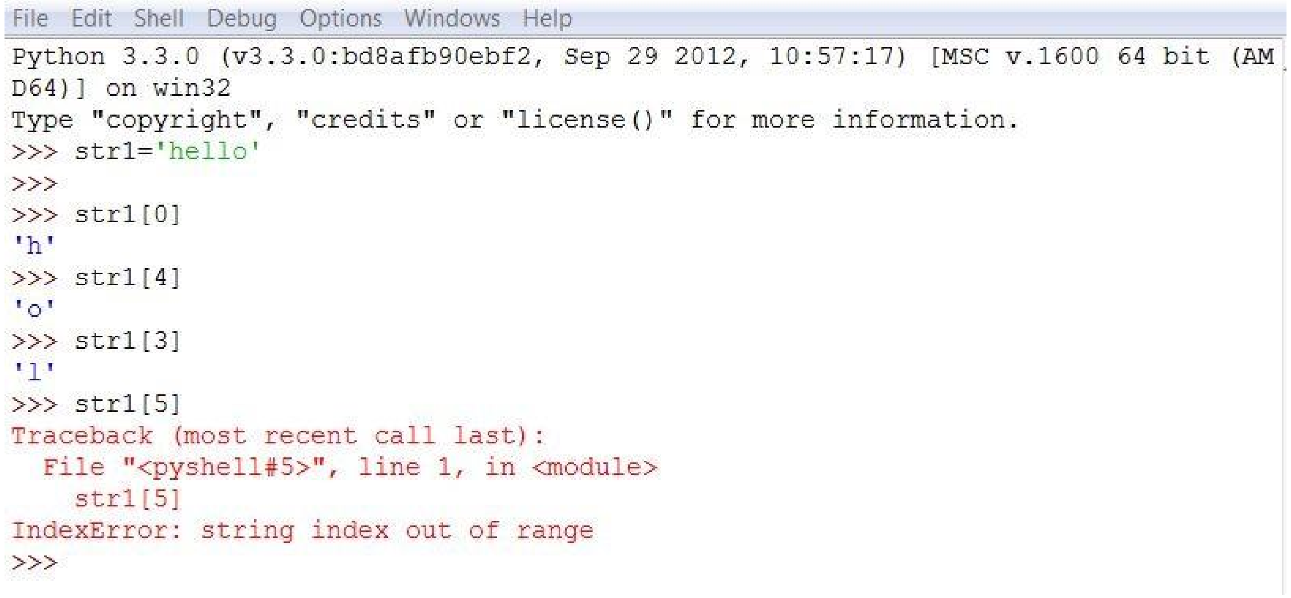


```
D64)] on win32
Type "copyright", "credi
>>> str1 = 'hello'
>>> type(str1)
<class 'str'>
>>> str1[0]
'h'
>>> str1[3]
'l'
>>> 
```

Python Methodologies for Data Sciences

Starting at zero, you can access all of the characters in a string by using the index, but don't go too far! As you can see, if you try to access a part of the string that does not exist you will get an error.

This used to be called an "out of bounds" error, but Python calls it out of range now. We'll find out why soon.

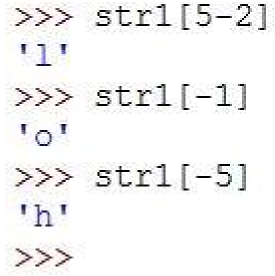


```
Python 3.3.0 (v3.3.0:bd8afb90ebf2, Sep 29 2012, 10:57:17) [MSC v.1600 64 bit (AM
D64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> str1='hello'
>>>
>>> str1[0]
'h'
>>> str1[4]
'o'
>>> str1[3]
'l'
>>> str1[5]
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    str1[5]
IndexError: string index out of range
>>>
```

Python Methodologies for Data Sciences

We can use arithmetic in an index if we like. We can also do ‘negative’ indexing (which starts at the end of the string and goes backward; -1 is the last element here)

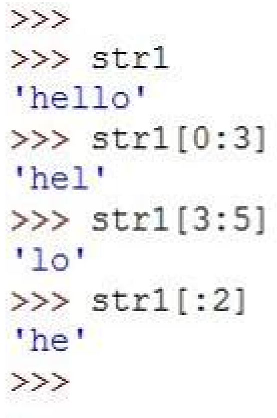
Try some on your own....



```
>>> str1[5-2]
'l'
>>> str1[-1]
'o'
>>> str1[-5]
'h'
>>>
```

Python Methodologies for Data Sciences

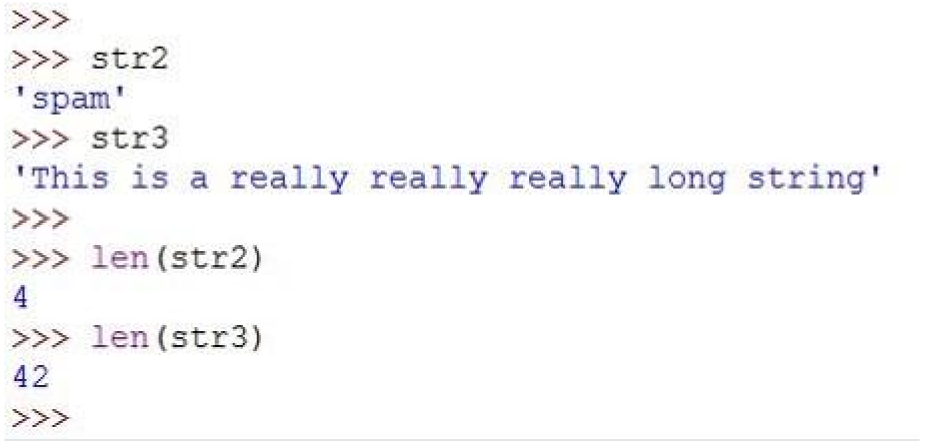
We can also do ‘Slicing’



```
>>>
>>> str1
'hello'
>>> str1[0:3]
'hel'
>>> str1[3:5]
'lo'
>>> str1[:2]
'he'
>>>
```

Python Methodologies for Data Sciences

We can also find out the length of a string using 'len()'



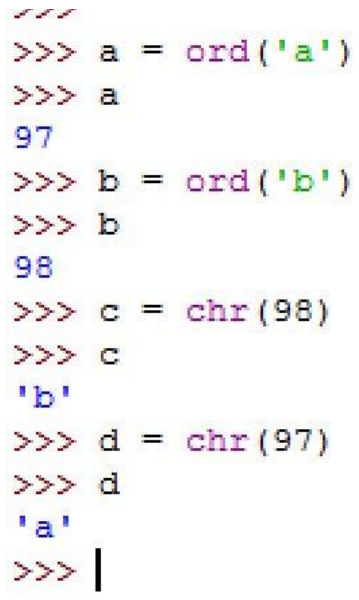
```
>>>
>>> str2
'spam'
>>> str3
'This is a really really really long string'
>>>
>>> len(str2)
4
>>> len(str3)
42
>>>
```

Python Methodologies for Data Sciences

See Z 132.

We can take characters and get their Unicode values, and take a value and get its Unicode character.

Use chr to look at Unicode 960 like they suggest in Z.



```
>>> a = ord('a')
>>> a
97
>>> b = ord('b')
>>> b
98
>>> c = chr(98)
>>> c
'b'
>>> d = chr(97)
>>> d
'a'
>>> |
```

Python Methodologies for Data Sciences

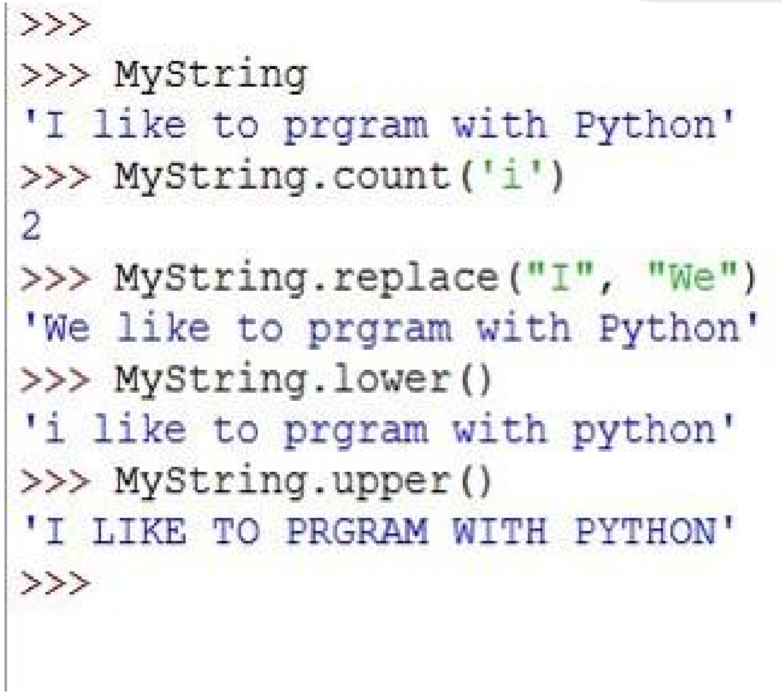
Now, here is a little bit of what we call JIT (Just in time). The 411 on these commands is as follows.

Strings are what programmers call objects.

Objects have certain functions that work for just them and their data. These functions are called Methods.

count, replace, lower and upper are methods that work with the string object.

To use them you type the variable name, a period, the name of the method and a parameter list (or two empty parens () if there are no parameters)



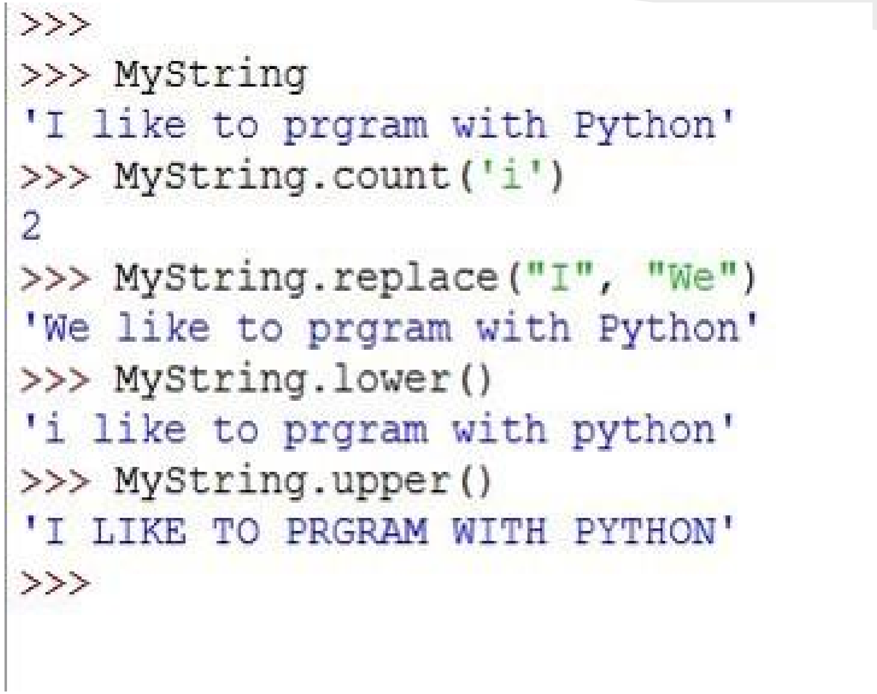
```
>>>
>>> MyString
'I like to prgram with Python'
>>> MyString.count('i')
2
>>> MyString.replace("I", "We")
'We like to prgram with Python'
>>> MyString.lower()
'i like to prgram with python'
>>> MyString.upper()
'I LIKE TO PRGRAM WITH PYTHON'
>>>
```

Python Methodologies for Data Sciences

We can see that the count method counts all the ‘i’s in the string.

replace does just that; my “I” is now a “We,” but it does not change the variable value. If we wanted this change to be permanent we would assign the results here to a new variable.

upper and lower are self-explanatory, yes? **If you do not save this to a variable the change does not remain.** Methods have return values and it’s up to the programmer to save those results to a variable if they desire (with strings, we’ll learn why later in these slides).

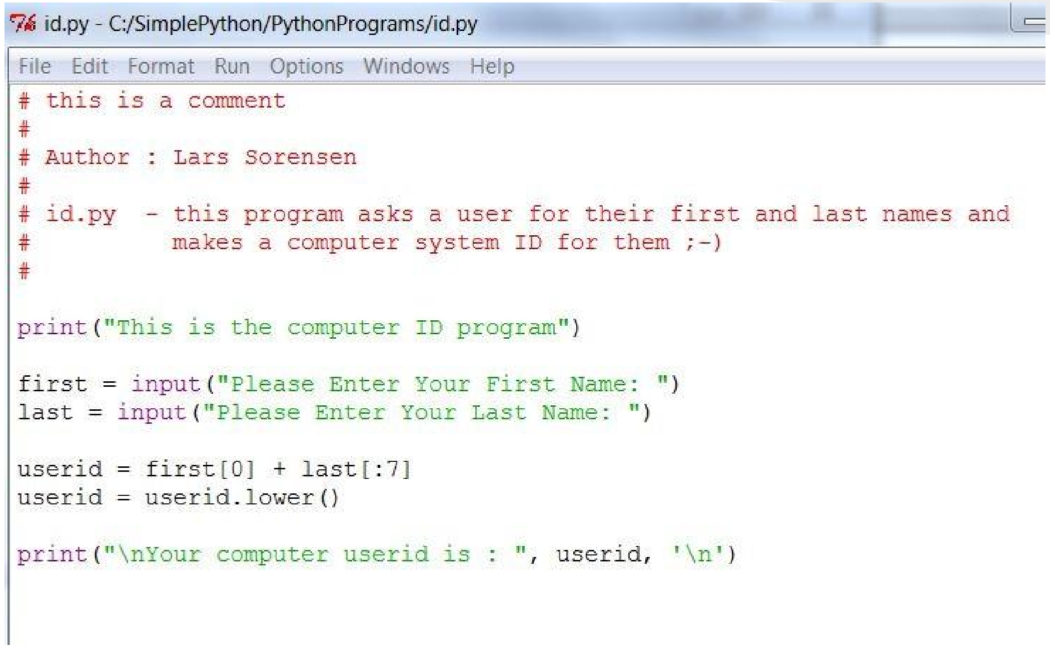


```
>>>
>>> MyString
'I like to prgram with Python'
>>> MyString.count('i')
2
>>> MyString.replace("I", "We")
'We like to prgram with Python'
>>> MyString.lower()
'i like to prgram with python'
>>> MyString.upper()
'I LIKE TO PRGRAM WITH PYTHON'
>>>
```

Python Methodologies for Data Sciences

This is Zelle’s userid program.

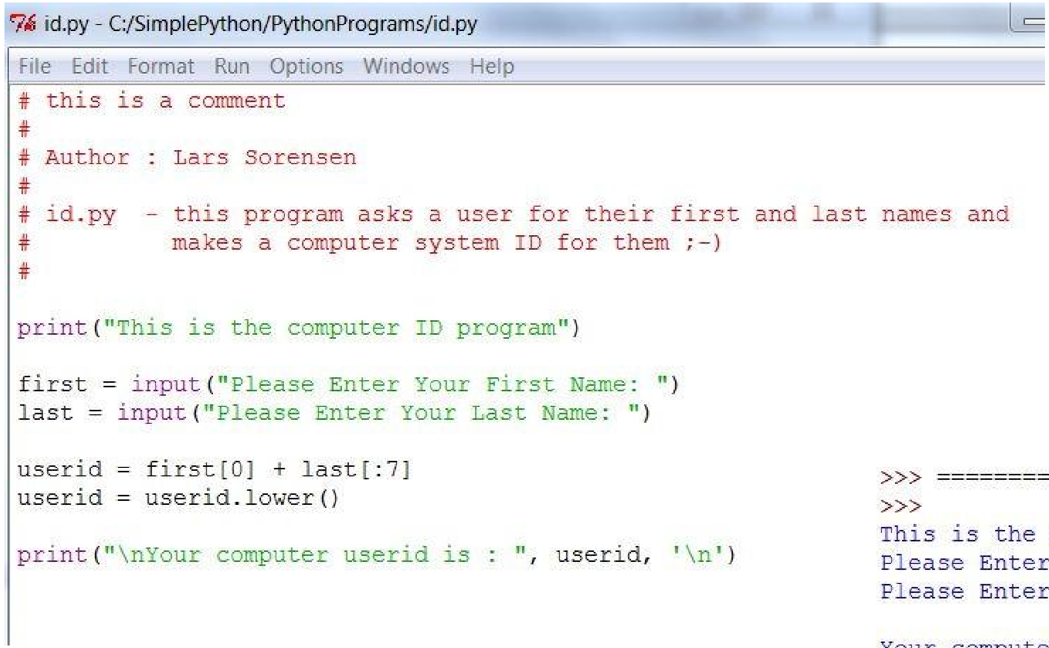
Type it in and run it.



id.py - C:/SimplePython/PythonPrograms/id.py

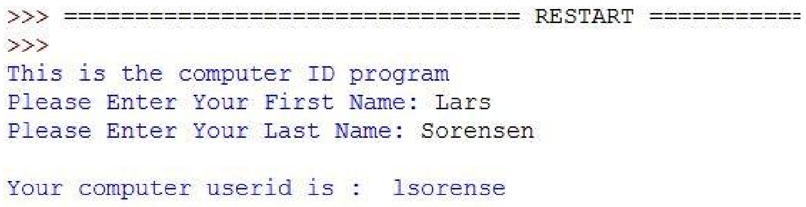
```python
# this is a comment
#
# Author : Lars Sorensen
#
# id.py  - this program asks a user for their first and last names and
#          makes a computer system ID for them ;-)
#

print("This is the computer ID program")

first = input("Please Enter Your First Name: ")
last = input("Please Enter Your Last Name: ")

userid = first[0] + last[:7]
userid = userid.lower()

print("\nYour computer userid is : ", userid, '\n')
```

Python Methodologies for Data Sciences


id.py - C:/SimplePython/PythonPrograms/id.py

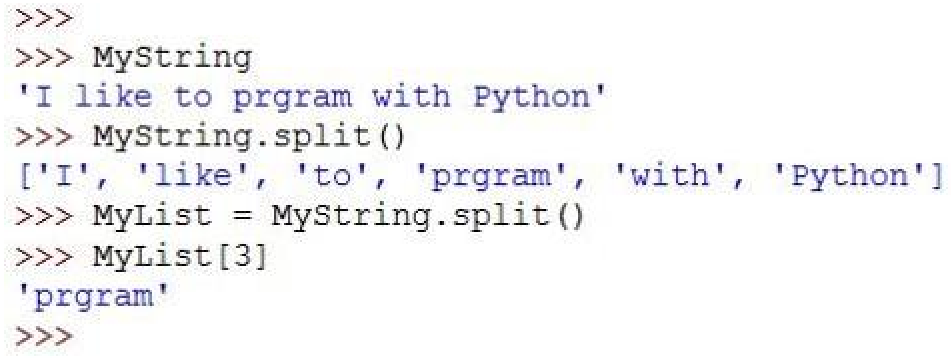
```
# this is a comment
#
# Author : Lars Sorensen
#
# id.py  - this program asks a user for their first and last names and
#          makes a computer system ID for them ;-)
#

print("This is the computer ID program")

first = input("Please Enter Your First Name: ")
last = input("Please Enter Your Last Name: ")

userid = first[0] + last[:7]
userid = userid.lower()

print("\nYour computer userid is : ", userid, '\n')
```

Can you explain what happened here?



```
>>> ================================ RESTART ===========
>>>
This is the computer ID program
Please Enter Your First Name: Lars
Please Enter Your Last Name: Sorensen

Your computer userid is :  lsorense
```

Python Methodologies for Data Sciences



```
>>>
>>> MyString
'I like to prgram with Python'
>>> MyString.split()
['I', 'like', 'to', 'prgram', 'with', 'Python']
>>> MyList = MyString.split()
>>> MyList[3]
'prgram'
>>>
```

split is a method that breaks up the words in a string and puts them into a List. Later, you will see that this is a very important tool, but the first thing we have to do is learn what a List is...

Lists

Python Methodologies for Data Sciences

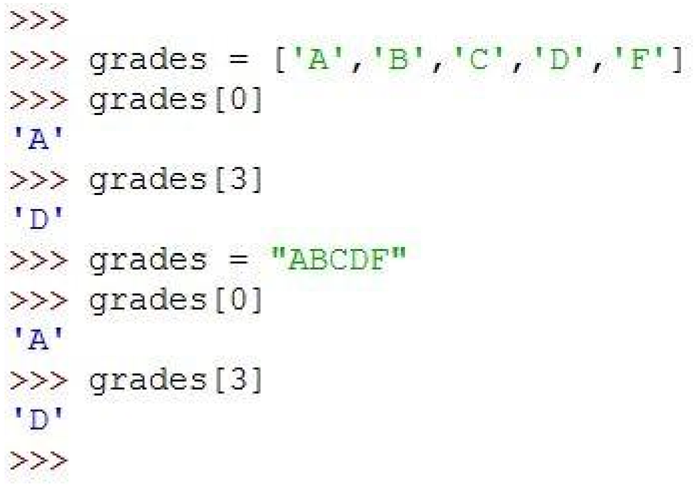
Look to the right. The first time I assign the variable *grades* a value it's five individual characters inside of brackets (not braces, brackets).

This creates a list. A list is similar to a string in that it is a sequence. Strings are sequences of characters that can be indexed or referred to by their place in line. Lists are the same in that regard.

You can see that underneath the list I re-assign the variable "grades" to a string. I can do the same indexing.

The question you might be asking now is "Why do we need lists if they act like strings?"

The answer is, they don't act like strings. They are strings on steroids.

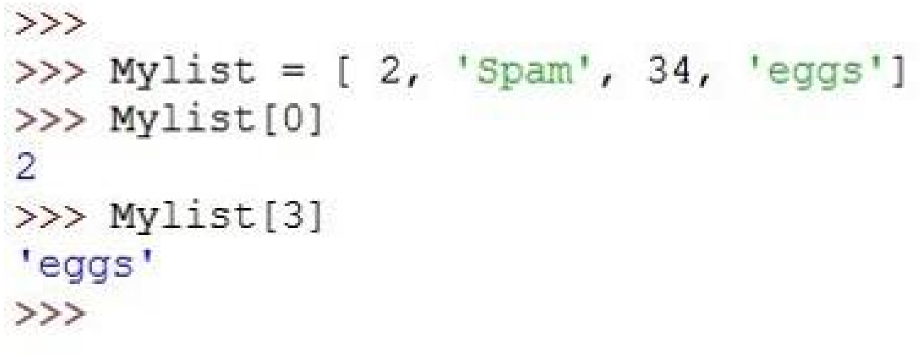


```
>>>
>>> grades = ['A','B','C','D','F']
>>> grades[0]
'A'
>>> grades[3]
'D'
>>> grades = "ABCDF"
>>> grades[0]
'A'
>>> grades[3]
'D'
>>>
```

Python Methodologies for Data Sciences

Here’s the first difference: A list can hold items with different data types. Right now, while we’re only dealing with Integers and Floats and Strings that may not seem like that big a deal, but later, when we have more complicated data types to deal with it’s going to become important.

In Python we call a sequence or collection of commonly kept items a list. In other languages you may have heard them referred to as *arrays*. What’s important, as we will see in the second half of our slides, is that these items are all kept under the same variable name but at a different index.



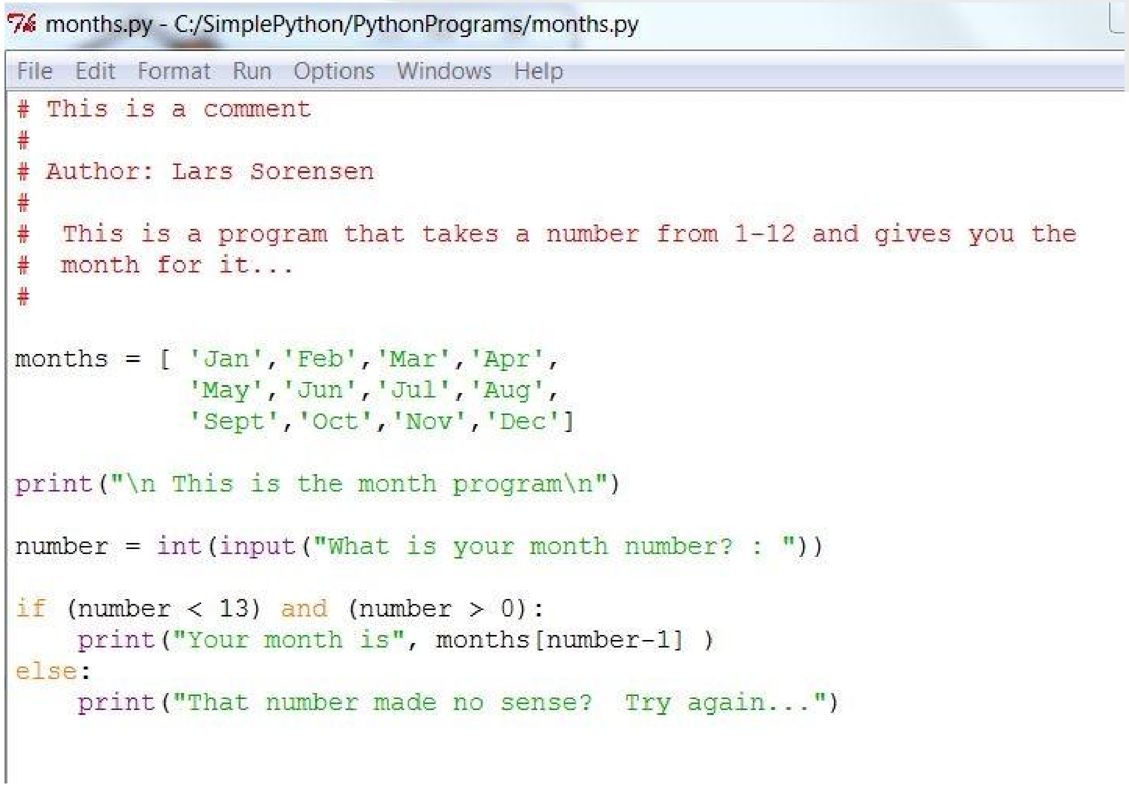
```
>>>
>>> Mylist = [ 2, 'Spam', 34, 'eggs']
>>> Mylist[0]
2
>>> Mylist[3]
'eggs'
>>>
```

Python Methodologies for Data Sciences

This is the month program from Zelle. Type it into IDLE and run it.

Note that we use a complex conditional to make sure that the input to the program makes sense. There is no month 22, why should I except that? This is an example of what's called "error trapping."

Notice also that we minus 1 from the index when we refer to our list. Computer Scientists start counting at 0, remember? months[0] = 'Jan' and months[1] = 'Feb'.


months.py - C:/SimplePython/PythonPrograms/months.py
File Edit Format Run Options Windows Help

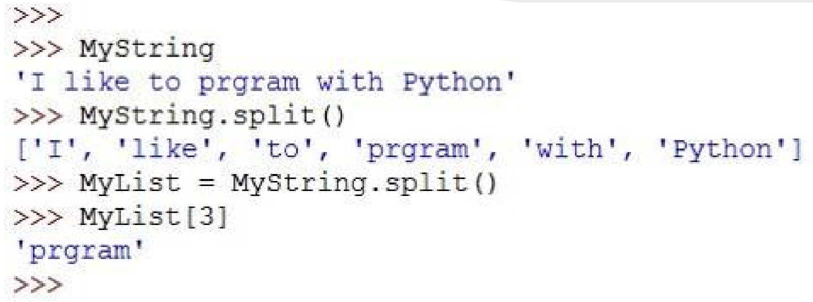
```
# This is a comment
#
# Author: Lars Sorensen
#
#  This is a program that takes a number from 1-12 and gives you the
#  month for it...
#

months = [ 'Jan','Feb','Mar','Apr',
           'May','Jun','Jul','Aug',
           'Sept','Oct','Nov','Dec']

print("\n This is the month program\n")

number = int(input("What is your month number? : "))

if (number < 13) and (number > 0):
    print("Your month is", months[number-1] )
else:
    print("That number made no sense?  Try again...")
```

Python Methodologies for Data Sciences

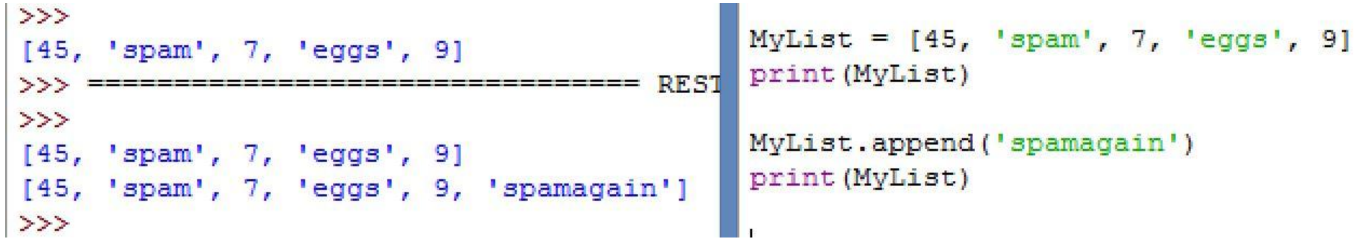
Lists have methods just like strings do. Two of the important ones are split and append.

split lets you take a string and break it's words out into a list. Computer folk often call individual words "tokens" and this makes life very easy when we need to examine sentences.

append lets you add to a list. There are also methods that let you remove things from a list as well.



```
>>>
>>> MyString
'I like to prgram with Python'
>>> MyString.split()
['I', 'like', 'to', 'prgram', 'with', 'Python']
>>> MyList = MyString.split()
>>> MyList[3]
'prgram'
>>>
```



```
>>>
[45, 'spam', 7, 'eggs', 9]
>>> ================================== REST
>>>
[45, 'spam', 7, 'eggs', 9]
[45, 'spam', 7, 'eggs', 9, 'spamagain']
>>>
```

```
MyList = [45, 'spam', 7, 'eggs', 9]
print(MyList)

MyList.append('spamagain')
print(MyList)
```

Python Methodologies for Data Sciences

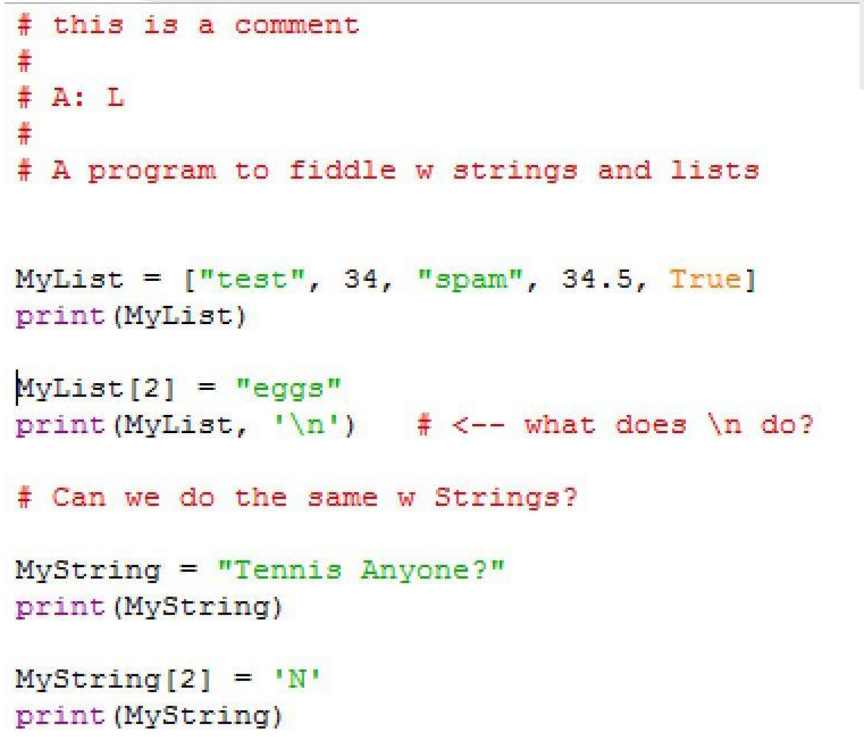
Now we see another difference between strings and lists.

Lists are "mutable." All that means is that we can change them and they can keep the same name. Strings do not have this property in Python: they are what's called "immutable."

If you want to change an item in a list you can use an assignment and re-assign that element of the list as we do to the right. We have taken the third item (index 2) and changed it from the string 'spam' to the string 'eggs'.

Underneath that we try the same thing with a string. Any guesses as to what will happen?

(Go into the SAKAI forums and tell me what '\n' does for an extra point towards your quizzes.)

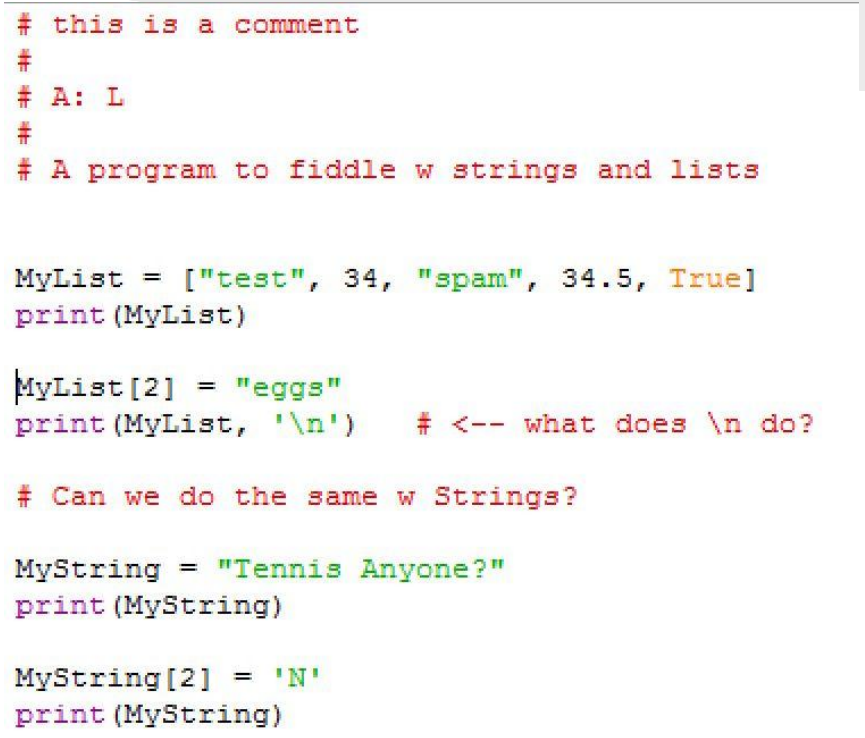


```
# this is a comment
#
# A: L
#
# A program to fiddle w strings and lists


MyList = ["test", 34, "spam", 34.5, True]
print(MyList)

MyList[2] = "eggs"
print(MyList, '\n')    # <-- what does \n do?

# Can we do the same w Strings?

MyString = "Tennis Anyone?"
print(MyString)

MyString[2] = 'N'
print(MyString)
```

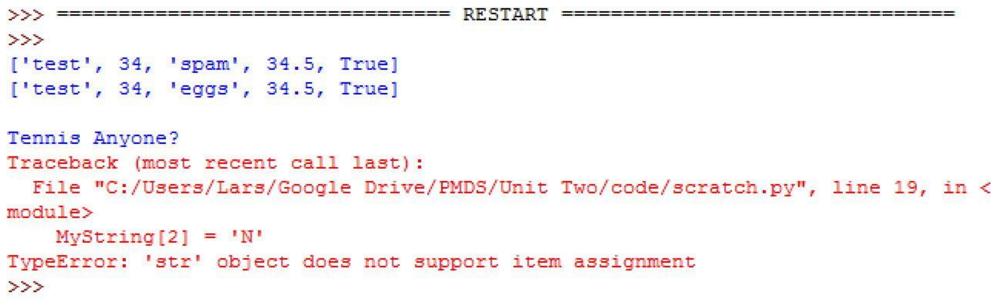
Python Methodologies for Data Sciences

That's right, it blows up into little Python meatballs.

Strings cannot be changed on the fly like lists can because lists are mutable and strings are not. Being able to store any data type you want as an item and being able to dynamically alter a list are two of the reasons why lists are one of the most used data type in Python. It really is in your best interest to use them, practice with them and understand all you can do with them...



```
# this is a comment
#
# A: L
#
# A program to fiddle w strings and lists


MyList = ["test", 34, "spam", 34.5, True]
print(MyList)

MyList[2] = "eggs"
print(MyList, '\n')    # <-- what does \n do?

# Can we do the same w Strings?

MyString = "Tennis Anyone?"
print(MyString)

MyString[2] = 'N'
print(MyString)
```

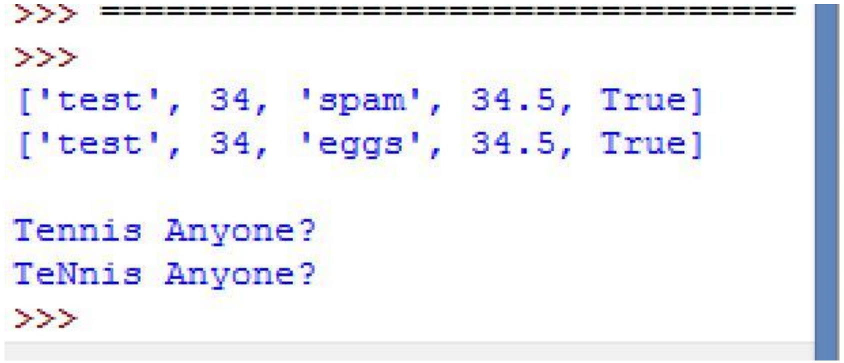


```
>>> ================================ RESTART ================================
>>>
['test', 34, 'spam', 34.5, True]
['test', 34, 'eggs', 34.5, True]

Tennis Anyone?
Traceback (most recent call last):
  File "C:/Users/Lars/Google Drive/PMDS/Unit Two/code/scratch.py", line 19, in <
module>
    MyString[2] = 'N'
TypeError: 'str' object does not support item assignment
>>>
```

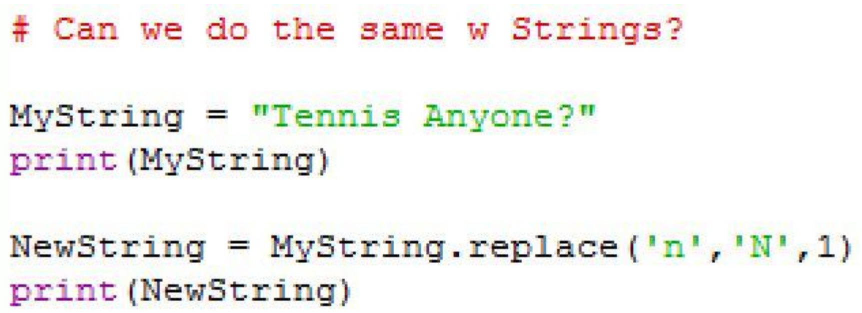
Python Methodologies for Data Sciences

No I'm not going to leave you hanging. Below is the way you can change a character in a string. You have to use the replace method and you have to assign the results to a new string. The old string cannot be changed, it's immutable, but we can have a method examine the existing string and write out a new one with the changes you want. That's what replace does. Nuff said.

Lookup how replace works on the web to find out what that 1 is doing at the end of the parameter list.



```
>>> ================================
>>>
['test', 34, 'spam', 34.5, True]
['test', 34, 'eggs', 34.5, True]

Tennis Anyone?
TeNnis Anyone?
>>>
```



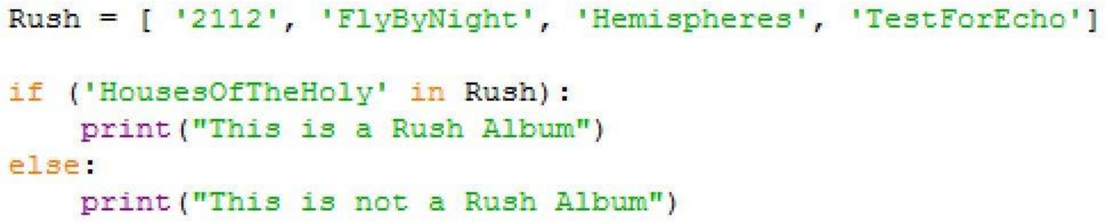
```
# Can we do the same w Strings?

MyString = "Tennis Anyone?"
print(MyString)

NewString = MyString.replace('n','N',1)
print(NewString)
```

Python Methodologies for Data Sciences

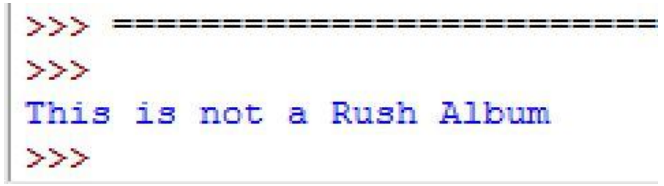
Okay. For the first time we will get a look at something that's actually useful for data science. Look at the code to the right.

The list I call Rush is a list of record albums from the band Rush. With a simple if statement I check to see if the string "HousesOfTheHoly" (It's a Led Zeppelin album, not Rush) is in the list.

Note the keyword 'in.' It's a beautiful thing 'in'. Even with a list of one million items you could use 'in' to test for inclusion in a list. Sound handy? You have no idea.



```
Rush = [ '2112', 'FlyByNight', 'Hemispheres', 'TestForEcho']

if ('HousesOfTheHoly' in Rush):
    print("This is a Rush Album")
else:
    print("This is not a Rush Album")
```



```
>>> ==============================
>>>
This is not a Rush Album
>>>
```

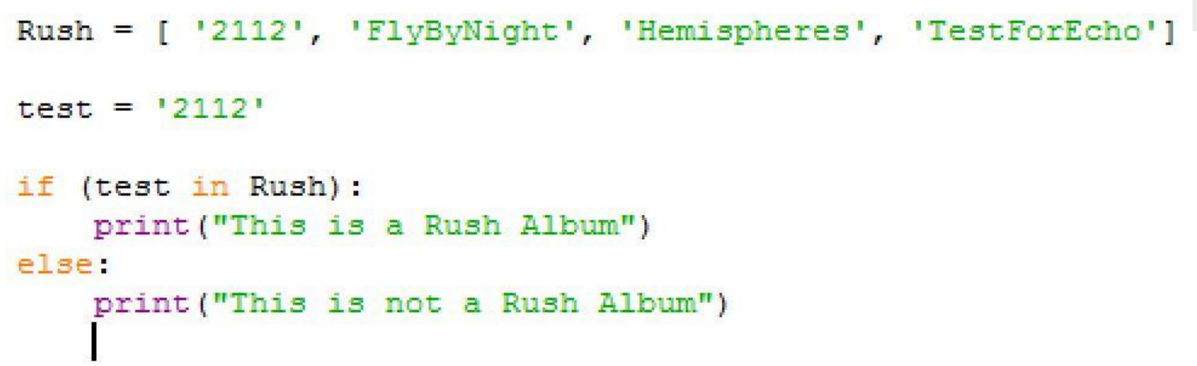
Python Methodologies for Data Sciences

We can also use variables and test for inclusion in a list with the variable's value.

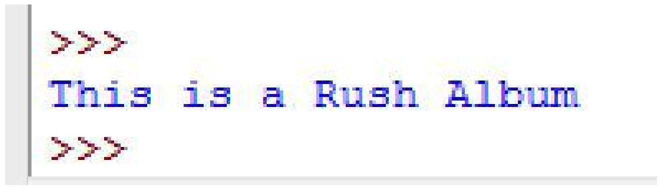
When we do loops (in, oh, about 5 minutes if you don't take a break... but do take the break) you are going to find out how powerful this is.

Finally, some next level stuff that we can apply to data science and real problem solving.

Just one more thing....



```
Rush = [ '2112', 'FlyByNight', 'Hemispheres', 'TestForEcho']

test = '2112'

if (test in Rush):
    print("This is a Rush Album")
else:
    print("This is not a Rush Album")
```



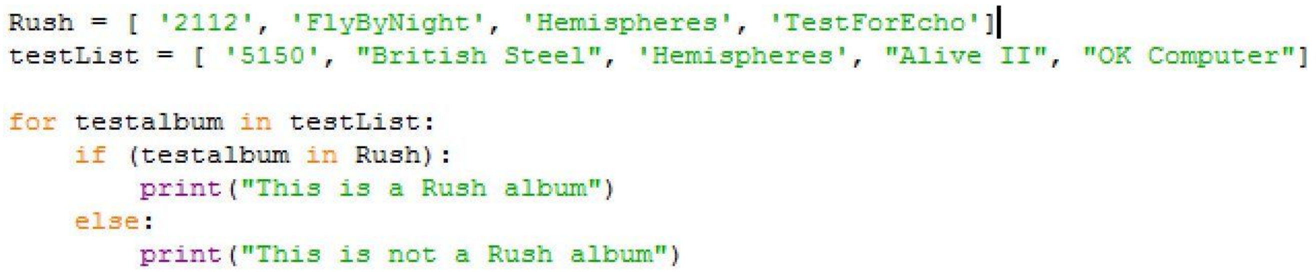
```
>>>
This is a Rush Album
>>>
```

Python Methodologies for Data Sciences

Looking ahead for just a heartbeat. Look at the code.

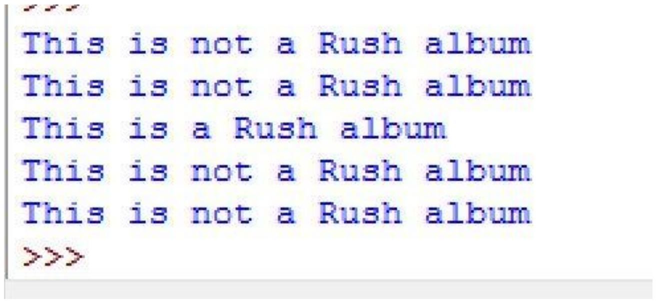
Two lists. One if statement. That 'for' keyword will be explained shortly, suffice it to say that we have a loop there that runs through the contents of the list testList and checks to see if the items are also in the list Rush.

Imagine if testList had a million items. It would still work.



```
Rush = [ '2112', 'FlyByNight', 'Hemispheres', 'TestForEcho']
testList = [ '5150', "British Steel", 'Hemispheres', "Alive II", "OK Computer"]

for testalbum in testList:
    if (testalbum in Rush):
        print("This is a Rush album")
    else:
        print("This is not a Rush album")
```

Now, after a unit and a half, we are finally going to get to the point where our programs are useful and even more fun. NOW we can think of data sciency things just a little bit. Most of the basics are out of the way. Once we have loops we'll have some powerful tools to use to solve problems. But first, take a break and absorb your strings and lists....



```
This is not a Rush album
This is not a Rush album
This is a Rush album
This is not a Rush album
This is not a Rush album
>>>
```

Python Methodologies for Data Sciences

Strings and Lists are incredibly important. Make sure you know what they are and what their differences are.

They are both sequences and as we do loops you are going to see how important that is.

First, take a break for a while. Then move on to the second half of these Unit 2 slides as we look at the main event, loops.

Loops

Python Methodologies for Data Sciences

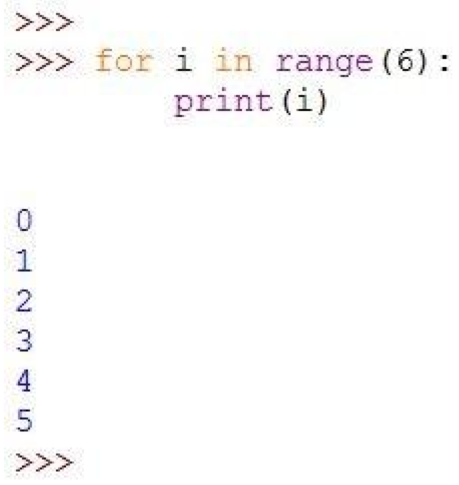
Now we're ready for the next level. We are going to look at "Iteration." Iteration is just the repetition of a process. Let's say you have a sequence, in our case a string or a list. What if you wanted to perform the same operation on every character in your string or every item in your list?

That's where loops come in. To the right you can see what's called a "for" loop. The range function is a special function that makes lists of numbers. Python only iterates over sequences, so if we want to do something six times we merely create a list numbered from 0 (remember, we're programmers, we start counting at 0) to 5.

Underneath the "for" loop we put the block of code we want executed for every item of the sequence. Here we merely print the variable i. When programmers learn to code we are taught to use i, j and k as the variables to use when we loop. I have no idea why, but I am passing this tradition on to you so when you see it elsewhere you'll know why.

Seems even programmers can't agree where it came from

(http://stackoverflow.com/questions/1147312/who-invented-i-j-k-as-integer-counter-variable-names



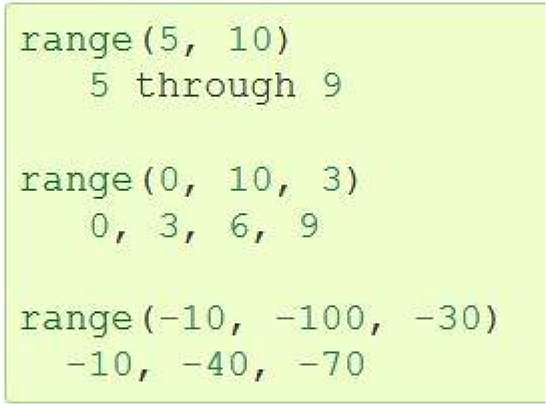
```
>>>
>>> for i in range(6):
        print(i)



0
1
2
3
4
5
>>>
```

Python Methodologies for Data Sciences

The key to using loops in Python at first is learning how to use the range function. Unlike other languages, Python only iterates or "loops" over lists so if we want to do something 10 times we need to use the range function to make a list from 0-9.

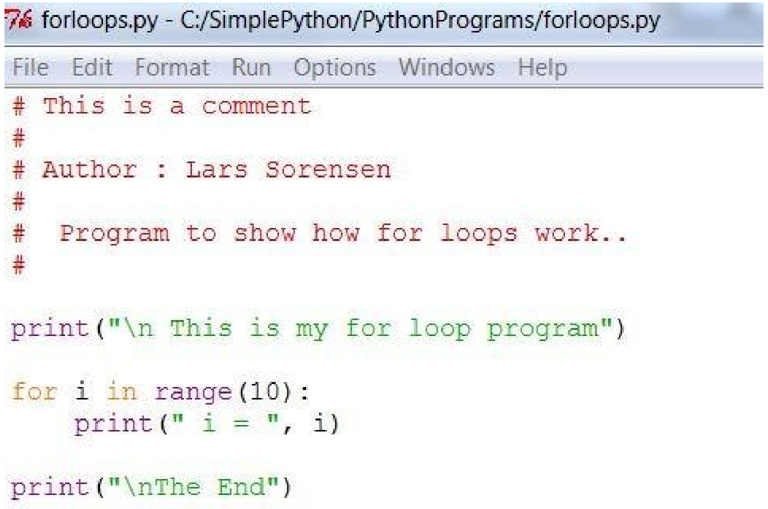
We can also use different increments and decrements like we do in the examples to the right. The first example has a start and an end. The second adds an increment, we start at 0, go until 10 and increment by 3. The last example shows that we can even go backwards with the range function.

Note that the last number of the range is never used. The front of the range is inclusive (start with 5) and the end of the range is exclusive (do not use 10). That's why we do not list -100 in the last example.

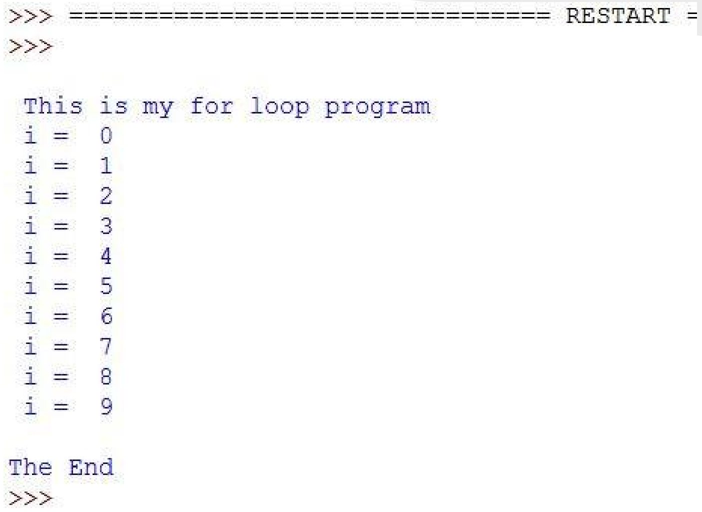


```
range(5, 10)
   5 through 9

range(0, 10, 3)
   0, 3, 6, 9

range(-10, -100, -30)
  -10, -40, -70
```

Python Methodologies for Data Sciences


forloops.py - C:/SimplePython/PythonPrograms/forloops.py
File Edit Format Run Options Windows Help

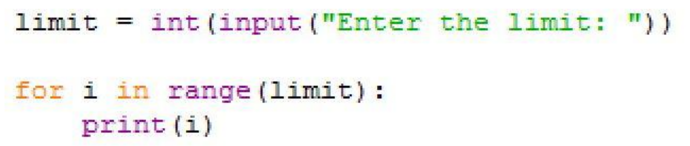
```python
# This is a comment
#
# Author : Lars Sorensen
#
#  Program to show how for loops work..
#

print("\n This is my for loop program")

for i in range(10):
    print(" i = ", i)

print("\nThe End")
```



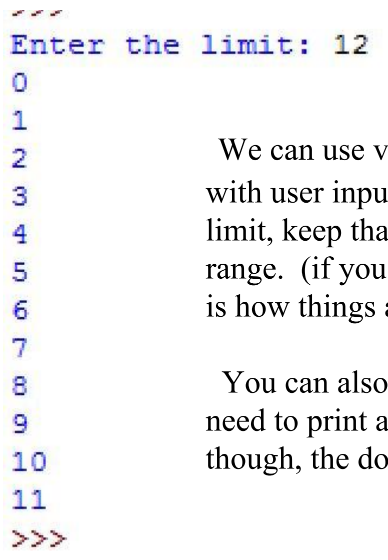
```
>>> ================================ RESTART =
>>>

 This is my for loop program
 i =  0
 i =  1
 i =  2
 i =  3
 i =  4
 i =  5
 i =  6
 i =  7
 i =  8
 i =  9

The End
>>>
```

Type in the above code and give it a try. Change the value and see what happens. What if you put in a negative number? What about zero?

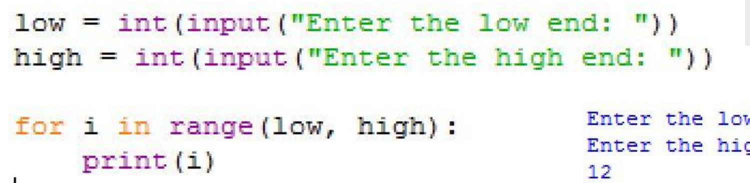
Python Methodologies for Data Sciences



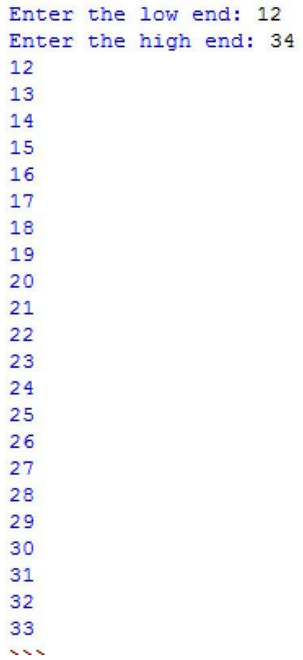
```
limit = int(input("Enter the limit: "))

for i in range(limit):
    print(i)
```



```
Enter the limit: 12
0
1
2
3
4
5
6
7
8
9
10
11
>>>
```



```
low = int(input("Enter the low end: "))
high = int(input("Enter the high end: "))

for i in range(low, high):
    print(i)
```

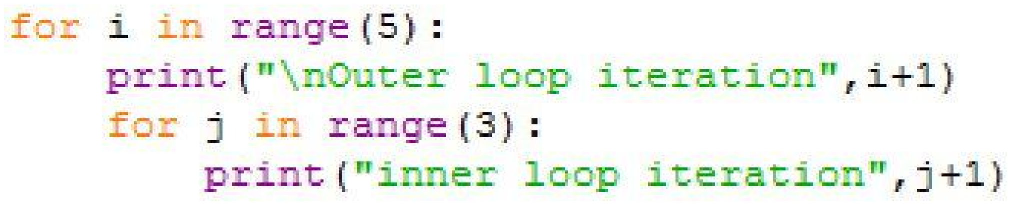


```
Enter the low end: 12
Enter the high end: 34
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
```

We can use variables in our range functions. Here we create two loops with user input. Notice how the last iteration is one less than the high end limit, keep that in mind, the range function is exclusive at the end of the range. (if you want to see 34 just create the range from (low, high+1), this is how things are usually handled.

You can also print 1-12 by adding 1 to the i in your print statement if you need to print a list from 1 through 12 (print(i+1)). This is a shortcut though, the do it properly use range(1,13).

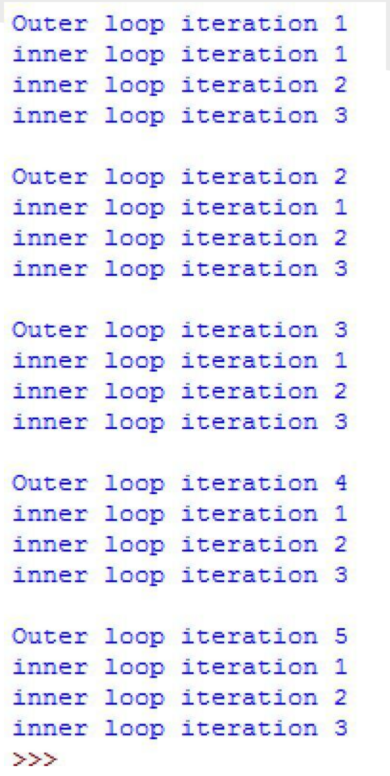
Python Methodologies for Data Sciences



```python
for i in range(5):
    print("\nOuter loop iteration",i+1)
    for j in range(3):
        print("inner loop iteration",j+1)
```

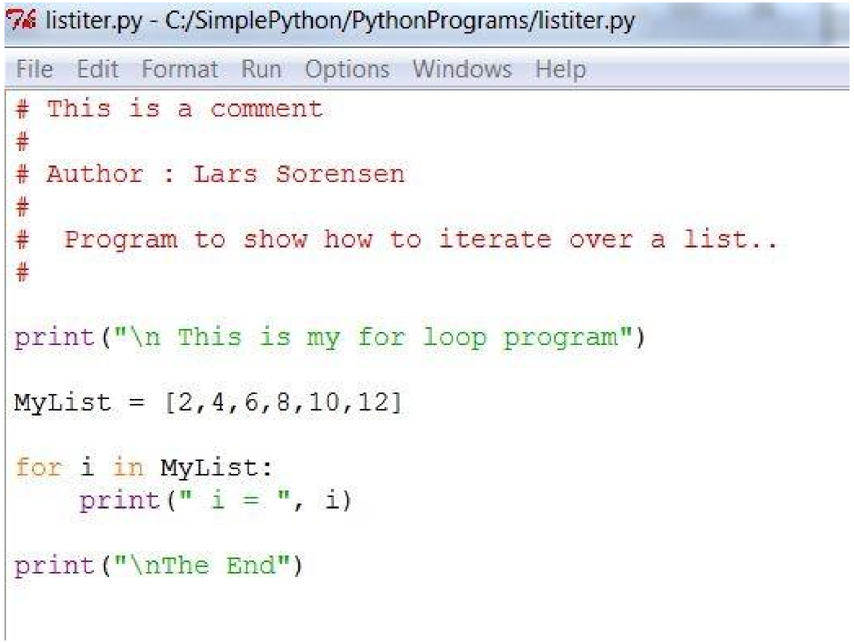
You can run loops inside of loops. We call these “nested” loops. They show up quite a bit in programming so it’s a good idea to understand what’s going here.

Later, we will look at two dimensional lists. They are just lists where the data is accessed with two indices, not just one. It’s how we represent a grid in programming. Anyway, nested loops are a good way to traverse (go across) one of these and look at every item. You’ll see more in the coming weeks. Until then just know that you can run another loop inside your top most loop.

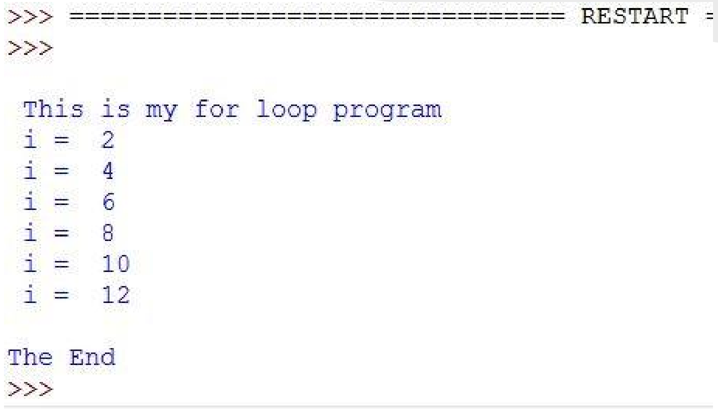


```
Outer loop iteration 1
inner loop iteration 1
inner loop iteration 2
inner loop iteration 3

Outer loop iteration 2
inner loop iteration 1
inner loop iteration 2
inner loop iteration 3

Outer loop iteration 3
inner loop iteration 1
inner loop iteration 2
inner loop iteration 3

Outer loop iteration 4
inner loop iteration 1
inner loop iteration 2
inner loop iteration 3

Outer loop iteration 5
inner loop iteration 1
inner loop iteration 2
inner loop iteration 3
>>>
```

Python Methodologies for Data Sciences


listiter.py - C:/SimplePython/PythonPrograms/listiter.py
File Edit Format Run Options Windows Help

```
# This is a comment
#
# Author : Lars Sorensen
#
#  Program to show how to iterate over a list..
#

print("\n This is my for loop program")

MyList = [2,4,6,8,10,12]

for i in MyList:
    print(" i = ", i)

print("\nThe End")
```



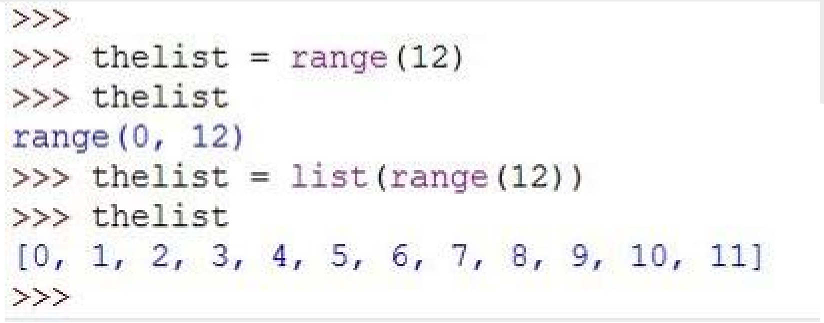
```
>>> ================================ RESTART =
>>>

 This is my for loop program
 i =  2
 i =  4
 i =  6
 i =  8
 i =  10
 i =  12

The End
>>>
```

You don’t have to use range. You can iterate over any list you have. Here I create a list that includes the numbers from 2 through 12 by 2.

Type this in and give it a try...

Python Methodologies for Data Sciences

Behind the Curtain



```
>>>
>>> thelist = range(12)
>>> thelist
range(0, 12)
>>> thelist = list(range(12))
>>> thelist
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
>>>
```

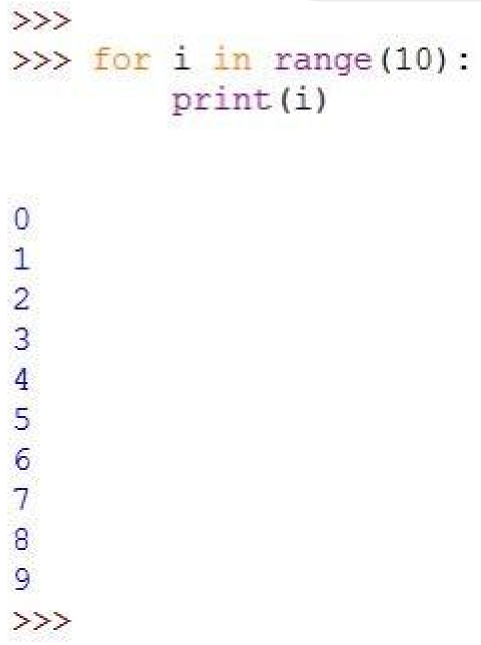
Here's some insider information. range doesn't really create a list. It's something in Python called a "generator." That means that it doesn't create a list, it just generates the next number in the sequence it has been given. This saves memory if we use insanely large numbers in our loops (Calculate Pi anyone?)

Loops can use generators just fine so this is usually not an issue, but just in case you want an honest to goodness list I show you how above. All we do is cast the results of range to a list (just like we casted with int or float). If this hurts your brain you have my permission to forget about it for a while, I just wanted you to have the real 411 from the beginning.... onwards...

Python Methodologies for Data Sciences

OK. The kind of loop to your right is called a "definite" loop. Come hell or high water that loop is going to "print(i)" only 10 times and then move on to the rest of the program.

What if we have situations where we don't know how many times we want to go through a process? What if we want to keep going until a certain condition is true?



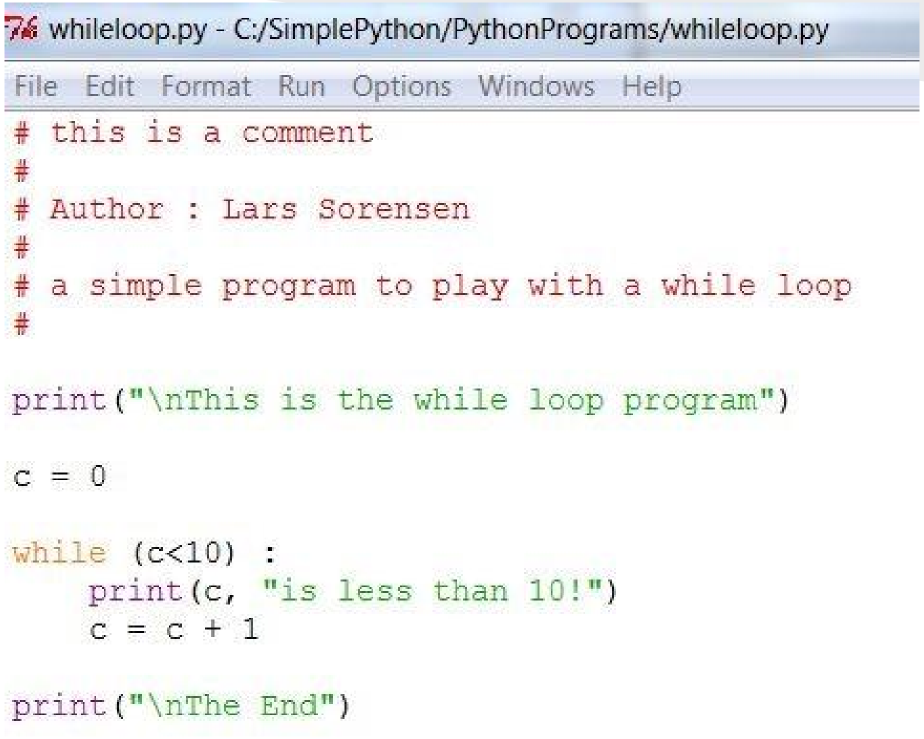
```
>>>
>>> for i in range(10):
        print(i)


0
1
2
3
4
5
6
7
8
9
>>>
```

Python Methodologies for Data Sciences

Type this program in and run it.

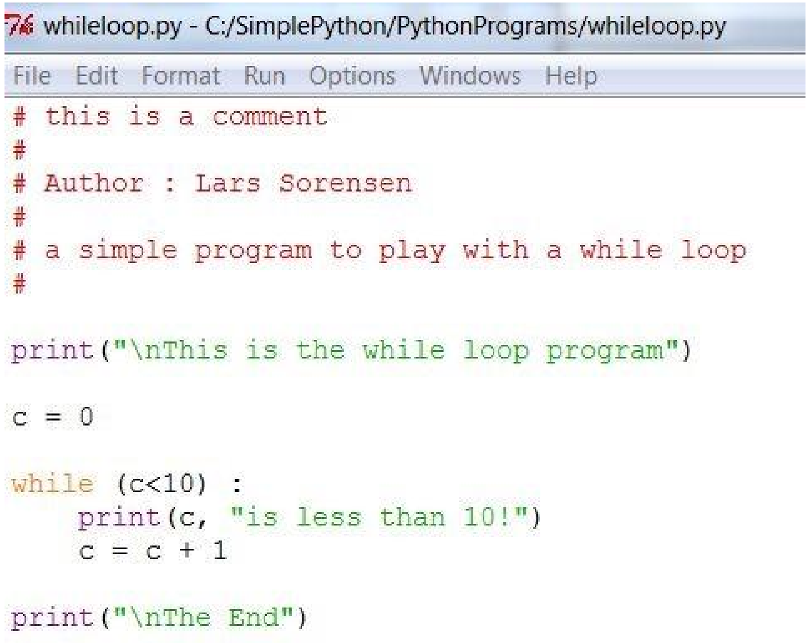
Now change the condition.

What would happen if you comment out the
c = c + 1
line?


whileloop.py - C:/SimplePython/PythonPrograms/whileloop.py
File Edit Format Run Options Windows Help

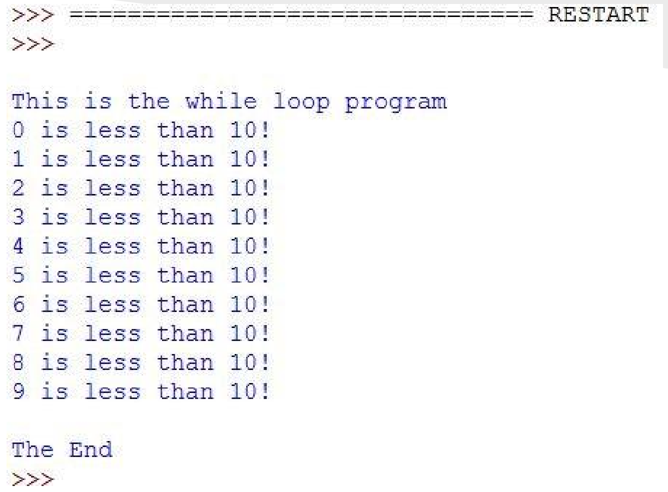
```
# this is a comment
#
# Author : Lars Sorensen
#
# a simple program to play with a while loop
#

print("\nThis is the while loop program")

c = 0

while (c<10) :
    print(c, "is less than 10!")
    c = c + 1

print("\nThe End")
```

Python Methodologies for Data Sciences



whileloop.py - C:/SimplePython/PythonPrograms/whileloop.py

```python
# this is a comment
#
# Author : Lars Sorensen
#
# a simple program to play with a while loop
#

print("\nThis is the while loop program")

c = 0

while (c<10) :
    print(c, "is less than 10!")
    c = c + 1

print("\nThe End")
```



```
>>> ================================ RESTART
>>>

This is the while loop program
0 is less than 10!
1 is less than 10!
2 is less than 10!
3 is less than 10!
4 is less than 10!
5 is less than 10!
6 is less than 10!
7 is less than 10!
8 is less than 10!
9 is less than 10!

The End
>>>
```

This is a while loop. It just keeps going until the condition it's watching is False. We kept adding 1 to c (that's called a counter, it's just a variable that increments 1 every time through a loop.) until c <10 was False (when c = 10). Then the loop stopped.

While loops are examples of "indefinite loops". We don't know how many times through they will go; they have to meet a condition.

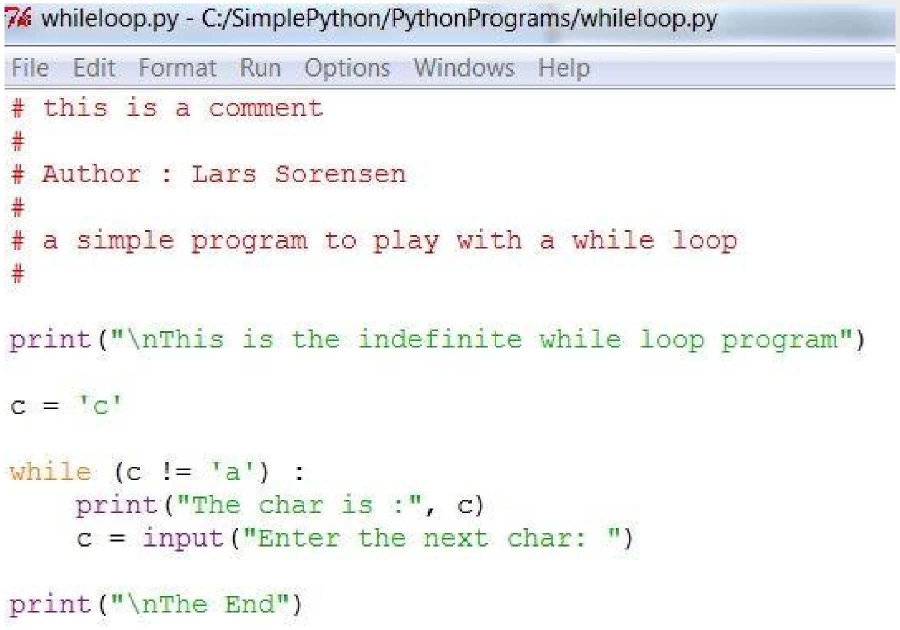
Python Methodologies for Data Sciences

Now, you might be saying:

“Wait a minutes. That last while loop was going to stop when c got to ten no matter what. That means it’s definite!”

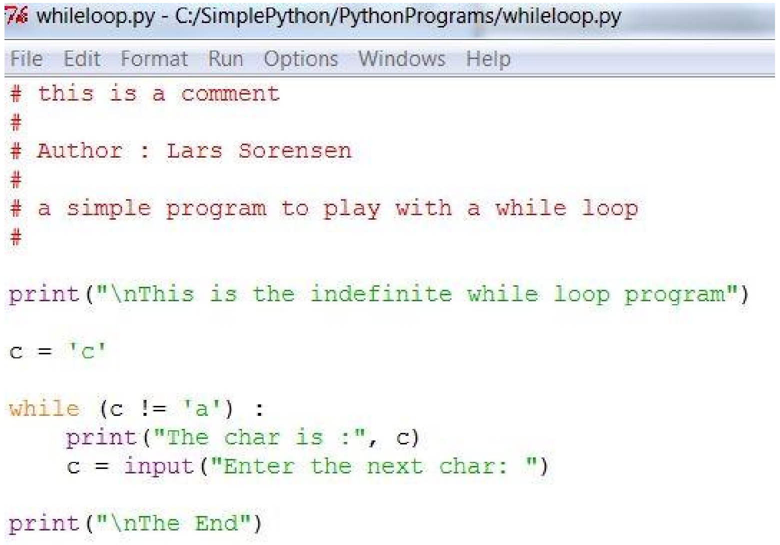
You’re right. In a sense we used an indefinite loop structure (while) to create a definite loop.

Check out this code. Type it in and run it. This is indefinite. It’s going to run until you enter an ‘a’

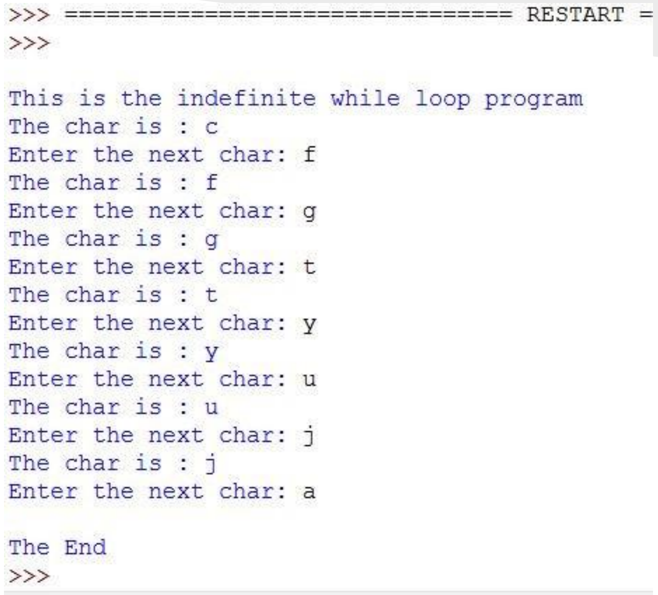

whileloop.py - C:/SimplePython/PythonPrograms/whileloop.py
File Edit Format Run Options Windows Help

```
# this is a comment
#
# Author : Lars Sorensen
#
# a simple program to play with a while loop
#

print("\nThis is the indefinite while loop program")

c = 'c'

while (c != 'a') :
    print("The char is :", c)
    c = input("Enter the next char: ")

print("\nThe End")
```

Python Methodologies for Data Sciences


whileloop.py - C:/SimplePython/PythonPrograms/whileloop.py
File Edit Format Run Options Windows Help

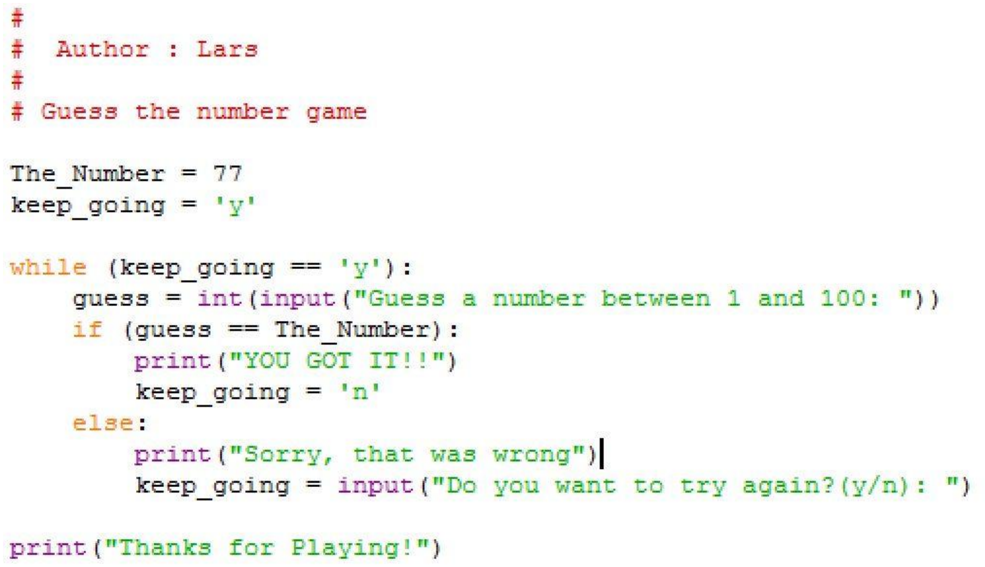
```
# this is a comment
#
# Author : Lars Sorensen
#
# a simple program to play with a while loop
#

print("\nThis is the indefinite while loop program")

c = 'c'

while (c != 'a') :
    print("The char is :", c)
    c = input("Enter the next char: ")

print("\nThe End")
```



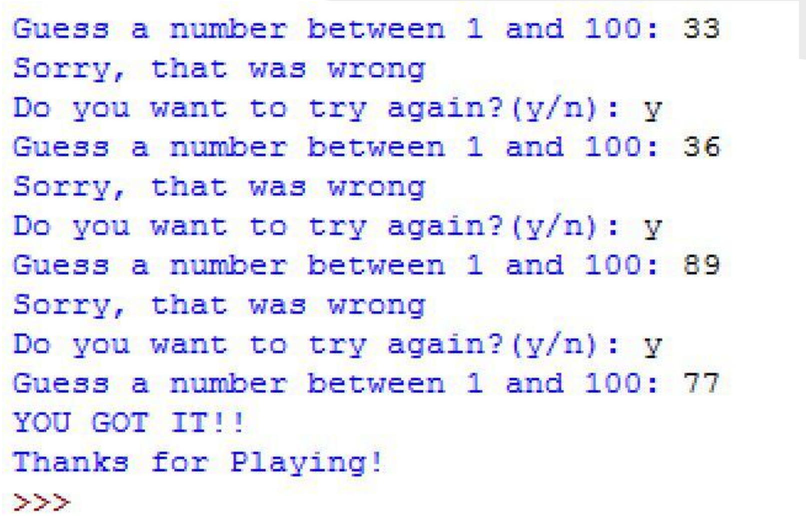
```
>>> ================================ RESTART =
>>>

This is the indefinite while loop program
The char is : c
Enter the next char: f
The char is : f
Enter the next char: g
The char is : g
Enter the next char: t
The char is : t
Enter the next char: y
The char is : y
Enter the next char: u
The char is : u
Enter the next char: j
The char is : j
Enter the next char: a

The End
>>>
```

Here’s the run. Can you think of a good use for this?

Python Methodologies for Data Sciences



```
#
#  Author : Lars
#
# Guess the number game

The_Number = 77
keep_going = 'y'

while (keep_going == 'y'):
    guess = int(input("Guess a number between 1 and 100: "))
    if (guess == The_Number):
        print("YOU GOT IT!!")
        keep_going = 'n'
    else:
        print("Sorry, that was wrong")
        keep_going = input("Do you want to try again?(y/n): ")

print("Thanks for Playing!")
```

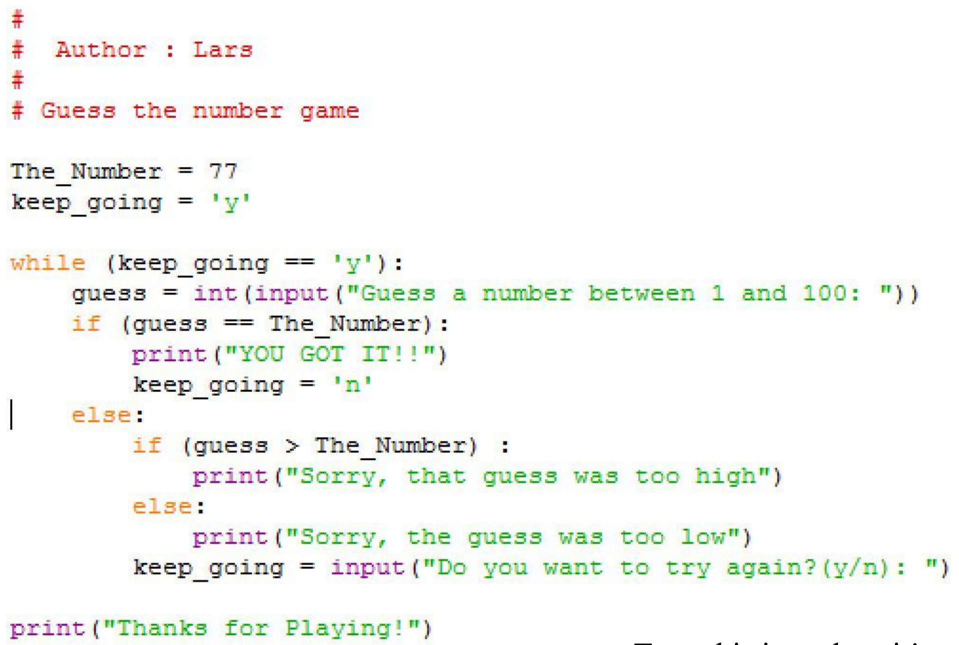


```
Guess a number between 1 and 100: 33
Sorry, that was wrong
Do you want to try again?(y/n): y
Guess a number between 1 and 100: 36
Sorry, that was wrong
Do you want to try again?(y/n): y
Guess a number between 1 and 100: 89
Sorry, that was wrong
Do you want to try again?(y/n): y
Guess a number between 1 and 100: 77
YOU GOT IT!!
Thanks for Playing!
>>>
```

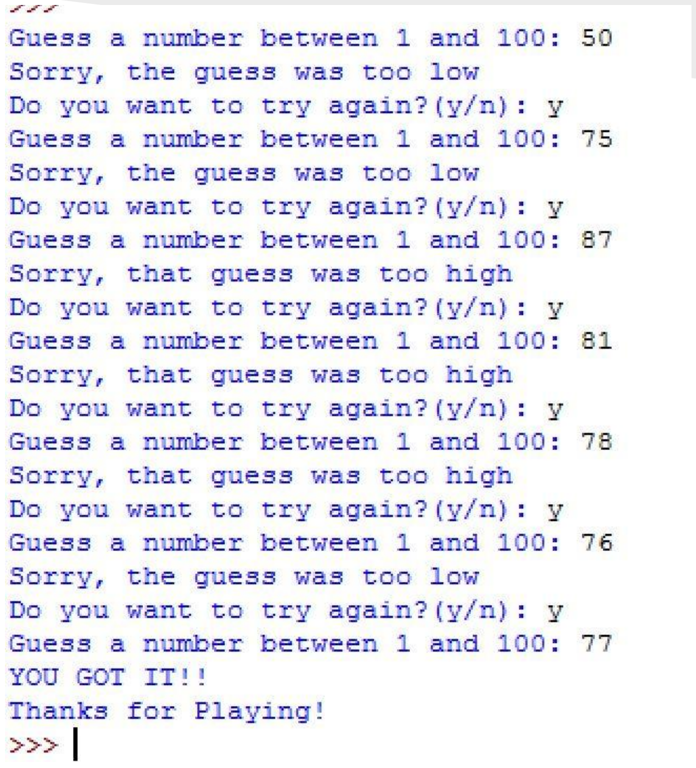
Now we have a way to keep our programs running and not have to keep starting them over and over again. Just put your code in a while loop, test to make sure your "flag" variable (keep_going) is 'y', and allow the user to make it an 'n' (or anything besides 'y') and you'll fall out of the loop, Print "Thanks for Playing" and you're done. This is how games STILL let you leave. When you input that "Q" or "Quit" it's being tested in a while loop!

Python Methodologies for Data Sciences

A few extra lines of code and it's a real high/low guess game!



```python
#
#  Author : Lars
#
# Guess the number game

The_Number = 77
keep_going = 'y'

while (keep_going == 'y'):
    guess = int(input("Guess a number between 1 and 100: "))
    if (guess == The_Number):
        print("YOU GOT IT!!")
        keep_going = 'n'
    else:
        if (guess > The_Number) :
            print("Sorry, that guess was too high")
        else:
            print("Sorry, the guess was too low")
        keep_going = input("Do you want to try again?(y/n): ")

print("Thanks for Playing!")
```



```
Guess a number between 1 and 100: 50
Sorry, the guess was too low
Do you want to try again?(y/n): y
Guess a number between 1 and 100: 75
Sorry, the guess was too low
Do you want to try again?(y/n): y
Guess a number between 1 and 100: 87
Sorry, that guess was too high
Do you want to try again?(y/n): y
Guess a number between 1 and 100: 81
Sorry, that guess was too high
Do you want to try again?(y/n): y
Guess a number between 1 and 100: 78
Sorry, that guess was too high
Do you want to try again?(y/n): y
Guess a number between 1 and 100: 76
Sorry, the guess was too low
Do you want to try again?(y/n): y
Guess a number between 1 and 100: 77
YOU GOT IT!!
Thanks for Playing!
>>>
```

Type this in and try it!
Can you make it better?

Other Kinds of Loops

We have looked at **Definite** loops (with for and the range function) and **Indefinite** loops (with while).

There are other kinds, but they are implemented with "for"s or "whiles"s

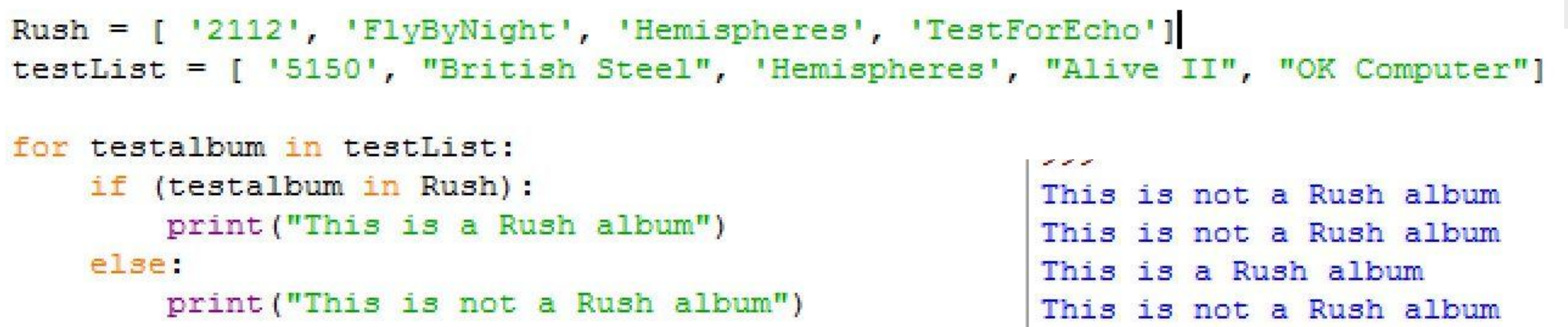
- **Interactive Loops** - Like the guessing game, the loop runs until user input says to stop
- **Sentinel Loops** - Loops that go through lists until they see a certain value. Here you would use != in the while condition. While (a != 999) for example
- **File Loops** - we'll see these in Unit 3. This is when you read a file until it's done even though you do not know how long it is before you start.

Finally!!!

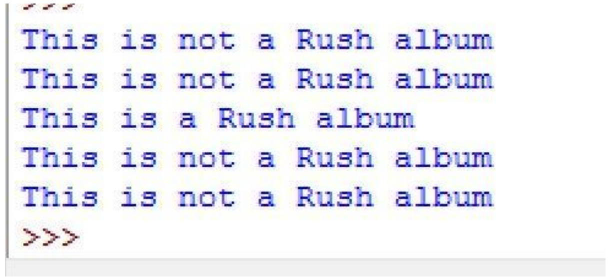
I think everyone who teaches programming waits for this moment. You have simple data under your belt. You know about conditionals. You know about lists and sequences of data. Now you have loops.

We can finally do some interesting things....

Python Methodologies for Data Sciences



```
Rush = [ '2112', 'FlyByNight', 'Hemispheres', 'TestForEcho']
testList = [ '5150', "British Steel", 'Hemispheres', "Alive II", "OK Computer"]

for testalbum in testList:
    if (testalbum in Rush):
        print("This is a Rush album")
    else:
        print("This is not a Rush album")
```



```
This is not a Rush album
This is not a Rush album
This is a Rush album
This is not a Rush album
This is not a Rush album
>>>
```

Now that we understand loops we can look at our list code again. Understand what's happening?

We go into a loop and for every item in testList the loop places that item in the variable "testalbum" and tests for inclusion in the list "Rush."

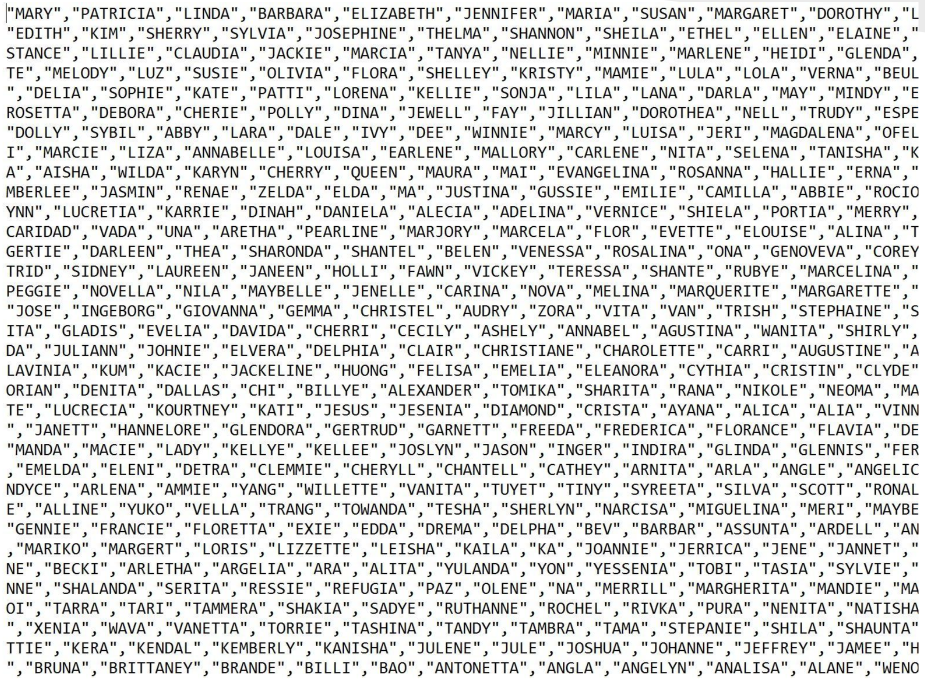
This is a simple, small example. What if testList had a million items? This could be rather handy when examining data, yes?

Python Methodologies for Data Sciences

What if we had data like this?

This is a list that contains 5,163 names. (well, it's a snippet of said list)

How can I test it to see if my name appears there?



```
"MARY","PATRICIA","LINDA","BARBARA","ELIZABETH","JENNIFER","MARIA","SUSAN","MARGARET","DOROTHY","L
"EDITH","KIM","SHERRY","SYLVIA","JOSEPHINE","THELMA","SHANNON","SHEILA","ETHEL","ELLEN","ELAINE","
STANCE","LILLIE","CLAUDIA","JACKIE","MARCIA","TANYA","NELLIE","MINNIE","MARLENE","HEIDI","GLENDA",
TE","MELODY","LUZ","SUSIE","OLIVIA","FLORA","SHELLEY","KRISTY","MAMIE","LULA","LOLA","VERNA","BEUL
","DELIA","SOPHIE","KATE","PATTI","LORENA","KELLIE","SONJA","LILA","LANA","DARLA","MAY","MINDY","E
ROSETTA","DEBORA","CHERIE","POLLY","DINA","JEWELL","FAY","JILLIAN","DOROTHEA","NELL","TRUDY","ESPE
"DOLLY","SYBIL","ABBY","LARA","DALE","IVY","DEE","WINNIE","MARCY","LUISA","JERI","MAGDALENA","OFEL
I","MARCIE","LIZA","ANNABELLE","LOUISA","EARLENE","MALLORY","CARLENE","NITA","SELENA","TANISHA","K
A","AISHA","WILDA","KARYN","CHERRY","QUEEN","MAURA","MAI","EVANGELINA","ROSANNA","HALLIE","ERNA","
MBERLEE","JASMIN","RENAE","ZELDA","ELDA","MA","JUSTINA","GUSSIE","EMILIE","CAMILLA","ABBIE","ROCIO
YNN","LUCRETIA","KARRIE","DINAH","DANIELA","ALECIA","ADELINA","VERNICE","SHIELA","PORTIA","MERRY",
CARIDAD","VADA","UNA","ARETHA","PEARLINE","MARJORY","MARCELA","FLOR","EVETTE","ELOUISE","ALINA","T
GERTIE","DARLEEN","THEA","SHARONDA","SHANTEL","BELEN","VENESSA","ROSALINA","ONA","GENOVEVA","COREY
TRID","SIDNEY","LAUREEN","JANEEN","HOLLI","FAWN","VICKEY","TERESSA","SHANTE","RUBYE","MARCELINA","
PEGGIE","NOVELLA","NILA","MAYBELLE","JENELLE","CARINA","NOVA","MELINA","MARQUERITE","MARGARETTE","
"JOSE","INGEBORG","GIOVANNA","GEMMA","CHRISTEL","AUDRY","ZORA","VITA","VAN","TRISH","STEPHAINE","S
ITA","GLADIS","EVELIA","DAVIDA","CHERRI","CECILY","ASHELY","ANNABEL","AGUSTINA","WANITA","SHIRLY",
DA","JULIANN","JOHNIE","ELVERA","DELPHIA","CLAIR","CHRISTIANE","CHAROLETTE","CARRI","AUGUSTINE","A
LAVINIA","KUM","KACIE","JACKELINE","HUONG","FELISA","EMELIA","ELEANORA","CYTHIA","CRISTIN","CLYDE"
ORIAN","DENITA","DALLAS","CHI","BILLYE","ALEXANDER","TOMIKA","SHARITA","RANA","NIKOLE","NEOMA","MA
TE","LUCRECIA","KOURTNEY","KATI","JESUS","JESENIA","DIAMOND","CRISTA","AYANA","ALICA","ALIA","VINN
","JANETT","HANNELORE","GLENDORA","GERTRUD","GARNETT","FREEDA","FREDERICA","FLORANCE","FLAVIA","DE
"MANDA","MACIE","LADY","KELLYE","KELLEE","JOSLYN","JASON","INGER","INDIRA","GLINDA","GLENNIS","FER
,"EMELDA","ELENI","DETRA","CLEMMIE","CHERYLL","CHANTELL","CATHEY","ARNITA","ARLA","ANGLE","ANGELIC
NDYCE","ARLENA","AMMIE","YANG","WILLETTE","VANITA","TUYET","TINY","SYREETA","SILVA","SCOTT","RONAL
E","ALLINE","YUKO","VELLA","TRANG","TOWANDA","TESHA","SHERLYN","NARCISA","MIGUELINA","MERI","MAYBE
"GENNIE","FRANCIE","FLORETTA","EXIE","EDDA","DREMA","DELPHA","BEV","BARBAR","ASSUNTA","ARDELL","AN
,"MARIKO","MARGERT","LORIS","LIZZETTE","LEISHA","KAILA","KA","JOANNIE","JERRICA","JENE","JANNET","
NE","BECKI","ARLETHA","ARGELIA","ARA","ALITA","YULANDA","YON","YESSENIA","TOBI","TASIA","SYLVIE","
NNE","SHALANDA","SERITA","RESSIE","REFUGIA","PAZ","OLENE","NA","MERRILL","MARGHERITA","MANDIE","MA
OI","TARRA","TARI","TAMMERA","SHAKIA","SADYE","RUTHANNE","ROCHEL","RIVKA","PURA","NENITA","NATISHA
","XENIA","WAVA","VANETTA","TORRIE","TASHINA","TANDY","TAMBRA","TAMA","STEPANIE","SHILA","SHAUNTA"
TTIE","KERA","KENDAL","KEMBERLY","KANISHA","JULENE","JULE","JOSHUA","JOHANNE","JEFFREY","JAMEE","H
","BRUNA","BRITTANEY","BRANDE","BILLI","BAO","ANTONETTA","ANGLA","ANGELYN","ANALISA","ALANE","WENO
```

Python Methodologies for Data Sciences

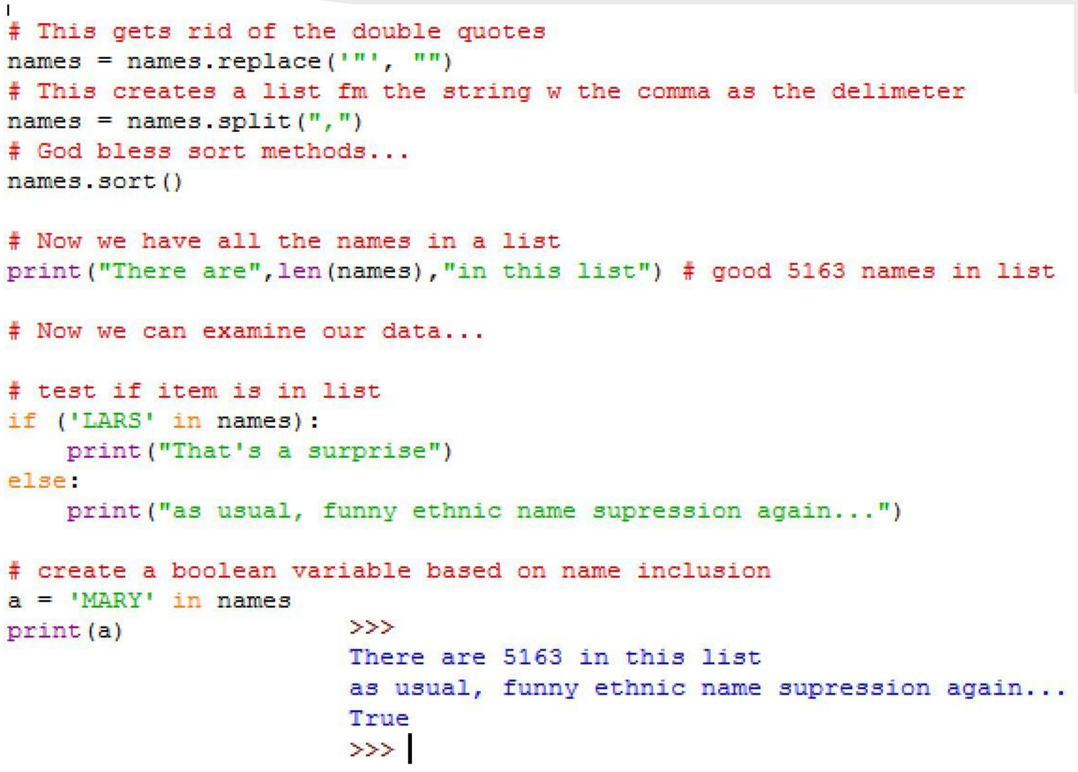
OK, I read this out of a file and we're not there yet so just know that I have a string variable, names, that is one long string with all the names in it.

I strip out the “”s, do a split so I turn my string into a list, and then I sort the list (didn't really have to do this, just showing you).

Now I have a list I can test with the ‘in’ keyword.

I test for my name (and get skunked, no surprise there) and I also create a boolean variable. Because “MARY” is in the list this condition is TRUE.

This program runs in less than a second.

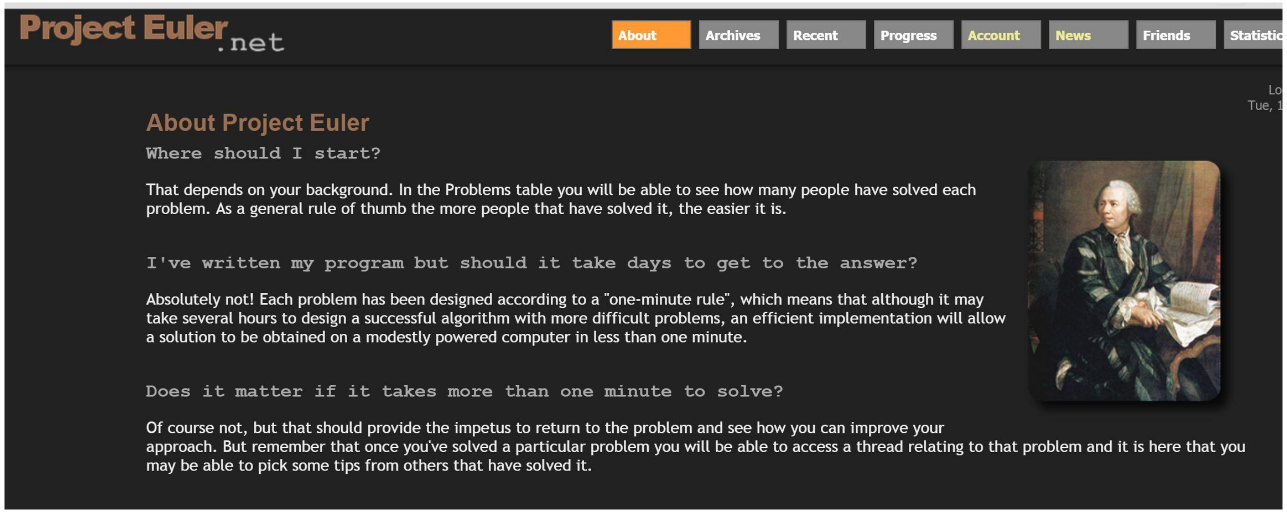


```
# This gets rid of the double quotes
names = names.replace('"', "")
# This creates a list fm the string w the comma as the delimeter
names = names.split(",")
# God bless sort methods...
names.sort()

# Now we have all the names in a list
print("There are",len(names),"in this list") # good 5163 names in list

# Now we can examine our data...

# test if item is in list
if ('LARS' in names):
    print("That's a surprise")
else:
    print("as usual, funny ethnic name supression again...")

# create a boolean variable based on name inclusion
a = 'MARY' in names
print(a)
```

```
>>>
There are 5163 in this list
as usual, funny ethnic name supression again...
True
>>> 
```

Python Methodologies for Data Sciences

Now we have the tools to work on some real problems! Go to projecteuler.net. If you want to, create a userid and login (you don't have to). With the tools you have now (variables, data types, arithmetic, conditions, lists, and loops) you have enough to tackle some of these problems



About Project Euler

Where should I start?

That depends on your background. In the Problems table you will be able to see how many people have solved each problem. As a general rule of thumb the more people that have solved it, the easier it is.

I've written my program but should it take days to get to the answer?

Absolutely not! Each problem has been designed according to a "one-minute rule", which means that although it may take several hours to design a successful algorithm with more difficult problems, an efficient implementation will allow a solution to be obtained on a modestly powered computer in less than one minute.

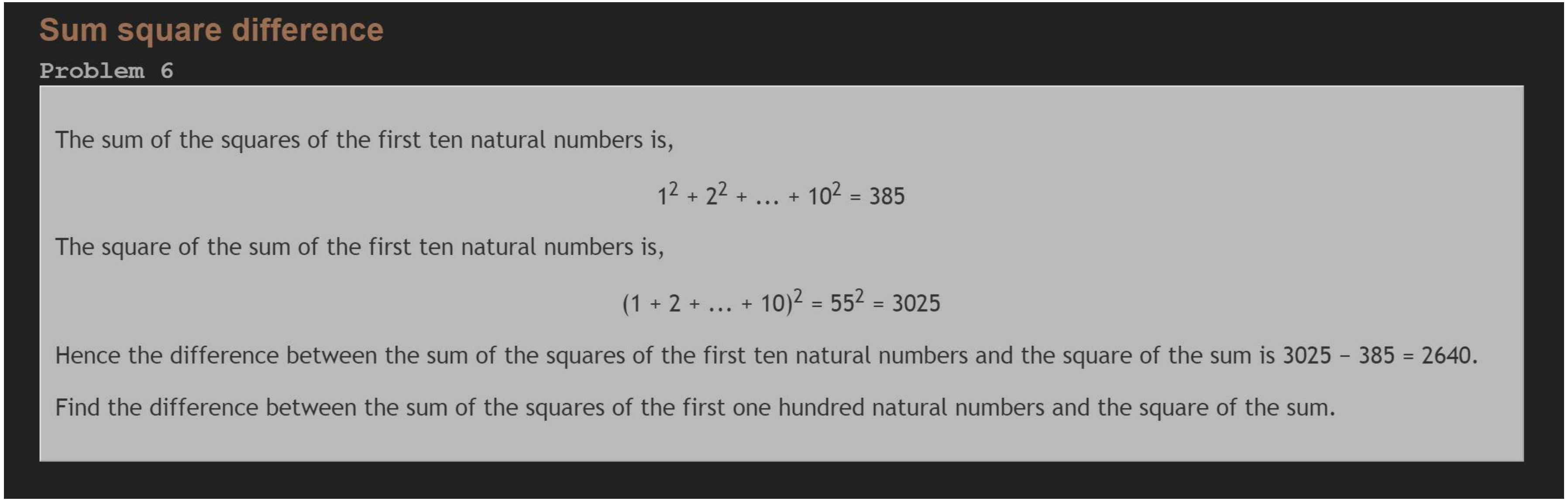
Does it matter if it takes more than one minute to solve?

Of course not, but that should provide the impetus to return to the problem and see how you can improve your approach. But remember that once you've solved a particular problem you will be able to access a thread relating to that problem and it is here that you may be able to pick some tips from others that have solved it.

Some of them are hairy, I know. We're going to look at an easy one.

Python Methodologies for Data Sciences

This is Euler #6. Give it a go. I will do a video and solve it next week, but try yourself first.



Sum square difference

Problem 6

The sum of the squares of the first ten natural numbers is,

$$1^2 + 2^2 + \ldots + 10^2 = 385$$

The square of the sum of the first ten natural numbers is,

$$(1 + 2 + \ldots + 10)^2 = 55^2 = 3025$$

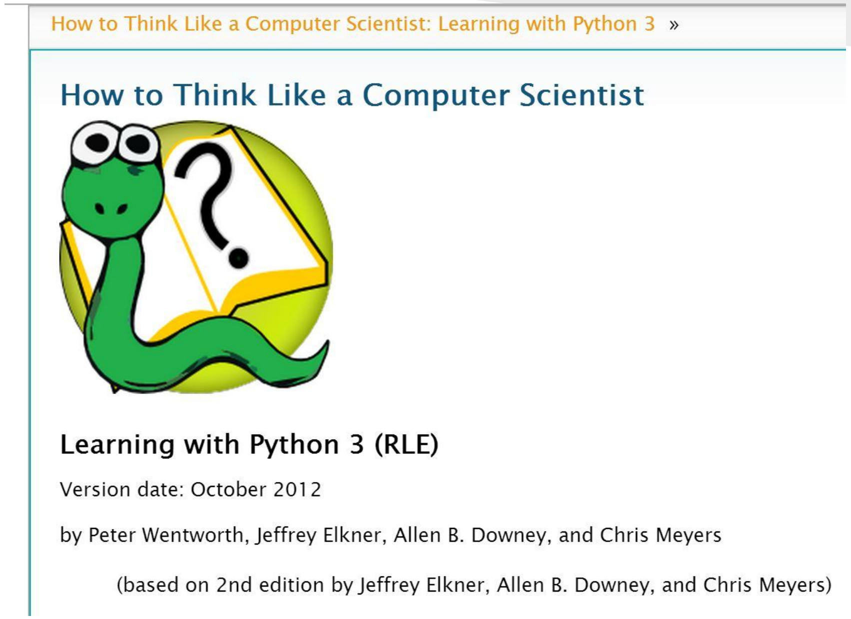
Hence the difference between the sum of the squares of the first ten natural numbers and the square of the sum is $3025 - 385 = 2640$.

Find the difference between the sum of the squares of the first one hundred natural numbers and the square of the sum.

Python Methodologies for Data Sciences

Other cool resources

I put a few free Python books in the resources section of SAKAI last week. Check them out! There will be no readings or assignments from them, I just want you to be aware of all the Python resources there are on the web.



How to Think Like a Computer Scientist: Learning with Python 3 »

How to Think Like a Computer Scientist

Learning with Python 3 (RLE)

Version date: October 2012

by Peter Wentworth, Jeffrey Elkner, Allen B. Downey, and Chris Meyers

(based on 2nd edition by Jeffrey Elkner, Allen B. Downey, and Chris Meyers)

http://openbookproject.net/thinkcs/python/english3e/

Dictionaries

For those of you keeping score at home, I know the syllabus mentioned that we would be covering dictionaries in this unit. I thought better of it as I was doing the slides.

I gave you sequences and then I gave you loops. Dictionaries are a data structure that is not a sequence per se and they do not lend themselves to being iterated over so I decided it made no sense to introduce the topic as you were wrapping your novice Python heads around lists and loops.

I will get to Dictionaries in Unit three or four. They are important and will be covered, I'm just going to do it later on.

Nuff said.

Python Methodologies for Data Sciences

A few simple exercises

1. Print all the multiples of 3 up until 60.

2. Print the numbers from 30 to 1 backwards

3. Create a game that asks the player to guess a secret word, revealing one letter of the word for every turn. At the end of the game tell the user how many guesses they made.

4. Print the nth term of the Fibonacci sequence where n is a number entered by the user. See pg 262 of Zelle for better description, this is exercise 1.