# Programming with Python

# Module 4

In this module, we will look at:

* The for loop
* Strings
* Tuples (Arrays)
* Working with Tuples

## The for loop

We have seen the while loop, but another type of loop available in Python is the "for" loop.

*"Python’s for statement iterates over the items of any sequence (a list or a string), in the order that they appear in the sequence" -- Python help files*

Here is an example:

import sys #used to reference the System code module

#Declare my variables

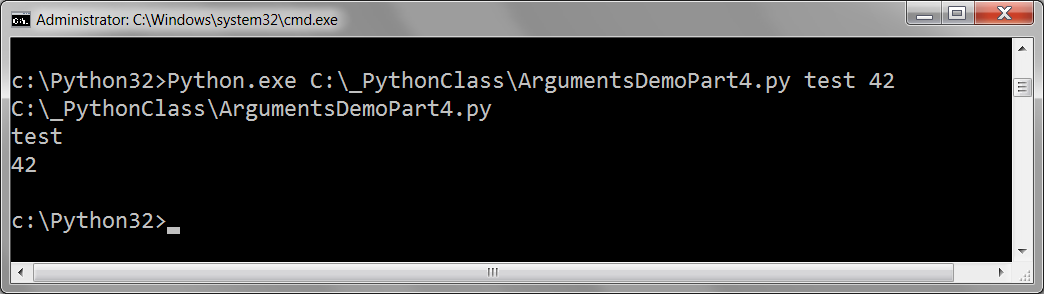
item = None

#Process the arguments

if(len(sys.argv) > 0):

**for** item **in** sys.argv:

print(item) #Output the values



## Strings

*"Strings contain Unicode characters. Their literals are written in single or double quotes: 'xyzzy', "frobozz". -- Python help files*

*"string objects are sequences of characters (represented by strings of length 1)" -- Python help files*

Strings are **one of six "sequence types"**: ***strings, range, tuples, and lists***

(also: byte sequences and byte arrays objects, but we will not cover those).

strData = **"**123**"**

rngData = **range**(1,4)

tplData = **(**1,2,3**)**

lstData = **[**1,2,3**]**

print("items in a string")

**for** item **in** strData:

print(item)

print("items in a range")

**for** item **in** rngData:

print(item)

print("items in a tuple")

**for** item **in** tplData:

print(item)

print("items in a list")

**for** item **in** lstData:

print(item)

**Note:** Though these six sequences are all very like each other, they are also a bit different from each other. We will explore these differences in the next couple of chapters.

### String indexes

You can identify an individual character in a string with a **subscript**(index)

strData = '1,2,3'

print(strData**[0]**)

print(strData**[1]**)

print(*strData***[2]**)

print(strData**[3]**)

print(strData**[4]**)

**NOTE**: The numbering system is **zero** based.

### String Methods

Here are some from the Python help file:

**“**

*str.****capitalize()***

*Return a copy of the string with its first character capitalized and the rest lowercased.*

*str.****lower()***

*Return a copy of the string converted to lowercase.*

*str.****upper()***

*Return a copy of the string converted to uppercase.*

*str.****strip([chars])***

*Return a copy of the string with the leading and trailing characters removed.*

*str.****replace****(old, new[, count])*

*Return a copy of the string with all occurrences of substring old replaced by new. If the optional argument count is given, only the first count occurrences are replaced.*

*str.****isalpha()***

*Return true if all characters in the string are alphabetic and there is at least one character* ***(in the string),*** *false otherwise.*

*str.****split([sep[, maxsplit]])***

*Return a list of the words in the string, using sep as the delimiter string. If maxsplit is given, at most maxsplit splits are done (thus, the list will have at most maxsplit+1 elements). If maxsplit is not specified, then there is no limit on the number of splits (all possible splits are made).” – Python Help Files*

Here are some examples of how these functions are used:

strData = **"test data"**

print(strData.**capitalize**())

print(strData.**lower**())

print(strData.**upper**())

print(strData.**replace**(" ", "--"))

strData = **" test data "**

print(strData.**strip**())

print("------ isalpha?")

strData = "**abc**"

print(strData.**isalpha**())#true

strData = "**123**"

print(strData.**isalpha**())#false

strData = "**abc123**"

print(strData.**isalpha**())#false

print("------ Split ")

strData = '1,2,3'

lstData = strData.split(',')

print(lstData[0])

print(lstData[1])

print(lstData[2])

## Substitution Parameters

Most languages, including Python allow the use of Substitution Parameters in a string.

*str.format()*

[*http://www.python-course.eu/python3\_formatted\_output.php*](http://www.python-course.eu/python3_formatted_output.php)

Here is an example:

print("The 1st parameter is {0} and the 2nd is {1}".**format**(47,11))

### Weird String Functions

Python has a few string functions that do not act as you would expect them to. This is because they are created to work with rare Unicode escape characters like “\u00B2”, where the ‘u’ indicates Unicode. Here are three examples:

*"str.****isdecimal()***

*Return true if all characters in the string are decimal characters and there is at least one character* ***(in the string),*** *false otherwise.*

*str.****isdigit()***

*Return true if all characters in the string are digits and there is at least one character* ***(in the string),*** *false otherwise.*

*str.****isnumeric()***

*Return true if all characters in the string are numeric characters, and there is at least one character* ***(in the string),*** *false otherwise." – Python Help Files*

Here are some examples:

strData = "123.4" *#Note whole numbers work, but decimal ones don't!*

print(strData.**isdecimal**())# false is a root of 10(base of 10)

print(strData.**isdigit**())# false

print(strData.**isnumeric**())# false

### The Immutable String

Once strings are made they cannot change their values, even if it looks like they can.

strData = '1,2,3'

print(strData)

strData = '123'

print(strData) #Actually a new string!

strData = '123' + '4'

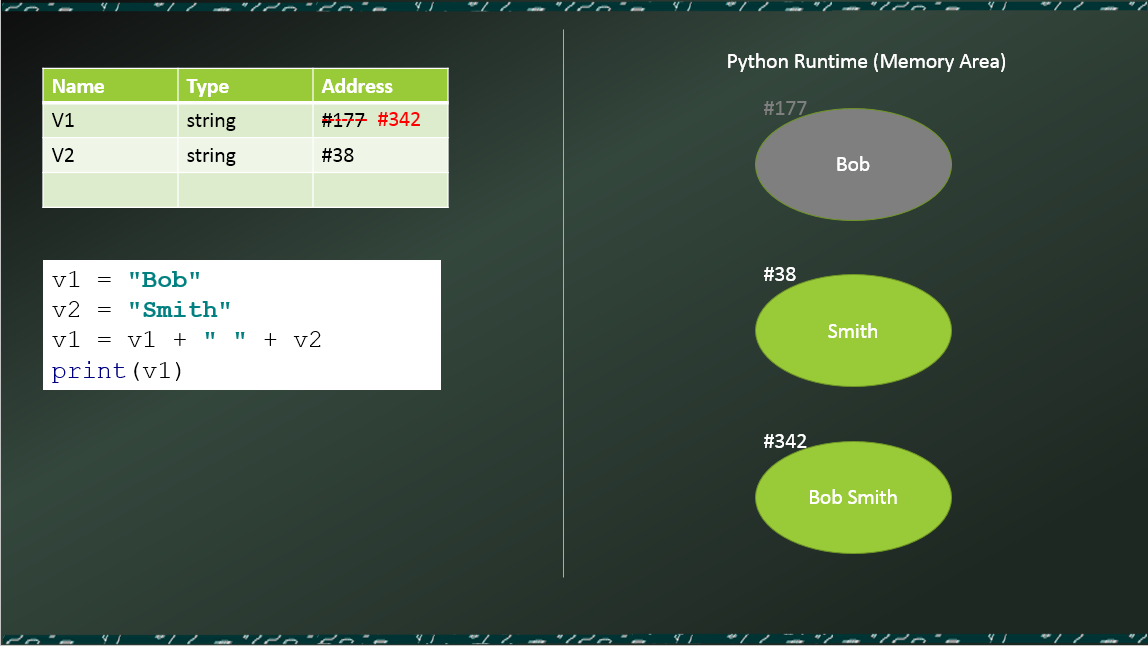
print(strData) #Actually another new string!

**#**strData[2] = 5 #You **cannot** assign a value after a string is made

print(strData)

**#**strData[5] = 5 #You **cannot** add more elements either

print(strData)



### Slicing strings

You can slice out portions of a string with the range operator.

strData = 'ABC123'

#Elements 0, 1, 2

print(strData**[2:4]**) **# What will this return???**

## Tuples (Arrays)

Tuples are a way of storing many pieces of data under one a single variable name. Tuples are a collection of all type of data, while strings are only a collection of characters.

print("ID, Name, Phone, Current Customer?")

tplData = **(**1,"Bob Smith","555-3445", True**)**

print(tplData[0])

print(tplData[1])

print(tplData[2])

print(tplData[3])

print("The whole tuple:", tplData)

You do not need the use parenthesis to create a tuple, but it is a good idea since it identifies this visually in your code. *Creating a tuple without the parenthesis is called* *tuple packing*.

tplData01 = **(**1,"Bob Smith","555-3445", True**)**

tplData02 = 2,"Sue Jones","555-7757", False

print("ID, Name, Phone, Current Customer?")

print("Employee01:", tplData01)

print("Employee02:", tplData02)

Tuples and loops

**For** loops are often used with tuples to extract data. Here is an example:

tplData01 = (1,"Bob Smith","555-3445", True)

tplData02 = 2,"Sue Jones","555-7757", False

print("ID, Name, Phone, Current Customer?")

print("--------------------------------------")

**for** item in tplData01:

print(item) #Output the values

print("-------------------------------")

**for** item in tplData02:

print(item) #Output the values

### Un-packing tuples

You can also un-pack a tuple into a sequence of variables with the following code.

tplData01 = (1,"Bob Smith","555-3445", True)

tplData02 = 2,"Sue Jones","555-7757", False

#Unpack tuple into singleton variables

print("--------------------------------------")

print("ID, Name, Phone, Current Customer?")

print("--------------------------------------")

intCustId, strName, strPhone, blnCurrent = **tplData01** #Unpack tuple

print("Customer Id:", intCustId)

print("Name:", strName)

print("Phone:", strPhone)

print("Current Customer:", blnCurrent)

print("--------------------------------------")

#Is the same as...

intCustId, strName, strPhone, blnCurrent = tplData02

print("Customer Id:", tplData02[0])

print("Name:", tplData02[1])

print("Phone:", tplData02[2])

print("Current Customer:", tplData02[3])

### Tuple with Multiple Dimensions

Tuples can contain other "nested" tuples. This means that there is an outer tuple and an inner tuple. Here is an example:

tplDataRow01 = (1,"Bob Smith","555-3445", True)

tplDataRow02 = 2,"Sue Jones","555-7757", False

tplDataTableA = (tplDataRow01, tplDataRow02)

print(tplDataTableA)

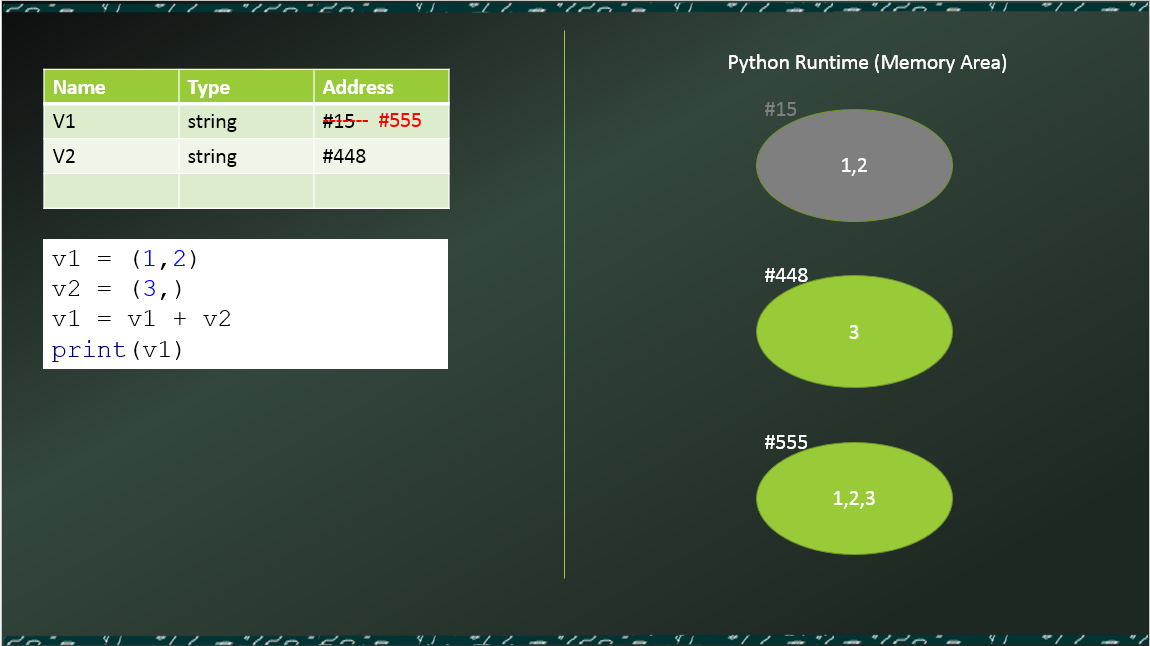
You can think of this as creating a table of values.

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 'Bob Smith' | '555-3445' | True |
| 2 | 'Sue Jones' | '555-7757' | False |

The inner tuple (set of columns) is considered a first *Dimension* (or *Rank*) the outer tuple (set of rows) is considered the second *Dimension* (or *Rank*).

### Tuples are Immutable

Once a tuple is created you cannot change the contents of the tuple. However, it seems that you can since you can add use concatenation as you do with Strings.



## LAB 4-1: Working with Tuples

Let's create some python code that works with tuples.

1. Create a script that uses a two-dimensional tuple to hold the following data:

|  |  |  |
| --- | --- | --- |
| Id | Name | Email |
| 1 | Bob Smith | BSmith@Hotmail.com |
| 2 | Sue Jones | SueJ@Yahoo.com |
| 3 | Joe James | JoeJames@Gmail.com |

1. Create a for loop that prints out each row of the data as follows:

ID, Name, Email

--------------------------------------

(1, 'Bob Smith', 'BSmith@Hotmail.com')

--------------------------------------

(2, 'Sue Jones', 'SueJ@Yahoo.com')

--------------------------------------

(3, 'Joe James', 'JoeJames@Gmail.com')

1. Now added a nested for-loop to extract the individual elements (columns) data and display it as follows.

ID

Name

Email

--------------------------------------

1

Bob Smith

BSmith@Hotmail.com

--------------------------------------

2

Sue Jones

SueJ@Yahoo.com

--------------------------------------

3

Joe James

JoeJames@Gmail.com

**Hint**: Use Pseudo code to start, and remember to try the built-in python function print(<Tuple>).

## Working with tuples

Let's look at two common ways you work with Tuples.

### The In operator

The *in* operator searches though a sequence and return a Boolean if found:

tplDataRow01 = (1,"Bob Smith","555-3445", True)

tplDataRow02 = (2,"Sue Jones","555-7757", False)

if "Bob Smith" in tplDataRow02:

print("Customer found")

else:

print("Customer Not found")

### Adding on to a tuple

You can add more data to a tuple, just like a string, but it creates another tuple behind the scenes. (YOU NEED THIS FOR YOUR HOMEWORK!!!)

tplData = ("1","2","3")

print(tplData)

strMoreData = ("4"), *#Note that the comma at the end makes this a tuple!!!*

tplData += strMoreData

print(tplData) #Actually another new tuple(just like strings!)

#tplData += '4' #Error: Can only concatenate tuple (not "str") to tuple

#tplData = tplData + '4' #Error: Can only concatenate tuple (not "str") to tuple

#tplData[2] = 4 #Error: Cannot assign a value after a string is made

#tplData[3] = 4 #Error: Cannot add on more elements after a string is made

## LAB 4-2: The In Operator

1. Create an script that uses a two-dimensional tuple to hold the following data: (this is the same data as the last lab’s)

|  |  |  |
| --- | --- | --- |
| Id | Name | Email |
| 1 | Bob Smith | BSmith@Hotmail.com |
| 2 | Sue Jones | SueJ@Yahoo.com |
| 3 | Joe James | JoeJames@Gmail.com |

1. Add code that searches for customers by name and returns a customer Id. You code should look similar to this:

strCustomerName = input("Enter a Customer Name: ")

blnFound = False

for Row in tplDataTableA:

if strCustomerName in tplDataTableA:

print(strCustomerName + " is Customer ID: " + str(Row[0]) )

blnFound = True

if blnFound == False:

print("Customer Not Found")

1. **EXTRA**: Add a loop that lets the user keep trying.

## Lists

Lists are very similar to Tuples, but are more flexible. They include more functions and make changing the data in the list easier. We will look at Lists in again in module 05.

v1 = [1,2]  
v2 = [3] *#Note that no comma is needed this time*v1 = v1 + v2  
print(v1)

# Review Chapter 4

"FOR LOOPS, STRINGS, AND TUPLES: THE WORD JUMBLE GAME

You’ve seen how variables are a great way to access information, but as your

programs grow in size and complexity, so can the number of your variables.

Keeping track of all of them can become a lot of work. Therefore, in

this chapter, you’ll learn about the idea of sequences and meet a new type, called

the tuple, which let’s you organize and manipulate information in ordered groups.

You’ll also see how a type you’ve already encountered, the string, is really a

sequence too. You’ll learn about a new kind of loop that’s built just for working

with sequences. Specifically, you’ll learn how to do the following:

• Construct for loops to move through a sequence

• Use the range() function to create a sequence of numbers

• Treat strings as sequences

• Use tuples to harness the power of sequences

• Use sequence functions and operators

• Index and slice sequences"

(Chapter 4, Python Programming for the Absolute Beginner, Third Edition, Dawson)