## 1.2.1.2: Use Interactive Interpreter as a Calculator

The three angle brackets (****>>>****) indicate that you are in Python's interactive interpreter. From here, you can do a variety of basic programming tasks including math operations. Table 1 shows the Python syntax to use for the most common math operators.

****Table 1: Math Operators and Python Syntax Comparison****

|  |  |  |
| --- | --- | --- |
| ****Operation**** | ****Math**** | ****Syntax**** |
| Addition | a+b | a+b |
| Subtraction | a-b | a-b |
| Multiplication | axb | a\*b |
| Division | a÷b | a/b |
| Exponents | a^b | a\*\*b |

Example 1 shows a few math operations using the Python syntax.

****Example 1: Basic Arithmetic Operations****

|  |
| --- |
| $$ ****python3**** Python 3.5.2 (default, Aug 18 2017, 17:48:00)  [GCC 5.4.0 20160609] on linux Type "help", "copyright", "credits" or "license" for more information. >>> ****2+3**** 5 >>> ****10-4**** 6 >>> ****2\*4**** 8 >>> ****20/5**** 4 >>> ****3\*\*2**** 9 |

Python uses the standard order of operations commonly known as ****PEMDAS****. Mathematical expressions are evaluated in the following order.

****P****arentheses  
****E****xponents  
****M****ultiplication and ****D****ivision  
****A****ddition and ****S****ubtraction

Try entering an expression with a complex order of operations in the interactive interpreter.

## 1.2.1.3: Use Interpreter to Print a String

A string is any sequence of characters such as letters, numbers, symbols, or punctuation marks. The interactive interpreter will directly output text that you enter as a string as long as you enclose the string in either single quotes (') or double quotes (").

The ****print**** command can be used in a script to output a string, as shown in Example 1.

****Example 1: Printing a String****

|  |
| --- |
| >>> ****"Hello World!"**** 'Hello World!' >>> ****'Hello World!'**** 'Hello World!' >>> ****print("Hello World!")**** Hello World! |

## 1.2.2.1: Basic Data Types

In programming, data types are a classification which tells the interpreter how the programmer intends to use the data. For example, the interpreter needs to know if the data the programmer entered is a number or a string. Although there are several different data types, we will focus only on the following:

* ****Integer**** - used to specify whole numbers (no decimals), such as 1, 2, 3, and so on. If an integer is entered with a decimal, the interpreter ignores the decimal. For example, 3.75 is interpreted as 3.
* ****Float**** - used to specify numbers that need a decimal value, such as 3.14159.
* ****String**** - any sequence of characters such as letters, numbers, symbols, or punctuation marks.
* ****Boolean**** - any data type that has a value of either True or False.

Use the ****type()**** function to determine the data type, as shown in Example 1.

****Example 1: Determine the Data Type****

|  |
| --- |
| >>> ****type(98)**** <class 'int'> >>> ****type(98.6)**** <class 'float'> >>> ****type("Hi!")**** <class 'str'> >>> ****type(True)**** <class 'bool'> |

## 1.2.2.2: Boolean Operators

The Boolean data type makes use of the operators shown in Table 1.

****Table 1: Boolean Operators****

|  |  |
| --- | --- |
| ****Operator**** | ****Meaning**** |
| ****>**** | Greater than |
| ****<**** | Less than |
| ****==**** | Equal to |
| ****!=**** | Not equal to |
| ****>=**** | Greater than or equal to |
| ****<=**** | Less than or equal to |

 In the IDLE shell, try out the different Boolean operators, as shown in Example 1.

****Example 1: Boolean Operators Comparisons****

|  |
| --- |
| >>> ****1<2**** True >>> ****1>2**** False >>> ****1==1**** True >>> ****1!=1**** False >>> ****1>=1**** True >>> ****1<=1**** True |

## 1.2.2.3: Creating and Using a Variable

The Boolean operator for determining whether two values are equal is the double equal sign (****==****). A single equal sign (****=****) is used to assign a value to a variable. The variable can then be used in other commands to recall value, as shown in Example 1.

****Example 1: Assigning a Value to Variable and Using It in a Command****

|  |
| --- |
| >>> ****x=3**** >>> ****x\*5**** 15 >>> ****"Cisco"\*x**** 'CiscoCiscoCisco' |

## 1.2.2.4: Concatenate Multiple String Variables

Concatenation is the process of combining multiple strings into one string. For example, the concatenation of "****foot****" and "****ball****" is "****football****". In Example 1, four variables are concatenated together in a ****print()**** statement with the plus sign (****+****). Notice that the space variable was defined for use as white space between the words.

Example 1: Concatenating String Variables

|  |
| --- |
| >>> ****str1="Cisco"**** >>> ****str2="Networking"**** >>> ****str3="Academy"**** >>> ****space=" "**** >>> ****print(str1+space+str2+space+str3)**** Cisco Networking Academy >>> |

****Challenge****: Try writing a ****print()**** statement with a space between the words without using a variable to create the space.

## 1.2.2.5: Converting Data Types

Concatenation does not work for joining different data types, as shown in Example 1.

****Example 1: Error Message When Concatenating Different Data Types****

|  |
| --- |
| >>> ****x=3**** >>> ****print("This value of X is " + x)**** Traceback (most recent call last):  File "<pyshell#27>", line 1, in <module>  print("This value of X is " + x) TypeError: Can't convert 'int' object to str implicitly |

Use the ****str()**** functionto convert the value of a variable to a string, as shown in Example 2.

****Example 2: Converting a Variable's Value to a String****

|  |
| --- |
| >>> ****x=3**** >>> ****print("The value of x is " + x)**** Traceback (most recent call last):  File "<pyshell#27>", line 1, in <module>  print("This value of X is " + x) TypeError: Can't convert 'int' object to str implicitly >>> ****print("The value of x is " + str(x))**** The value of x is 3 >>> ****type(x)**** <class 'int'> |

Notice that the data type for the variable x is still an integer. To convert the data type, reassign the variable to the new data type, as shown in Example 3.

****Example 3: Converting the Data Type of a Variable****

|  |
| --- |
| >>> ****x=3**** >>> ****print("The value of x is " + x)**** Traceback (most recent call last):  File "<pyshell#27>", line 1, in <module>  print("This value of X is " + x) TypeError: Can't convert 'int' object to str implicitly >>> ****print("The value of x is " + str(x))**** The value of x is 3 >>> ****type(x)**** <class 'int'> >>> ****x=str(x)**** >>> ****type(x)**** <class ‘str'> |

You may want to display a float to a specific number of decimal places instead of the full number. To do this, use the     
****"{:.2f}".format**** function, as shown in Example 4.

****Example 4: Format a Float to 2 Decimal Places****

|  |
| --- |
| >>> ****num = 22/7**** >>> ****print(num)**** 3.142857142857143 >>> ****print("{:.2f}".format(num))**** 3.14 >>> |

## 1.2.3.1: Lists

In programming, a list variable is used to store multiple pieces of ordered information. Example 1 shows how to create a list variable called ****hostnames****.

* Create a list using brackets ****[ ]****and enclosing each item in the list with quotes.
* Separate the items with a comma.
* Use the ****type()****command to verify the data type.
* Use the ****len()****command return the number of items in a list.
* Call the list variable name to display it’s contents.

****Example 1: Creating a List****

|  |
| --- |
| >>> ****hostnames=["R1","R2","R3","S1","S2"]**** >>> ****type(hostnames)**** <class 'list'> >>> ****len(hostnames)**** 5 >>> ****hostnames**** ['R1', 'R2', 'R3', 'S1', 'S2'] |

Also called an array in some programming environments, an item in a list can be referenced and manipulated using its index, as shown in Example 2.

* The first item in a list is indexed as zero, the second is indexed as one, and so on.
* The last item can be referenced with index ****[-1]****.
* Replace an item by assigning a new value to the index.
* Use the ****del****command to remove an item from a list.

****Example 2: Referencing and Manipulating a List****

|  |
| --- |
| >>> ****hostnames[0]**** 'R1' >>> ****hostnames[-1]**** 'S2' >>> ****hostnames[0]="RTR1"**** >>> ****hostnames**** ['RTR1', 'R2', 'R3', 'S1', 'S2'] >>> ****del hostnames[3]**** >>> ****hostnames**** ['RTR1', 'R2', 'R3', 'S2'] >>> |

## 1.2.3.2: Dictionaries

Dictionaries are unordered lists of objects. Each object contains a key/value pair. In Example 1, a dictionary ****ipAddress**** is created with three key/value pairs to specify the IP address values for three routers.

* Create a dictionary using the braces ****{ }****.
* Each dictionary entry includes a key and a value.
* Separate a key and its value with a colon.
* Use quotes for keys and values that are strings.

****Example 1: Creating a Dictionary****

|  |
| --- |
| >>> ****ipAddress = {"R1":"10.1.1.1","R2":"10.2.2.1","R3":"10.3.3.1"}**** >>> ****type(ipAddress)**** <class 'dict'> |

Unlike lists, objectives inside a dictionary cannot be referenced by their sequence number. Instead, you reference a dictionary object using its key, as shown in Example 2.

* The key is enclosed with brackets ****[ ]****.
* Keys that are strings can be referenced using single or double quotes, as shown for R1 and S1 in Example 2.
* Add a key/value pair by setting the new key equal to a value.
* Use a *****key* in *dictionary***** statement to verify if a key exists in the dictionary.

****Example 2: Referencing and Modifying a Dictionary****

|  |
| --- |
| >>> ****ipAddress**** {'R1': '10.1.1.1', 'R2': '10.2.2.1', 'R3': '10.3.3.1'} >>> ****ipAddress['R1']**** '10.1.1.1’ >>> ****ipAddress["S1"]="10.1.1.10"**** >>> ****ipAddress**** {'R1': '10.1.1.1', 'R2': '10.2.2.1', 'R3': '10.3.3.1', 'S1': '10.1.1.10'} >>> ****"R3" in ipAddress**** True >>> |

 Values in a key/value pair can be any other data type including lists and dictionaries. For example, if R3 has more than one IP address, how would you represent that inside the ****ipAddress**** dictionary? Create a list for the value of the R3 key, as shown in Example 3.

****Example 3: Embedding a List in a Dictionary****

|  |
| --- |
| >>> ****ipAddress["R3"]=["10.3.3.1","10.3.3.2","10.3.3.3"]**** >>> ****ipAddress**** {'S1': '10.1.1.10', 'R2': '10.2.2.1', 'R1': '10.1.1.1', 'R3': ['10.3.3.1', '10.3.3.2', '10.3.3.3']} >>> |

## 1.2.3.3: Activity - Troubleshoot List and Dictionary Code

Open the script ****02\_list-dicts.py****, which is also shown in Example 1. This script creates a list of the BRICS countries (****B****razil, ****R****ussia, ****I****ndia, ****C****hina, and ****S****outh Africa). It then creates a dictionary for each country's capital(s). A list is used for South Africa's three capitals. The script is then supposed to print the country list, capital dictionary, and the 2nd listed capital for the value of South Africa. However, there are errors in the script.

****Note****: Click [here](https://lms.netacad.com/pluginfile.php/72006439/mod_page/content/7/Ch1_Files.zip" \t "/Users/dbnull/Documents\\x/_blank) to download the Chapter 1 files if you did not get them when completing the [Lab - PC Setup for Workshop](https://lms.netacad.com/pluginfile.php/72006439/mod_page/content/7/1.0.1.2 Lab - PC Setup for Workshop.pdf" \t "/Users/dbnull/Documents\\x/_blank).

****Example 1: Script for 02\_list-dicts.py****

|  |
| --- |
| country=["Brazil","Russia","India","China","South Africa"] capitals={"Brazil":"Brasilia","Russia":"Moscow","India":"New Delhi",  "China":"Beijing","South Africa":["Pretoria","Cape Town","Bloemfontein"]} print("country") print("capitals") print(capitals["South Africa"[1]]) |

Troubleshoot the code until the script runs without errors, as shown in Example 2.

****Example 2: Output for Script 02\_list-dicts.py****

|  |
| --- |
| ======= RESTART: /home/user/Documents/GitHub/02\_list-dicts.py ========== ['Brazil', 'Russia', 'India', 'China', 'South Africa'] {'South Africa': ['Pretoria', 'Cape Town', 'Bloemfontein'], 'India': 'New Delhi', 'Russia': 'Moscow', 'China': 'Beijing', 'Brazil': 'Brasilia'} Cape Town >>> |

## 1.2.5.1: If/Else Function

In programming, conditional statements check if something is true and then carry out instructions based on the evaluation. If the evaluation is false, different instructions are carried out.

Example 1 demonstrates the if/else function in a simple script:

****Example 1: The If/Else Function****

|  |
| --- |
| nativeVLAN = 1 dataVLAN = 100 if nativeVLAN == dataVLAN:  print("The native VLAN and the data VLAN are the same.") else:  print("The native VLAN and the data VLAN are different.") |

Create this script for your files:

* Open a blank script and save it as ****04\_if-vlan.py****.
* Copy the script in Example 1.
* Run the script and troubleshoot any errors. Your output should look like Example 2.
* Change the values to test the ****else**** print statement and run the script again.

****Example 2: Output for First Run of Script 04\_if-vlan.py****

|  |
| --- |
| ========== RESTART: /home/user/Documents/GitHub/04\_if-VLAN.py ========== The native VLAN and the data VLAN are different. >>> |

Modify the variables so that ****nativeVLAN**** and ****dataVLAN**** have the same value. Save and run the script again. Your output should look like Example 3.

****Example 3: Output for Second Run of Script 04\_if-vlan.py****

|  |
| --- |
| ========== RESTART: /home/user/Documents/GitHub/04\_if-VLAN.py ========== The native VLAN and the data VLAN are the same. >>> |

## 1.2.5.2: If/Elif/Else Function

What if we have more than two conditional statements to consider? In this case, we can use ****elif**** statements in the middle of the ****if/else**** function. An ****elif**** statement is evaluated if the ****if**** statement is false and before the ****else**** statement. You can have as many ****elif**** statements as you would like. However, the first one matched will be executed and none of the remaining ****elif**** statements will be checked. Nor will the ****else**** statement.

The script in Example 1 asks the user to input the number of an IPv4 ACL and then checks whether that number is a standard IPv4 ACL, extended IPv4 ACL, or neither standard or extended IPv4 ACL.

****Note****: The data type for the input function is changed from the default string to an integer so that the ****if**** and ****elif**** evaluations will work.

****Example 1: The If/Elif/Else Function****

|  |
| --- |
| aclNum = int(input("What is the IPv4 ACL number? ")) if aclNum >= 1 and aclNum <= 99:  print("This is a standard IPv4 ACL.") elif aclNum >=100 and aclNum <= 199:  print("This is a extended IPv4 ACL.") else:  print("This is not a standard or extended IPv4 ACL.") |

 Create this script for your files:

* Open a blank script and save it as ****05\_if-acl.py****.
* Copy the script in Example 1.
* Run multiple times to test each statement. Troubleshoot any errors. Your output should look like Example 2.

****Example 2: Output for Script 05\_if-acl.py****

|  |
| --- |
| =========== RESTART: /home/user/Documents/GitHub/05\_if-ACL.py =========== What is the IPv4 ACL number? ****10**** This is a standard IPv4 ACL. >>> =========== RESTART: /home/user/Documents/GitHub/05\_if-ACL.py =========== What is the IPv4 ACL number? ****100**** This is a extended IPv4 ACL. >>> =========== RESTART: /home/user/Documents/GitHub/05\_if-ACL.py =========== What is the IPv4 ACL number? ****2000**** This is not a standard or extended IPv4 ACL. >>> |

## 1.2.5.3: For Loop

The Python ****for****command is used to loop or iterate through the elements in a list or perform an operation on a series of values. Example 1 demonstrates how a for loop can be used to print the elements in a list. The variable name ****item**** is arbitrary and can be anything the programmer chooses.

****Example 1: For Loop****

|  |
| --- |
| >>> ****devices=["R1","R2","R3","S1","S2"]**** >>> ****for item in devices:****  ****print(item)****    R1 R2 R3 S1 S2 >>> |

What if you only want to list the items that begin with the letter R? An if statement can be embedded in a for loop to achieve this, as shown in Example 2.

****Example 2: For Loop with Embedded If****

|  |
| --- |
| >>> for item in devices:  ****if "R" in item:****  ****print(item)****   R1 R2 R3 >>> |

You can also use a combination of the for loop and if statement to create a new list. Example 3 shows how to use the ****append()**** method to create a new list called switches.

****Example 3: Use a For Loop to Create a New List****

|  |
| --- |
| >>> ****switches=[]**** >>> ****for item in devices:****  ****if "S" in item:****  ****switches.append(item)****    >>> ****switches**** ['S1', 'S2'] >>> |

## 1.2.5.4: Create a While Loop

Instead of running a block of code once, as in an if statement, you can use a while loop. A while loop keeps executing a code block as long as a boolean expression remains true. This can cause a program to run endlessly if you do not make sure your script includes a condition for the while loop to stop. While loops will not stop until the boolean expression evaluates as false.

In Example 1, the while loop counts from 1 to a number entered by the user.

* Open a blank script and save it as ****06\_while-loop.py****.
* Create a program with a while loop that counts to a user’s supplied number.
  + Convert the string to an integer: ****x = int(x)****.
  + Set a variable to start the count: ****y = 1****.
  + ****While y <= x****, print the value of y and increment y by 1.

****Example 1: Counting Using a While Loop****

|  |
| --- |
| x=input("Enter a number to count to: ") x=int(x) y=1 while y<=x:  print(y)  y=y+1 |

Example 2 shows the output for the script when the user enters 10.

****Example 2: Output for First Run of 06\_while-loop.py****

|  |
| --- |
| ========== RESTART: /home/user/Documents/GitHub/07\_while-loop.py ========== Enter a number to count to: ****10**** 1 2 3 4 5 6 7 8 9 10 >>> |

Instead of using ****while y <= x****, we can modify the while loop to use a Boolean check and break to stop the loop when the check evaluates as false, as shown in Example 3.

****Example 3: Modify the While Loop to Use Break****

|  |
| --- |
| x=input("Enter a number to count to: ") x=int(x) y=1 while True:  print(y)  y=y+1  if y>x:  break |

 Modify the ****06\_while-loop.py**** script as shown in Example 3 and run it. You should not have any errors and your output should look similar to Example 2.

## 1.2.5.5: Use a While Loop to Check for User Quit Command

What if we want the program to run as many times as the user wants until the user quits the program? To do this, we can embed the program in a while loop that checks if the user enters a quit command, such as ****q**** or ****quit****.

In your ****06\_while-loop.py**** script, make the following changes, as shown in Example 1.

* Add another while loop to the beginning of the script which will check for a quit command.
* Add an if function to the while loop to check for ****q**** or ****quit****.

****Example 1: Add a While Loop to Check for Quit Command****

|  |
| --- |
| while True:  x=input("Enter a number to count to: ")  if x == 'q' or x == 'quit':  break   x=int(x)  y=1  while True:  print(y)  y=y+1  if y>x:  break |

 Your output should look similar to Example 2 in which the user entered two different values before quitting the program.

****Example 2: Output for 06\_while-loop.py****

|  |
| --- |
| ========= RESTART: /home/user/Documents/GitHub/06\_while-loop\_sol.py ========= Enter a number to count to: ****3**** 1 2 3 Enter a number to count to: ****5**** 1 2 3 4 5 Enter a number to count to: ****quit**** >>> |

## 1.2.6.1: Read an External File

In addition to user input, you can access a database, another computer program, or a file to provide input to your program. The ****open()**** function can be used to access a file using the following syntax:

|  |
| --- |
| ****open****(name, [mode]) |

The name parameter is the name of the file to be opened. If the file is in a different directory than your script, you will also need to provide path information. For our purposes, we are only interested in three mode parameters:

* ****r**** - read the file (default mode if mode is omitted).
* ****w**** - write over the file, replacing the content of the file.
* ****a**** - append to the file.

Complete the following steps to read and print a file:

1. Open a blank script and save it as ****07\_file-access.py****.
2. Create a script to read and print the content of a file, as shown in Example 1.
3. The ****devices.txt**** file should be in the same directory as your script.
4. After printing the contents of the file, use the ****close()**** function to remove it from the computer's memory.

****Note****: The contents of the file are set to a variable named ****file****. However, that variable can be called anything the programmer chooses.

****Example 1: Read and Print a File****

|  |
| --- |
| file=open("devices.txt","r") for item in file:  print(item) file.close() |

Run the script and troubleshoot, if necessary. Your output should look like Example 2.

****Example 2: Output for the Script 07\_file-access.py****

|  |
| --- |
| ========== RESTART: /home/user/Documents/GitHub/07\_file-access.py ========== Cisco 819 Router  Cisco 881 Router  Cisco 888 Router  Cisco 1100 Router  Cisco 4321 Router  Cisco 4331 Router  Cisco 4351 Router  Cisco 2960 Catalyst Switch  Cisco 3850 Catalyst Switch  Cisco 7700 Nexus Switch  Cisco Meraki MS220-8 Cloud Managed Switch  Cisco Meraki MX64W Security Appliance  Cisco Meraki MX84 Security Appliance  Csico Meraki MC74 VoIP Phone  Cisco 3860 Catalyst Switch  >>> |

****Note****: Click [here](https://lms.netacad.com/pluginfile.php/72006491/mod_page/content/7/Ch1_Files.zip" \t "/Users/dbnull/Documents\\x/_blank) to download the Chapter 1 files if you did not get them when completing the [Lab - PC Setup for Workshop](https://lms.netacad.com/pluginfile.php/72006491/mod_page/content/7/1.0.1.2 Lab - PC Setup for Workshop.pdf" \t "/Users/dbnull/Documents\\x/_blank).

## 1.2.6.2: Remove Blank Lines from the Output

You may have noticed that Python added a blank line after each entry. We can remove this blank line using the ****strip()**** method. Edit your ****07\_file-access.py**** script as shown in Example 1.

****Example 1: Stripping the Blank Line****

|  |
| --- |
| file=open("devices.txt","r") for item in file:  item=item.strip()  print(item) file.close() |

|  |
| --- |
| ========== RESTART: /home/user/Documents/GitHub/07\_file-access.py ========== Cisco 819 Router Cisco 881 Router Cisco 888 Router Cisco 1100 Router Cisco 4321 Router Cisco 4331 Router Cisco 4351 Router Cisco 2960 Catalyst Switch Cisco 3850 Catalyst Switch Cisco 7700 Nexus Switch Cisco Meraki MS220-8 Cloud Managed Switch Cisco Meraki MX64W Security Appliance Cisco Meraki MX84 Security Appliance Csico Meraki MC74 VoIP Phone Cisco 3860 Catalyst Switch >>> |

Run and, if necessary, troubleshoot your script. Your output should look like Example 2.

****Example 2: Output for the Script 07\_file-access.py****

## 1.2.6.3: Copy File Content Into a List Variable

Most of the time when programmers access an external resource such as a database or file, they are wanting to copy that content into a local variable that can then be referenced and manipulated without impacting the original resource.

The ****devices.txt**** file is a list of Cisco devices that can easily be copied into a Python list using the following steps:

1. Create an empty list.
2. Use the ****append**** parameter to copy file content to the new list.

Modify your ****07\_file-access.py**** as shown in Example 1.

****Example 1: Copying a File into a List****

|  |
| --- |
| devices=[] file=open("devices.txt","r") for item in file:  item=item.strip()  devices.append(item) file.close() print(devices) |

Run and, if necessary, troubleshoot your script. Your output should look like Example 2.

|  |
| --- |
| ========== RESTART: /home/user/Documents/GitHub/07\_file-access.py ========== ['Cisco 819 Router', 'Cisco 881 Router', 'Cisco 888 Router', 'Cisco 1100 Router', 'Cisco 4321 Router', 'Cisco 4331 Router', 'Cisco 4351 Router', 'Cisco 2960 Catalyst Switch', 'Cisco 3850 Catalyst Switch', 'Cisco 7700 Nexus Switch', 'Cisco Meraki MS220-8 Cloud Managed Switch', 'Cisco Meraki MX64W Security Appliance', 'Cisco Meraki MX84 Security Appliance', 'Csico Meraki MC74 VoIP Phone', 'Cisco 3860 Catalyst Switch'] |

****Example 2: Output for the Script 07\_file-access.py****

## 1.2.6.4: Activity – Create a Script to Allow User to Add Devices

What if you want to add more devices to the ****devices.txt**** file? You can open the file in append mode and then ask the user to provide the name of the new devices. Complete the following steps to create a script:

1. Open a new file and save it as ****07\_file-access\_actvity.py****.
2. For the ****open()**** function use the mode ****a****, which will allow you to append a item to the ****devices.txt**** file.
3. Inside a ****while True:****loop, embed an ****input()**** function command that asks the user for the new device.
4. Set the value of the user's input to a variable named ****newItem****.
5. Use an if statement that breaks the loop if the user types ****exit****and prints the statement****"All done!"****.
6. Use the command ****file.write(newItem + “\n”)**** to add the new user provided device.

Run and troubleshoot your script until you get output similar to Example 1.

****Example 1: Output for Script 07\_file-access\_activity.py****

|  |
| --- |
| ==== RESTART: /home/user/Documents/GitHub/07\_file-access\_sol\_activity.py ==== Enter device name: ****Cisco 1941 Router**** Enter device name: ****Cisco 2950 Catalyst Switch**** Enter device name: ****exit**** All done! >>> |

## 1.3.1.1: Application Programming Interface (API)

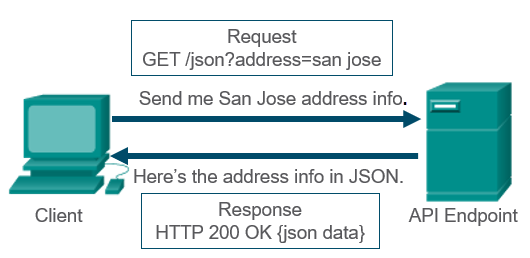
|  |
| --- |
| An API allows one piece of software talk to another. An API is analogous to a power outlet. Without a power outlet, what would you have to do to power your laptop?   * Open the wall * Unsheath wires * Splice wires together * Understand all the wires in the wall   An API defines how a programmer can write one piece of software to talk to an existing application’s features or even build entirely new applications. |

## 1.3.1.3: Web Services Interface using HTTP

|  |
| --- |
| Web browsers use Hypertext Transfer Protocol (HTTP) to request (GET) a web page. If successfully requested (HTTP status code 200), web servers respond to GET requests with a Hypertext Markup Language (HTML) coded web page, as shown in Figure 1. |
| ****Figure 1: HTTP Process for Requesting a Web Page****  http-services |

## 1.3.1.4: RESTful API using HTTP

|  |
| --- |
| Representation State Transfer (REST) APIs use HTTP to interface with RESTful services. In Figure 1, the HTTP request asks for JavaScript Object Notation (JSON) formatted data. If the request is successfully constructed according to the API documentation, the server will respond with JSON data. |
| ****Figure 1: RESTful Request for JSON Data**** |



## 1.3.1.5: Anatomy of a RESTful Request

|  |
| --- |
| Figure 1 shows the anatomy of a RESTful request to the MapQuest directions API. The full parameters are not shown.  ****Figure 1: RESTful Request Anatomy**** |
|  |
| The parts of the request are as follows:   * ****API Server****: The URL for the server that answers REST requests * ****Resources****: Specifies the API that is being requested * ****Format****: Usually JSON or XML * ****Parameters****: Specifies what data is being requested |

## 1.3.2.1: JSON Response Data

1.3.2.1: JSON Response Data

|  |
| --- |
| JavaScript Object Notation (JSON) is a format for storing and exchanging text between a server and a client application. It is easy to read and write. JSON is a very popular format that web services and APIs use to provide public data because it is easy to parse and can be used with most modern programming languages including Python.  Click [here](http://api.open-notify.org/iss/v1/?lat=30.26715&lon=-97.74306" \t "/Users/dbnull/Documents\\x/_blank) to see publicly available information for the International Space Station (ISS). The example calls the ISS Pass Predictions API and returns the time a duration for the next 5 passes of the ISS over Austin, Texas.  ****Note****: Search the Internet to learn more about the ISS APIs.  You should get JSON response data similar to what is shown in Figure 1, which looks a lot like a collection of Python lists and dictionaries. This similarity makes it easy to convert JSON objects and arrays into Python data structures.  ****Figure 1: ISS JSON Response Data for Austin, Texas**** |
| IMG_256 |
| ****Note****: To be able to expand and collapse the JSON data, make sure you should have the [JSONView](https://chrome.google.com/webstore/detail/jsonview/chklaanhfefbnpoihckbnefhakgolnmc" \t "/Users/dbnull/Documents\\x/_blank) extension installed in your Chrome browser.  In JSON, objects are indicated by curly braces and resemble Python dictionaries. JSON arrays are held in square brackets and resemble Python lists. To simplify the discussion, we will refer to the JSON structures using the familiar Python terms. Keep in mind however, that JSON data is usually converted to Python data structures before it is used by Python programs.  Collapse the JSON data, as shown in Figure 2. At the highest level, the JSON data is contained in a dictionary, as indicated by the curly braces ****{ }****. Notice that ****message****, ****request****, and ****response****are key/value pairs in the dictionary. The ****request**** element contains another dictionary and the ****response**** element contains a list, as indicated by the square brackets ****[ ]****.  ****Figure 2: ISS API Dictionary Root Elements**** |
| IMG_257 |
| Expand the dictionary inside the ****response****element to view each of the key/value pairs in the dictionary (Figure 3).  ****Figure 3: Elements in the Response List**** |
| IMG_258 |
| To be able to parse JSON data, it is important to understand this feature of embedding dictionaries and lists within each other. For example, to extract the rise time when the ISS passes over Austin, Texas, you would need to specify three levels of embedded lists and dictionaries, as shown in the script in Example 1.  ****Example 1: Script to Retrieve and Extract the Rise Time when ISS Passes over Austin**** |
| |  | | --- | | import urllib.parse  import requests  import time  url = 'http://api.open-notify.org/iss/v1/?lat=30.26715&lon=-97.74306'  json\_data = requests.get(url).json()epoch = json\_data['response'][0]['risetime']  next\_pass = time.strftime("%a, %d %b %Y %H:%M:%S %Z", time.localtime(epoch))  print("The next ISS pass will be: " + (next\_pass)) | |
| Run the above code in your Python instance to test it. You should get output that gives the timestamp for the ISS pass over Austin, Texas. Change the coordinates to reflect your location. Make sure you use the correct positive and/or negative designations for your chosen latitude and longitude. |

## 1.3.2.2: XML Response Data

|  |
| --- |
| E****x****tensible ****M****arkup ****L****anguage (XML) extends the functionality of HTML allowing web programmers to construct custom tags. To get XML data instead of JSON from the MapQuest API, replace ****json**** with ****xml**** in the URL structure as shown here:  https://www.mapquestapi.com/directions/v2/route?****outFormat=xml****&key=****KEY****&from=Chicago&to=Boston  ****Note****: The above URL does not work. You would first need to replace ****KEY**** with a valid key to call the MapQuest API. You will get a key in the next topic.  You can see in the XML response in Figure 1 that the embedded data has the same basic structure as the previous JSON response. However, in XML, it is not easy to distinguish between dictionaries and lists. Instead, each element is enclosed by an opening and closing tag, such as ****<results>**** and ****</results>****. Parsing XML data is more difficult than parsing JSON. Therefore, we will be using JSON data exclusively in this course.  ****Figure 1: XML Response Data for Direction from Chicago to Boston**** |
|  |

## 1.3.3.1: Demonstration - MapQuest Directions API Application

In this topic, you will create an application that retrieves JSON data from the MapQuest Directions API, parses the data, and formats it for output to the user. Click [here](https://lms.netacad.com/pluginfile.php/72006583/mod_page/content/9/1.3.3.1 Lab - Parsing JSON with a Python Application.pdf?time=1603822852909" \t "/Users/dbnull/Documents\\x/_blank) to download a PDF of the full lab.

You will use the GET Route request from the MapQuest Directions API. Review the GET Route Directions API documentation here:

[https://developer.mapquest.com/documentation/directions-api/route/get/](https://developer.mapquest.com/documentation/directions-api/route/get/" \t "/Users/dbnull/Documents\\x/_blank)

Example 1 shows output from a functioning version of the program. Notice that the user entered ask for directions for two different trips.

To build this application, you will complete the following objectives:

* Obtain a MapQuest API Key.
* Import necessary modules.
* Create API request variables and construct a URL.
* Add user input functionality.
* Add a quit feature so that the user can end the application.
* Display trip information for time, distance, and fuel usage.
* Iterate through the JSON data to extract and output the directions.
* Display error messages for invalid user input.

****Example 1: Output for the MapQuest Directions API Application****

|  |
| --- |
| ========= RESTART: /home/user/Documents/GitHub/08\_parse-json\_sol.py =========  Starting Location: ****Washington****  Destination: ****Baltimore****  URL: https://www.mapquestapi.com/directions/v2/route?key=your\_api\_key&from=Washington&to=Baltimore  API Status: 0 = A successful route call.  Directions from Washington to Baltimore  Trip Duration: 00:49:19  Kilometers: 61.32  Fuel Used (Ltr): 6.24  =============================================  Start out going north on 6th St/US-50 E/US-1 N toward Pennsylvania Ave/US-1 Alt N. (1.28 km)  Turn right onto New York Ave/US-50 E. Continue to follow US-50 E (Crossing into Maryland). (7.51 km)  Take the Balt-Wash Parkway exit on the left toward Baltimore. (0.88 km)  Merge onto MD-295 N. (50.38 km)  Turn right onto W Pratt St. (0.86 km)  Turn left onto S Calvert St/MD-2. (0.43 km)  Welcome to BALTIMORE, MD. (0.00 km)  =============================================  Starting Location: ****Moscow****  Destination: ****Beijing****  URL: https://www.mapquestapi.com/directions/v2/route?key=your\_api\_key&from=Moscow&to=Beijing  API Status: 0 = A successful route call.  Directions from Moscow to Beijing  Trip Duration: 84:31:10  Kilometers: 7826.83  Fuel Used (Ltr): 793.20  =============================================  Start out going west on Кремлёвская набережная/Kremlin Embankment. (0.37 km)  Turn slight right onto ramp. (0.15 km)  Turn slight right onto Боровицкая площадь. (0.23 km)  [output omitted]  Turn left onto 广场东侧路/E. Guangchang Rd. (0.82 km)  广场东侧路/E. Guangchang Rd becomes 东长安街/E. Chang'an Str. (0.19 km)  Welcome to BEIJING. (0.00 km)  =============================================  Starting Location: ****Washington****  Destination: ****Beijing****  URL: https://www.mapquestapi.com/directions/v2/route?key=your\_api\_key&from=WashingtonTurn+right+onto+%E5%89%8D%E9%97%A8%E8%A5%BF%E5%A4%A7%E8%A1%97%2FQianmen+West+Street.+%281.01+km%29&to=Beijing  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Staus Code: 402; Invalid user inputs for one or both locations.  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Starting Location: ****Washington****  Destination: ****Balt****  URL: https://www.mapquestapi.com/directions/v2/route?key=your\_api\_key&from=Washington&to=Balt  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Staus Code: 602; Refer to:  https://developer.mapquest.com/documentation/directions-api/status-codes  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Starting Location: ****Washington****  Destination:  URL: https://www.mapquestapi.com/directions/v2/route?key=your\_api\_key&from=Washington&to=  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Staus Code: 611; Refer to:  https://developer.mapquest.com/documentation/directions-api/status-codes  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Starting Location: ****q****  >>> |

****Note****: Your instructor may demonstrate the program and show you the script to create it. In addition, your instructor may allow you to have the solution script, ****08\_parse-json\_sol.py****. However, you will create this script step by step in this topic of the course.

## 1.3.3.1: Demonstration - MapQuest Directions API Application

In this topic, you will create an application that retrieves JSON data from the MapQuest Directions API, parses the data, and formats it for output to the user. Click [here](https://lms.netacad.com/pluginfile.php/72006583/mod_page/content/9/1.3.3.1 Lab - Parsing JSON with a Python Application.pdf?time=1603822852909" \t "/Users/dbnull/Documents\\x/_blank) to download a PDF of the full lab.

You will use the GET Route request from the MapQuest Directions API. Review the GET Route Directions API documentation here:

[https://developer.mapquest.com/documentation/directions-api/route/get/](https://developer.mapquest.com/documentation/directions-api/route/get/" \t "/Users/dbnull/Documents\\x/_blank)

Example 1 shows output from a functioning version of the program. Notice that the user entered ask for directions for two different trips.

To build this application, you will complete the following objectives:

* Obtain a MapQuest API Key.
* Import necessary modules.
* Create API request variables and construct a URL.
* Add user input functionality.
* Add a quit feature so that the user can end the application.
* Display trip information for time, distance, and fuel usage.
* Iterate through the JSON data to extract and output the directions.
* Display error messages for invalid user input.

****Example 1: Output for the MapQuest Directions API Application****

|  |
| --- |
| ========= RESTART: /home/user/Documents/GitHub/08\_parse-json\_sol.py =========  Starting Location: ****Washington****  Destination: ****Baltimore****  URL: https://www.mapquestapi.com/directions/v2/route?key=your\_api\_key&from=Washington&to=Baltimore  API Status: 0 = A successful route call.  Directions from Washington to Baltimore  Trip Duration: 00:49:19  Kilometers: 61.32  Fuel Used (Ltr): 6.24  =============================================  Start out going north on 6th St/US-50 E/US-1 N toward Pennsylvania Ave/US-1 Alt N. (1.28 km)  Turn right onto New York Ave/US-50 E. Continue to follow US-50 E (Crossing into Maryland). (7.51 km)  Take the Balt-Wash Parkway exit on the left toward Baltimore. (0.88 km)  Merge onto MD-295 N. (50.38 km)  Turn right onto W Pratt St. (0.86 km)  Turn left onto S Calvert St/MD-2. (0.43 km)  Welcome to BALTIMORE, MD. (0.00 km)  =============================================  Starting Location: ****Moscow****  Destination: ****Beijing****  URL: https://www.mapquestapi.com/directions/v2/route?key=your\_api\_key&from=Moscow&to=Beijing  API Status: 0 = A successful route call.  Directions from Moscow to Beijing  Trip Duration: 84:31:10  Kilometers: 7826.83  Fuel Used (Ltr): 793.20  =============================================  Start out going west on Кремлёвская набережная/Kremlin Embankment. (0.37 km)  Turn slight right onto ramp. (0.15 km)  Turn slight right onto Боровицкая площадь. (0.23 km)  [output omitted]  Turn left onto 广场东侧路/E. Guangchang Rd. (0.82 km)  广场东侧路/E. Guangchang Rd becomes 东长安街/E. Chang'an Str. (0.19 km)  Welcome to BEIJING. (0.00 km)  =============================================  Starting Location: ****Washington****  Destination: ****Beijing****  URL: https://www.mapquestapi.com/directions/v2/route?key=your\_api\_key&from=WashingtonTurn+right+onto+%E5%89%8D%E9%97%A8%E8%A5%BF%E5%A4%A7%E8%A1%97%2FQianmen+West+Street.+%281.01+km%29&to=Beijing  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Staus Code: 402; Invalid user inputs for one or both locations.  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Starting Location: ****Washington****  Destination: ****Balt****  URL: https://www.mapquestapi.com/directions/v2/route?key=your\_api\_key&from=Washington&to=Balt  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Staus Code: 602; Refer to:  https://developer.mapquest.com/documentation/directions-api/status-codes  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Starting Location: ****Washington****  Destination:  URL: https://www.mapquestapi.com/directions/v2/route?key=your\_api\_key&from=Washington&to=  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Staus Code: 611; Refer to:  https://developer.mapquest.com/documentation/directions-api/status-codes  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Starting Location: ****q****  >>> |

****Note****: Your instructor may demonstrate the program and show you the script to create it. In addition, your instructor may allow you to have the solution script, ****08\_parse-json\_sol.py****. However, you will create this script step by step in this topic of the course.

## 1.3.3.2: Authenticating a RESTful Request

Before building the application, you will need to obtain a key from MapQuest developer site, which you will do in the next step. Previously, you used no authentication to access the ISS Pass Predictions API. However, many APIs require some form of authentication.

Authenticating a RESTful request is done in one of four ways:

* ****None****: The API resource is public and anybody can place the request. This is the method you have used up to this point.
* ****Basic HTTP****: The username and password are passed to the server in an encoded string. This method is less common than token and OAuth authentication.
* ****Token****: A secret key generally retrieved from the Web API developer portal.
* ****Open Authorization (OAuth)****: An open standard for retrieving an access token from an Identity Provider. The token is then passed with each API call.

For your purposes, you will use token authentication. To add a token to your RESTful request, simply add ****key=your\_api\_key****, as shown in Figure 1.

****Figure 1: Adding a Token to a RESTful Request****



## 1.3.3.4: Importing Modules

To begin our script for parsing JSON data, you will need to import two modules from the Python library: ****requests**** and ****urllib.parse****. The ****request**** module provides functions for retrieving JSON data from a URL. The ****urllib.parse**** module provides a variety of functions that will enable us to parse and manipulate the JSON data you receive from a request to a URL.

1. Open a blank script file and save it as ****08\_parse-json1.py****.
2. Import the ****urllib.parse**** and ****requests**** modules, as shown in Example 1.

****Example 1: Importing Modules****

|  |
| --- |
| import urllib.parse import requests |

## 1.3.3.5: Create Variables for API Request

The first step in creating our API request is to construct the URL that our program will use to make the call. Initially, the URL will be the combination of the following variables, as shown in Example 1:

* ****main\_api**** - the main URL that you are accessing
* ****orig**** - the parameter to specify your point of origin
* ****dest****- the parameter to specify your destination
* ****key**** - the MapQuest API key you retrieved from the developer website.

Add the variables in Example 1 to your ****08\_json-parse1.py**** script, replacing ****"your\_api\_key"**** with your API key.

****Example 1: Create the Variable to Construct the URL****

|  |
| --- |
| main\_api = "https://www.mapquestapi.com/directions/v2/route?" orig = "Washington" dest = "Baltimaore" key = "your\_api\_key" |

Those variables are combined to create the ****url**** variable, as shown in Example 2. Use the ****urlencode**** method to properly format the address value. This function builds the parameters part of the URL and converts possible special characters in the address value (e.g. space into “+” and a comma into “%2C”).

Add the ****url**** variable in Example 2 to your ****08\_json-parse1.py**** script.

****Example 2: Create the Variable for the URL****

|  |
| --- |
| url = main\_api + urllib.parse.urlencode({"key": key, "from":orig, "to":dest}) |

Now you are ready to make the request by creating the ****json\_data**** variable, as shown in Example 3. The variable makes use of the ****get**** method of the ****requests**** module and specifies JSON as the requested format. The ****print**** statement is used to test that the request was successful.

Add the ****json\_data**** variable and ****print**** statement in Example 3 to your ****08\_json-parse1.py**** script.

****Example 3: Create the JSON Request Variable****

|  |
| --- |
| json\_data = requests.get(url).json() print(json\_data) |

## 1.3.3.6: Activity - Test the URL Request

Run your ****08\_json-parse1.py**** script and verify it works. Troubleshoot your code, if necessary. Although your output might be slightly different, you should get a JSON response similar what is shown in Example 1.

****Example 1: Output for Testing the URL Request****

|  |
| --- |
| ========== RESTART: /home/user/08\_parse-json1.py ========== {'route': {'distance': 38.089, {'route': {'distance': 38.089, 'hasHighway': True, 'hasUnpaved': False, 'hasAccessRestriction': False, 'options': {'mustAvoidLinkIds': [], 'maxWalkingDistance': -1, 'manmaps': 'true', 'urbanAvoidFactor': -1, 'stateBoundaryDisplay': True, 'cyclingRoadFactor': 1, 'routeType': 'FASTEST', 'countryBoundaryDisplay': True, 'drivingStyle': 2, 'highwayEfficiency': 22, 'narrativeType': 'text', 'routeNumber': 0, 'tryAvoidLinkIds': [], 'generalize': -1, 'returnLinkDirections': False, 'doReverseGeocode': True, 'avoidTripIds': [], 'timeType': 0, 'sideOfStreetDisplay': True, 'filterZoneFactor': -1, 'walkingSpeed': -1, 'useTraffic': False, 'unit': 'M', 'tr  [output omitted]  >>> |

## 1.3.3.7: Print the URL and Check the Status of the JSON Request

Now that you know the JSON request is working, you can add some more functionality to the application.

1. Save your script as ****08\_json-parse2.py****.
2. Delete the ****print(json\_data)**** statement as you no longer need to test that the request is properly formatted.
3. Add the statements below, which will do the following:

* Print the constructed URL so that the user can see the exact request made by the application.
* Parse the JSON data to obtain the ****statuscode**** value
* Print the ****statuscode**** value and a message if the request is successful. The ****\n**** adds a blank line after the message.

****Example 1: Print the URL and Request Status****

|  |
| --- |
| print("URL: " + (url))  json\_data = requests.get(url).json()  json\_status = json\_data["info"]["statuscode"]  if json\_status == 0:  print("API Status: " + str(json\_status) + " = A successful route call.\n") |

## 1.3.3.8: Activity - Test Status and URL Print Commands

Run your ****08\_json-parse2.py**** script and verify it works. Troubleshoot your code, if necessary. You should get output similar to what is shown in Example 1.

****Example 1: Output for Printing the URL and Request Status****

|  |
| --- |
| ========== RESTART: /home/user/08\_parse-json2.py ==========  URL: https://www.mapquestapi.com/directions/v2/route?key=your\_api\_key&from=Washington&to=Baltimore  API Status: 0 = A successful route call.  >>> |

## 1.3.3.9: Add User Input for Address

Up to this point, you have used Washington and Baltimore as the static values for the location variables. However, the application requires that the user input these.

Complete the following steps to update your application:

1. Save your script as ****08\_json-parse3.py****.
2. Delete the current ****orig**** and ****dest****
3. Rewrite the ****orig**** and ****dest**** to be within a while loop in which it requests user input for the starting location and destination. The while loop allows the user to continue to make requests for different directions.
4. Be sure all the remaining code is indented within the while loop, as shown in Example 1.

****Example 1: While Loop for User Input****

|  |
| --- |
| while True:  orig = input("Starting Location: ")  dest = input("Destination: ")  url = main\_api + urllib.parse.urlencode({"key": key, "from":orig, "to":dest})  print("URL: " + (url))  json\_data = requests.get(url).json()  json\_status = json\_data["info"]["statuscode"]  if json\_status == 0:  print("API Status: " + str(json\_status) + " = A successful route call.\n") |

## 1.3.3.10: Activity - Test User Input

Run your ****08\_json-parse3.py**** script and verify it works. Troubleshoot your code, if necessary. You should get output similar to what is shown in Example 1.

You will add quit functionality later. For now, enter ****Ctrl+C**** to quit the program.

****Example 1: Output for Testing User Input for the Address****

|  |
| --- |
| ========== RESTART: /home/user/08\_parse-json3.py ========== Starting Location: ****Washington****  Destination: ****Baltimore****  URL: https://www.mapquestapi.com/directions/v2/route?key=your\_api\_key&from=Washington&to=Baltimore  API Status: 0 = A successful route call.  Starting Location: ****<Ctrl+C>**** |

****Note****: The output has been modified to remove the API key.

## 1.3.3.11: Add Quit Functionality

Instead of entering ****Ctrl+C**** to quit the program, you will add the ability for the user to enter ****q**** or ****quit**** as keywords to quit the program.

Complete the following steps to update your application:

1. Save your script as ****08\_json-parse4.py****.
2. Add an if statement after the address variable to check if the user enters ****q**** or ****quit****, as shown in Example 1.

****Example 1: Add an If Statement to Check for Quit Commands****

|  |
| --- |
| while True:  orig = input("Starting Location: ")  if orig == "quit" or orig == "q":  break  dest = input("Destination: ")  if dest == "quit" or dest == "q":  break |

## 1.3.3.12: Activity - Test Quit Functionality

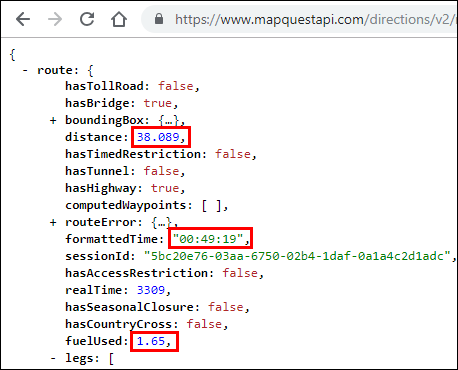
Run your ****08\_json-parse4.py**** script twice to verify that both ****quit**** and ****q**** will end the application. Troubleshoot your code, if necessary. You should get output similar to what is shown in Example 1.

****Example 1: Output for Testing Quit Functionality****

|  |
| --- |
| Starting Location: ****q****  >>>  Starting Location: ****quit****  >>>  Starting Location: ****Washington****  Destination: ****q****  >>>  Starting Location: ****Washington****  Destination: ****quit****  >>> |

## 1.3.3.13: Parse and Display Trip Data

|  |
| --- |
| Copy your URL into your web browser. If you collapse all the JSON data, you will see that there are two root dictionaries: ****route and info****.  Expand the ****route****dictionary and investigate the rich data. There are values to indicate whether the route has toll roads, bridges, tunnels, highways, closures, or crosses into other countries. You should also see values for distance, the total time the trip will take, and fuel usage, as highlighted below. To parse and display this, specify the ****route**** dictionary and select key/value pair you want to print. |



|  |
| --- |
| Complete the following steps to print this trip data   1. Save your script as ****08\_json-parse5.py****. 2. Below the API status print command, add print statements that display the from and to locations, as well as the ****formattedTime****, ****distance****, and ****fuelUsed**** 3. Add a print statement that will display a double line before the next request for a starting location as shown below. |
| |  | | --- | | if json\_status == 0:  print("API Status: " + str(json\_status) + " = A successful route call.\n")  print("Directions from " + (orig) + " to " + (dest))  print("Trip Duration: " + (json\_data["route"]["formattedTime"]))  print("Miles: " + str(json\_data["route"]["distance"]))  print("Fuel Used (Gal): " + str(json\_data["route"]["fuelUsed"])) | |
| Run ****08\_json-parse5.py**** to see the following output: |
| |  | | --- | | Starting Location: ****Washington****  Destination: ****Baltimore****  URL: https://www.mapquestapi.com/directions/v2/route?to=Baltimore&key=Your\_api\_key&from=Washington  API Status: 0 = A successful route call.  Directions from Washington to Baltimore  Trip Duration: 00:49:19  Miles: 38.089  Fuel Used (Gal): 1.65  =============================================  Starting Location: ****q****  >>> | |
| MapQuest uses the imperial system and there is not a request parameter to change data to the metric system. Therefore, you should probably convert your application to display metric values, as shown below. |
| |  | | --- | | print("Kilometers: " + str((json\_data["route"]["distance"])\*1.61))  print("Fuel Used (Ltr): " + str((json\_data["route"]["fuelUsed"])\*3.78)) | |
| Run the modified ****08\_json-parse5.py**** script to see the following output: |
| |  | | --- | | Starting Location: ****Washington****  Destination: ****Baltimore****  URL: https://www.mapquestapi.com/directions/v2/route?key=Your\_api\_key&to=Baltimore&from=Washington  API Status: 0 = A successful route call.  Directions from Washington to Baltimore  Trip Duration: 00:49:19Kilometers: 61.32329Fuel Used (Ltr): 6.236999999999999  =============================================  Starting Location: ****q****  >>> | |
| Use the ****"{:.2f}".format**** argument to format the float values to 2 decimal places before converting them to string values, as shown below. Each statement should be on one line. |
| |  | | --- | | print("Kilometers: " + str("{:.2f}".format((json\_data["route"]["distance"])\*1.61)))  print("Fuel Used (Ltr): " + str("{:.2f}".format((json\_data["route"]["fuelUsed"])\*3.78))) | |
| Run the modified ****08\_json-parse5.py**** script to see the following output: |
| |  | | --- | | Starting Location: ****Washington****  Destination: ****Baltimore****  URL: https://www.mapquestapi.com/directions/v2/route?key=your\_api\_key&to=Baltimore&from=Washington  API Status: 0 = A successful route call.  Directions from Washington to Baltimore  Trip Duration: 00:49:19  Kilometers: 61.32  Fuel Used (Ltr): 6.24  =============================================  Starting Location: ****q****  >>> |  1.3.3.14: Test the Parsing and Formatting Functionality  |  | | --- | | Run your ****08\_json-parse5.py**** script to verify it works. Troubleshoot your code, if necessary. Make sure you have all the proper opening and closing parentheses. You should get output similar to the following. | | |  | | --- | | Starting Location: ****Washington****  Destination: ****Baltimore****  URL: https://www.mapquestapi.com/directions/v2/route?key=your\_api\_key&to=Baltimore&from=Washington  API Status: 0 = A successful route call.  Directions from Washington to Baltimore  Trip Duration: 00:49:19  Kilometers: 61.32  Fuel Used (Ltr): 6.24  =============================================  Starting Location: ****q****  >>> | |  1.3.3.15: Inspect the Maneuvers JSON Data  |  | | --- | | Now you are ready to display the step-by-step directions from the starting location to the destination. Locate the ****legs****list inside the route dictionary. The ****legs**** list includes one big dictionary with most of the JSON data.  Find the ****maneuvers**** list and collapse each of the seven dictionaries inside, as shown below. |   IMG_256Expand the first dictionary in the ****maneuvers****list. Each dictionary contains a ****narrative**** key with a value, such as “Start out going north...”, as shown below. You need to parse the JSON data to extract the value for the ****narrative**** key to display inside your application.   1.3.3.16: Add a For Loop to Iterate Through the Maneuvers JSON Data  |  | | --- | | Complete the following steps to update your application:   1. Save your script as ****08\_json-parse6.py****. 2. Add a for loop below the second double line print statement. The for loop iterates through each ****maneuvers**** list and does the following:    1. Prints the ****narrative****    2. Converts miles to kilometers with ****\*1.61****.    3. Formats the kilometer value to print only two decimal places with the ****"{:.2f}".format**** 3. Add a print statement that will display a double line before the next request for a starting location as shown below.   ****Note****: The second double line print statement is not indented within the for loop. It therefore is part of the previous if statement that checks the ****statuscode**** parameter. | | |  | | --- | | if json\_status == 0:  print("API Status: " + str(json\_status) + " = A successful route call.\n")  print("Directions from " + (orig) + " to " + (dest))  print("Trip Duration: " + str(json\_data["route"]["formattedTime"]))  print("Kilometers: " + str("{:.2f}".format((json\_data["route"]["distance"])\*1.61)))  print("Fuel Used (Ltr): " + str("{:.2f}".format((json\_data["route"]["fuelUsed"])\*3.78)))  print("=============================================")  for each in json\_data["route"]["legs"][0]["maneuvers"]:  print((each["narrative"]) + " (" + str("{:.2f}".format((each["distance"])\*1.61) + " km)"))  print("=============================================\n") | |  1.3.3.17: Activity - Test JSON Iteration and Application Functionality  |  | | --- | | Run your ****08\_json-parse6.py**** script and verify it works. Troubleshoot your code, if necessary. You should get an output similar to the following: | | |  | | --- | | Starting Location: ****Washington****  Destination: ****Baltimore****  URL: https://www.mapquestapi.com/directions/v2/route?key=Your\_api\_key&to=Baltimore&from=Washington  API Status: 0 = A successful route call.  Directions from Washington to Baltimore  Trip Duration: 00:49:19  Kilometers: 61.32  Fuel Used (Ltr): 6.24  =============================================  Start out going north on 6th St/US-50 E/US-1 N toward Pennsylvania Ave/US-1 Alt N. (1.28 km)  Turn right onto New York Ave/US-50 E. Continue to follow US-50 E (Crossing into Maryland). (7.51 km)  Take the Balt-Wash Parkway exit on the left toward Baltimore. (0.88 km)  Merge onto MD-295 N. (50.38 km)  Turn right onto W Pratt St. (0.86 km)  Turn left onto S Calvert St/MD-2. (0.43 km)  Welcome to BALTIMORE, MD. (0.00 km)  =============================================  Starting Location: ****q****  >>> | |  1.3.3.18: Check for Invalid User Input  |  | | --- | | Now you are ready to add one final feature to your application to report an error when the user enters invalid data. Recall that you started an if loop to make sure the returned ****statuscode**** equals 0 before parsing the JSON data: | | |  | | --- | | json\_status = json\_data["info"]["statuscode"]  if json\_status == 0:  print("API Status: " + str(json\_status) + " = A successful route call.\n") | | |  | | |  | | --- | | Starting Location: ****Washington****  Destination: ****Beijing****  URL: https://www.mapquestapi.com/directions/v2/route?to=Beijing&key=your\_api\_key&from=Washington  Starting Location: ****Washington****  Destination: ****Balt****  URL: https://www.mapquestapi.com/directions/v2/route?to=Balt&key=your\_api\_key&from=Washington  Starting Location: ****Washington****  Destination: ****[no user input]****  URL: https://www.mapquestapi.com/directions/v2/route?to=&key=your\_api\_key&from=Washington  Starting Location: ****q**** | | | Save your script as ****08\_jsont-parse7.py****.  To provide error information when this happens, add ****elif**** and ****else**** statements to your if loop. After the last double line print statement under the ****if json\_status == 0****, add the following ****elif**** and ****else**** statements: | | |  | | --- | | for each in json\_data["route"]["legs"][0]["maneuvers"]:  print((each["narrative"]) + " (" + str("{:.2f}".format((each["distance"])\*1.61) + " km)"))  print("=============================================\n")  elif json\_status == 402:  print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  print("Status Code: " + str(json\_status) + "; Invalid user inputs for one or both locations.")  print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n")  else:  print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  print("Status Code: " + str(json\_status) + "; Refer to:")  print("https://developer.mapquest.com/documentation/directions-api/status-codes")  print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n") | | | The ****elif**** statement prints if the ****statuscode**** value is 402 for an invalid location. The ****else**** statement prints for all other ****statuscode**** values, such as no entry for one or more locations. The else statement ends the if/else loop and returns the application to the while loop. |  1.3.3.19: Activity - Test Full Application Functionality  |  | | --- | | Run your ****08\_json-parse7.py**** script and verify it works. Troubleshoot your code, if necessary. Test all the features of the application. For error testing, you should get output similar to the following: | | |  | | --- | | Starting Location: ****Washington****  Destination: ****Beijing****  URL: https://www.mapquestapi.com/directions/v2/route?from=Washington&to=Beijing&key=your\_api\_key  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Staus Code: 402; Invalid user inputs for one or both locations.  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Starting Location: ****Washington****  Destination: ****Balt****  URL: https://www.mapquestapi.com/directions/v2/route?from=Washington&to=Balt&key=your\_api\_key  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Staus Code: 602; Refer to:  https://developer.mapquest.com/documentation/directions-api/status-codes  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Starting Location: ****Washington****  Destination: ****[no user input]****  URL: https://www.mapquestapi.com/directions/v2/route?from=Washington&to=&key=your\_api\_key  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Staus Code: 611; Refer to:  https://developer.mapquest.com/documentation/directions-api/status-codes  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Starting Location: ****q****  >>> | |  1.3.4.1: Option 1 - Modify the MapQuest API App Use your existing MapQuest Directions API and complete at least one of the following tasks:  ****Primary Skills:****   * Parse different data and change the output   ****Challenge Skills:****   * Add user input for metric or imperial.   ****Master Skills:****   * Build a web interface that users access to interact with the application  1.3.4.2: Option 2 - Create an App for Sunrise and Sunset Create an application that accesses the Sunrise and Sunset API at the following URL:  [https://api.sunrise-sunset.org/json?lat=48.8584&lng=2.2945](https://api.sunrise-sunset.org/json?lat=48.8584&lng=2.2945" \t "/Users/dbnull/Documents\\x/_blank)  Example returns sunrise and sunset times in UTC for Paris, France (Eiffel Tower)  ****Primary Skills:****   * Display the sunrise and sunset times for a give set of latitude and longitude coordinates   ****Challenge Skills:****   * Add a calculation to convert from UTC time to local time. This can be a prompt to the user, "How many hours..."   ****Master Skills:****   * Automate the conversion of time to the local time zone (no user input) (TZ API: https://maps.googleapis.com/maps/api/timezone/json?location=48.8584,2.2945&timestamp=1458000000) * Find an open API that will return coordinates for a user entered zip code  1.3.4.3: Option 3 - Create an App for the ISS Create an application that accesses the API for data about the International Space Station (ISS) at the following URL:  [http://api.open-notify.org/iss-pass.json?lat=40.71&lon=-74](http://api.open-notify.org/iss-pass.json?lat=40.71&lon=-74" \t "/Users/dbnull/Documents\\x/_blank)  API returns the number of times today ISS will pass over a given location (NYC in the example).  API documentation can be found here:  [http://open-notify.org/Open-Notify-API/](http://open-notify.org/Open-Notify-API/" \t "/Users/dbnull/Documents\\x/_blank)  ****Primary Skills:****   * Display the number of passes for today * Print the number of passes and the ISO 8061 time for each pass * Use API documentation to determine what the values mean. For example, what does duration:476 mean? Then display that information. For example: print("The duration is 476 ..???)   ****Challenge Skills:****   * Use the API documentation to do one or more of the following * Display the coordinates of the current location of ISS * Display the number of people currently in space   Master Skills:   * Build a web app that shows a map of the current location of ISS.  1.3.4.4: Option 4 - Create an App for Any API For expert or master level students, come up with your own idea. Find an API, do the authentication necessary, and create an application. 2.0.1.1: Chapter 2: Model Driven Programmability In this chapter, you will learn some basic network programmability concepts and learn how to program a network device using YANG device models and the RESTCONF and NETCONF APIs. The workshop will consist of three main parts:   * Network Programmability * YANG Models * Experimenting with RESTCONF * Experimenting with NETCONF * Security considerations   At the end of this workshop you will be able to:   * Explain the concept of APIs and the importance of RESTful APIs for software integration. * Explain how network programmability enhances network management and automation. * Explain the advantages of using model based device APIs, compared with a traditional CLI based approach for network automation. * Explain how YANG device models define the structure, syntax and validation rules for device data. * Interact with networking devices using RESTCONF and NETCONF API interfaces. * Use Python with combination of RESTCONF and NETWORK APIs to retrieve and update the device’s configuration.       With the networking devices that we are familiar with, you may tend to think of a router or switch as a single device. However, the major functions of these devices can be divided into two planes:   * ****Control plane**** - Makes forwarding decisions. The control plane contains Layer 2 and Layer 3 route forwarding mechanisms. Information sent to the control plane is processed by the CPU. * ****Data plane**** - Also called the forwarding plane, this plane is used to forward traffic flows. Routers and switches use information from the control plane to forward incoming traffic out the appropriate egress interface.   By separating the control plane and data plane, network programmers can centralize information that devices use to make forwarding decisions and perform other functions. 2.1.1.2: Programmable API interfaces for the network In Software Defined Architectures, the SDN controller provides an abstraction layer on top of the physical network, making it easier to configure, maintain and troubleshoot. The controller itself comes usually with built-in features that help network administrators out of the box. Moreover, the controllers expose a Northbound RESTful API interface that enables developers to build their own custom applications and features on top of the abstraction layer. Such applications can extend the build-in functionality of the controller or solve specific business requirements.  On the Southbound side, the controller uses various protocols to communicate with the networking devices. Common Southbound protocols are SSH, SNMP, but also emerging new model driven protocols as well such as RESTCONF and NETCONF. |