REST API Basics

In this Learning Lab you will learn the basics of consuming a REST API, and how to use Postman to explore a REST API.

Objectives

Completion Time: 20 minutes

- **Understand the basics of consuming REST APIs**
- **Learn how to use the Postman client to make API calls**
- **Learn how to make calls to the Spark API**

Prerequisites

In this lab, we are going to the Cisco Spark APIs and Postman to explore making API calls.

Background

- **If you are unfamiliar with Python and APIs or are new to programming, this is a great place to start! If you are a wizard of such things, feel free to move to the next module.**
- **This lab will use Postman which is a web based tool to make the REST API calls. Postman allows engineers to easily discover the capabilities and syntax of a REST API allowing for an easier transition from API syntax to real code. If you do not already have Postman installed, you will need to install this tool.**

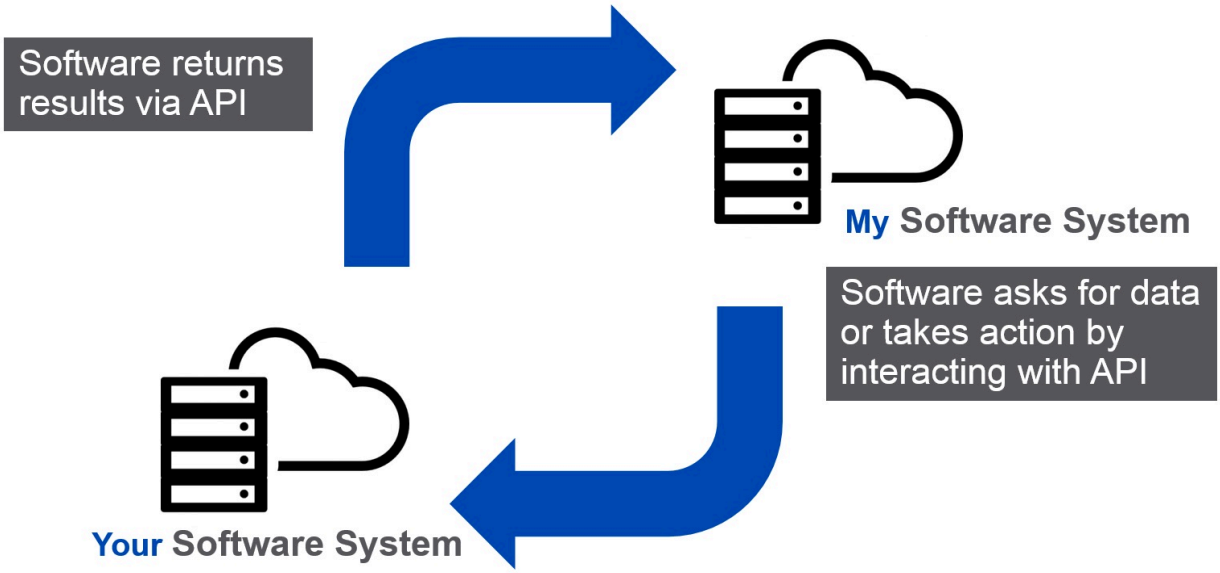
Overview of APIs

We are going to begin our discussion on network programmability by discussing what an API is and why they are important. Let's start by making sure we have a good foundation by diving into some definitions.

An API (Application Programming Interface) is a way for two

pieces of software to talk to each other. Think about the ways you typically interface with software. For example, you might open up a web interface to access your email. You might have a specific workflow to open up messages and file them away for later. Each of these workflows has a specific " interface" or way in which you achieve a certain task.

An API is similar in concept. Instead of humans interfacing with software, software interfaces with software. Rather than having a human point and click-through a workflow, an API exposes functionality to another application.



Why would you want to do this? One reason is that this allows for the development of rich applications with a wide variety of functionality. Let's go through an example.

Suppose you are the creator of a Restaurant Recommendation app and you want to easily integrate the ability to return a list of relevant restaurants in the area with a map application that displays where the restaurants are relative to your location. Would you create this functionality from scratch? Probably not. Doing so would likely take you away from your core expertise. Also, think about all of the risk and learning curve required to build something like that from the ground up. Instead, it would be better to leverage a third party that already offers that functionality, and integrate the functionality into your applications.

A good example is a Maps Server. Rather than build map functionality from scratch, you could use an API provided the Maps Server to integrate map functionality quickly into your application.

APIs help developers create apps that benefit the end user.

The role of an API is to act like a contract that enforces a specification. Take a look at the image below.

What would it be like to power a laptop without an outlet?

- Open wall
- Unsheath wires
- Splice wires together
- Understand all the wires in the wall

The outlet is a service that conforms to specifications.

- sockets deliver 120 volts of alternating current (AC) operating at 60Hz
- Sets expectation on behalf of consuming devices and provider.

Much like a wall outlet, an API enforces a specification of an interface. An API ensures that software adheres to the proper specifications much like an outlet ensures that devices adhere to the proper electrical specifications regardless of the vendor or device.

Step 1. Get Access To The Spark REST APIs

We are going to use the Spark REST API to start our programmability journey. Spark is a collaboration tool that allows people to chat and share files, plus other capabilities.

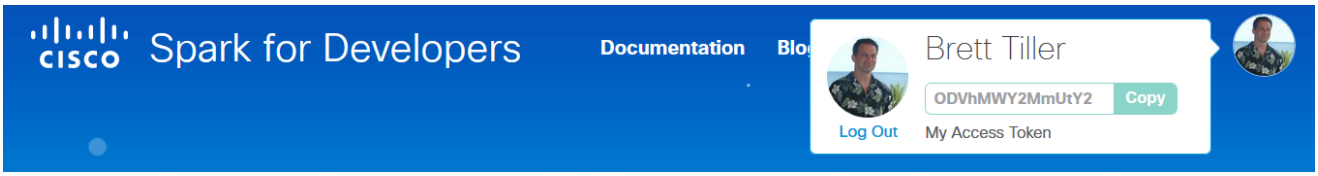
You need to set up a Spark developer account in order to explore the Spark REST API!

1 Go to the Spark Developer website.

2 Click on Sign Up button to create an account.

3 Log in with your new credentials and click on the member icon located on the top right corner of the page and click on the 'Copy' button to copy your Access Token.

4



5

6

7

The token allows you to make Spark API calls.

Great! Now that we have the basics of APIs down, let's dive into REST APIs to see how they work.

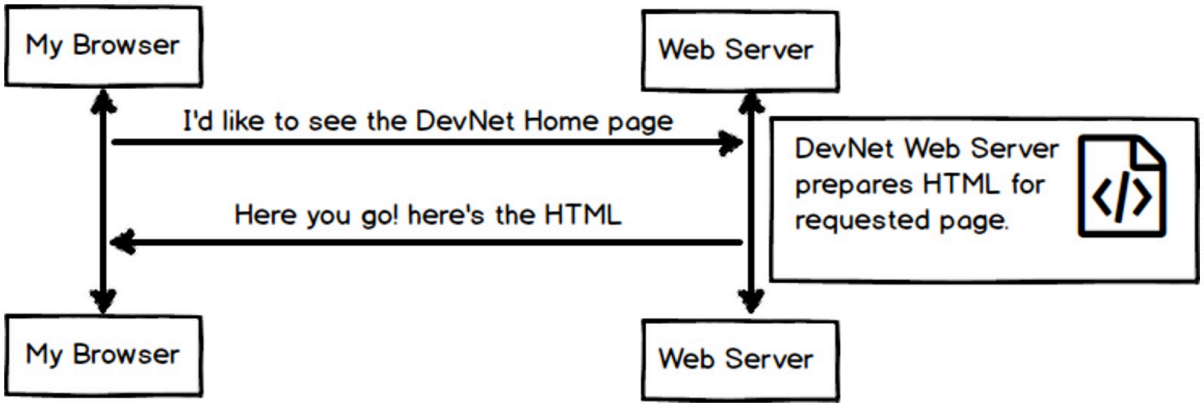
Step 2. What is a REST web service?

In general, a web service is a way for two systems to communicate through a defined interface. In the past 20 years, there have been two major types of Web Services – REST or SOAP. In the last 10 years, the REST approach has become increasingly popular.

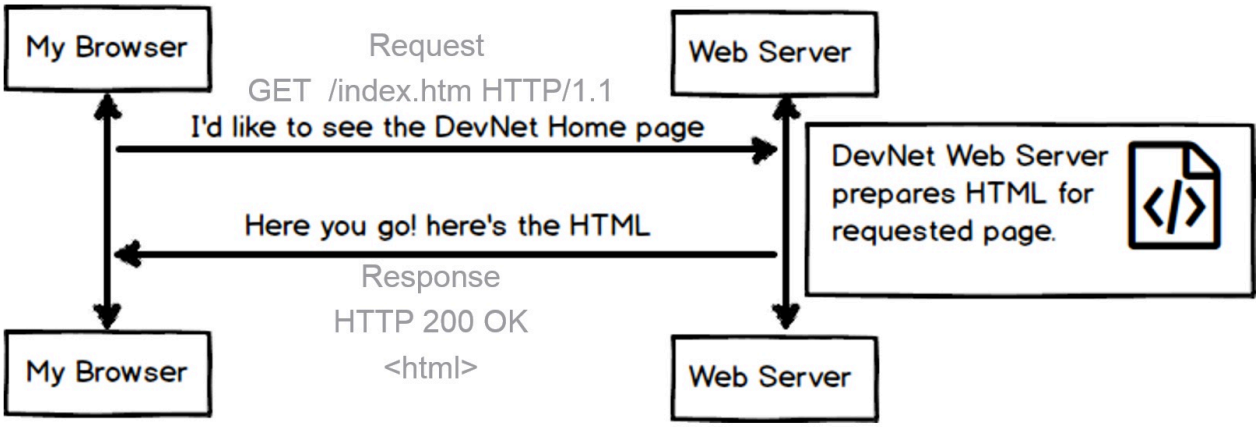
What is REST? REST (Representational State Transfer) is an architecture style for designing networked applications. A REST web service is a web service that is as easy to call as making an

HTTP request.

Look at the diagram below. It shows how a browser retrieves web pages. Normally, after a user requests a particular resource in a browser, the appropriate web server responds with the proper HTML to display the page to your client browser.

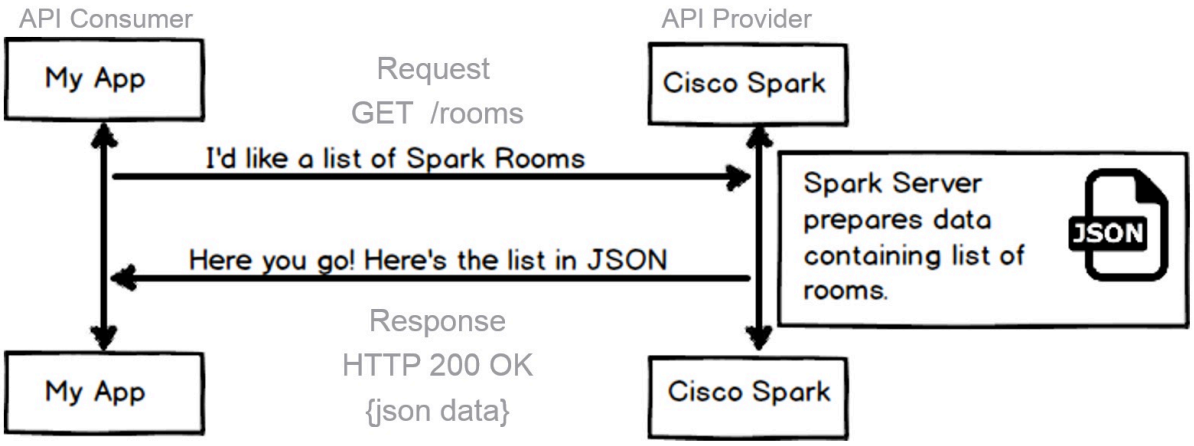


Behind the scenes, HTTP(S) uses CRUD (Create, Read, Update, Delete) operations on the wire to request data. In the example below, our browser is issuing a GET operation to read the associated web page. The web server returns the associated data and an HTTP response to the client browser.



RESTful interfaces offer these same CRUD (Create, Read, Update, Delete) operations using HTTP(S). Browsers are replaced by software to interface with the RESTful service. The

diagram below illustrates the same concept; however, browsers are replaced by software leveraging the REST API.

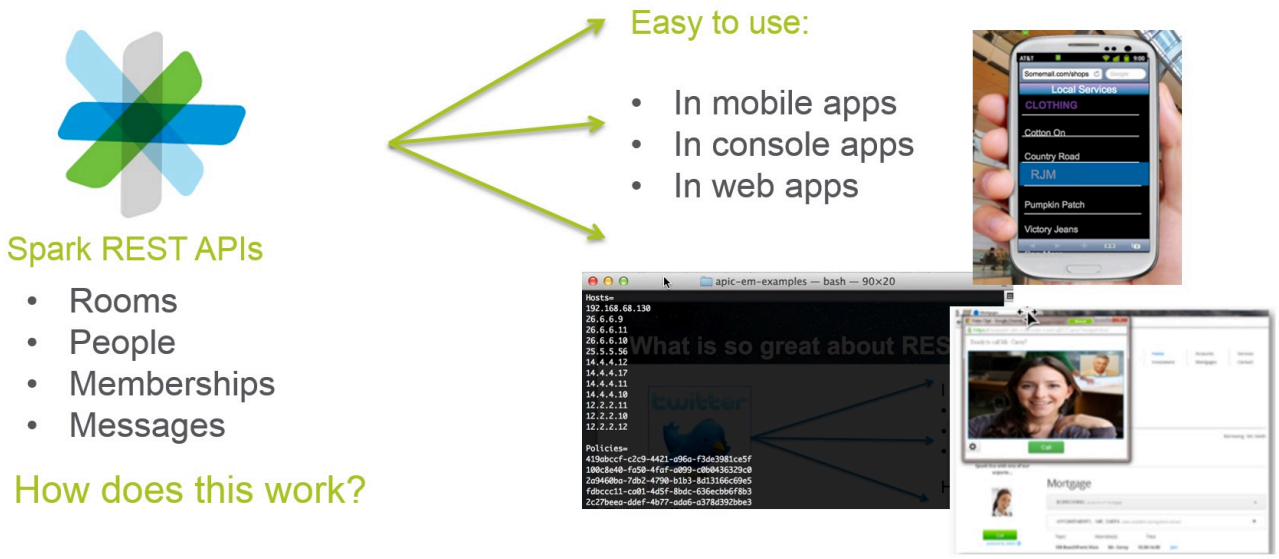


If you want to know more about REST in general, this is a great REST tutorial .

What is so great about REST?

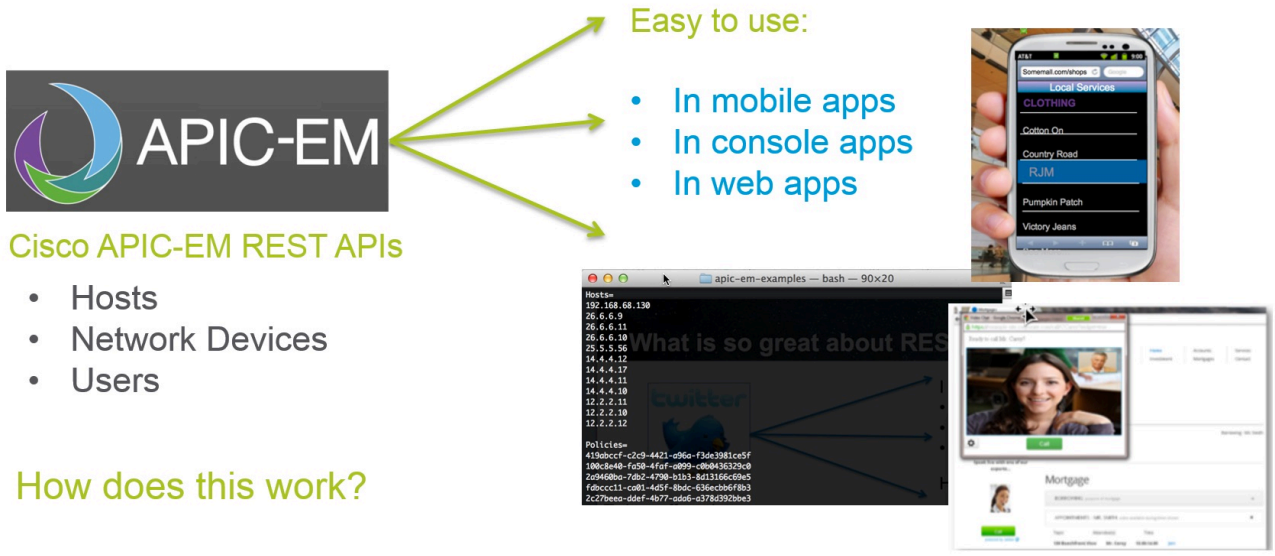
REST is easy to use on any platform!

Let's talk about what makes REST APIs so great. First off, the concepts are transferable across a number of platforms. This lab shall focus on the Spark REST API as a tool to learn REST APIs. Spark is a communication platform for collaboration.



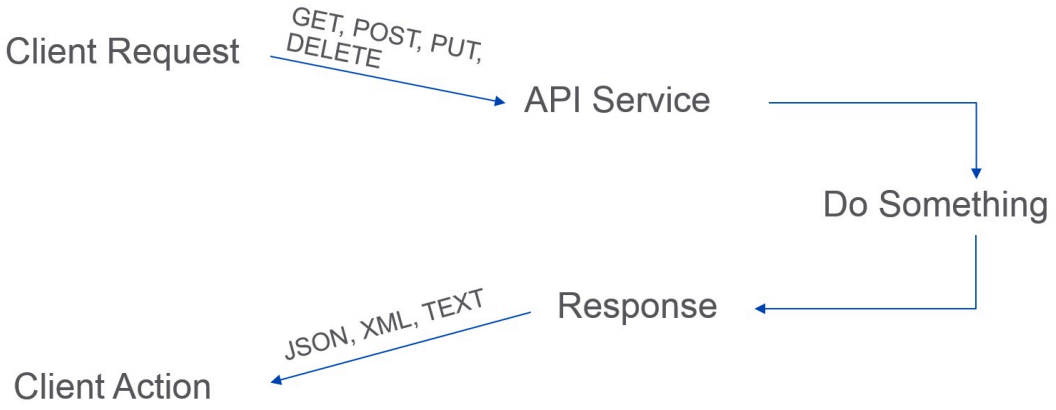
However, REST is also used by a number of networking

platforms such as Cisco's APIC-EM controller. While the API and data model of APIC-EM differs, the underlying tools are the same.



How does this work?

Now let's cover how REST APIs work. REST is centered around the HTTP request and response model. Consuming an API is just as simple as making an HTTP request.



For example, if we make a request to an API Service, then result of the request will be returned to us in the response. The data returned in the response is usually JSON or XML.

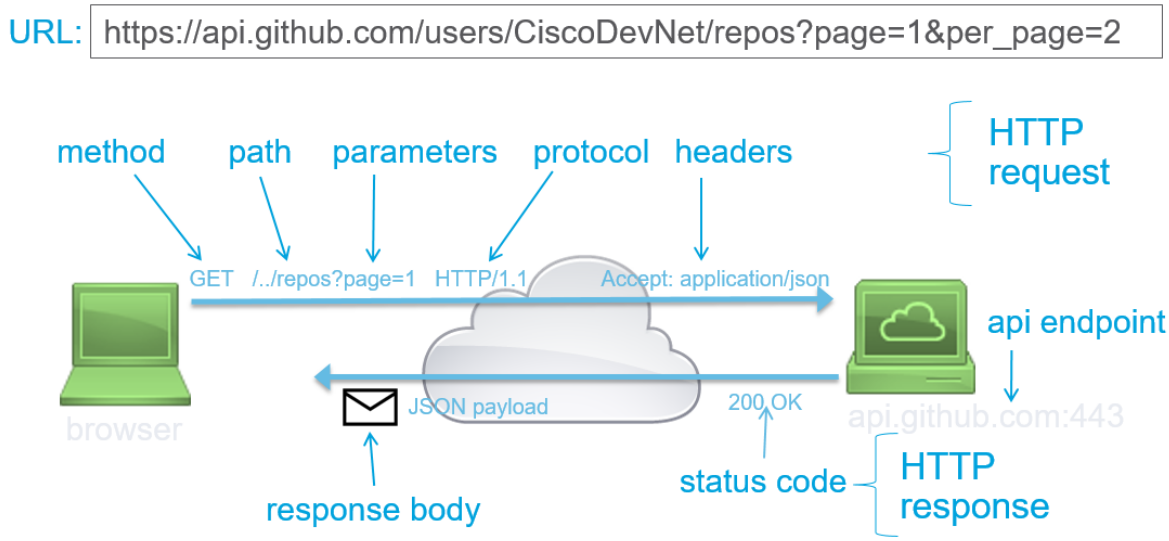
(JSON -- JavaScript Object Notation, is a lightweight text-based open standard designed for human-readable data interchange.)

Step 3. Examine the Anatomy of a REST API Query

To construct a request, you need to know the following information for the API that you are calling. You can find this information in the API reference documentation.

- Method
  - GET - Retrieve data
  - POST - Create something new
  - PUT - Update data
  - DELETE - Delete data
- URL
  - The URL for the endpoint you want to call
  - Example: http://api.ciscospark.com/v1/rooms
- URL Parameters
  - The parameters that you can pass as part of the URL.
- Authentication
  - You need to know the authentication type to use. Basic HTTP, token based, and OAuth are common types.
  - Authentication credentials
- Custom Headers
  - Does the API require you to send any HTTP Headers?
  - Example: Content-Type: application/json
- Request Body
  - JSON or XML containing data that is needed to complete request can be sent in the body of the request

Anatomy of a REST API query



A Little About Authentication

There are different types of authentication for REST APIs. Authentication is used to control access and access rights to the REST APIs. For example, some users might have read-only access which means that they can use only the parts of the API that read data. Other users might have both read and write access. This means they can use the API to perform operations that not only read data but also add, edit, and delete data. These access rights are typically based on user assigned roles such as Administrator where a user would have full rights to change the data. For example, a plain User role might have read-only access rights.

Types of Authentication Controls

- **None: the Web API resource is public, anybody can place call. Generally the case for GET methods, rarely for POST, PUT, DELETE.**
- **Basic HTTP: The username and password are passed to the server in an encoded string.**

Authorization: Basic ENCODEDSTRING

See Basic Authentication for more information.

- Token: A secret key generally retrieved from the Web API developer portal.

  The keyword may change from one Web API to another: Bearer, token..

  Passed with each API call.

  See Token Based Authentication for more information.
- OAuth: A sequence flow is initiated to retrieve an access token from an Identity Provider. The token is then passed with each API call.

  Open standard. User rights are associated with the token (OAuth scope).

  The token expires. It can be revoked. It can also be re-issued via a refresh token.

  See OAuth for more information.

API Reference Documentation

The API Reference Documentation lists all of the publicly available API methods and provides the details on how to make each request. When you start to work with a new API, the API Reference is one of the most important sources of information. Here's the API Reference Guide for Spark .

What is in the Response?

The API Reference Guide includes information about the attributes to be sent and returned. The returned data is defined in the Response portion which includes the HTTP status codes along with the data format and attributes.

8 HTTP Status Codes

HTTP status codes are used to return success, error, or other statuses.

http://www.w3.org/Protocols/HTTP/HTRESP.html

Some common examples are:

200 OK

202 Accepted/Processing

401 Not Authorized

9 Content

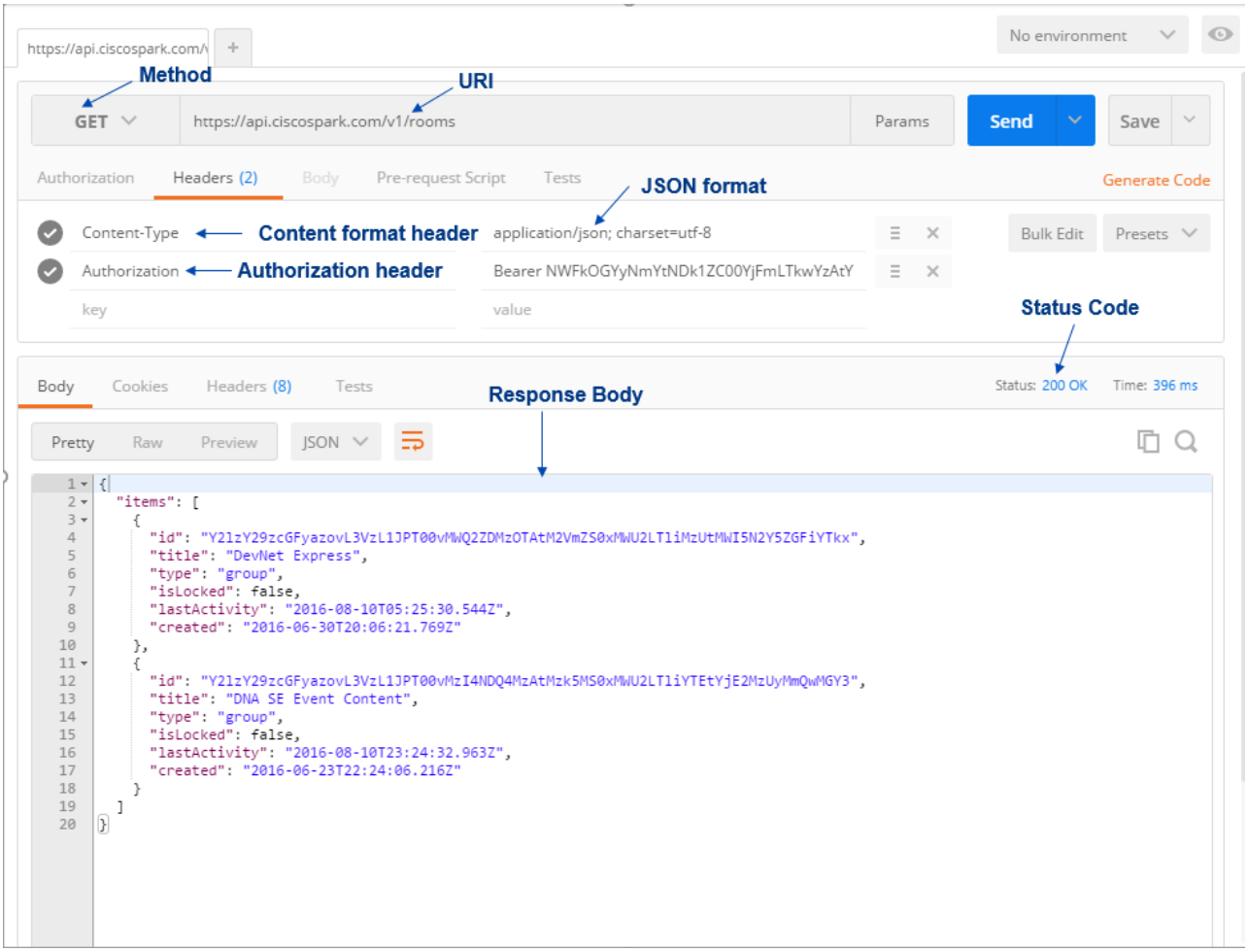
Often returned in different formats based upon the request.

Common formats are: JSON, XML and Text.

JSON (most commonly used)

```
{
 "data": [{
      "company": "Cisco Systems",
      "event": "DevNet Express",
      "location": "Las Vegas, NV, USA"
}, {
     "company": "Cisco Systems",
     "event": "Cisco Live America",
     "location": "San Jose, CA, USA"
}]
}
```

Here is an example that shows a REST API request and response all in the same window:



Step 4. Use Postman to Call the Spark API

As briefly mentioned earlier Postman is an HTTP web user interface (Web UI) client that allows you to make HTTP calls. There are many Web UI clients that have similar functionality, but for our purposes we'll focus on Postman.

What is Postman?

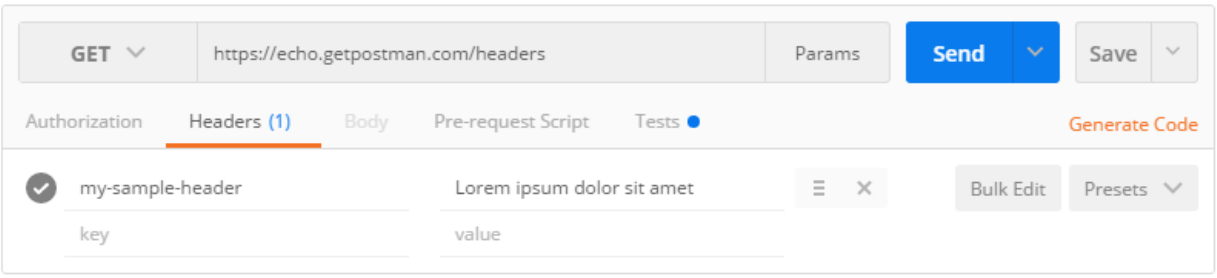
Postman is a Google Chrome application. It provides an easy-to-use interface for learning and interacting with REST APIs. Users can send API calls and receive responses in the same window. This application can also be used to generate codes for different languages such as python. Postman is useful for entry-

level users. To download and run Postman follow this link. We'll use Postman throughout this and following modules. Take a minute to make sure it is installed on your workstation.

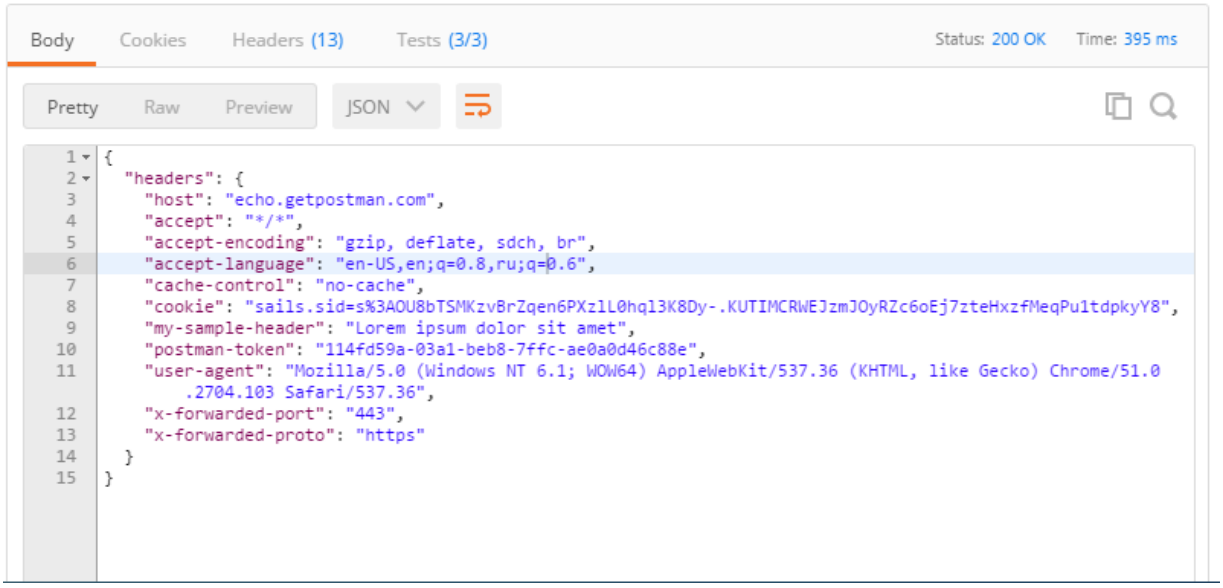
The Postman Window

Postman has three main working areas. On the left side of the window you can see History and Collections tabs. The History tab shows a list of your previous calls. If you want to save a particular API call from your History tab, you can click on the call to highlight it and then click the `Save to collection` link to specify where you want to save it.

In the middle of the screen is where all the magic happens. This is where you make the API calls by setting the call method and providing URL information. Under the Headers tab you should define all of the necessary key-value pairs to make a successful call. When all of the parameters are set, click the Send button.



If the request is correctly formed, after you click Send you should see a response on the bottom section of the window. The Response pane gives you information such as the Status code of the response, how it is formatted (JSON, XML, etc.), as well as the body of the response message.

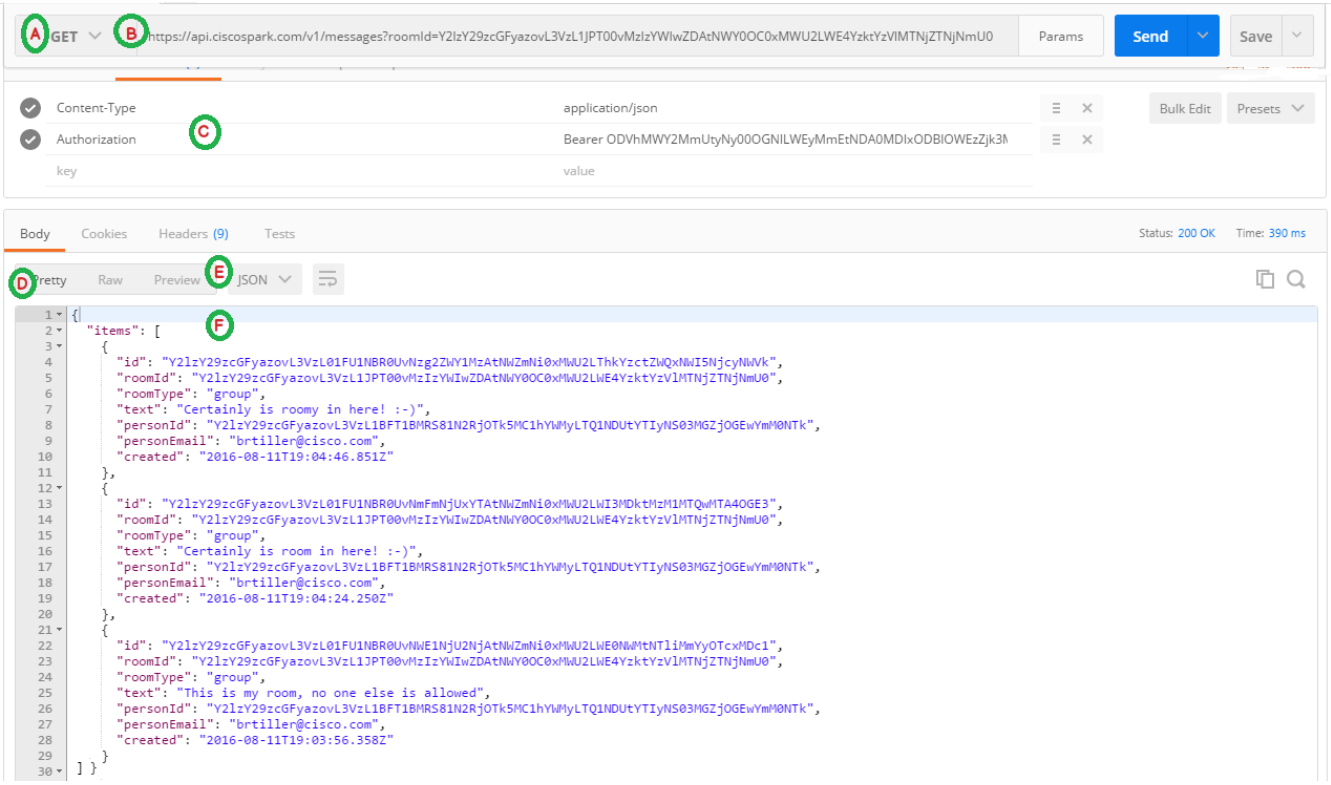


Postman Test Run: Call the Spark API

Let's make a call to the Spark API. Look at the screenshot below, it depicts an API call to Spark server. The letters show each step and explain the areas of the Postman window.

A. Shows the REST method to use to send the request. You can select the method from a drop down list. As previously discussed the most common methods are: GET, POST, PUT and DELETE. In this case we want to read in a list of messages

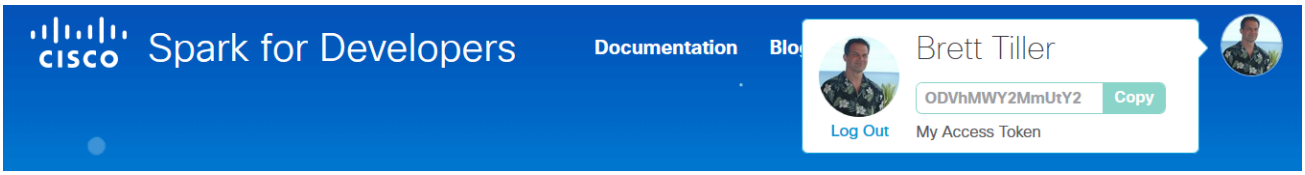
posted by a particular user to a particular room. That means for this example we want to use the GET method.
B. The URL to be called is specified here. The base URL for SPARK is https://api.ciscospark.com/v1/ . What follows next is the function to be called which is messages. Next is the roomId follow by the ID itself in the format of ?roomid=<the room ID>
C. Spark requires an Access Token in order to make function calls. For that reason, we have added the key named Authorization to the header. We entered the value Bearer <the access token>. The Content-Type key specifies what type of formatted content is being sent to the HTTP Server which is designated in the URL by api.ciscospark.com . In this case no content is being sent so this header, though commonly used, is unnecessary. At this point the request is ready to be sent and the user would press the Send button.
D and E. The Body contains the data returned from the request. Because the returned format has been selected to be JSON, there's various types of ways to format this output. Raw output displays the data in the format received. Pretty output which is selected here, reformats the data to make it much easier to read.
F. The returned JSON content in Pretty format.



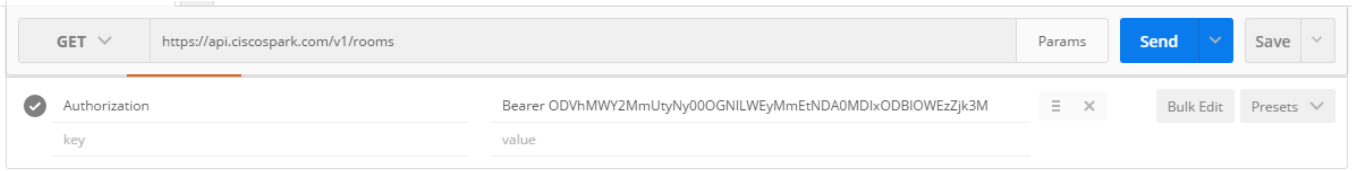
Give it a Try!

Follow the steps below to make your first Spark API call using Postman.

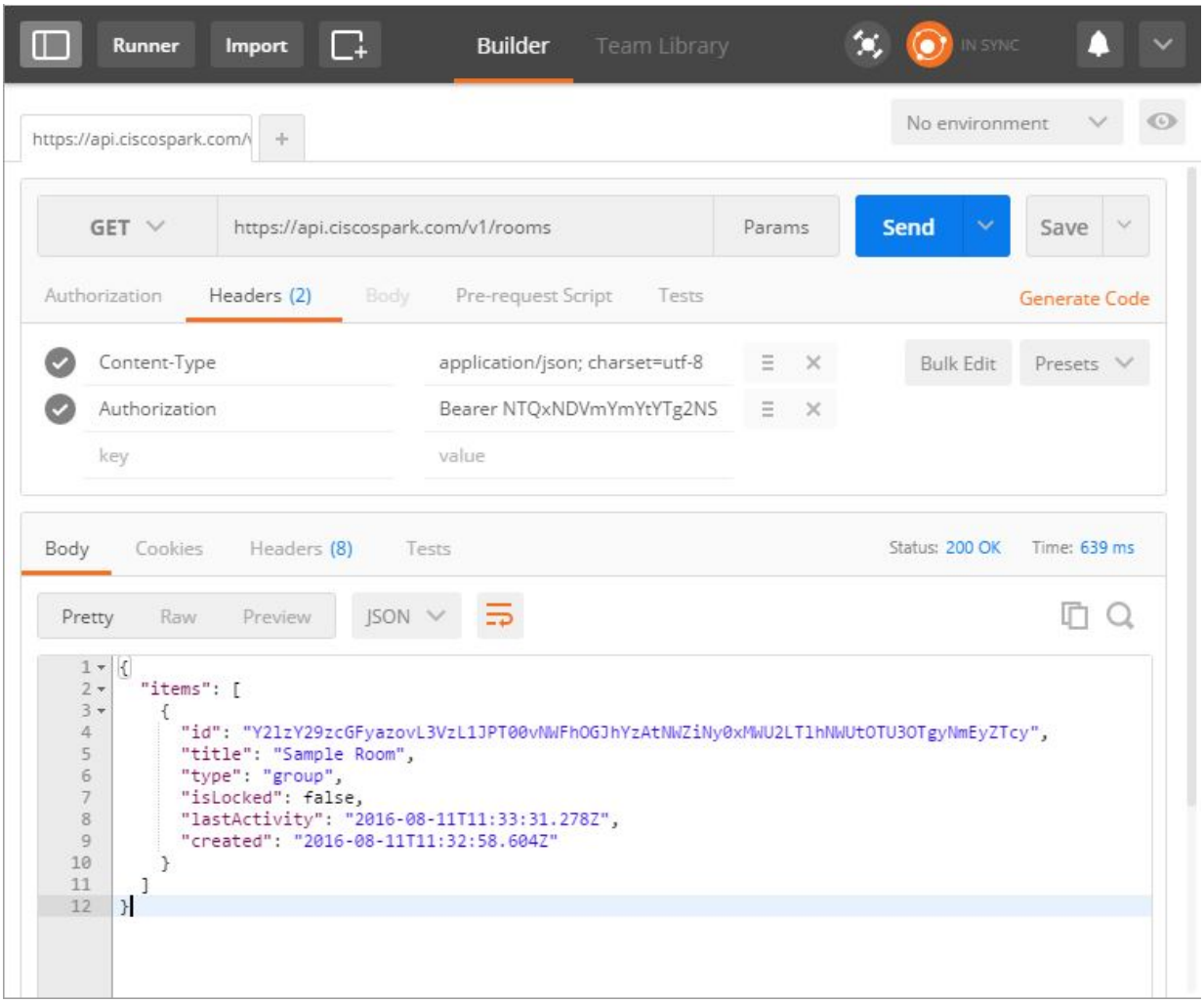
- We're going to make a function call to get the list of Spark rooms that you belong to. Access Spark with your Spark desktop application or the web client at http://web.ciscospark.com, and log in. If you do not belong to a room, simply create one, and you will automatically be a member. Or ask someone you know to invite you to their room.
- You'll also need your Access Token to make the function call. Go to https://developer.ciscospark.com and either sign up if you haven't done so or log in. Once you are logged in, click on the member icon located on the top right corner of the page and click on the 'Copy' button.
-



- Open Postman.
- Select 'GET' from the drop down method.
- In the URL field enter " https://api.ciscospark.com/v1/rooms"
- In the header field add the key " " Authorization" .
- In the value field add the word Bearer followed by a space, then paste in your Access Token.



- Press the Send button.
- Result should be similary to the screenshot show below.



You should receive the list of rooms to which you belong. The format of the data is in the JSON format. If you received an error recheck the steps, making sure that the method, URL, and header information are correct.

Congratulation you've just made your first API call!

Spark – Application Registration and Authentication

In this Learning Lab, you will learn about Spark's APIs and how to use Postman to make API calls.

Objectives

Completion Time: 20 minutes

- **Understand the basics of consuming REST APIs**
- **Learn how to use the Postman REST Client to make API calls**

Prerequisites

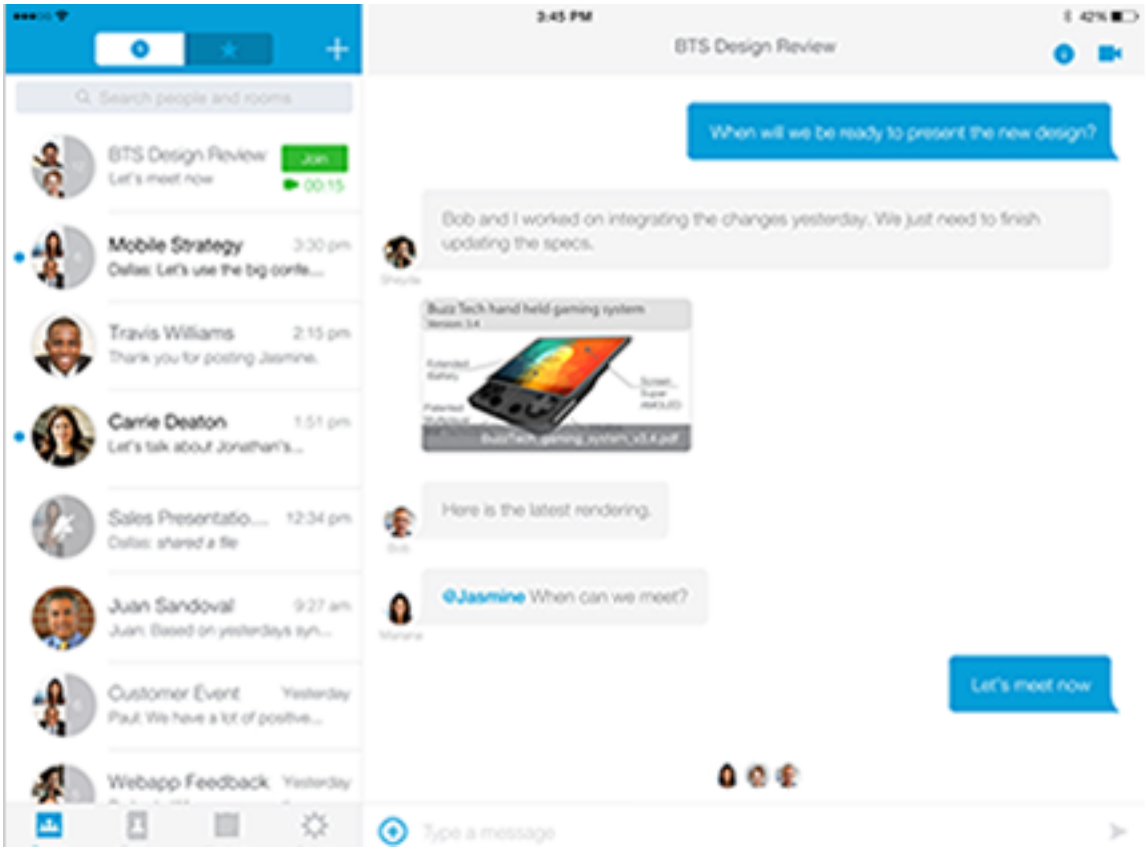
In this lab we are going to use Postman which is a HTTP client tool used to make the REST API calls. Download and install the application on your workstation if you have not done it already.

Overview

Cisco Spark is a cloud service providing persistent chat, room-based collaboration, WebRTC video conferencing, and more. Developers can easily integrate solutions with Spark via the Spark REST API. The APIs may be used to do many things such as adding Spark messaging features to an application user interface or automate sending Spark messages to Spark rooms based upon business system or real-world events.
Application developers integrating with Spark must register their applications via the Cisco Spark for Developers portal - defining the application name, permissions, and OAuth2 redirect URL (more on this later). During registration the system generates a Client ID and Client Secret pair which are later used by the application to access the Spark OAuth2 authentication service. During this lab you will create a new custom application registration, and generate a Client ID and Secret pair.

Step 1. Sign up for Cisco Spark and take a tour

- From the Spark home page at https://web.ciscospark.com enter your email address, and follow the instructions for creating a Spark account. Note: you will need to access your email account as Spark will send a 'You have a Cisco Spark account' email message with a 'Create Password' verification link. This step must be completed in order to log into Spark.
- Log into Spark and take a quick look around. Feel free to create a new room and invite a friend (Note: If they are not already signed up, they will need to complete the email verification step too.)

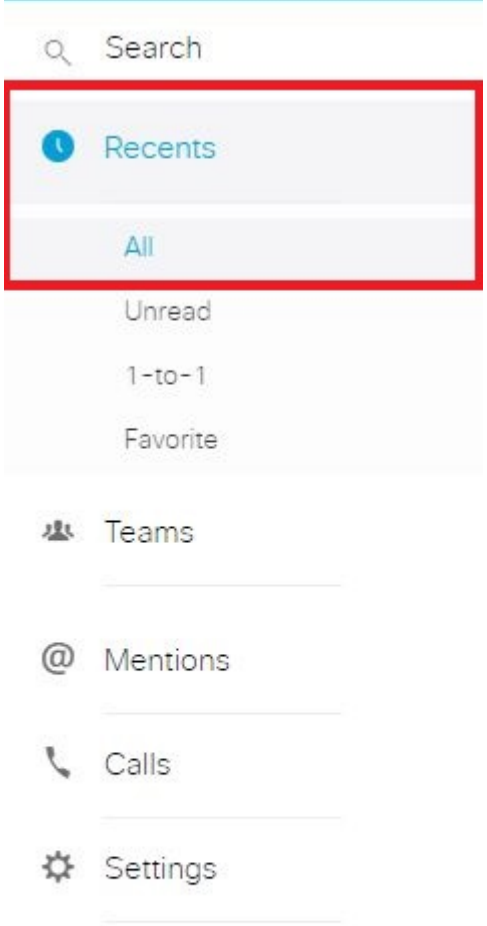


How To Create a Room in Cisco Spark

Creating a room in Cisco Spark is very easy. Person needs to invite two people to a new conversation and the room will be

crated automatically.

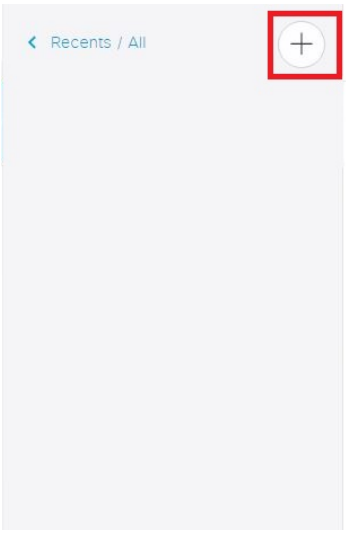
10 From the Cisco Spark's main page, click on the Recents



menu and select All option.

11

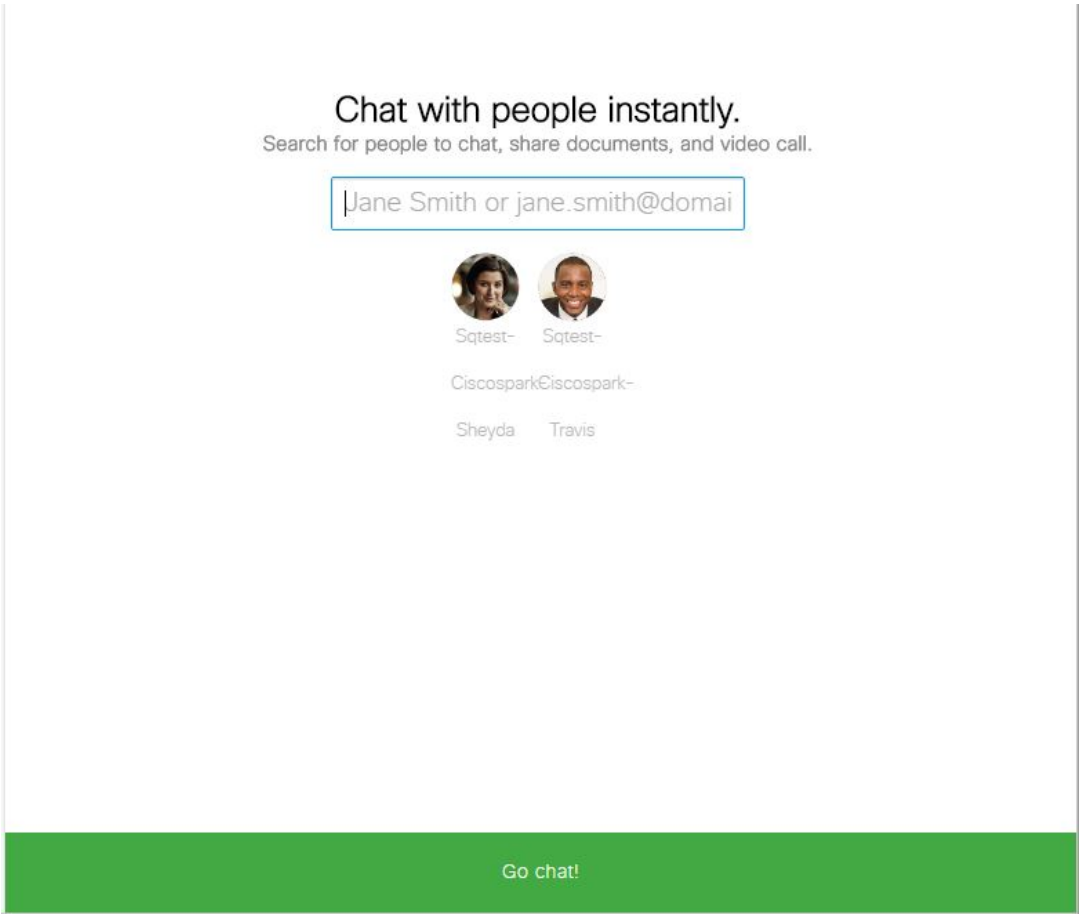
12 In the Recents / All menu click on the + button to invite



people to start a new conversation.

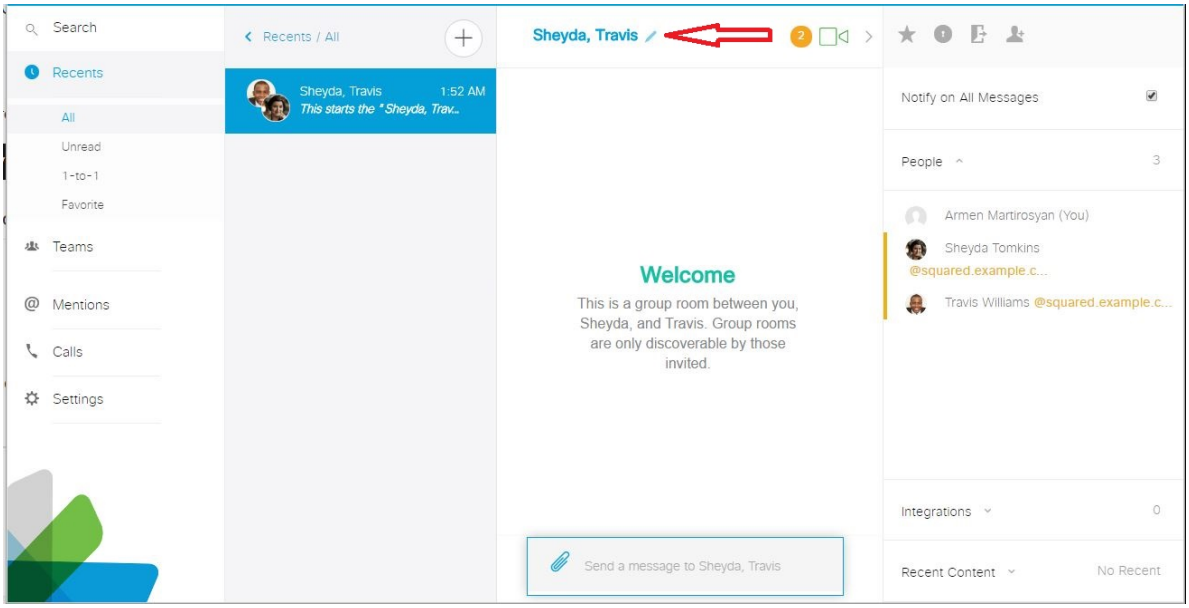
13

14 In order to create a room, you need to invite two or more people to a conversation. In order to do that, you need to provide either a person's firstname and lastname or provide an email address. If the person has a Cisco Spark account, his/her name should appear in the list. If the person does not have Cisco Spark account, you need to provide their email address. Spark notifies the invited person to join the conversation. Spark prompts the invited person to create an account. For demonstration purposes you can use these two email addresses: `sqtest-ciscospark-travis@squared.example.com` and `sqtest-ciscospark-sheyda@squared.example.com`. After adding both participants, click on the green bar on the bottom of the page with the Go chat! title.



15

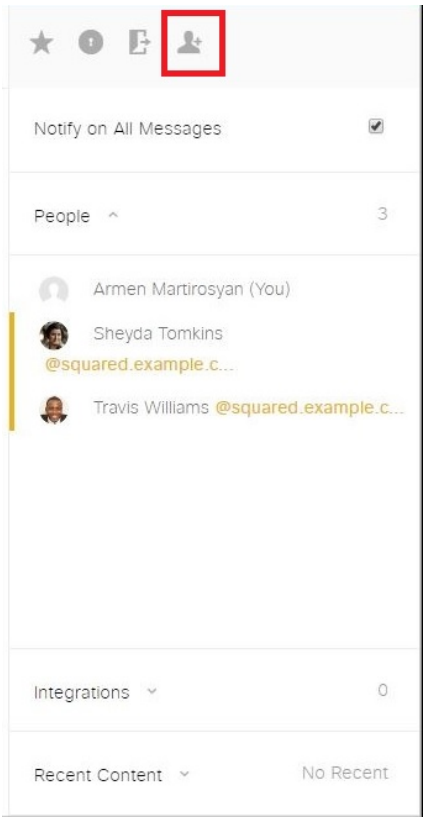
16 Congratulations! You have just created a new room in Cisco Spark. Let's go one step further. Let's rename the room to something more meaningful. In order to do that, select the newly created room in the Recents / All menu and click on the pen icon next to the participants name. Change the name to Marketing Ideas



17

Give it a Try!

1 On the top right side of the page, click on the Add People icon

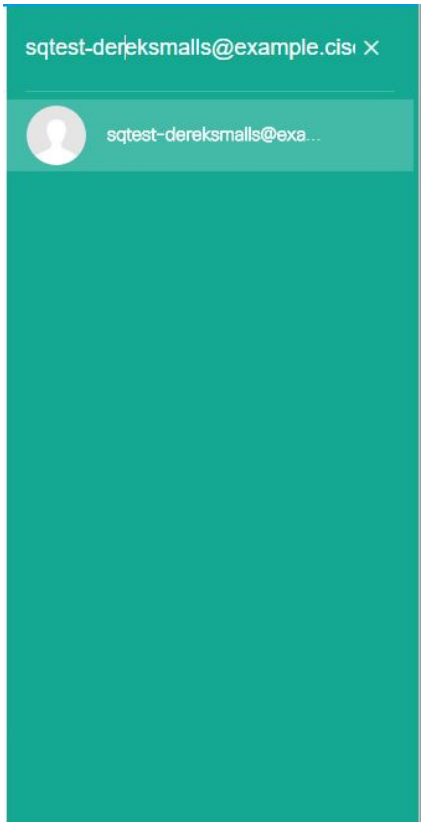


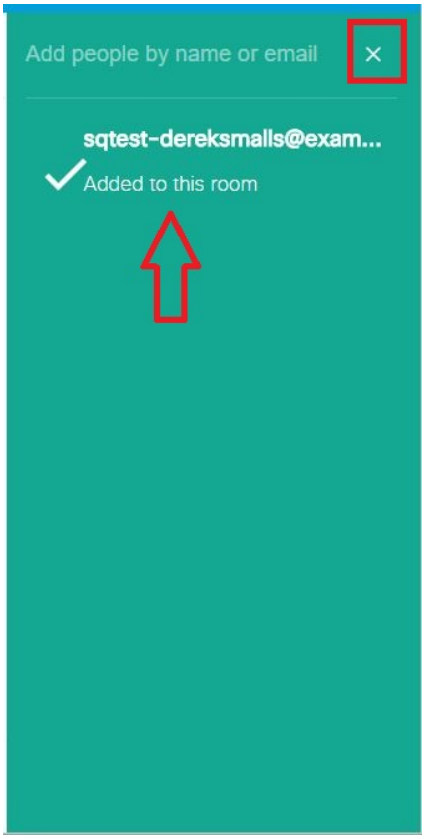
to add new participants.

2

3 In the Add people by name or email field, put [sqtest-]

dereksmalls@example.cisco.com then click on person's avatar icon to add him to the room. You should see a message indicating that the person was added to the room. After you are done adding people, click on the X icon next to the Add people by name or email field to close the panel.





4

5

Great! Now you know how to create rooms in Cisco Spark and add participants to it. In the next section, we will show you how to create a room using Cisco Spark's REST API calls.

Step 2: Listing Spark Rooms

The Spark user experience is centered on persistent collaboration rooms where groups of users can communicate via instant messaging, voice, video or by sharing files.

We can get a complete listing of the existing Spark rooms that your user is a member of first by using developer.ciscospark.com UI, then by running Postman to query Spark API's `/rooms` REST resource:

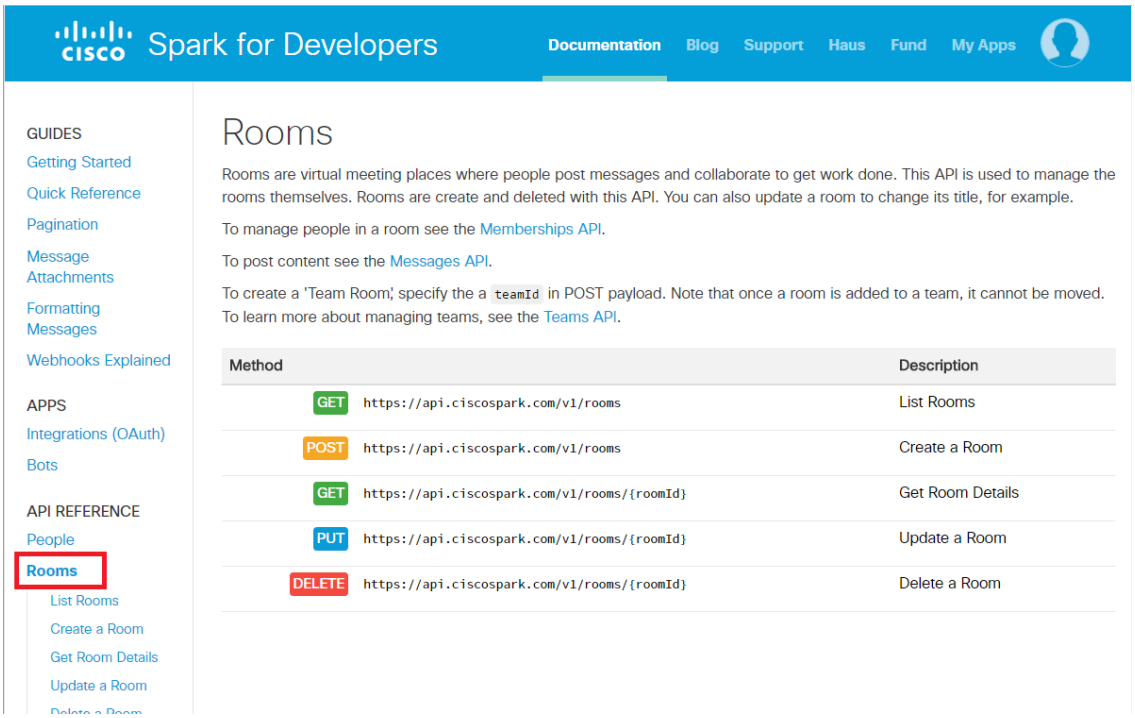
- Spark's Developers page contains API Reference page which

we are going to use to understand particular API call:

Open your browser and navigate to https://developer.ciscospark.com/getting-started.html.

Authenticate if the page asks you to.

On the left panel, under API Reference, find Rooms link and click on it.

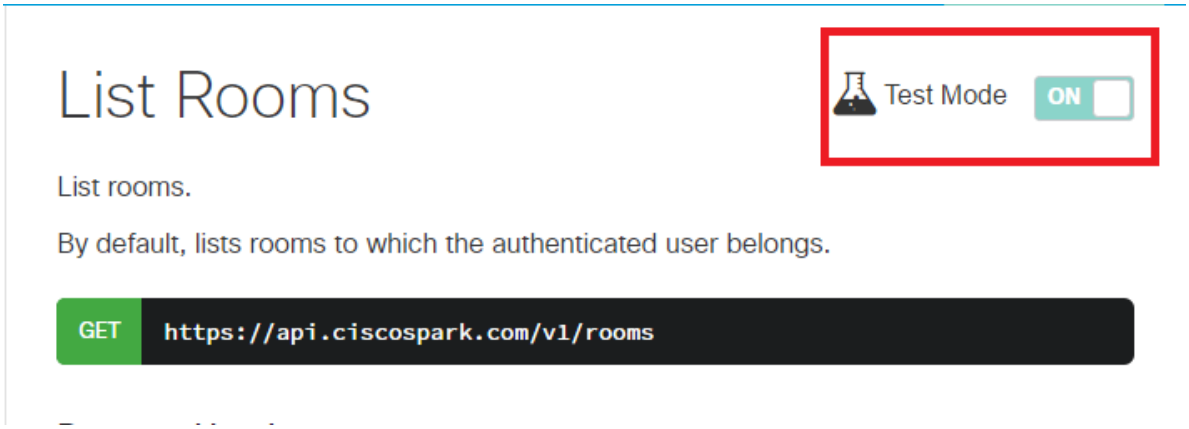


- Note the methods you have at your disposal for this level. We are going to use `GET https://api.ciscospark.com/v1/rooms` API resource to retrieve list of rooms associated with our account.

  From the available options, click on the `GET https://api.ciscospark.com/v1/rooms

Method		Description
GET	https://api.ciscospark.com/v1/rooms	List Rooms
POST	https://api.ciscospark.com/v1/rooms	Create a Room
GET	https://api.ciscospark.com/v1/rooms/{roomId}	Get Room Details
PUT	https://api.ciscospark.com/v1/rooms/{roomId}	Update a Room
DELETE	https://api.ciscospark.com/v1/rooms/{roomId}	Delete a Room

- Now, we need to enable Spark's Test Mode so we can be able to run API calls from the web browser.



-
- After activating the Test Mode function, the web page will change its look. Lets explore all the sections, so we can understand its purposes

  Request headers contains information that will be passed to the API server in the headers section. As we can see, `Content-type` was set to `application/json; charset=utf-8` and `Authorization` key has the value starting with `Bearer` following with the access token.

Request Headers

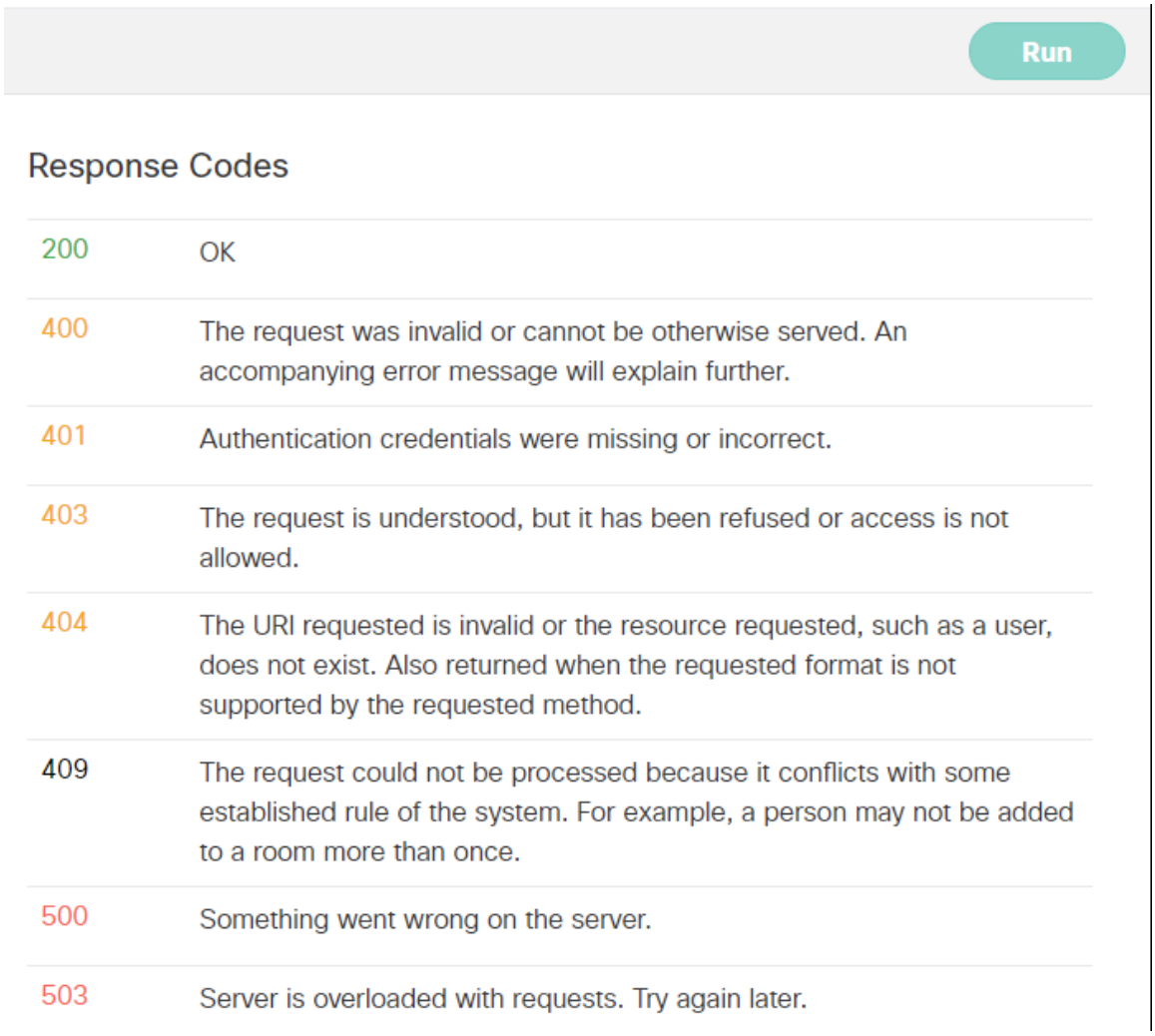
Content-type	application/json; charset=utf-8
Authorization	Bearer NTQxNDVmYmYtYTg2NS00ZGI5LThkMjk

Then we have Query Parameters. This field is used when we want to query information regarding particular value. The query parameter is mandatory, then Required column will have a checkbox next to the parameter.

Query Parameters

Name	Type	Your values	Required
teamId	string		
max	integer		
type	string		

At the end, we see detailed explanation of each response code you might get if you run particular API call. Click run button when you are ready



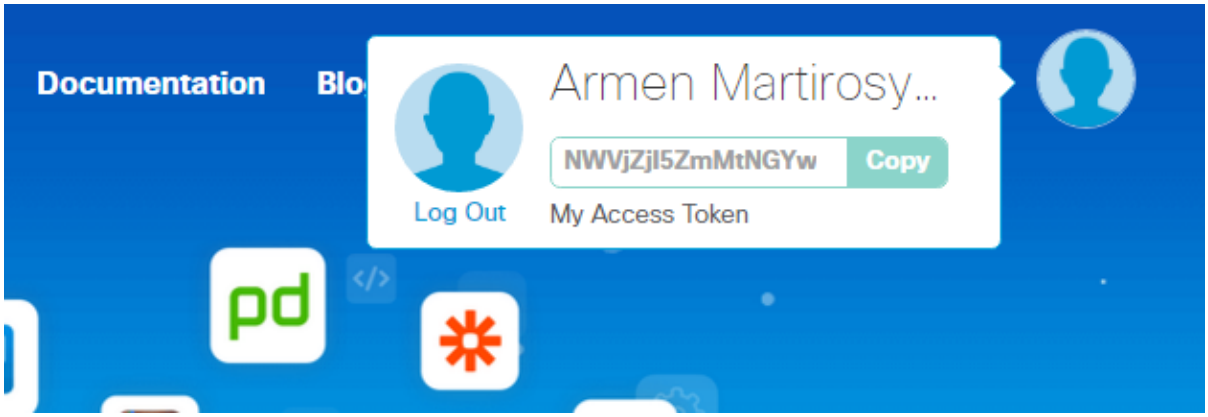
- On the right side of the page, you should see the response from the Spark. Notice the response code. As we can see, we have only one room associated with our account. If your account is accosiated with multiple rooms, then all of those rooms will be displayed in the response body.

Great! We just saw how to use Spark's API documentation and how to make REST API calls from within it.

Spark API with Postman

Let's try to create a new Spark room using Postman.

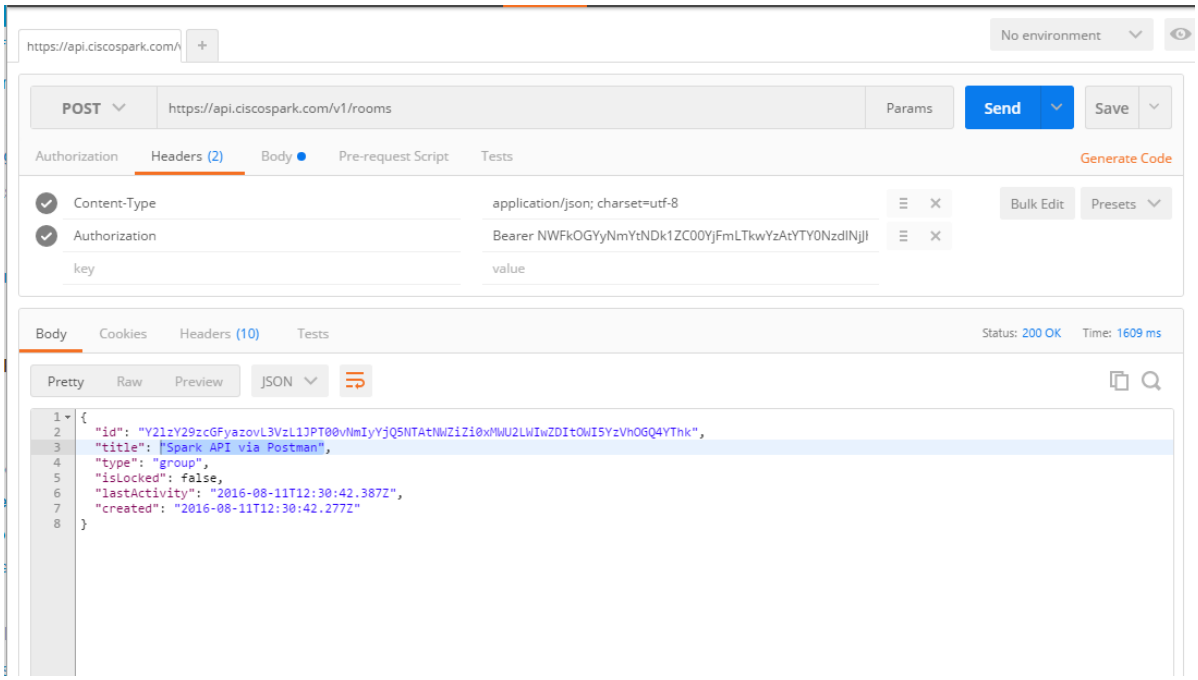
- Retrieve your Spark API Access Token. This token is your permanent developer token, and can be used to access Spark APIs during exploration and development:
  - Open a browser tab, navigate to the https://developer.ciscospark.com, and Login
  - Click on your profile image
  - Click Copy to copy your access token to the clipboard



- On your workstation locate and activate Postman which is a REST Client. (If necessary, click the Reset button to clear any previously entered data.)
- In Postman, construct a POST request for the /rooms resource
  - From the request method drop-down box, select POST.
  - In the field labeled Enter request URL, type `https://api.ciscospark.com/v1/rooms`. This is the URL of the API's /rooms resource.
  - Select the Headers tab.
  - In the field labeled key, type 'Authorization'.
  - In the field labeled value, enter 'Bearer {access token}'. Replace {access token} with your Spark API access token. Note the space between 'Bearer' and the access token.
  - In a separate row of the header, in the key field specify `Content-Type` as the key name and set the value to `application/json; charset=utf-8`
  - Select the Body tab.
  - Select raw. This indicates that you are going to type the body of the request as plain text.
  - Select JSON. This indicates that the plain text is going to follow the JSON format.

Type the body of the request: `{"title" : "Spark API via Postman"}` (You can refer to the Spark API documentation regarding the mandatory parameters in the request body.)

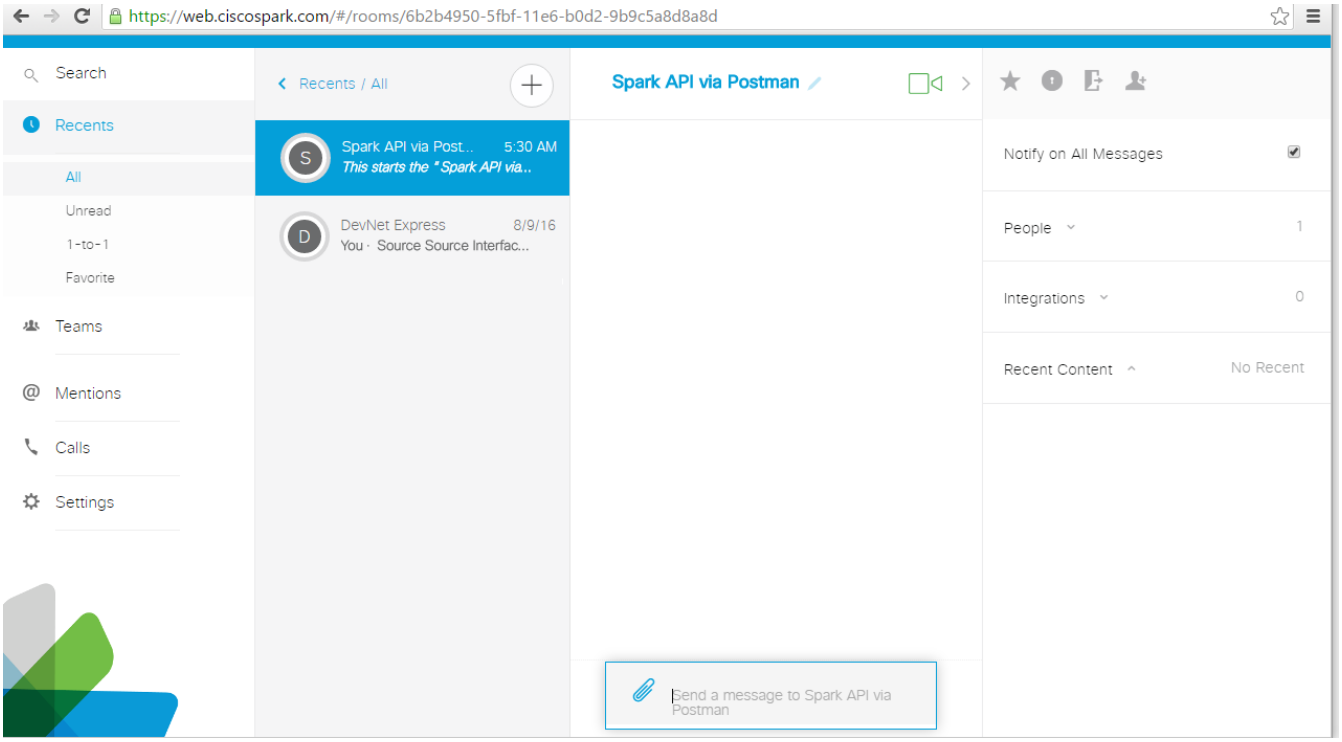
Click Send. The response returns the information about the new room with the title Spark API via Postman.



- Confirm that new room has been created via your web browser:

  Navigate to https://web.ciscospark.com/ and Login.

  You should see the new room in your list of available rooms



Congratulations! You've just created a new room for team collaboration.

Give it a Try!

Change the method in Postman to modify the name of the room you've just created. Refer to the Spark API documentation for required parameters.

Interacting with Spark's API using Python

In this learning module, we will interface with Spark's API using Python

Objectives

Completion Time: 20 minutes

- **Understand how to make REST API calls using the Python**

scripting language

- Post a message in the Spark room using Python

Prerequisites

Python

- To run the code samples, you need to have Python3 installed on your machine.
  - On how to install Python3 on your workstation, please refer to Module 00 where installation steps are covered in great detail.
- Your workstation should have the python `request` module installed. If you are not sure, open a command line terminal and issue `sudo python3 -m pip install requests` command for Linux based machines or `python -3 -m pip install requests` command for Windows based machines.

Access to the Spark REST API

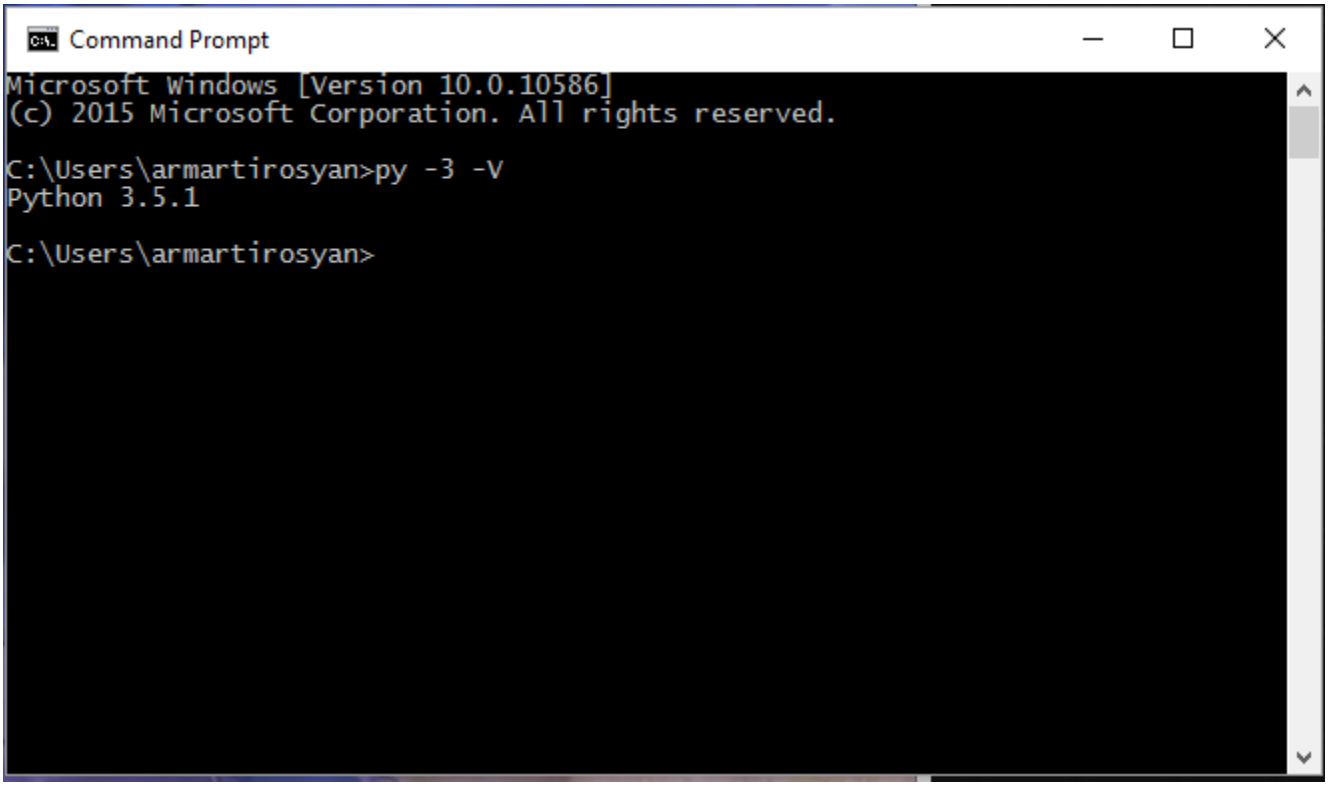
18 Be sure to set up a Spark developer account, so you can explore the Spark REST API! Go to the Spark Developer website then sign up to receive your access token which will allow you to make Spark API calls.

This lab also includes steps in the How to Set up Your Computer section to set up your development environment.

Step 1. Introduction to Python

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Python has simple, easy to learn syntax which makes writing and reading code easy. Python supports modules and packages which encourages program modularity and code reuse.

Lets check if our workstation has Python version 3.x installed on it and that it is ready to use. Open the command line terminal and type command: `py -3 -V` . The command should show you the Python version installed on the workstation. Note: if you are using Linux or OS X the command would be: `python3 -V` .



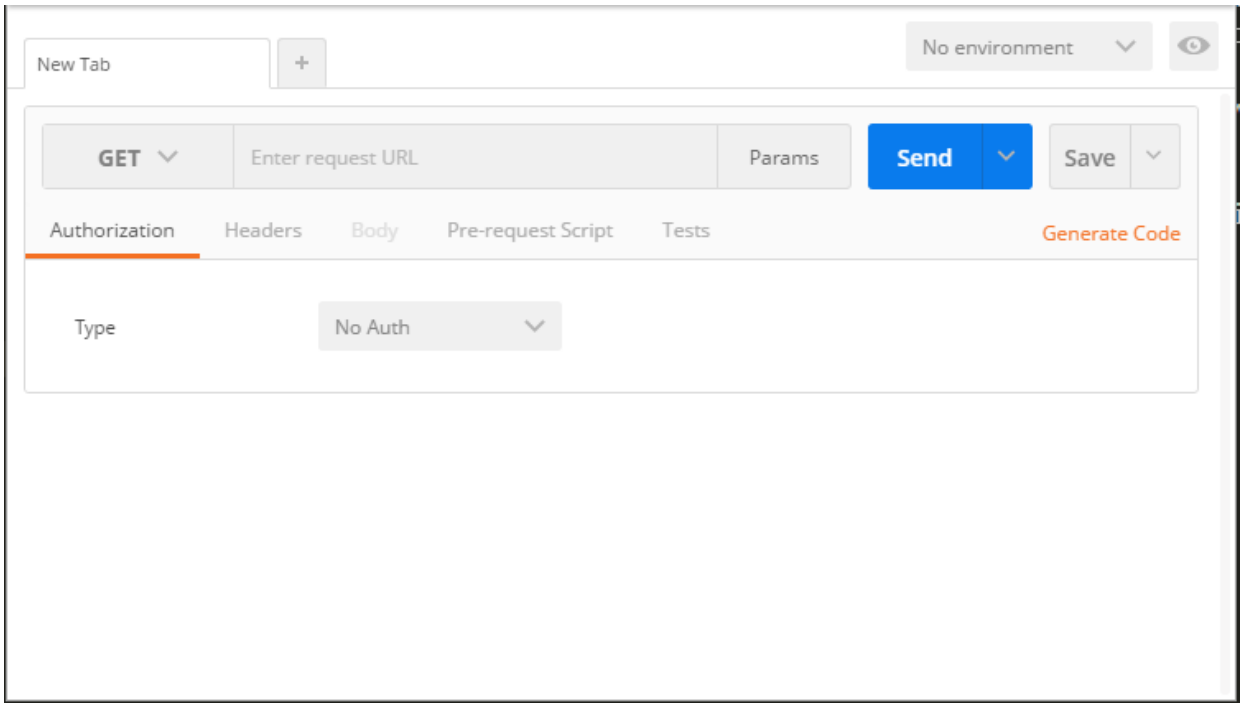
Great! We have Python working on our workstation.

Step 2. Generating Code Using Postman

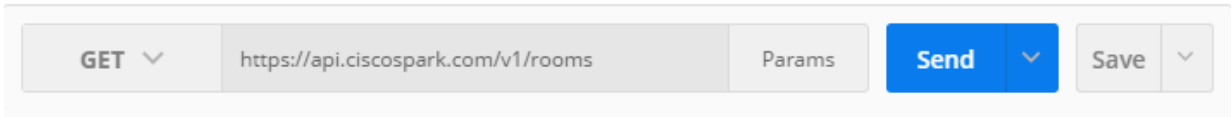
In the previous lab, we showed you how to make API requests using the Postman application. Also, we mentioned that it is possible to generate code for different coding languages using Postman. In this step we will show you how to do exactly that.

We assume that you already have the Postman application installed on your workstation and you know how to use it. If not, please take a minute and review previous lab since it covers that topic.
To generate code, we need to populate all fields in Postman.
Lets begin.

- From your workstation open Postman application.
-



-
-
- Our plan is to get a list of the rooms associated with the user making the inquiry. To do that we need to set the method to `GET` and URL to `https://api.ciscospark.com/v1/rooms`
-

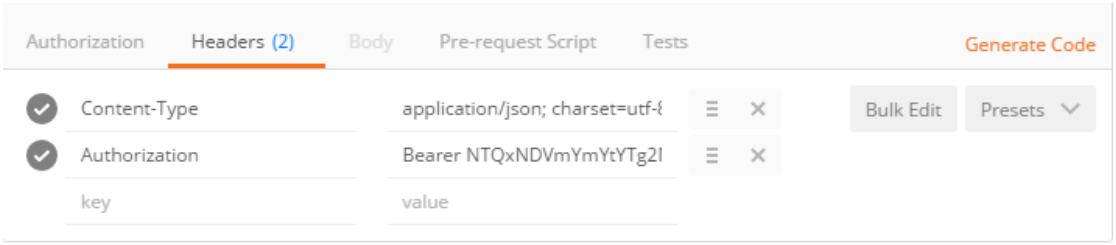


-
-
- Now, we need to define request headers information. Click on the

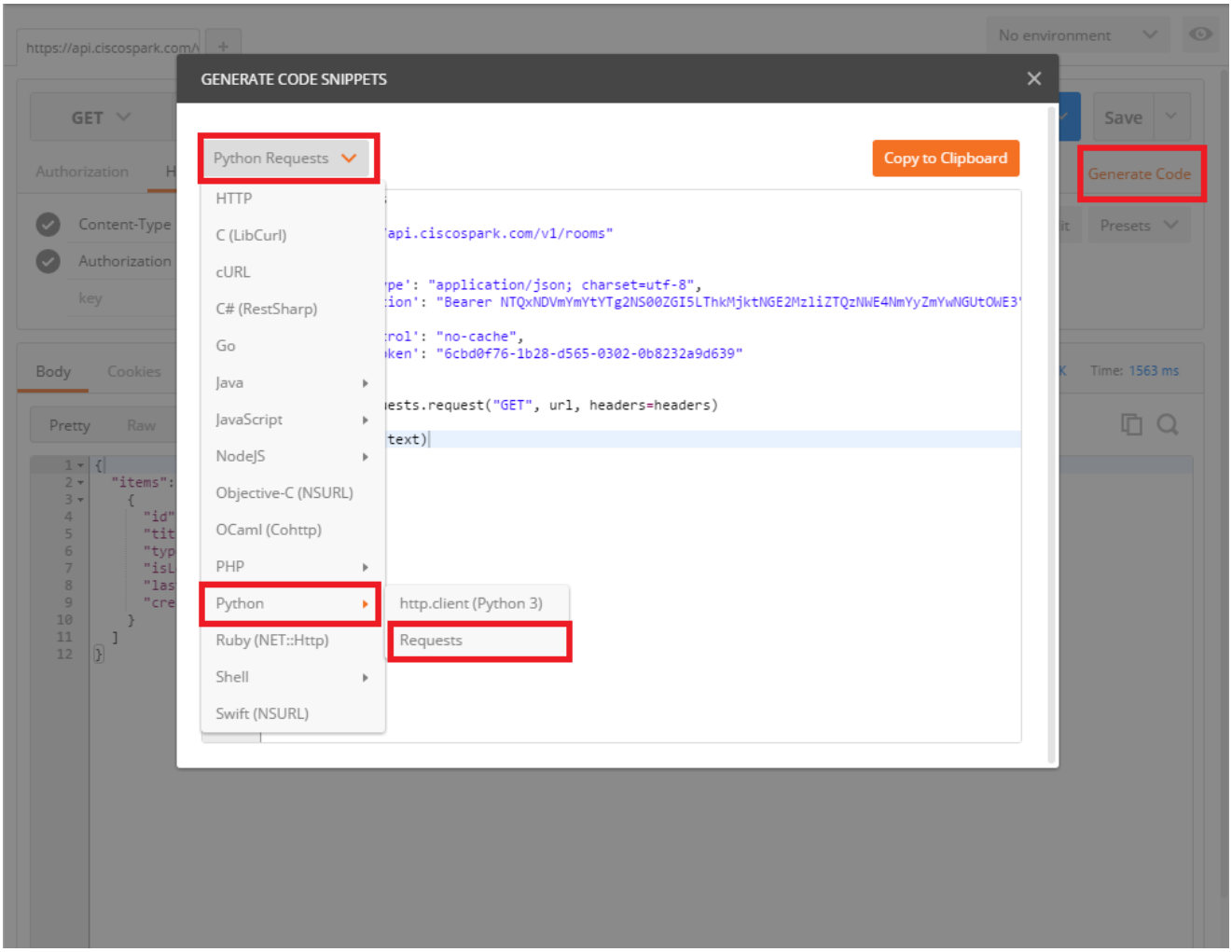
headers tab and provide two key value pairs. For this step you need to have access token obtained from Spark Developer website

`Content-Type` and `application/json; charset=utf-8`

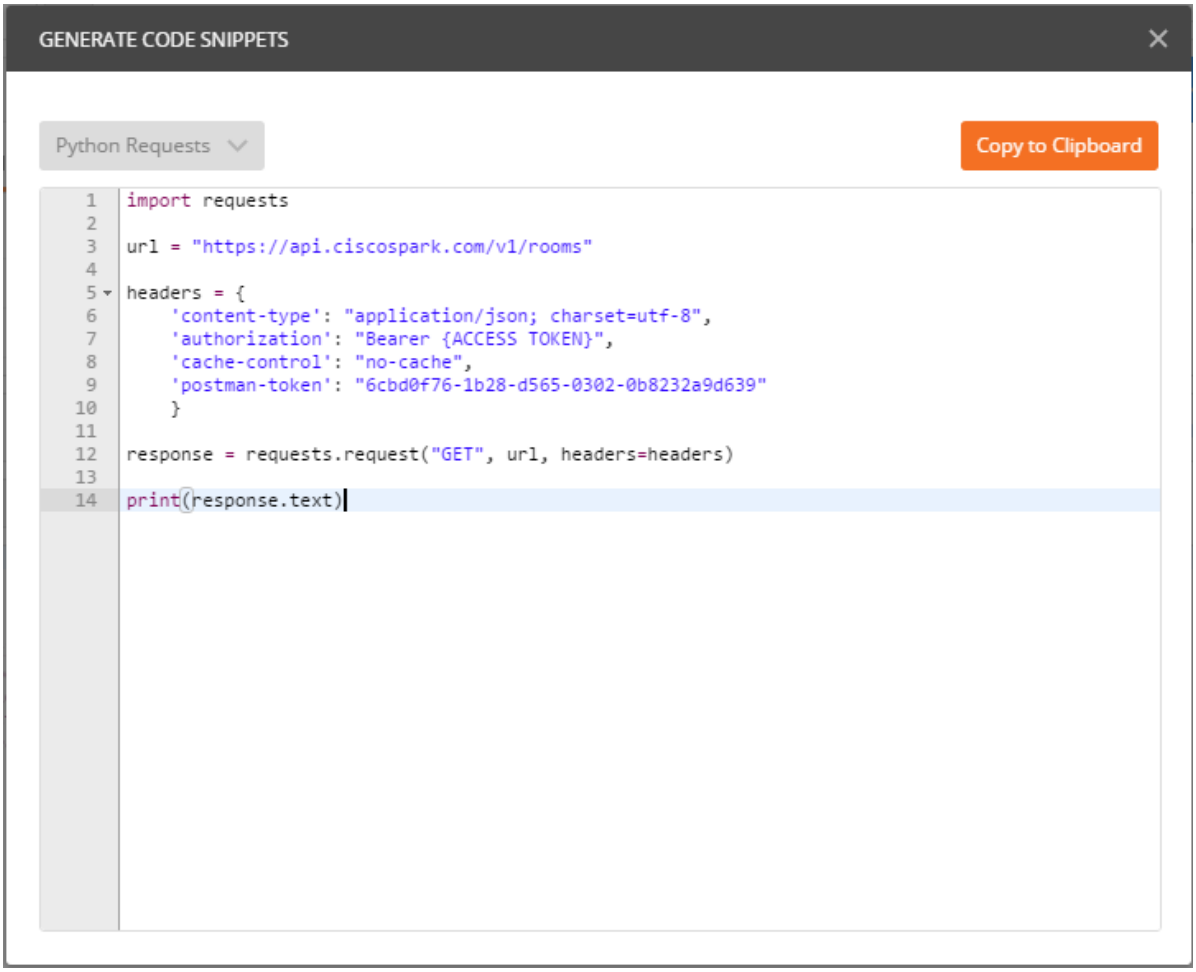
`Authorization` and `Bearer {access token}`



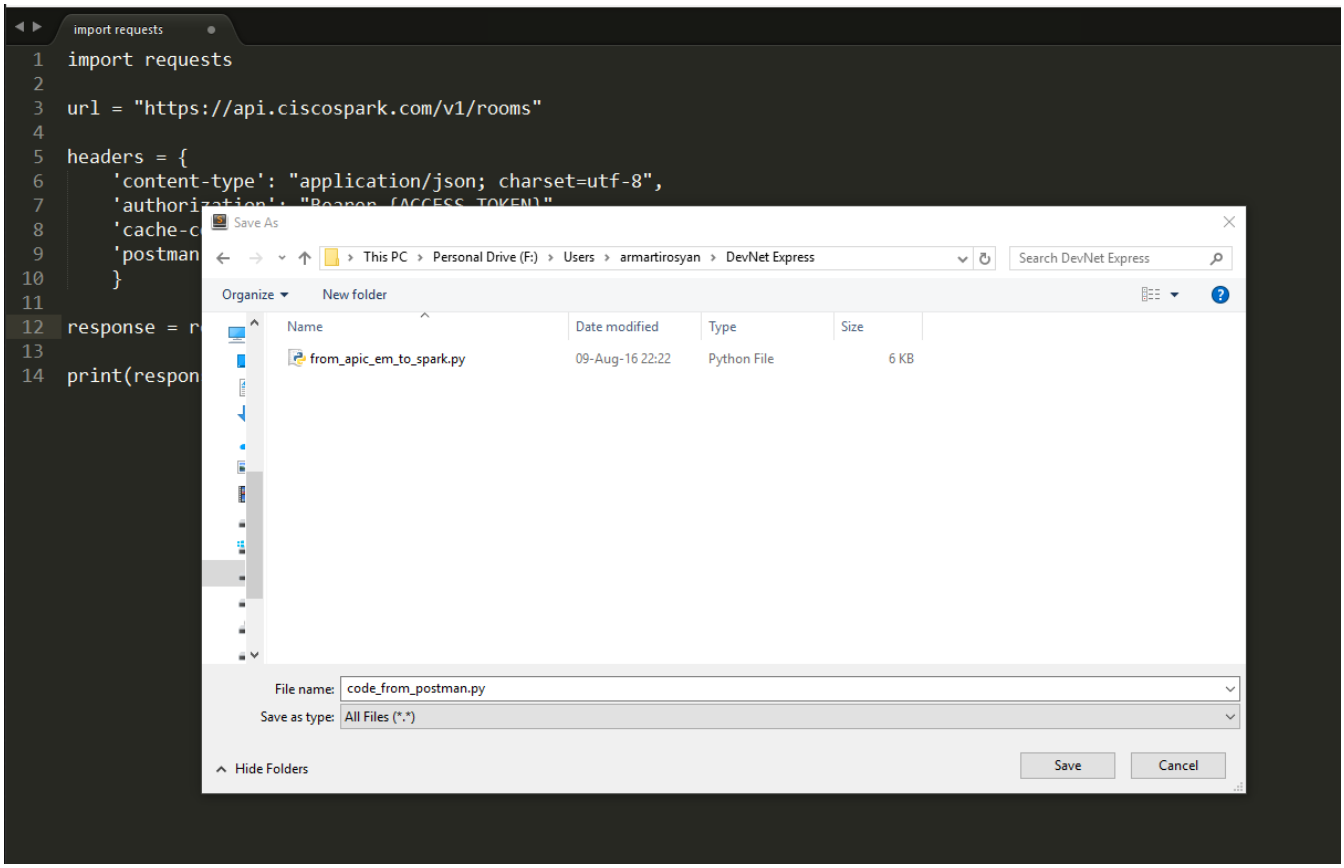
- Good! We are ready to generate the code using Postman. Under the `Save` button you should find `Generate Code` link. Click on it. In the Generate Code Snippets window select Python -> requests language.
-



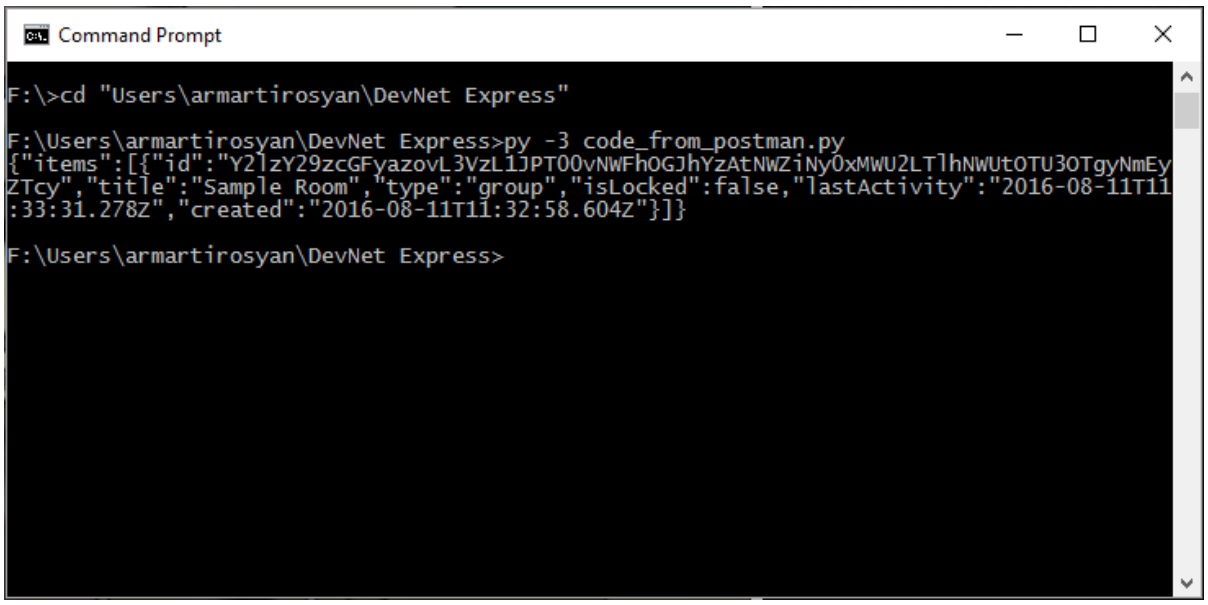
-
-
-
- Now the entries are converted to Python code using `request` module to make the API call. Click `Copy to Clipboard` button to copy the code.
-



- Paste the contents of the clipboard to your favorite text editor and save it as a Python file in your working directory.



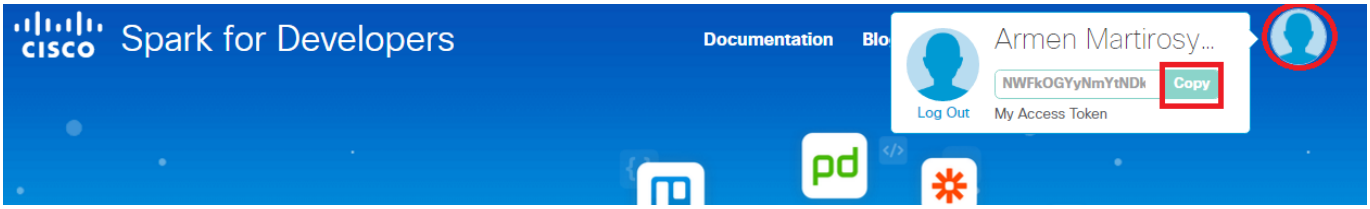
-
-
-
- Lets give it a try and see if it works. Open the command line terminal and navigate to your working directory. Issue `py -3 <FILE-NAME.py>` command. You should see a response from the Spark server.
-



Terrific! You just learned how to generate code using Postman. In the next section of this lab we will show you how to write the code yourself.

Step 3. Writing Python Script

Now, we will write the code ourselves. We will do it step-by-step, and eventually the code will send an API request to Spark and retrieve user's name and lastname based on the email information provided. In the code, we will be using `/people` API call and you can find detailed information regarding the call here. Before we proceed, we need to obtain Access Token from https://developer.ciscospark.com/. We will Login with our username and password then click on our avatar picture. It will show us the access token needed to make the API calls. Click on the copy button.



- To make API calls using python, we need to import two modules: `requests` and `json`.

```
# Importing necessary modules
import requests
import json
```

- Now, we should be able to call all of the functions we need that are packaged in the modules. Optionally, we can turn off warning messages related to SSL certificates. It can be achieved with the code below.

```
# Importing necessary modules
import requests
import json

#Disable warnings
requests.packages.urllib3.disable_warnings()
```

- Below we will define variables which will be used later in our code.

```
# Importing necessary modules
import requests
import json

#Disable warnings
requests.packages.urllib3.disable_warnings()
```

```
# Variables

url = "https://api.ciscospark.com/v1"
api_call ="/people"
access_token = "Bearer {access-token}" #Replace the
    {access-token} with your personal access token.
```

It is time to define our header and parameter information.

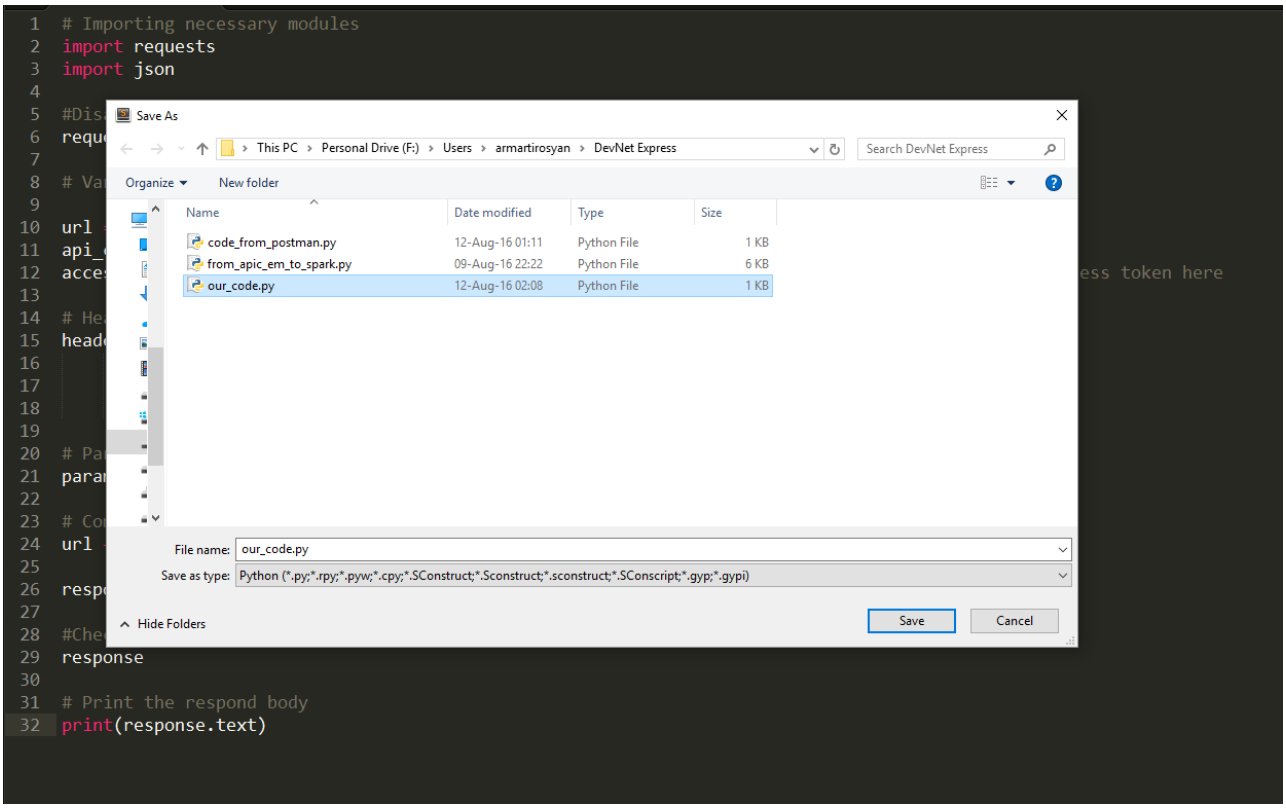
```
 # Importing necessary modules
import requests
import json

#Disable warnings
requests.packages.urllib3.disable_warnings()

# Variables

url = "https://api.ciscospark.com/v1"
api_call ="/people"
access_token = "Bearer {access-token}" #Replace the
    {access-token} with your personal access token.

# Header information
headers = {
            "content-type" : "application/json;
    charset=utf-8",
            "authorization" : access_token
          }
# Parameter variable
param = "?email=sqtest-ciscospark-
    travisuser@squared.example.com"
```

The result of our call we will assign to a `response` variable.

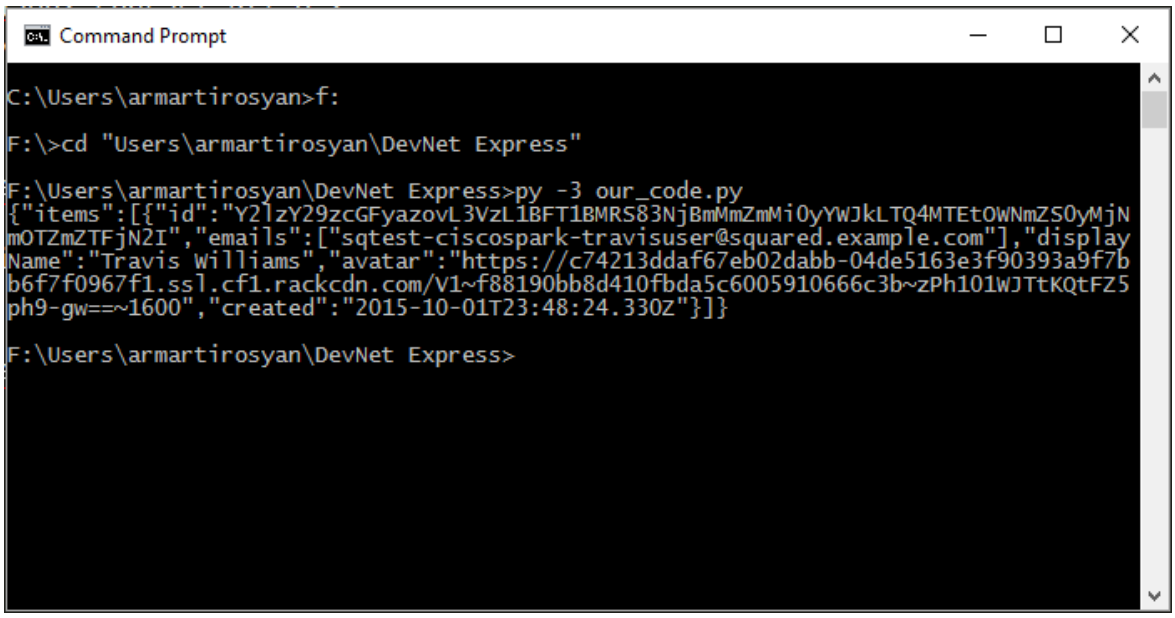
```
 # Importing necessary modules
import requests
import json

```

```
#Disable warnings
requests.packages.urllib3.disable_warnings()

# Variables

url = "https://api.ciscospark.com/v1"
api_call ="/people"
access_token = "Bearer {access-token}" #Replace the
    {access-token} with your personal access token.

# Header information
headers = {
            "content-type" : "application/json;
    charset=utf-8",
            "authorization" : access_token,
          }

# Parameter variable. The email belongs to a bot user, but
    we can use it for our code
param = "?email=sqtest-ciscospark-
    travisuser@squared.example.com"

# Combine URL, API call and parameters variables
url +=api_call+param

response = requests.get(url, headers=headers,
    verify=False)

# Print the respond body
print(response.text)
```

- Our code is ready to be used. We can copy and save it in our working directory as a python file.



- To see how it works, open command line interface and navigate to the working directory. Then issue `py -3 our_code.py` command. If everything was written correctly then we should see output in our terminal screen.



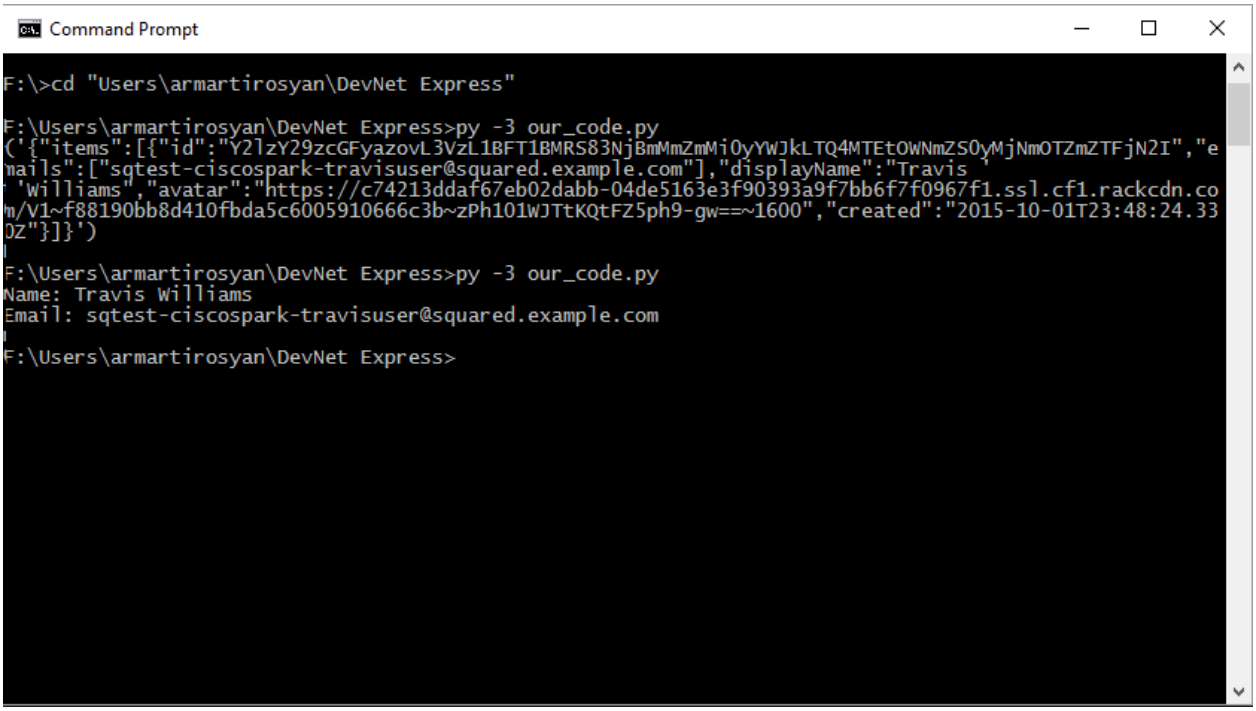
- As you can see the output is not very readable. Lets change our code, so it would print only information that interests us. To do that add `.json()` at the end of the `response = requests.get(url, headers=headers, verify=False)` line. Then you can add a for loop to iterate over items stored in the `response` dictionary and print only names and the email address. The additional code looks like this `for item in response["items"]:`
- `print('Name: ' + item['displayName'])`
- `print('Email: ' + item['emails'][0])`

The final code should look like this:

```
# Importing necessary modules
import requests
import json

#Disable warnings
requests.packages.urllib3.disable_warnings()

# Variables

url = "https://api.ciscospark.com/v1"
```

```
api_call ="/people"
access_token = "Bearer {access-token}" #Replace the
    {access-token} with your personal access token.

# Header information
headers = {
            "content-type" : "application/json;
    charset=utf-8",
            "authorization" : access_token,
          }

# Parameter variable. The email belongs to a bot user, but
    we can use it for our code
param = "?email=sqtest-ciscospark-
    travisuser@squared.example.com"

# Combine URL, API call and parameters variables
url +=api_call+param

response = requests.get(url, headers=headers,
    verify=False).json()

# Print user's name and email address from respond body.
for item in response["items"]:
    print('Name: ' + item['displayName'])
    print('Email: ' + item['emails'][0])
```

- Make the changes to the code and run it again. The output will be much cleaner and more readable.



Congratulations! You've written python code that can make REST API calls and retrieve information from Spark. If you don't fully understand the code above do not worry about it. The next lab goes through great detail and explains the moving pieces of the Python, so you can better understand what is happening in our own code.

Coding with Python and Parsing JSON

In this Learning Lab, we will go deeper into Python syntax and techniques for parsing JSON in python.

Objective

Completion Time: 45 minutes

- **Understand and apply the fundamentals of coding in Python**
- **Learn how to parse simple to slightly complex JSON using Python**

Prerequisites

Background

- We recommend that you complete the Coding 101 Rest Basics Learning Lab and Coding 102 Rest Python Learning Lab before you start this lab.

Python

19 To run the code samples, you need to have Python 3 installed on your machine.

20 If you are working on a DevNet Learning Lab PC at a DevNet event, Python 3.x + is already installed.

21 See How to Set up Your Own Computer section above for how to install Python on your own machine.

Python Requests Library

6 Some of the code samples use the Python Requests Library to simplify making REST API calls.

7 If you are working on a DevNet Learning Lab PC at a DevNet event, the Requests Library is already installed.

8 See How to Set up Your Own Computer section above for how to install the Requests Library on your own machine.

Clone Git Repo

- If you are working on a DevNet Learning Lab PC at a DevNet event,
  - Open the Git Command window by clicking on the Git CMD icon on the Task Bar or click on the Start button, then in the Run bar type: git cmd and press the Enter key.
- If you are working from your own workstation, on your desktop open a command terminal.
- Go to the root directory by typing: `cd \`
- Create a directory called 'C:\DevNetCode\yourname' by typing: `mkdir DevNetCode\<your-name>`

- For example: `mkdir DevNetCode\brTiller`

- Go to the new directory by typing: `cd \DevNetCode\<your-name>`
  - For example: `cd \DevNetCode\brTiller`
- Clone the devnet-express-code-samples repository from GitHub. Enter the command below.`git clone https://github.com/CiscoDevNet/devnet-express-code-samples`

```
C:\DevNetCode\brTiller>git clone https://github.com/CiscoDevNet/devnet-express-code-samples.git
Cloning into 'devnet-express-code-samples'...
remote: Counting objects: 148, done.
remote: Total 148 (delta 0), reused 0 (delta 0), pack-reused 148Receiving objects:  60% (89/148)
Receiving objects: 100% (148/148), 41.09 KiB | 0 bytes/s, done.
Resolving deltas: 100% (78/78), done.
Checking connectivity... done.
```

- Inside the directory you created you should now see directory 'devnet-express-code-samples'.

```
C:\DevNetCode\brTiller>dir
 Volume in drive C is System
 Volume Serial Number is 808B-CEEA

 Directory of C:\DevNetCode\brTiller

12/15/2015  03:16 PM    <DIR>          .
12/15/2015  03:16 PM    <DIR>          ..
12/15/2015  03:16 PM    <DIR>          devnet-express-code-samples
               0 File(s)              0 bytes
               3 Dir(s)  55,545,159,680 bytes free
```

Step 1. Python Version, Scripts and Interpretor

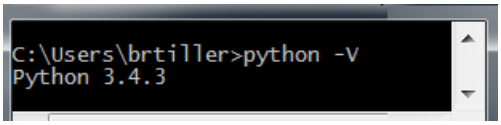
Checking Your Python Version

It's quite easy to check the version of Python installed on your system. Simply access a terminal window, then at the command

prompt type: python -V and press the return key. If you have multiple versions of python installed on your system or python 3 you may need to type either py -3 -V or python3 -V depending upon your operating system. See the graphic below for details.

- **Running Python 3: Checking your version**
  - python -V or py -3 -V [Windows]
  - python3 –V [Linux, OS X]

Here's an example of how to get the Python version.

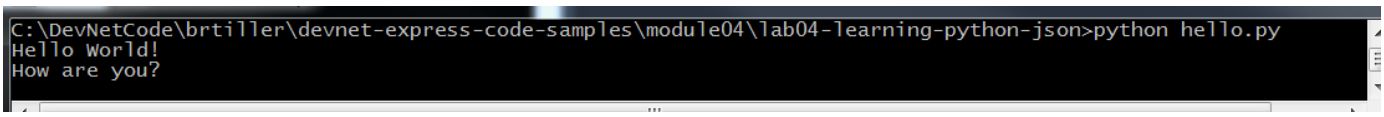


Running Your Python Script

To run a Python script is quite simple. At the terminal window command prompt enter python then enter the full name of the python script and press the return key. If you are not at the directory where the python script resides you will have to specify the full directory location of the script as well.

- **Running a python script**
  - python script.py or py -3 script.py [Windows]
  - python3 script.py [Linux, OS X]

Here's an example of running a Python Script.

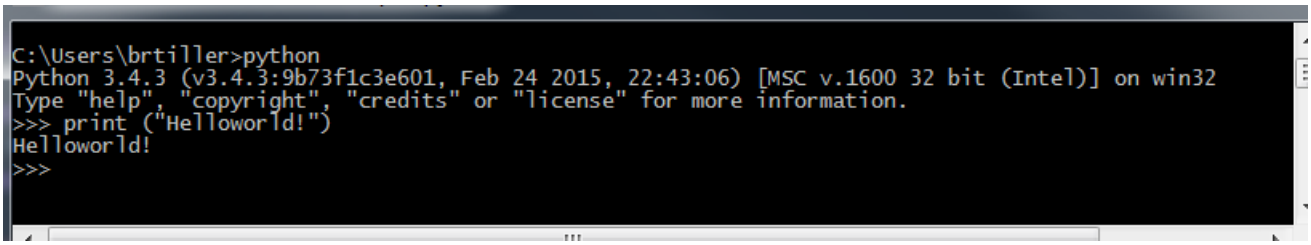


Using the Python Interpretor

At the command prompt you can enter the Python Interpretor by simply typing python and pressing the return key. Inside the Interpretor you can write and run very basic Python code. To exit type Cntrl-Z or quit().

- **Running Python Interpreter on the Command line**
  - python or py -3 [Windows]
  - python3 [Linux, OS X]
  - To exit: Type quit() or Ctrl Z

Here's an example of using the Python interpretor



Give it a try!

Get the Python Version. Use the interpretor

1. Open a terminal
2. Enter the appropriate python command for your operating system to get the version. For example, in Windows you would type python -V

then press the return key.

3. Enter the appropriate python command to start the interpretor. For example, in Windows you would type python then press the return key.
4. Enter print(" Hello World! How are you?") and press the return key.
5. To exit press the Ctrl key then z or enter quit()

Run a Python Script

1. Open a terminal and go to the directory you created earlier named: DevNetCode\<your-name>
   - In the Prerequisites section you had cloned the source code files from the git repository into this directory. Inside the directory you created you should see subdirectory devnet-express-code-samples. If that subdirectory does not exist, go back to the prerequisites section and follow the steps to create the directory and clone the git repository.
2. Go to directory module04\lab04-learning-python-json. In the terminal type: cd \DevNetCode\<your-name>\devnet-express-code-samples\module04\04-lab04-learning-python-json
3. To run the script type the python command and then the filename at the command prompt, and press the return key.
   - On Windows type: py -3 hello.py. Or type: python hello.py
   - On Mac OS or Linux type: python3 hello.py
4. The program should execute or display an error message.

Next Step: Learn about Python Scope, Variables, Operators and Simple Conditional Statements

Step 2. Python - Scope, Variables, Operators and Simple Conditional Statements

In this step we'll cover the very basics of scope, variables and conditional statements.

Python Basics

If you've written code in other languages you've noticed that statement blocks define scope (the range that a statement affects or variable exists) which is set by curly braces. These braces are used to define the start and end of the statement, and therefore, its scope. Python doesn't use curly braces to define scope. Instead it uses indentation. The indentation can be in the form of spaces or tabs, but must be consistent or the Python interpretor will complain.

Consider the Python code below.

```
print ("Hello World!")
num = 1
if num < 1:
   print ("I'm less than 1!")
   print ("Goodbye Cruel World!")
```

1. On the first line we print to the screen the text Hello World! which is commonly what developers do when they write their first application.
2. On the second line we create a variable which we call 'num' and assign it the value of 1. By the way we could have called that variable 'brett', 'a' and many other things. You can read more about naming variables.
3. On the next line we check the value of the variable we called 'num' asking if it is less than 1? The less than symbol **'<'** is an operator. Python has many operator types with the most of commonly used being Comparison, Assignment and Arithmetic. This 'if' statement is called a *conditional statement*because we are checking a condition. If the condition is true, then whatever is inside the scope of the conditional statement gets executed; otherwise it all gets skipped.
4. Notice that the next two statements are indented. That indentation signifies that they are inside the scope of the conditional statement.

Quiz. Scroll Down for Answers

1. If the python code above were run, what would be printed to the screen?
2. If you unindented the line print("Goodbye Cruel World!") what would be printed to the screen?
3. If you only changed the value of num to 0, what would be printed to the screen?

Answers

1. Hello World!
2. Hello World! Goodbye Cruel World!
3. Hello World! I'm less than 1! Goodbye Cruel World!

Give it a try!

1. Open a terminal and go to the directory created back in Step 1 named: **DevNetCode\<your-name>**
    - In the **Prerequisites section** of Step 1 you had cloned the source code files from the git repository into this directory. Inside the directory you created you should see subdirectory **devnet-express-code-samples**. If that subdirectory does not exist, go back to the Step 1 prerequisites section and follow the steps to create the directory and clone the git repository.
2. Go to directory **module04\lab04-learning-python-json**. In the terminal type: **cd \DevNetCode\<your-name>\devnet-express-code-samples\module04\04-lab04-learning-python-json**
3. To run the script type the python command and then the filename at the command prompt, and press the return key.
    - *On Windows type*: **py -3 helloworld.py**. Or type: **python helloworld.py**
    - *On Mac OS or Linux type*: **python3 helloworld.py**
4. The program should execute or display an error message.

Make these changes and run the helloworld.py script.

1. Open the file **helloworld.py**. For example, in Windows type: **notepad helloworld.py**
2. Make the change specified in the instructions in question 2 of the Quiz.
3. Save the file. If encoding type is an option select **UTF-8**.
4. Run the script.
5. Repeat steps 1 - 4, but make the change specified in the instructions in question 3 of the Quiz.

Write your first program

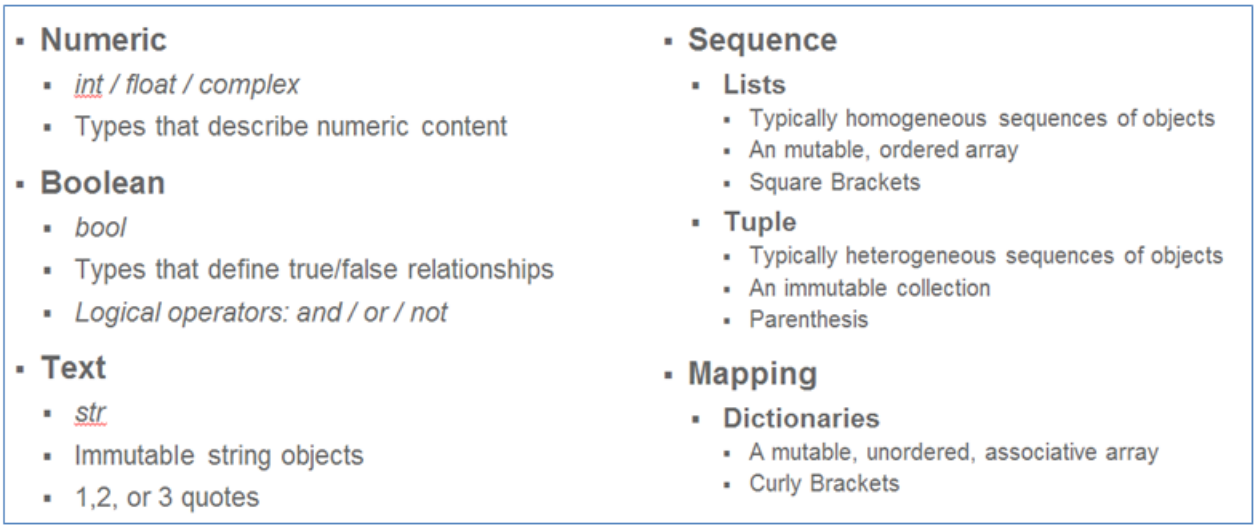
1. Create a file called myworld.py .
2. Write python code that does the following:
   a. Assigns a value to a variable
   b. Checks the value in the variable “Remember if (variable < value):”
   c. Prints statements within the *if conditional block* when the condition is true. Statements under the if conditional statement must be indented!

Next Step: Learn about Python Data Types

Step 3. Python Data Types

In this step we'll explore Python data types. There are many data types listed in the figure below. There are simple numeric ones like integers and floats. For our purposes the differences between a float and an integer is that a float has a decimal in it. For example 52.856 is a float while 52 is an integer. Boolean

data types can only be either True or False. Text is a data type with a non-numeric value. Text is more commonly called string. For example: " brett" is a string, so is 'brett123', but 123 is an integer.



There are also more complex data types such as list, tuples and dictionaries. Let's delve into these types.

A list contains any number of sequential elements and is defined by square brackets []. Here's a list ['Martha','Betty',5] . Assigning a list to a variable is as simple as var=['Martha','Betty',5]. Lists are mutable which means that they can be changed. You can add data to a list, modify and delete it or even sort it. Notice that lists can have different data types inside them.

A tuple contains any number of sequential elements and is defined by parenthesis (). Here's a tuple ('Brett',9,'Cisco'). Assigning a tuple to a variable is simple: var1=('Brett',9,'Cisco'). Tuples are similar to lists, but they are immutable which means they cannot be changed. You might use a tuple to group similar data. For example, referring to the tuple defined above the similarity is that my name is Brett and I've worked at Cisco for 9 years. Because tuples are immutable they are more efficient

meaning that their data can be accessed more quickly than lists. While lists and tuples are defined a little differently, they both have their data accessed the same manner which is by square brackets []. The first data element always starts at position zero. In the example in the list that was defined earlier if I entered the python statement print(var[0]), it would display Martha. If I entered the python statement print(var[1]), it would display Betty and so on. The same syntax applies to tuples. For the tuple defined above if I entered the python statement print(var1[0]), it would display Brett. If I entered the python statement print(var1[1]), it would display 9 and so on

A dictionary is a different than lists and tuples. Each element in a dictionary must contain a key followed by a value. This key value association is typically referred to as name-value pairs. Dictionaries are defined by curly braces {}. Here's a dictionary {" car" :" corvette" ," age" :7, " food" :" pizza" } . Assigning a dictionary to a variable is simple:

myvar={" car" :" corvette" ," age" :7, " food" :" pizza" } . The value in a dictionary is accessed by its key. For example if I entered the python statement print(myvar[" car"]), it would display corvette. If I entered the python statement print(myvar[" food"]), it would display pizza and so on. Dictionaries are sometimes also called maps or associative arrays .

Examples of how to define each Python data type is shown in the table below.

Type	Example
• Numeric: Integer, Float	x = 10 x = 1.0
• String	x= 'Mike'
• Boolean	y = True x = False
• List	my_list = [10, 20, 30]
• Tuple	my_tuple = ('Brett', 'Cisco', 'Cary',2015)
• Dictionary	my_dict = {"one":1, "two":2}
• Lists in Lists	my_list2=[[10,20,30], ['Cisco Live', 'May', 2016]]

Give it a try!

- In Step 2 you created and ran python script myworld.py . Now let's add to this script. a. Add a list with two elements and assign it to a variable. b. Add a tuple with three elements and assign it to a variable.. c. Add a dictionary with two name value pairs and assign it to a variable.. d. Print one value from each data type you created.
- Now open file data-types-values.py . Which values should be displayed for each data type? Run the script to check your answers.
- Modify the file data-type-values.py to create errors. For example, change the line print(my_dict[" green"]) to print(my_dict[" mean"]) . What error is displayed?

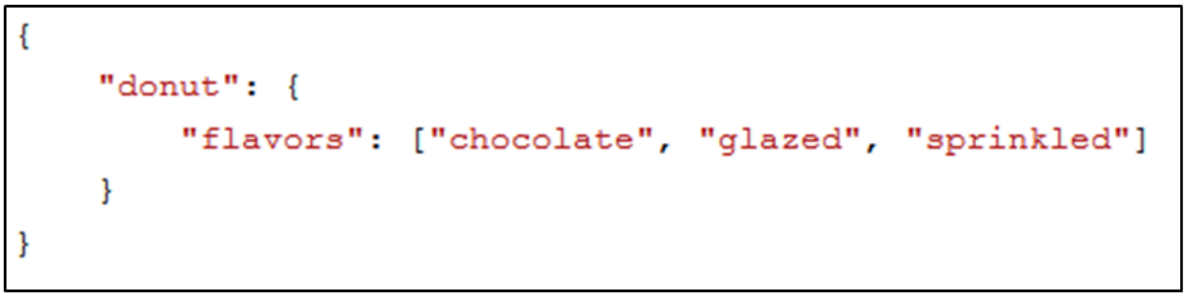
Next Step: Learn about JSON and How to Parse it Using Python

Step 4. Parsing JSON With Python

We are going to build on your new knowledge of Python data types, and learn about JSON and how to parse it with Python. JSON which stands for Java Script Object Notation consists of text-based name-value pairs making it simple for applications to store and exchange data. It's designed to be lightweight and readable as well as minimal so there's not excessive text which is sometimes been the complaint of XML which is another text-based protocol for exchanging and storing data.
The structure and parsing of JSON is very similar to Python dictionaries and lists.

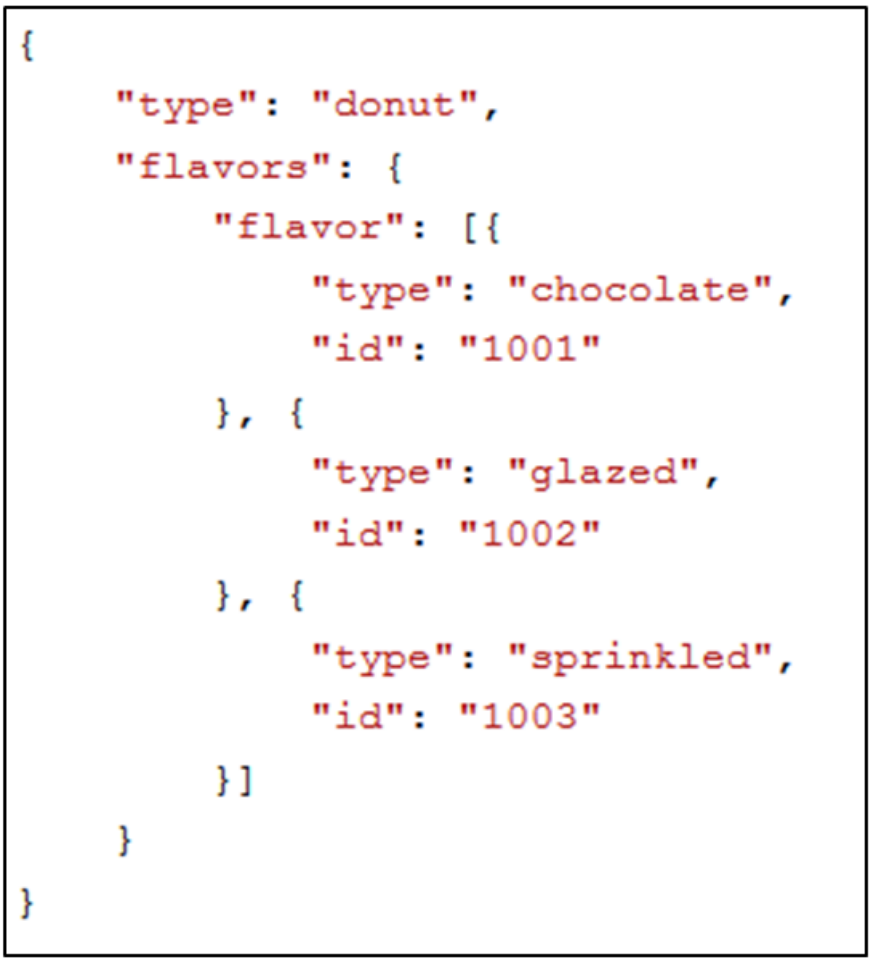
- In JSON data is set up in name value pairs just like a Python dictionary. However, in JSON dictionaries are referred to as objects. For example, {" car" :" volvo" , " fruit" :" apple" } is a JSON object. The object is assigned to a variable in the same manner as well: var={" car" :" volvo" , " fruit" :" apple" }. In addition, data is accessed the same way as when accessing data in a dictionary. For example, to get the value for fruit we would enter var[" fruit"] which would return apple. To display this value we enter print(var[" fruit"]) .
-
- As with Python, JSON also uses lists which it refers to as arrays. In JSON an array is typically nested in an object. For example, {" donut" :[" chocolate" ," glazed" ," sprinkled"]} . Let's assign this JSON object to a variable:
  var1={" donut" :[" chocolate" ," glazed" ," sprinkled"]}. Notice here the donut is the key, and the value is the array of the donut flavors. If I wanted to get the chocolate donut, (you're getting hungry aren't you?), I would access it by entering var1[" donut"] [0] which would return chocolate because it is the first element in the array. I could display the text by entering print(var1[" donut"] [0]).
-

- JSON can also nest objects in objects with arrays inside. For example:
  {" donut" :{" flavors" :[" chocolate" ," glazed" ," sprinkled"]}}. With this nesting we see that donut is the key with the object flavors being it's value. But notice that flavors is also a key and its value is the array of donut flavors. As a result, to get the chocolate donut we need to dig a little deeper. First we assign the JSON to a variable
  myvar={" donut" :{" flavors" :[" chocolate" ," glazed" ," sprinkled"]}} . We access chocolate by entering myvar[" donut"][" flavors"][0] . We can display it by entering print(myvar[" donut"][" flavors"][0]) . For viewing purposes JSON is typically formatted to separate data types from each other to make it easier to read. So the example above would typically look like the picture below, but is still parsed the same way.



- JSON will also include objects inside a list. For example:
  {" type" :" donut" ," flavors" :{" flavor" :[{" type" :" chocolate" ," id" :" 1001" },
  {" type" :" glazed" ," id" :" 1002" },{" type" :" sprinkled" ," id" :" 1003" }]}}. See if you can determine which elements are keys, values and are both keys and values. We need to dig a little deeper still to get our chocolate donut. First we assign it to a variable
  myvar1={" type" :" donut" ," flavors" :{" flavor" :[{" type" :" chocolate" ," id" :" 1001" },
  {" type" :" glazed" ," id" :" 1002" },{" type" :" sprinkled" ," id" :" 1003"

}]}} . We access chocolate by entering myvar1[" flavors"][" flavor"][0][" type"] . We can display it by entering print(myvar1[" flavors"][" flavor"][0][" type"]). As mentioned earlier for viewing purposes JSON is typically formatted. The example above would typically look like the picture below, but is still parsed the same way.



```
{
    "type": "donut",
    "flavors": {
        "flavor": [{
            "type": "chocolate",
            "id": "1001"
        }, {
            "type": "glazed",
            "id": "1002"
        }, {
            "type": "sprinkled",
            "id": "1003"
        }]
    }
}
```

Quiz. Scroll Down for Answers

fruit={" cherry" :" red" , " lemon" :" yellow" , " lime" :" greeen" }

food={“vegetables”:[“carrots”,“kale”,“cucumber”,”tomato”]}

cars={“sports”:{“Porsche”:“Volkswagon”,“Viper”:“Dodge”,“Corvette”:“Chevy”}}

donut={" type" :" donut" ," flavors" :{" flavor" :[{" type" :" chocolate" ," id" :" 1001" },

{" type" :" glazed" ," id" :" 1002" },{" type" :" sprinkled" ," id" :" 1003 " }]}}

Write the code to access and then display the requested items below

- Access the value for " lemon" from the fruit variable.
- Access " tomato" from the food variable.
- Access the value for " Viper" from the cars variable.
- Access the glazed value from the donut variable.
-
-

Answers

```
22  fruit[" lemon" ] print(fruit[" lemon" ])
23  food[" vegetables" ][3] print(food[" vegetables" ][3])
24  cars[" sports" ][" Viper" ] print(cars[" sports" ][" Viper" ])
25  donut[" flavors" ][" flavor" ][1][" type" ]
    print(donut[" flavors" ][" flavor" ][1][" type" ])
```

Give it Try!

```
9 Go to directory \DevNetCode\<your-name>\devnet-express-code-
    samples\module04\04-lab04-learning-python-json
10  Open file json_parse-1.py
11  Write Python code to print out one value for each of the json
    objects. Run the code.
```

Next Step: Learn how to write loops in Python.

Step 5. Loops

In this step we're going to learn about writing loops and functions in Python. We'll use this new skill to more easily parse JSON data that contains multiple elements in an array, tuple or dictionary.

There are many reasons to write loops in Python or any coding

language for that matter. The most common reason is that you have a list of data that you want to process. You might have noticed in the previous steps that when you wanted to access an object inside a JSON array that you had to specify the element number. Imagine how difficult that would be if you had to do the same thing for a hundred or a thousand objects! Loops make processing lists much easier because the loop will iterate through each object which you can then process via the source code that you write.

Let's start by looking at different types of loops in python

The first type of loop uses the range function to specify a fixed amount of iterations that will occur in a loop. The first example specifies there should be 5 iterations. By default Python will start counting from 0 so going from 0 to 4 is 5 iterations. In the next example we provide a starting and ending point. Here there will be three iterations since python excludes the last number, so the iterations will be 2,3,4. The in keyword is important because this keyword causes the value iterated over in the range to be assigned to the variable count. In this case that means that the variable name count is incremented with each iteration.

Similarly in the example for fruit in basket, basket is an array that we iterate over and with each iteration the value found is assigned to the variable named fruit which we can then parse. In the final two examples, while loops don't assign a value to a variable and are typically used for processing inputs or data that is not in a list. For example, the while True infinite loop might be used when waiting for user input.

for count in range(5)	Loop from 0 to 4
for count in range(2,5)	Loop from 2 to 4
for fruit in basket	For var in List statement
while count < 5	Must increment count
while True	Infinite loop. End with break statement

Using this new information about loops let's see how we can now process more JSON data much more quickly. In step 4 you parsed the JSON data about donuts to get the chocolate flavor. However, it would be much more time consuming and difficult if you wanted to get every donut flavor and it's corresponding id without using a loop. See the example below.

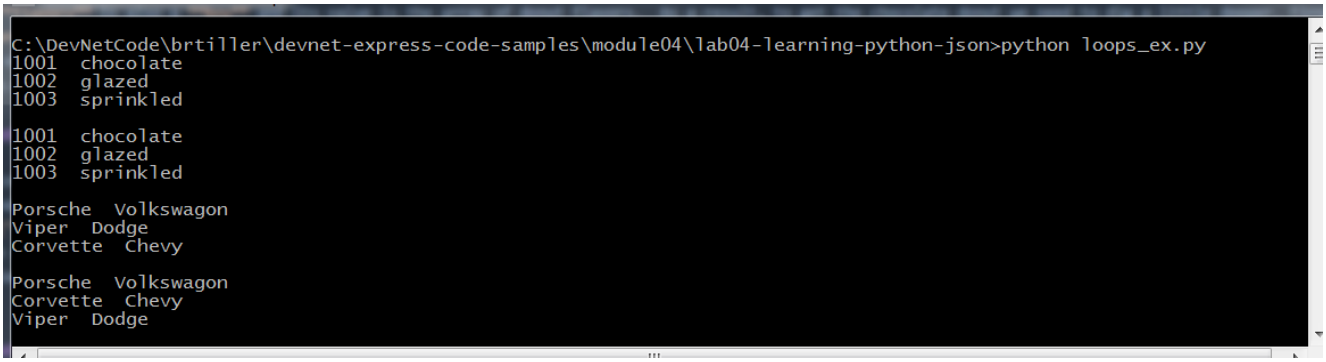
```
donut={"type":"donut","flavors":{"flavor":[{"type":"chocola
te","id":"1001"},
{"type":"glazed","id":"1002"},{"type":"sprinkled","id":"100
3"}]}}

#Parse JSON without loops
print(donut["flavors"]["flavor"][0]["id"] + "   " +
donut["flavors"]["flavor"][0]["type"])
print(donut["flavors"]["flavor"][1]["id"] + "   " +
donut["flavors"]["flavor"][1]["type"])
print(donut["flavors"]["flavor"][2]["id"] + "   " +
donut["flavors"]["flavor"][2]["type"])
print()

#Parse JSON with loops
for hungry in donut["flavors"]["flavor"]:
    print(hungry["id"] + "   " + hungry["type"])
```

You can also parse data using loops that are not included in arrays. For example in step 4 you parsed JSON data about cars. However, what if you wanted to print out the make of each car along with its corresponding model? You could use a loop to print out each without a problem. Let's see how it's done below with the old way versus the better way.

```
cars={"sports":{"Porsche":"Volkswagon","Viper":"Dodge","Cor
```

```
vette":"Chevy"}}

#Parse JSON without loops
print("Porsche  " + cars["sports"]["Porsche"])
print("Viper  " + cars["sports"]["Viper"])
print("Corvette  " + cars["sports"]["Corvette"])
print()

#Parse JSON with loops
for auto in cars["sports"]:
    print(auto + "  " + cars["sports"][auto])
```

Here's the output from running our script parsing each JSON data with the old and new way.



Let's review what we've learned so far! Review the code below and see if you can figure out what will be printed to the screen.

```
print ("Hello World!")
print()

for i in range(5):
    print (str(i) + " I'm alive!")
print()

basket=["apple","peach","pear","cherry"]
for fruit in basket:
    print (fruit)
print()

my_color={"red":1,"blue":2,"green":3}
for color in my_color:
    print (color + " %d" % my_color[color]);
```

```
print()

name = "Brett"
if name == "Brett":
    print ("Brett who?")
else:
    print ("Nice name!")
```

Give it Try!

- Go to directory \DevNetCode\<your-name>\devnet-express-code-samples\module04\04-lab04-learning-python-json
- Run the python script:loops.py . Then open the file to review the source code. Close it when you are done.
- Run the python script:loops_ex.py . Then open the file to review the source code. Close it when you are done.
- Open file json_parse-1.py . Using loops write Python code to print out all of the values for each of the json objects. Run the code.

Next Step: Putting it all Together with Python Functions

Step 6. Writing Your Own Functions

A function is a block of code that is run only when it's explicitly called. For example, print() is a function written in Python that you've called many times. Functions are written to modularize code to make it easy to read, easier to debug because it's located in one place and to reuse. Essentially, you don't want to write code over and over again that does the same thing. Instead you would put it into a function and then call that function whenever you need it.

Let's look at the structure of a function, then we'll look at a simple example. In Python a function is defined in the follow manner shown below. The keyword def specifies that a function is defined which is then followed by the name of the function

and optional arguments that are passed into it.

```
def <function name>():
    code

def <function name>(arguments):
    code
```

Let's look at some simple examples of functions. The first function named my_function simply prints a statement. The second function brett takes an argument called val which it passes to the function range and uses for looping.

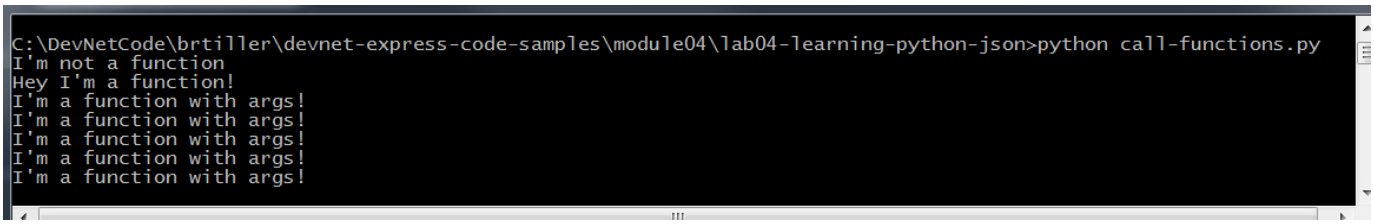
```
def my_function():
        print("Hey I'm a function!")


def brett(val):
    for i in range(val):
        print("I'm a function with args!")
```

Now let's look at these simple functions in a script to see how they're called. When this script is run, starting from the top of the script, the Python interpretor looks at what it should run now. It sees the call to print and executes that. It then sees the next two defined functions, makes note of them, but does not run

them because they are not explicitly called. Continuing down the script it then sees the call to function my_function and executes it. Finally, it sees the call to function brett with the argument of 5 passed in and executes it.

```
print ("I'm not a function")

def my_function():
        print("Hey I'm a function!")

def brett(val):
    for i in range(val):
        print("I'm a function with args!")

my_function()
brett(5)
```

As a result the output occurs as shown below.



Putting it all Together

In previous labs of this module you ran scripts that used the Spark APIs, but that was before you knew how to write Python. Now you've learned the basics about variables, operators, scope, parsing JSON, loops and functions. Review the functions and how they are called. Check out the dictionaries that are created and assigned to variables.

spark-room.py

This sample code uses the Spark rooms REST function call to retrieve a list of spark rooms that the bearer of the access token

belongs to.

```
# import requests library
import requests

#import json library
import json

accessToken = "" #put your access token here between the
quotes.


def setHeaders():
    accessToken_hdr = 'Bearer ' + accessToken
    spark_header = {'Authorization': accessToken_hdr,
'Content-Type': 'application/json; charset=utf-8'}
    return spark_header


def getRooms(theHeader):
    uri = 'https://api.ciscospark.com/v1/rooms'
    resp = requests.get(uri, headers=theHeader)
    print("SparkAPI: ")
    return resp.json()


header=setHeaders()
value=getRooms(header)
#format output to make easier to read then display it.
print (json.dumps(value, indent=4, separators=(',', ': ')))
```

To run this code sample:

- Go to directory \DevNetCode\<your-name>\devnet-express-code-samples\module04\04-lab04-learning-python-json
- Open the file spark-room.py. For example, in Windows type: notepad spark-room.py
- Assign your Spark Access Token to the variable " accessToken" . Recall that you can retrieve your Access Token by accessing and logging into URL https://developer.ciscospark.com/ and clicking on your member icon on the top right side of the web page.
- Save the file. If encoding type is an option select UTF-8.
- Type the python command and then the filename spark-room.py. and

press the return key.

- The program should execute or display an error message.

You should see a result displaying rooms to which you belong. Example output is below. For purposes of brevity some records have been removed.

Give it a Try

- Go to directory \DevNetCode\<your-name>\devnet-express-code-samples\module04\lab04-learning-python-json
- Run the python script:call-functions.py . Then open the file to review the source code. Close it when you are done.
- Run the python script spark-room.py. If you belong to one or more rooms, add at least one python function with a loop to parse and display the name of each room. Run the code.
- Open the python script spark-room-out.py. Add at least one python function with a loop to parse and display all of the data for each

room as assigned to the rooms variable. Run the file. Please note that the teamId and id field values have been changed to prevent curious developers from attempting to access room content.

Congratulations! You've completed Coding with Python and Parsing JSON!

The Mission

Instant messaging is a great way to interact with humans. Cisco Spark offers a platform for group chat, content sharing, and desktop sharing. Cisco Spark also offers REST APIs, end-to-end encryption and clients across multiple platforms making it a very viable choice for operational and DevOps workflows.

The reason for this lab is to start illustrating how developers interact with messaging tools like Cisco Spark. Imagine if an automated bot could send you a text when the network was down or to deploy an automated configuration change that has passed the engineering team's rigorous sanity checks. Developers perform this type of task routinely. They use messaging applications to approve deployments to production or receive alerts from tools when new code doesn't pass unit or integration tests. These alerts can be sent and acted upon from your phone, tablet, or messaging client.

Objective

Completion Time: 30 minutes

Use Python and Spark APIs to :

- **Create a Spark room**

- Add a user to the room and post messages into the room
- Retrieve and display data from the room

Prerequisites

Background

- We recommend that you first complete the learning labs in this module that preceded this mission.

Python

26 To run the code samples, you need to have Python 3 installed on your machine.

27 If you are working on a DevNet Learning Lab PC at a DevNet event, Python 3.x + is already installed.

28 See How to Set up Your Own Computer section above for how to install Python on your own machine.

Python Requests Library

12 Some of the code samples use the Python Requests Library to simplify making REST API calls.

13 If you are working on a DevNet Learning Lab PC at a DevNet event, the Requests Library is already installed.

14 See How to Set up Your Own Computer section above for how to install the Requests Library on your own machine.

Clone Git Repo

- If you are working on a DevNet Learning Lab PC at a DevNet event,
  - Open the Git Command window by clicking on the Git CMD icon on the Task Bar or click on the Start button, then in the Run bar type: git cmd and press the Enter key.
- If you are working from your own workstation, on your desktop open a command terminal.
- Go to the root directory by typing: `cd \`
- Create a directory called 'C:\DevNetCode\yourname' by typing:

`mkdir DevNetCode\<your-name>`

- For example: `mkdir DevNetCode\brTiller`

- Go to the new directory by typing: `cd \DevNetCode\<your-name>`
  - For example: `cd \DevNetCode\brTiller`
- Clone the devnet-express-code-samples repository from GitHub. Enter the command below.`git clone https://github.com/CiscoDevNet/devnet-express-code-samples.git`

```
C:\DevNetCode\brTiller>git clone https://github.com/CiscoDevNet/devnet-express-c
Cloning into 'devnet-express-code-samples'...
remote: Counting objects: 148, done.
remote: Total 148 (delta 0), reused 0 (delta 0), pack-reused 148Receiving object
Receiving objects: 100% (148/148), 41.09 KiB | 0 bytes/s, done.
Resolving deltas: 100% (78/78), done.
Checking connectivity... done.
```

- Inside the directory you created you should now see directory 'devnet-express-code-samples'.

```
C:\DevNetCode\brTiller>dir
 Volume in drive C is System
 Volume Serial Number is 808B-CEEA

 Directory of C:\DevNetCode\brTiller

12/15/2015  03:16 PM    <DIR>          .
12/15/2015  03:16 PM    <DIR>          ..
12/15/2015  03:16 PM    <DIR>          devnet-express-code-samples
               0 File(s)              0 bytes
               3 Dir(s)  55,545,159,680 bytes free
```

Your Mission

In this mission you will be given a partially completed Python script named spark_mission.py that creates a room, creates a new user, posts a message to a room and gets the room details. However, the script does not work properly because it's not completed. To finish the script you'll need to complete the tasks below. For assistance review the sample code provided in this module from the previous labs you completed. You will also want to review the Spark API Reference Guide . See the Tasks section below for more details.

spark_mission.py

```
import json
import sys
import requests

#MISSION: FILL IN THE REQUESTED DETAILS
ACCESS_TOKEN     = None #Replace None with your access
token. Shroud with quotes.
ROOM_NAME        = None #Replace None with the name of the
room to be created. Shroud with quotes.
YOUR_MESSAGE     = None #Replace None with the message that
you will post to the room. Shroud with quotes.


#sets the header to be used for authentication and data
format to be sent.
def setHeaders():
    accessToken_hdr = 'Bearer ' + ACCESS_TOKEN
    spark_header = {'Authorization': accessToken_hdr,
'Content-Type': 'application/json; charset=utf-8'}
    return (spark_header)


# creates a new room and returns the room id.
def createRoom(the_header,room_name):
    roomInfo = {"title":room_name}
    uri = 'https://api.ciscospark.com/v1/rooms'
```

```
    resp = requests.post(uri, json=roomInfo,
headers=the_header)
    var = resp.json()
    print("createRoom JSON: ", var)
    #MISSION: REPLACE None WITH CODE THAT PARSES AND
RETURNS THE ROOM ID.
    return(None)

# adds a new member to the room.  Member e-mail is
test@test.com
def addMembers(the_header,roomId):
    member = {"roomId":roomId,"personEmail":
"test@test.com", "isModerator": False}
    uri = 'https://api.ciscospark.com/v1/memberships'
    resp = requests.post(uri, json=member,
headers=the_header)
    print("addMembers JSON: ", resp.json())

#posts a message to the room
def postMsg(the_header,roomId,message):
    message = {"roomId":roomId,"text":message}
    uri = 'https://api.ciscospark.com/v1/messages'
    resp = requests.post(uri, json=message,
headers=the_header)
    print("postMsg JSON: ", resp.json())

#MISSION: WRITE CODE TO RETRIEVE AND DISPLAY DETAILS ABOUT
THE ROOM.
def getRoomInfo(the_header,roomId):
    print("In function getRoomInfo")
    uri = None
    if uri == None:
        sys.exit("Please add the uri call to get room
details.  See the Spark API Ref Guide")
    resp = requests.get(uri, headers=the_header)
    print("Room Info: ",resp.text)
    resp = resp.json()
    print("MISSION: Please add code to parse and display
details about the room.")


if __name__ == '__main__':
    if ACCESS_TOKEN==None or ROOM_NAME==None or
YOUR_MESSAGE==None:
        sys.exit("Please check that variables ACCESS_TOKEN,
```

```
ROOM_NAME and YOUR_MESSAGE have values assigned.")
    header=setHeaders()
    #passing the ROOM_NAME for the room to be created
    room_id=createRoom(header,ROOM_NAME)
    if room_id == None:
        sys.exit("Please check that function createRoom
returns the room ID value.")
    #passing roomId to members function here to add member
to the room.
    addMembers(header,room_id)
    #passing roomId to message function here to Post
Message to a room.
    postMsg(header,room_id,YOUR_MESSAGE)
    #MISSION: ADD FUNCTION CALL getRoomInfo(header,room_id)
    print("MISSION: ADD FUNCTION CALL
getRoomInfo(header,room_id)")
```

Your Tasks

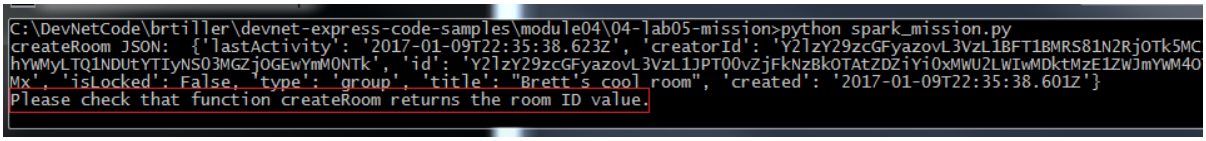
1. Open a terminal and go to the directory created earlier: DevNetCode\<your-name>

In the Prerequisites section you had cloned the source code files from the git repository into this directory. Inside the directory you created you should see subdirectory devnet-express-code-samples. If that subdirectory does not exist, go back to the prerequisites section and follow the steps to create the directory and clone the git repository.

2. Go to directory module04\lab05-mission. In the terminal type: cd \DevNetCode\<your-name>\devnet-express-code-samples\module04\lab05-mission
3. Open the file spark_mission.py. For example, in Windows type: notepad spark_mission.py
4. Whereever the uppercase word MISSION appears there is work to be done. A description will follow detailing the work required. Details are below as well.

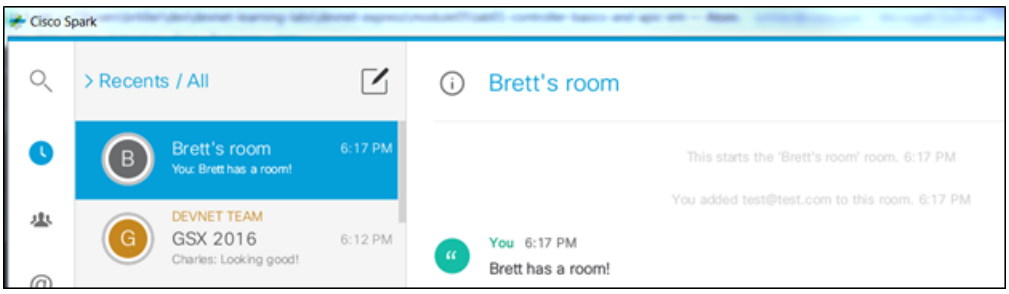
5 Fill in the data requested for ACCESS_TOKEN, ROOM_NAME and YOUR_MESSAGE. Recall that you can retrieve your Access Token by accessing and logging into URL https://developer.ciscospark.com/ and clicking on your member icon on the top right side of the web page. Also, to avoid any unexpected results, please use ASCII characters while naming the room.

6 Finish writing the code in function createRoom. See ## MISSION: WRITE CODE TO RETRIEVE AND DISPLAY DETAILS ABOUT THE ROOM. You will need to parse and return the room ID. Hint: Run the script which will fail with message: " Please check that function createRoom returns the room ID value." . Check the returned JSON listed as 'createRoom JSON', and figure out how to parse the room ID in the createRoom function. Hint: It's just one line of code.

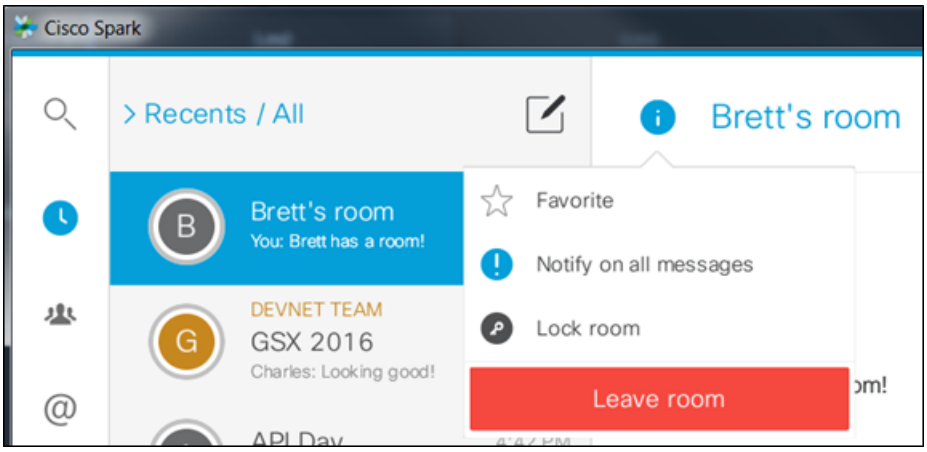


7 Write the code in function getRoomInfo. Specify the Spark REST API function in the uri variable. Use the Spark API Reference Guide to determine which function call to use. After making the function call write code to parse the REST data and display each of the values for the room. Finally, call the function.

8 After completing the script either open your Spark application or go to https://web.ciscospark.com . Then log in to see your newly created room with your message.



You can remove the room(s) by leaving them. Click on the (i) icon and select " Leave Room"



Congratulations! You've completed your Mission!