## 1. Requests

### 1.1 About Requests:

The `request` module accomplishes similar tasks as `urllib`, but it makes everything much easier. Because the syntax is a lot simpler, many companies and organizations prefer to use it. The NSA, Amazon, Google, PayPal, Twitter, and many others claim to use Requests internally.

```python
import urllib
req = urllib.request.urlopen("http://cs1301.com/")
print(req.read().decode("utf-8"))
import requests
req = requests.get("http://cs1301.com/")
print(req.text)
```

In both of the examples above, we found the source code of the home page for cs1301.com. That said, the `requests` module required a lot less effort on our part. As url requests get more complicated, the built-in `urllib` module gets harder and harder to use.

Unfortunately, python does not come with the requests module pre-installed, but it's free to download and use. In this document we'll walk you through the installation on your computer, so you can use it for future assignments/projects. We will also give you a brief introduction on how to start using the Requests module.

### 1.2 Quick Installation Guide:

If you're really comfortable with your computer, use these instructions. Otherwise, check out the more detailed instructions on the next page.

1. Open a Terminal Window or Command Prompt
2. Run: `pip3.6 install requests`
3. Test your installation:
    - Open a python shell and run: `import requests`

### 1.3 Detailed Installation Guide (skip if 1.2 was successful):

**Step #1** Open a Terminal Window or Command Prompt:
The command line is the best way to interact directly with your machine without the need for a graphical user interface. If you're using a Mac, this is called the *Terminal*. On Windows, this is called the *Command Prompt*.

Developers tend to prefer this, since there's a lot of different things we might want to ask the computer to do. Making a button for every possible command simply wouldn't be feasible. To install the `request` module. We need to open a command line window.

Authors: Daniel Barrundia and Joshua Diaddigo

On a Mac:

1. Press the command key and then your spacebar at the same time.
2. Type "Terminal" in the resulting window.
3. Press Enter.

On Windows 10, 8 or 8.1:

1. Right-click on the start button.
2. Choose Command Prompt.

On Windows 7, Vista or XP:

1. Click Start.
2. Click All Programs.
3. Click Accessories.
4. Choose Command Prompt.

***Step #2*** Update *pip* (skip this step, come back to it if you run into problems later):
On a Mac: copy/paste the following line into your terminal and press enter to run it:

```
pip3.6 install -U pip setuptools
```

On a Windows computer: copy/paste the following line into your command prompt window and press enter to run it:

```
python -m pip install -U pip setuptools
```

This step was successful if your output ends with something like this:

```
Successfully installed <lots of package names>
```

You will encounter an error here if you do not have python3.6 installed. It is important that you do. Head over to https://www.python.org/downloads/ and download and run the Python 3.6.0 installer.

**Step #3** Install Requests:
Copy/paste the following line into your command line window and press enter to run it:

```
pip3.6 install requests
```

You should see output similar to this:

```
Collecting requests
         Downloading requests-2.13.0-py2.py3-none-any.whl (584kB)
    100% |████████████████████████████████| 593kB 1.8MB/s
Installing collected packages: requests
Successfully installed requests-2.13.0
```

**Step #4** Test your installation:
At this point you should be able to open a Python shell and run:

```
import requests
```

If this does not result in a `ModuleNotFoundError`, you're good to go! If you are encountering an error, but step 3 resulted in the correct output, make sure you are testing with a Python3.6 shell. It is possible to have multiple versions of Python on your computer at once, but module installations will only apply to one version.

**1.4 Start Using Requests**

The best way to learn anything is to do some research first. This is Requests' official website and it has a fantastic User Guide.

http://docs.python-requests.org/en/master/#the-user-guide

Read and experiment with it as much as it as you want. For the purposes of this course you will just learn the basics, but you are always encouraged to go above and beyond, after all this is Georgia Tech.

http://docs.python-requests.org/en/master/user/quickstart/

## 2. APIs

**2.1 What is an API?**

API stands for Application Programming Interface. At the most basic level, the idea behind an API is that multiple programmers can work together without knowing much about how the other programmers' code is working. A unique aspect to an API is that it always involved sending web requests. In a way, an API is like a website that is built for your computer to browse (as opposed to being built for humans). Check out this and short article that explains what an API is:

https://medium.freecodecamp.com/what-is-an-api-in-english-please-b880a3214a82#.iwpcxbpz6

Using the `requests` module (from Part 1) to do API calls is really simple and powerful! I have created a small script for you to play around and check out the useful stuff you can do with a robust API. Hopefully you will learn how to do your own API calls and showoff the applications you create.

**2.2 What are APIs used for?**

Sometimes an API is used to present information to other developers. Imagine, for example, a satellite is collecting information that lots of scientists would like to use. In this situation, those with access to the satellite might create a web API that other scientists can send requests to. In this way, everyone could access the data without necessarily having access to the satellite.

Other times, an API is setup to process data. In this situation, a company might have created a really cool algorithm that they want people to be able to benefit from. However, in an effort to protect their intellectual property, the company might not want to give away the algorithm. In this situation, the company might make an API that receives input from the algorithm and outputs the result. This way, developers can use the algorithm without knowing how it works.

**2.3 What is JSON?**

JSON, at its most basic level, is simply a string representing information. Specifically, it is a string formatted in such a way that everyone agrees on how the information should be formatted. It is often used to make APIs simpler to work with.

An API will always return values as a string – just because that is how web requests are formatted. Imagine that one person made an API that sends "Your result is: 5" to the user. But then another person might make an API that sends "The result is: 5" to the user. Yet another person might make an API that simply sends "5". Writing a python program to get the result out of that string might become really tiresome. It would become even more complicated if we wanted the API to return something more complicated like a list in string form. For this reason, JSON was made.

JSON has specific ways that it represents things like lists, dictionaries and strings such that there is never any ambiguity about what the value and type of the data should be (even though it is currently in the form of a string).

**2.4 Brief API Tutorial**
For the purposes of this tutorial we will be using the MetaWeather API.

https://www.metaweather.com/

MetaWeather is an automated weather data aggregator that takes the weather predictions from various forecasters and calculates the most likely outcome. MetaWeather provides an API that delivers JSON over HTTPS for access to their data. You can jump ahead and read the documentation they provide for their API, do not worry if you don't understand anything yet. This is what this tutorial is for.

https://www.metaweather.com/api/

Let's start coding. Create a new python script and import the modules we will be using:

```python
import requests
from pprint import pprint
```

Note: If importing the `requests` module is giving you an exception, you must go back to Part1.

Now let's create some global variables for the URLs and a dictionary that maps some cities to known WOEID (Where on Earth ID):

```python
MAIN_URL = "https://www.metaweather.com"
location_search_url = "/api/location/search/"
location_url = "/api/location/{}/"
location_day_url = "/api/location/{}/{}/"

"""
Where to find WOEID (Where On Earth ID)? http://woeid.rosselliot.co.nz/
"""
LOCATIONS = {
    'Atlanta':'2357024',
    'London':'44418',
    'New York':'2459115',
    'San Francisco':'2487956',
    'Los Angeles': '2442047'
    }
```

Now that we have this we can actually start making some calls. Get familiar with how a JSON object looks like. Let's try this:

```
r = requests.get(MAIN_URL+location_url.format(LOCATIONS['Atlanta']))
data = r.json()
pprint(data)
```

Note: doing `pprint(data)` is not a typo. We want to use the pretty print library that python provides, to actually understand the data: https://docs.python.org/3/library/pprint.html

You should get something like this as a result:

```
{u'consolidated_weather': [{u'air_pressure': 1010.77,
                            u'applicable_date': u'2017-03-13',
                            u'created': u'2017-03-13T19:04:47.450570Z',
                            u'humidity': 83,
                            u'id': 5458355648724992,
                            u'max_temp': 6.665,
                            u'min_temp': 3.125,
                            u'predictability': 75,
                            u'the_temp': 7.66,
                            u'visibility': 5.970134415016304,
                            u'weather_state_abbr': u'lr',
                            u'weather_state_name': u'Light Rain',
                            u'wind_direction': 87.18987537005906,
                            u'wind_direction_compass': u'E',
                            u'wind_speed': 8.132140028235108},
                           ...] ,
 u'latt_long': u'33.748310,-84.391113',
 u'location_type': u'City',
 u'parent': {u'latt_long': u'32.678280,-83.222954',
             u'location_type': u'Region / State / Province',
             u'title': u'Georgia',
             u'woeid': 2347569},
 u'sources': [{u'crawl_rate': 180,
               u'slug': u'bbc',
               u'title': u'BBC',
               u'url': u'http://www.bbc.co.uk/weather/'},
              ...],
 u'sun_rise': u'2017-03-13T07:50:18.080762-04:00',
 u'sun_set': u'2017-03-13T19:44:03.013104-04:00',
 u'time': u'2017-03-13T16:01:17.427360-04:00',
 u'timezone': u'America/New_York',
 u'timezone_name': u'LMT',
 u'title': u'Atlanta',
 u'woeid': 2357024}
```

The variable `data` is now a dictionary with the response of our API call. We had to convert our response, which was a `json` object, to a python dictionary. The requests module does this for us, using the `json` module:

https://docs.python.org/3/library/json.html

When doing API calls, if you have a URL you can put it in your browser so you can see what response you should be getting. Try putting this in your browser:

https://www.metaweather.com/api/location/2357024/

Your browser should display the json object.

Let's start making some sense of all this data! One of the keys of our dictionary is: "consolidated_weather". The value associated with it is a list of the weathers of today and 5 days in the future. Let's see how we can get today's weather:

```
r = requests.get(MAIN_URL+location_url.format(LOCATIONS['Atlanta']))
data = r.json()
today = data['consolidated_weather'][0]['the_temp']
print("The current weather is: {} Celsius".format(today))
```

Our program should now give us the current weather.

What should we do if we wanted to get, not only today's weather, but also the weather of the next 5 days? What about getting the min and max temperature? Is it going to rain? We can answer all of these questions and more with simple API calls. Let's do another simple example:

```
r = requests.get(MAIN_URL+location_url.format(LOCATIONS['Atlanta']))
data = r.json()
print("This is your forecast for today and the next 5 days:")

for day in data['consolidated_weather']:
    date = str(day['applicable_date'])
    max_temp = int(day['max_temp'])
    min_temp = int(day['min_temp'])
    state = str(day['weather_state_name'])
    print("Day: {} - {} ({},{})".format(date, state,min_temp,max_temp))
```

Your result should look like this:

```
This is your forecast for today and the next 5 days:
Day: 2017-03-13 - Light Rain (3,6)
Day: 2017-03-14 - Heavy Cloud (0,9)
Day: 2017-03-15 - Light Cloud (-3,6)
Day: 2017-03-16 - Light Cloud (-2,10)
Day: 2017-03-17 - Heavy Cloud (5,17)
Day: 2017-03-18 - Showers (8,19)
```

^ Yes. It looks like a cold week here in Atlanta.

In T-Square under *>Resources >Class Notes* you can find a script for you to modify and keep testing other useful methods you decide to build.