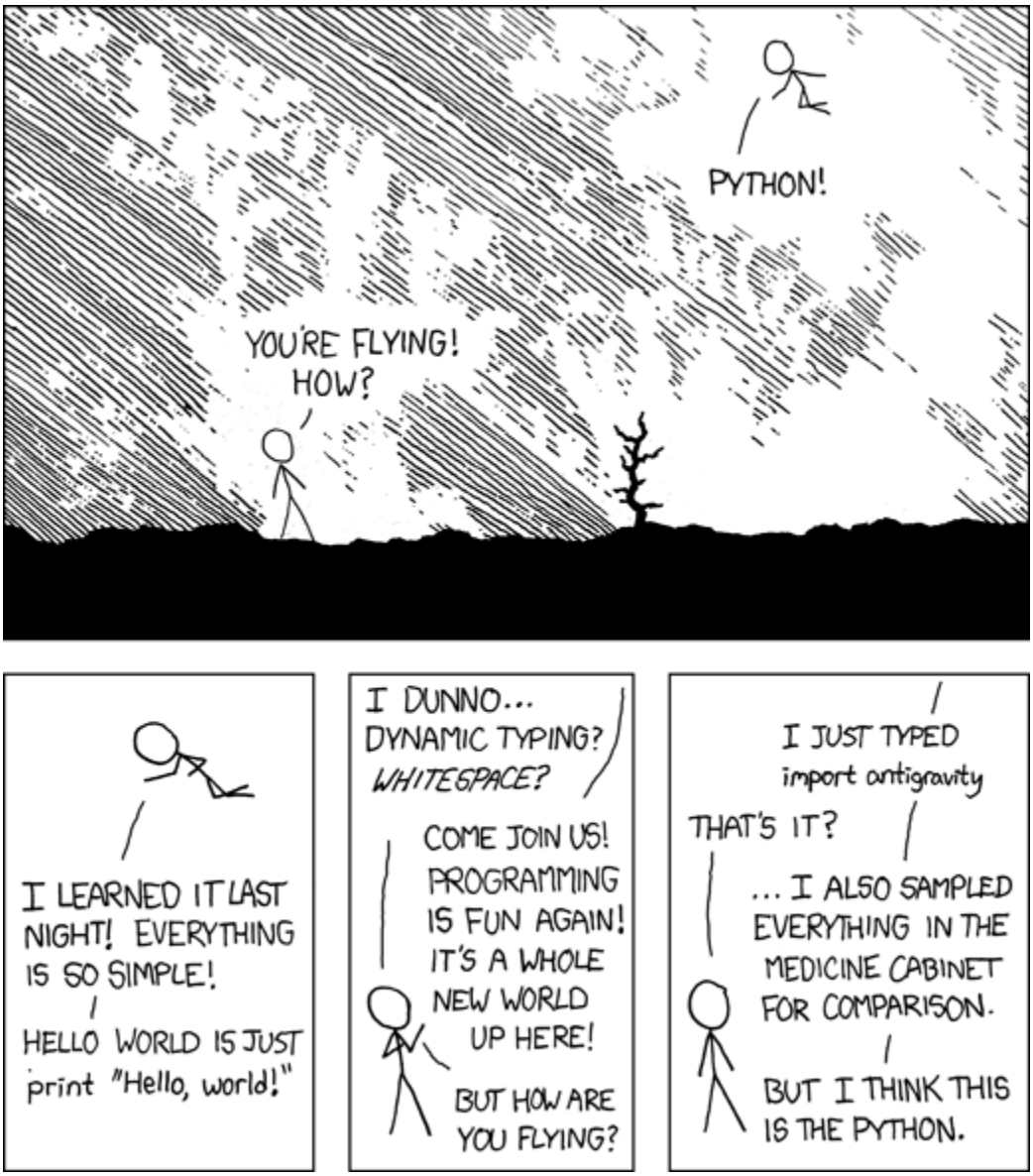
Controlling the Web with Python

Will Koehrsen Mar 10, 2018 · 9 min read

An adventure in simple web automation

Problem: Submitting class assignments requires navigating a maze of web pages so complex that several times I've turned an assignment in to the wrong place. Also, while this process only takes 1–2 minutes, it sometimes seems like an insurmountable barrier

Solution: Use Python to automatically submit completed assignments! Ideally, I would be able to save an assignment, type a few keys, and have my work uploaded in a matter of seconds. At first this sounded too good to be true, but then I discovered selenium, a tool which can be used with Python to navigate the web for you.



Obligatory XKCD

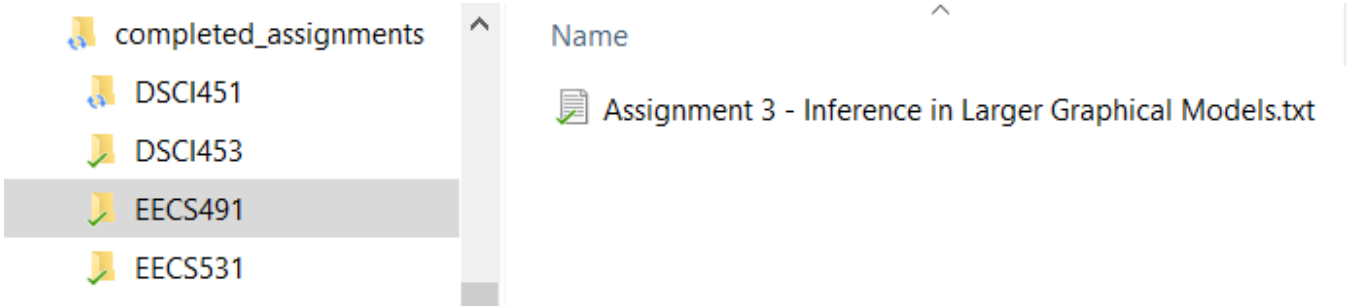
sequence of steps, this is a great chance to write a program to automate the process for us. With selenium and Python, we just need to write a script once, and which then we can run it as many times and save ourselves from repeating monotonous tasks (and in my case, eliminate the chance of submitting an assignment in the wrong place)!

Here, I'll walk through the solution I developed to automatically (and correctly) submit my assignments. Along the way, we'll cover the basics of using Python and selenium to programmatically control the web. While this program does work (I'm using it every day!) it's pretty custom so you won't be able to copy and paste the code for your application. Nonetheless, the general techniques here can be applied to a limitless number of situations. (If you want to see the complete code, it's available on GitHub).

Approach

Before we can get to the fun part of automating the web, we need to figure out the general structure of our solution. Jumping right into programming without a plan is a great way to waste many hours in frustration. I want to write a program to submit completed course assignments to the correct location on Canvas (my university's "learning management system"). Starting with the basics, I need a way to tell the program the name of the assignment to submit and the class. I went with a simple approach and created a folder to hold completed assignments with child folders for each class. In the child folders, I place the completed document named for the particular assignment. The program can figure out the name of the class from the folder, and the name of the assignment by the document title.

Here's an example where the name of the class is EECS491 and the assignment is "Assignment 3 — Inference in Larger Graphical Models".



The first part of the program is a loop to go through the folders to find the assignment and class, which we store in a Python tuple:

```
# os for file management
import os

# Build tuple of (class, file) to turn in
submission_dir = 'completed_assignments'

dir_list = list(os.listdir(submission_dir))

for directory in dir_list:
    file_list = list(os.listdir(os.path.join(submission_dir,
directory)))
    if len(file_list) != 0:
        file_tup = (directory, file_list[0])

print(file_tup)
```

```
('EECS491', 'Assignment 3 - Inference in Larger Graphical
Models.txt')
```

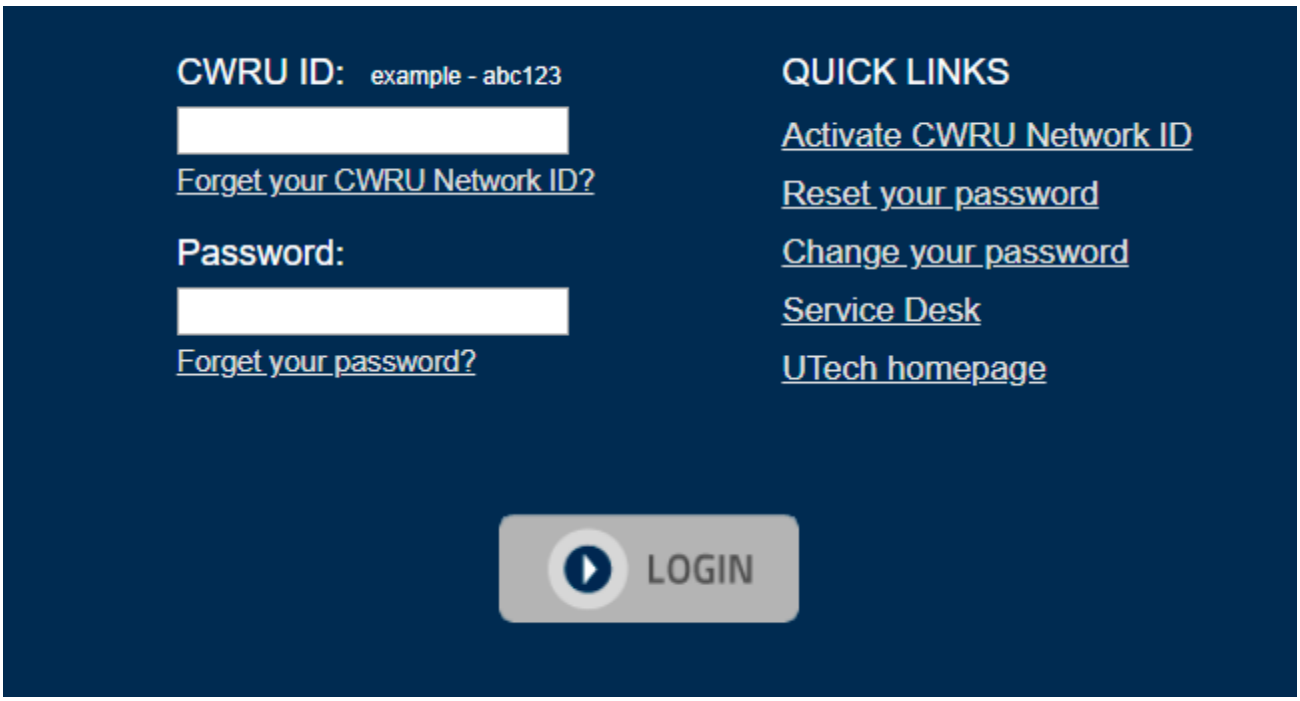
This takes care of file management and the program now knows the program and the assignment to turn in. The next step is to use selenium to navigate to the correct webpage and upload the assignment.

Web Control with Selenium

To get started with selenium, we import the library and create a web driver, which is a browser that is controlled by our program. In this case, I'll use Chrome as my browser and send the driver to the Canvas website where I submit assignments.

```
import selenium

# Using Chrome to access web
driver = webdriver.Chrome()

# Open the website
driver.get('https://canvas.case.edu')
```

When we open the Canvas webpage, we are greeted with our first obstacle, a login box! To get past this, we will need to fill in an id and a password and click the login button.



Imagine the web driver as a person who has never seen a web page before: we need to tell it exactly where to click, what to type, and which buttons to press. There are a number of ways to tell our web driver what elements to find, all of which use selectors. A selector is a unique identifier for an element on a webpage. To find the selector for a specific element, say the CWRU ID box above, we need to inspect the webpage. In Chrome, this is done by pressing “ctrl + shift + i” or right clicking on any element and selecting “Inspect”. This brings up the Chrome developer tools, an extremely useful application which shows the HTML underlying any webpage.

saw the following in developer tools. The highlighted line corresponds to the id box element (this line is called an HTML tag).

HTML in Chrome developer tools for the webpage

This HTML might look overwhelming, but we can ignore the majority of the information and focus on the `id = "username"` and `name="username"` parts. (these are known as attributes of the HTML tag).

To select the id box with our web driver, we can use either the `id` or `name` attribute we found in the developer tools. Web drivers in selenium have many different methods for selecting elements on a webpage and there are often multiple ways to select the exact same item:

```
# Select the id box
id_box = driver.find_element_by_name('username')

# Equivalent Outcome!
id_box = driver.find_element_by_id('username')
```

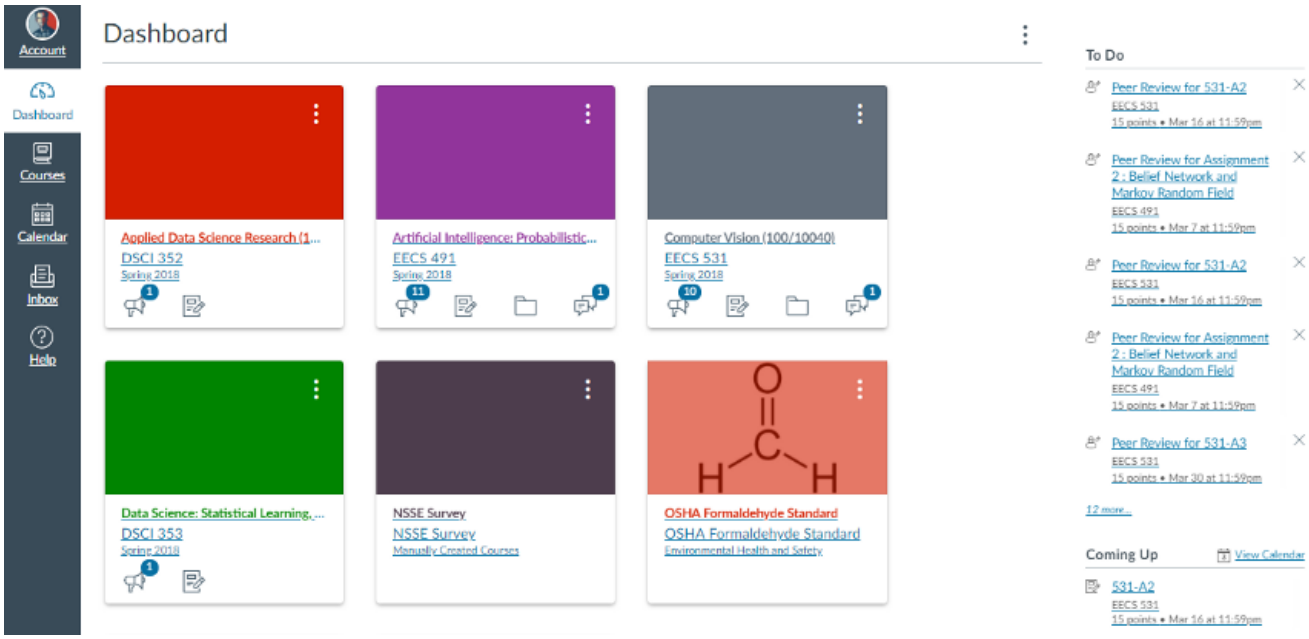
Our program now has access to the `id_box` and we can interact with it in various ways, such as typing in keys, or clicking (if we have selected a button).

```
# Send id information
id_box.send_keys('my_username')
```

We carry out the same process for the password box and login button, selecting each based on what we see in the Chrome developer tools. Then, we send information to the

```
# Find password box
pass_box = driver.find_element_by_name('password')

# Send password
pass_box.send_keys('my_password')

# Find login button
login_button = driver.find_element_by_name('submit')

# Click login
login_button.click()
```

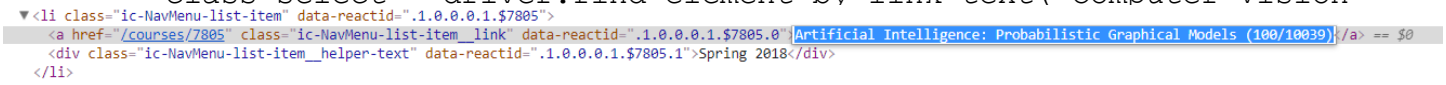
Once we are logged in, we are greeted by this slightly intimidating dashboard:



We again need to guide the program through the webpage by specifying exactly the elements to click on and the information to enter. In this case, I tell the program to select courses from the menu on the left, and then the class corresponding to the assignment I need to turn in:

```
# Find and click on list of courses
courses_button =
```

```
# Get the name of the folder
folder = file_tup[0]

# Class to select depends on folder
if folder == 'EECS491':
    class_select = driver.find_element_by_link_text('Artificial
Intelligence: Probabilistic Graphical Models (100/10039)')

elif folder == 'EECS531':
    class_select = driver.find_element_by_link_text('Computer Vision
```



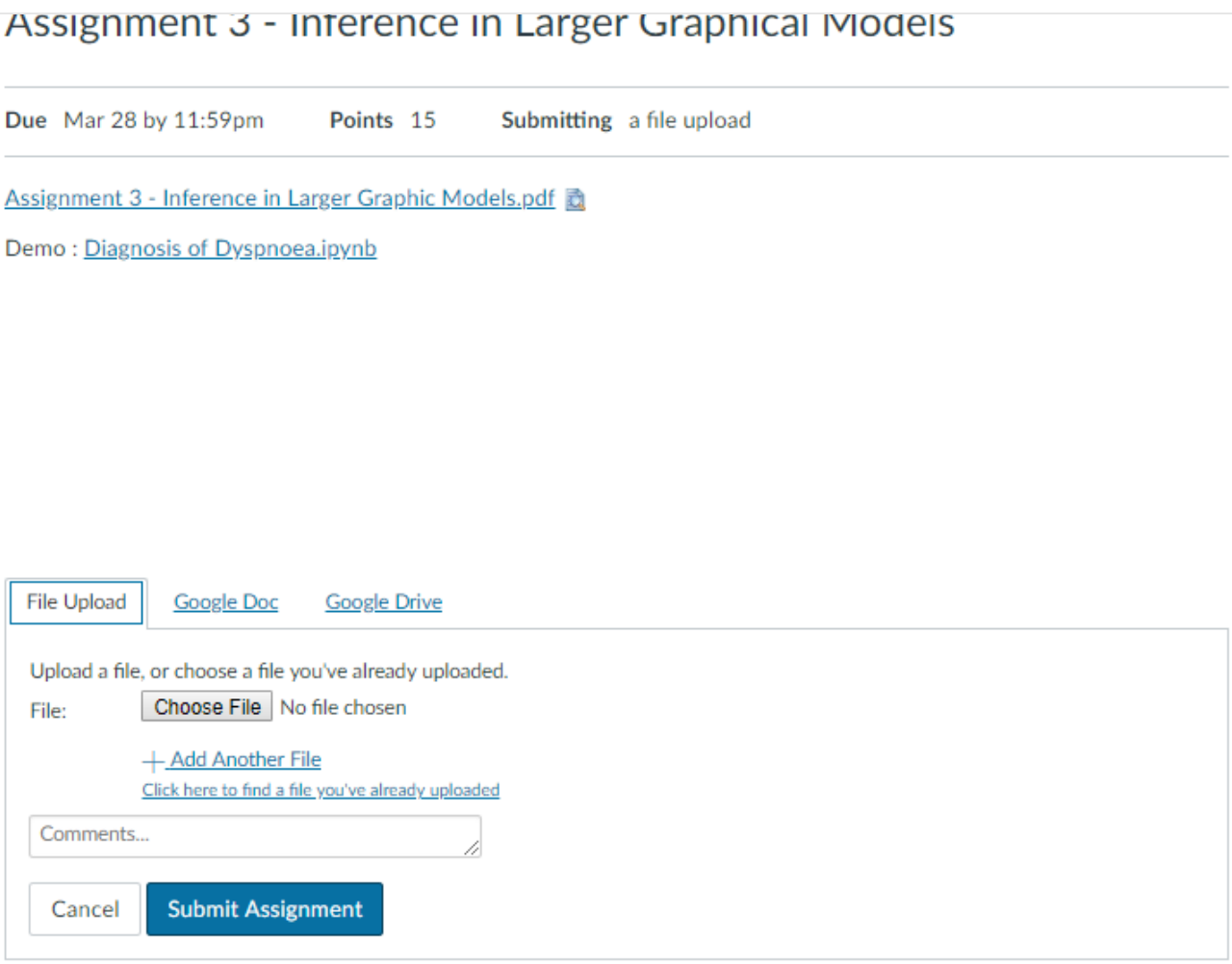
Inspecting the page to find the selector for a specific class

```
# Click on the specific class
class_select.click()
```

This workflow may seem a little tedious, but remember, we only have to do it once when we write our program. After that, we can run as many times as we want and the program will navigate through all these pages for us.

The program finds the correct class using the name of the folder we stored in the first step. In this case, I use the `find_element_by_link_text` to find the specific class. The “link text” for an element is just another selector we can find by inspecting the page. :

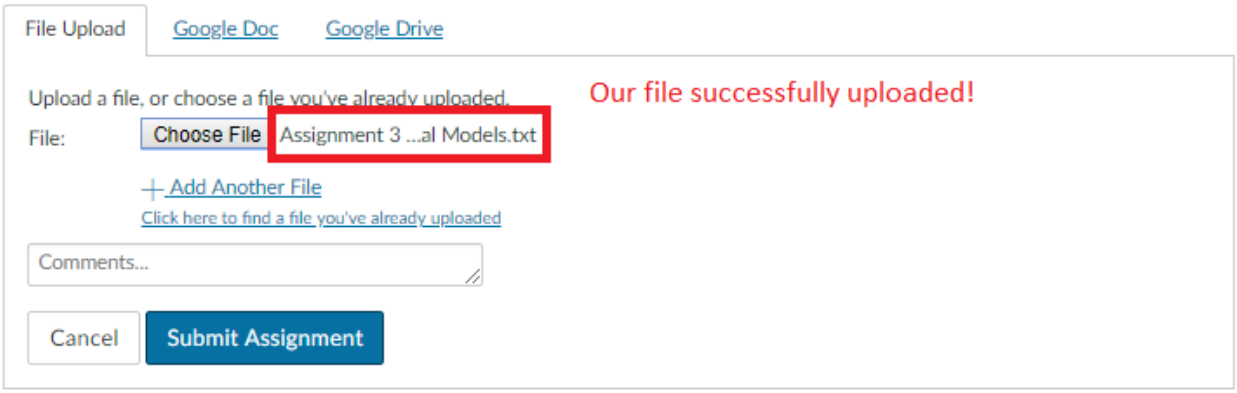
We use the same ‘inspect page — select element — interact with element’ process to get through a couple more screens. Finally, we reach the assignment submission page:



At this point, I could see the finish line, but initially this screen perplexed me. I could click on the "Choose File" box pretty easily, but how was I supposed to select the actual file I need to upload? The answer turns out to be incredibly simple! We locate the `Choose File` box using a selector, and use the `send_keys` method to pass the exact path of the file (called `file_location` in the code below) to the box:

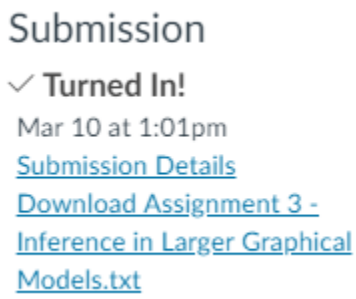
```
# Choose File button
choose_file = driver.find_element_by_name('attachments[0]
[uploaded_data]')

# Complete path of the file
file_location = os.path.join(submission_dir, folder, file_name)

# Send the file location to the button
```

That's it! By sending the exact path of the file to the button, we can skip the whole process of navigating through folders to find the right file. After sending the location, we are rewarded with the following screen showing that our file is uploaded and ready for submission.



Now, we select the "Submit Assignment" button, click, and our assignment is turned in!

```
# Locate submit button and click
submit_assignment = driver.find_element_by_id('submit_file_button')
submit_assignent.click()
```



Cleaning Up

File management is always a critical step and I want to make sure I don't re-submit or lose old assignments. I decided the best solution was to store a single file to be submitted in the `completed_assignments` folder at any one time and move files to

location:

```
    # Location of files after submission
Assignment 3 - Inference in Larger Graphical Models.txt Assignment for Class EECS491 successfully submitted at 2018-03-10 13:01:12.
Submitted assignment available at C:/Users/Will Koehrsen/Desktop/submitted_assignments\EECS491\Submitted Assignment 3 - Inference in Larger Graphical Models.txt.
    submitted_file_name)

    # Rename essentially copies and pastes files
    os.rename(file_location, submitted_file_location)
```

While the program is running, I can watch Python go to work for me:

Conclusions

The technique of automating the web with Python works great for many tasks, both general and in my field of data science. For example, we could use selenium to automatically download new data files every day (assuming the website doesn't have an API). While it might seem like a lot of work to write the script initially, the benefit comes from the fact that we can have the computer repeat this sequence as many times as want in exactly the same manner. The program will never lose focus and wander off to Twitter. It will faithfully carry out the same exact series of steps with perfect consistency (which works great until the website changes).

example is relatively low-risk as I can always go back and re-submit assignments and I usually double-check the program's handiwork. Websites change, and if you don't change the program in response you might end up with a script that does something completely different than what you originally intended!

In terms of paying off, this program saves me about 30 seconds for every assignment and took 2 hours to write. So, if I use it to turn in 240 assignments, then I come out ahead on time! However, the payoff of this program is in designing a cool solution to a problem and learning a lot in the process. While my time might have been more effectively spent working on assignments rather than figuring out how to automatically turn them in, I thoroughly enjoyed this challenge. There are few things as satisfying as solving problems, and Python turns out to be a pretty good tool for doing exactly that.

As always, I welcome feedback and constructive criticism. I can be reached on Twitter @koehrsen_will.

Python Education Programming Internet Towards Data Science