**Challenge Five**

So this file is going to be a little different from the others, since I want to introduce you to a topic that’s a little bit harder to grasp, but I think is probably one of the most rewarding skills to learn about. That being said, this is something that I myself have just recently learned very robustly, so don’t feel bad if it seems to be very complicated. Remember, if you have any questions you can always feel free to contact me, although for this topic, I may know about as much as you do.

Alright so let’s hop right in.

So right off the bat you’ll notice that this python file doesn’t look like the others. I didn’t include a tester file for the problem or any long-winded functions with lots of notes. The purpose of this file is mostly explicative, rather than for you to test your understanding of Python 3 concepts. What I want you to do is to follow along with what I’ve written in this Word document file, and try your best to come up with each solution before reading it on here.

The topic of this challenge is called **Web Scraping**. Basically web scraping is when your program goes online to find a certain piece of information, then returns with a result or an analysis of many results. Cool, right? I think this is one of the most powerful / interesting things you can learn from basic programming.

So for example you could create a program that goes to Dictionary.com and gets the definition of a word. Or you could get a program that returns the first paragraph of any Wikipedia article. Or maybe a program that goes to BestBuy and compares product prices.

So with web scraping you could, for example, create a program that goes to Dictionary.com and gets the definition of a word. Or you could make a program that returns the first paragraph of any Wikipedia article. Or maybe a program that goes to BestBuy and compares product prices. There are endless possibilities.

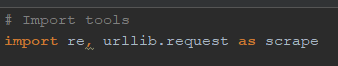
Today I’m going to take you through an example web scraping project step-by-step, and hopefully by the end you will have created something you’ll be proud of. **In this challenge we will create a program that, given your astrological sign, will return your horoscope!**

1. **Importing tools**
2. **Reconnaissance**
3. **Taking user input**
4. **Creating a partial URL / Base URL**
5. **Downloading the page**
6. **Regular expressions intro**
7. **Picking the matching text**
8. **Regular expressions outro**
9. **Displaying to the user**

**A. Importing tools**

Alright, so because the internet is such a vast and complicated place, python developers before us have already created tools to navigate it, via code! One of those tools is the urllib module, which gives us some pretty cool abilities, like the ability to download a page and sort through its HTML code. The other tool which we’ll be using is just called re. Re stands for ‘Regular Expression,’ and we’ll talk more about what that means later down, in part f.

Let me just break down that import statement in your challenge file to make sure that everything is clear.

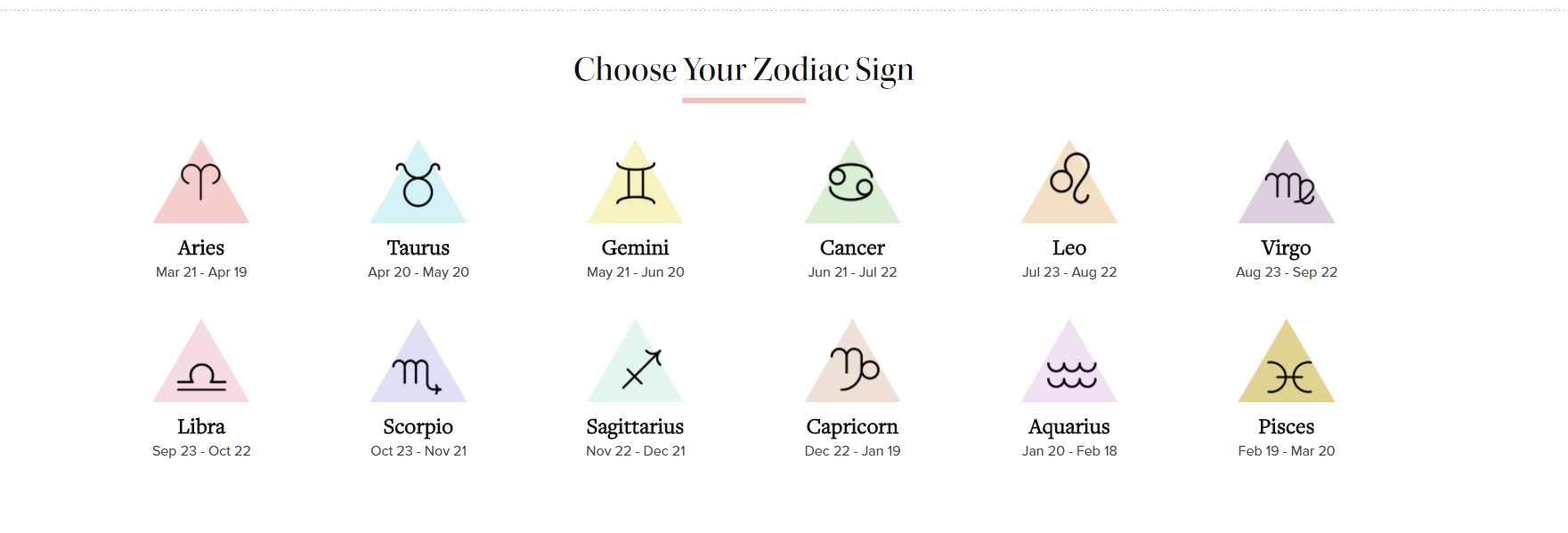


So in a nutshell, the “import” keyword tells Python that you want to use a library not included by default in Python 3. Then you can include all of the names of the modules you want to use, separated by commas, in this case we are using the libraries ‘re’ and ‘urllib.request’. Another trick to know, that makes things easier for later, is that you can use the ‘as’ keyword to assign an alias to a library. In other words, by doing ‘import urllib.request as scrape’ now every time I want to use the urllib.request library, all I have to do is type in ‘scrape’ instead.

**B. Reconnaissance**

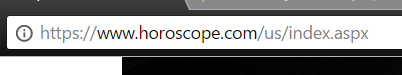
Alright, now let’s look around on the internet to see if we can find a site with a URL that’s easy to manipulate. Oh, ok! This one looks pretty good: [www.horoscope.com](http://www.horoscope.com)

But how did I know that this site had a url that we could easily manipulate? Well take a second a check out the site’s content:

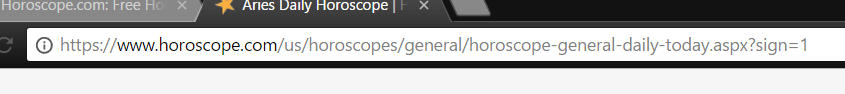


If you scroll down a little bit on the site’s homepage, you’ll see this cute little selection screen for your possible zodiac signs. Let’s click on the first one to see what happens to the URL…

Alright, so our URL went from this on the homepage:

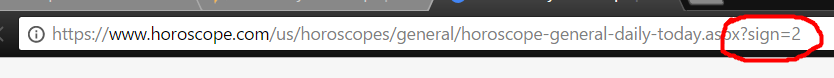


To this, on the Aries zodiac sign page:

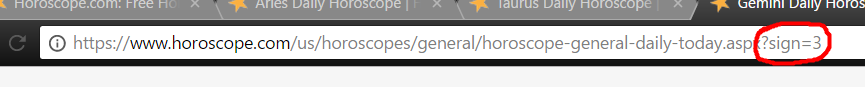


Alright, so there’s not really much of a pattern between those two pages. Let’s try another link.

What if I click on the Taurus zodiac sign page?...



Ah-ha! It looks like each zodiac sign is assigned a number index! Let’s check the next zodiac sign just to make sure…



Yep! Awesome. So now we know that the URL for each zodiac sign is about the same, except for the number at the very end! This will become helpful to us later! For now let’s focus on accepting the input of the user, and assigning his input a number!

**C. Taking user input**

Alright, this part should be fairly easy for you by now. In order for us to later manipulate the URL that we’ve discovered, let’s ask the user for his zodiac sign, then assign him a number based on what he chooses. For example, if I were to type in ‘scorpio’ as my zodiac sign, you should store the integer 8 in the variable **zodiac\_number**. It might be a good idea to produce an error message if a user misspells his zodiac sign, but I’ll leave that up to you.

For your reference, each zodiac sign and its corresponding number are below.

Aries - 1

Taurus - 2

Gemini - 3

Cancer - 4

Leo - 5

Virgo - 6

Libra - 7

Scorpio - 8

Sagittarius - 9

Capricorn - 10

Aquarius - 11

Pisces - 12

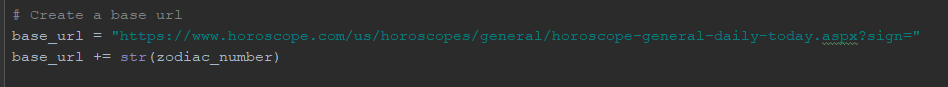
**D. Creating a partial URL / Base URL**

Alright, now let’s assume that the user has already inputted his zodiac sign, and the corresponding number has been stored in a variable called **zodiac\_number**. Remember this url from earlier? 

Well, now let’s store this as a string, into a variable. Create a new variable called **base\_url** and store the following string into it:

“**https://www.horoscope.com/us/horoscopes/general/horoscope-general-daily-today.aspx?sign=**”

You’ll notice that I omitted the number at the end of the url. Let’s add that number to the end of the string now, using string concatenation. Try this yourself, but if you’re lost, your code should look something like this:

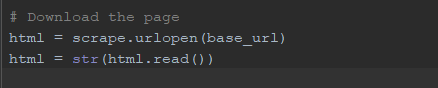


So now, stored in the variable base\_url, we have the full url of the horoscope page of the user. Next, we’ll download this page, so that later we can print out the user’s horoscope.

**E. Downloading the page**

Let’s create another variable. We can call this one html, because here we will download the HTML code from the horoscope website. In a nutshell, HTML stands for “HyperText Markup Language,” and basically tells your browser how to organize text and images on-screen. By storing this information as a string in a variable, we can see most of this site’s information, including the user’s horoscope!

Copy the following into your code editor:



All you really need to know about these lines is that by doing this, we store the html code found at the url **base\_url** and store it within the variable **html**. If you were to print out the html variable, you would see all of the html code from the site all garbled together.

**F. Regular Expressions intro**

Now this is probably the trickiest part, so don’t worry about understanding everything the first time you read this section. Regular expressions will be explained with more detail later on.

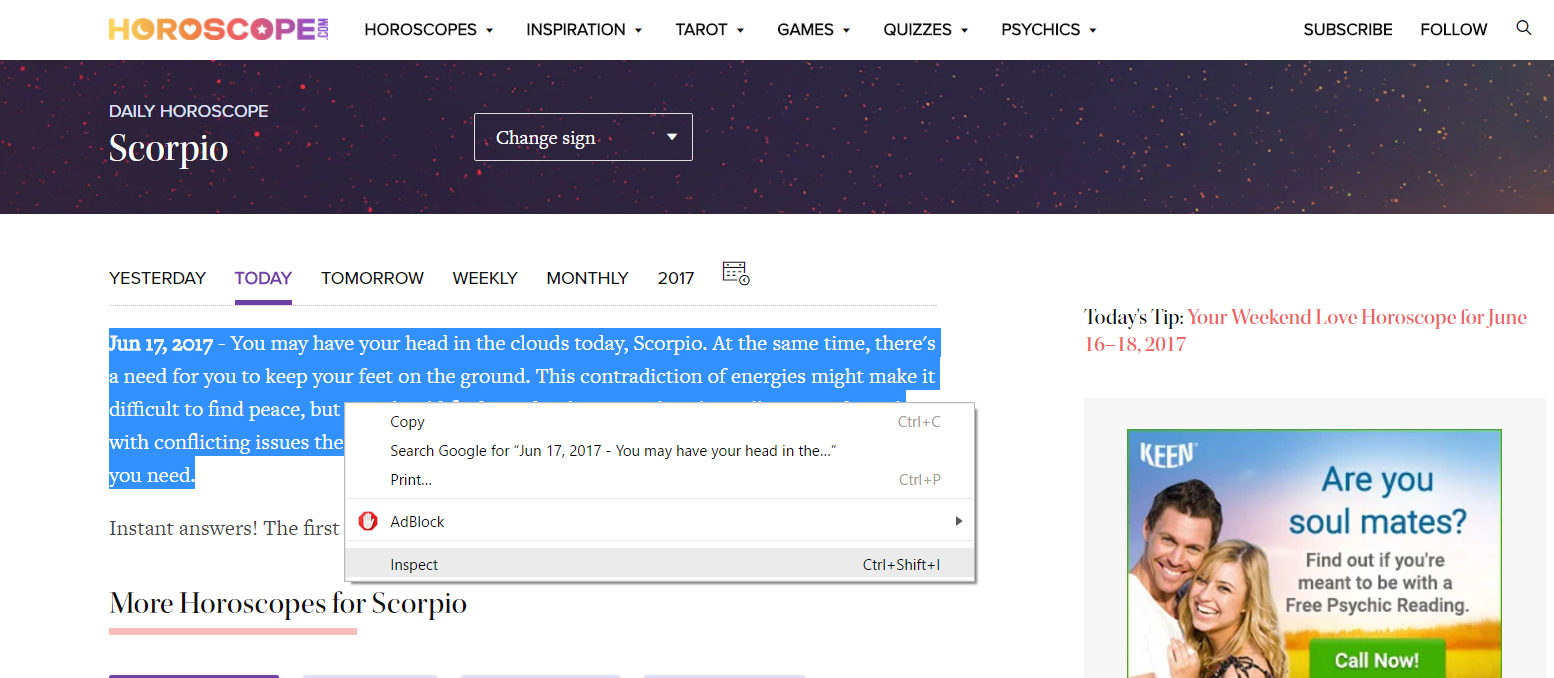
Regular expressions are basically a language within every programming language. Basically their purpose is to sort through large amounts of text, and to pull out information that they match with. For example, regular expressions could be used to pull all of the telephone numbers from a digital telephone book. All you would have to do is create a regular expression that recognizes phone numbers are a combination of 7 digits, with hyphens in between them. Humans do basically the same thing, if you were to even glance at the following number: (787)-123-4567, you would recognize that it’s a phone number.

Learning how to use regular expressions is a little bit beyond the scope of what I’m trying to do here (like I said, it’s like learning another language in-and-of itself) but for this project I want you to know a couple of things:

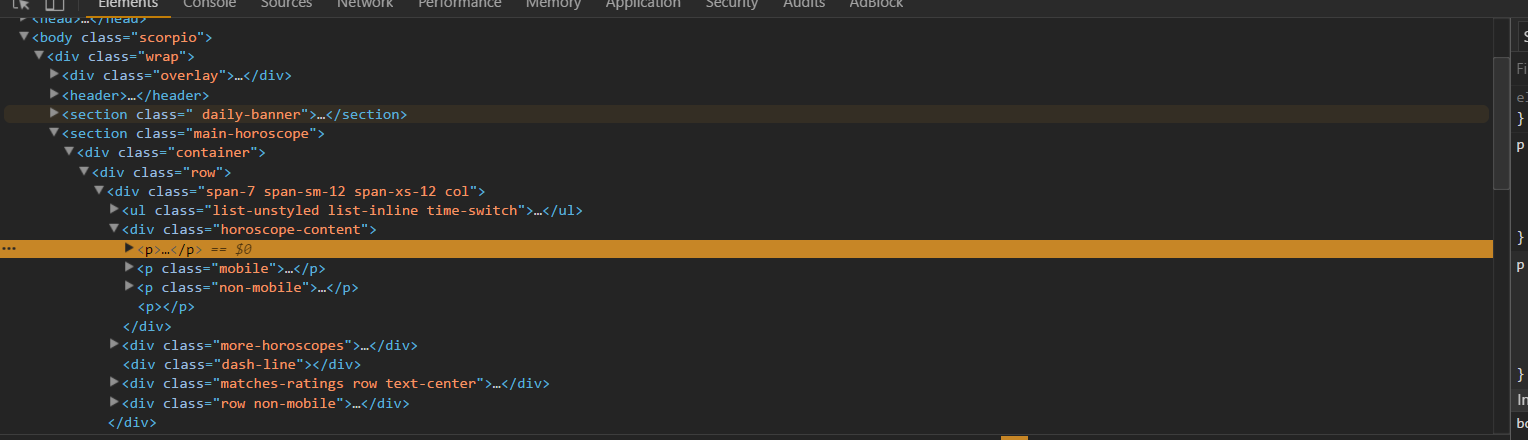
1. For our purposes, the phrase **.+?** is going to be the only real part of regular expressions that we’ll have to know. Basically **.+?** just means, match this phrase with anything.
2. The **\** (backslash) character is called an **escape character**. Basically what that means is that if you put a backslash before other characters, instead of being read as code, Python will read the characters literally. So for example in the string **“This is a quotation mark \“ “** the \” is just read as a quotation mark, and not as the end of the string.

Ok, so I know this is a lot to take in, and that you might not understand it all, but that’s ok! We are going to come back to this later, so for now let’s move on to the next part of this project.

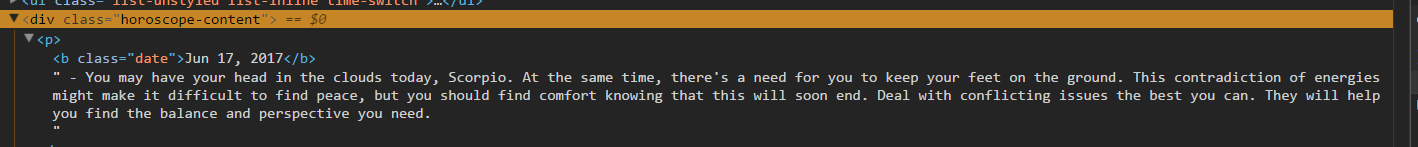
**G. Picking the Matching Text**

Let’s return to the webpage with the horoscope text stored on it. Highlight the text that we want to download from the page, right-click, and hit “Inspect.” 

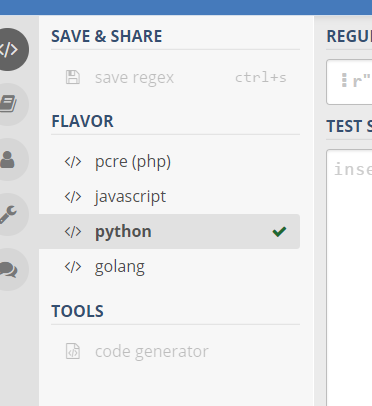
A similar panel to this one should show up after hitting “Inspect.”

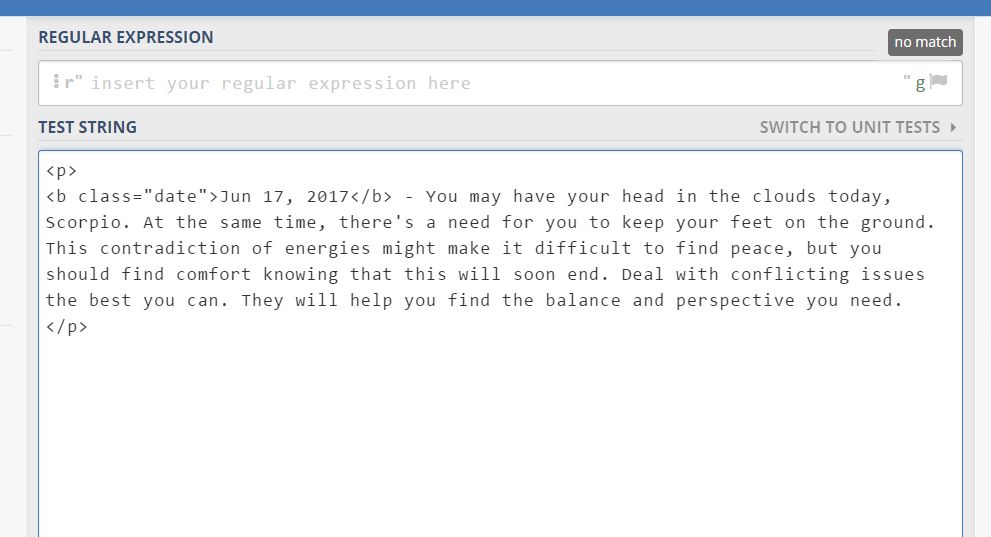


This is the HTML code of the website! Neat, right? If you open up the highlighted <p> tags (by clicking on the arrow on the left) you should notice that inside of the html code is the text that we’ve been looking for! Take a look:



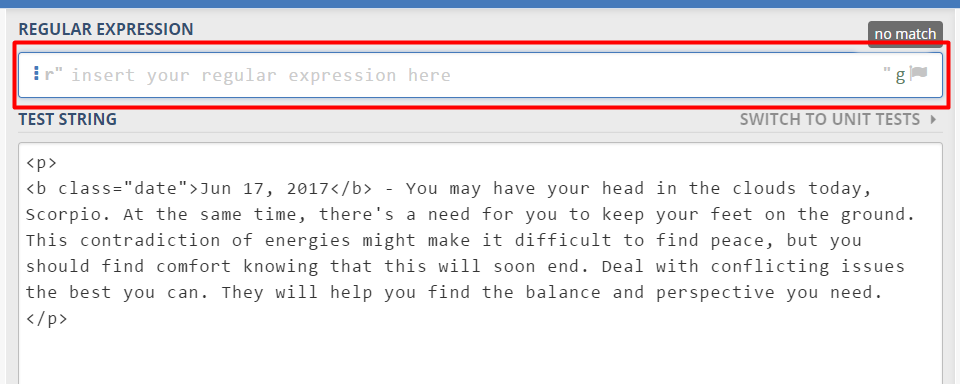
Right-click on the <p> tag, go under copy, and select Outer HTML. Now open up the following site: https://regex101.com/

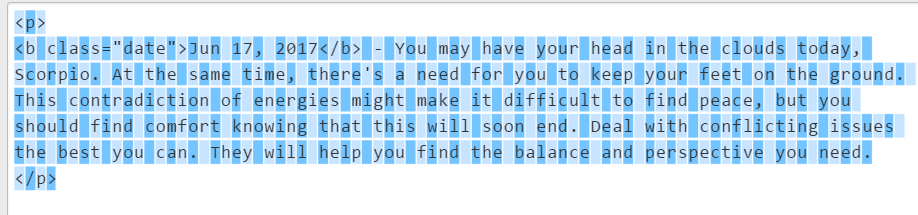
I like to use this website to make sure my regular expressions work before I start testing them. Make sure to go on the left panel of the site and select python as your programming language: 

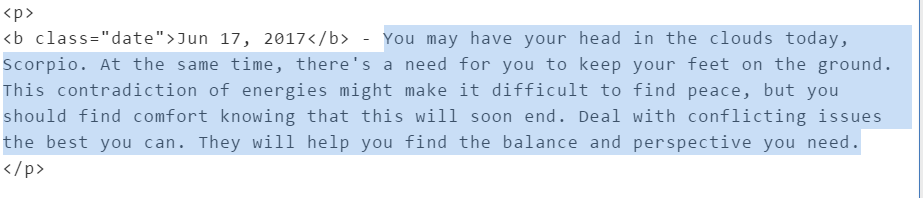
Paste into the section that says Test String. It should look something like this: 

**H. Regular expressions outro**

Now we need to enter our regular expression into the box above the “Test String” section

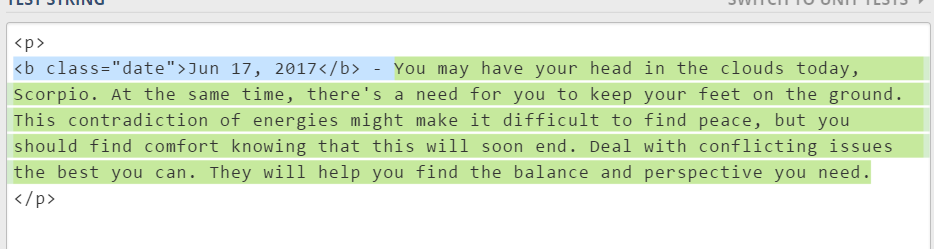


You’ll notice, if I enter the phrase **.+?**  every single character will match with the regular expression. 

You can refine the search by adding more information to the string. Remember, this is the only text I want to take from the HTML code: 

Copy and paste the following line into the regular expression field:

**<b class=\"date\">.+?</b> - (.+)**

I know, I know, that looks really intimidating, but watch what happens once you plug that line in: 

Python matches the text we are looking for! I’m not going to break down how this works into too much detail, since I think this is too complicated of a subject, and I am too poor of a teacher to explain this. If you want to know more about how/why this works, please check out: <https://automatetheboringstuff.com/chapter7/>

Now let’s move onto the last section.

**I. Displaying to the User**

Now that we know what the regular expression should be:

**<b class=\"date\">.+?</b> - (.+)**

Make your code look like the following:

