Introduction to programming with Python

Session 9

Objectives

- Quick review of what HTML is
- The find() string method
- Regular expressions
- Installing external libraries
- Using a web parser: BeautifulSoup
- Submitting data to a form using MechanicalSoup
- Fetching data in real time

The HTML language

- HTML is the standard language for creating content on the web.
- Every webpage is written in HTML.
- To see the source code of the webpage you are currently seeing, either right click and select "View page source", or from the menu of your browser, click on View and "View Source". Alternatively, you can use keyboard shortcut Ctrl+U (Command+U on a mac).

Example

Profile_Aphrodite.htm

```
<html>
<head>
  <meta http-equiv="Content-Type"
          content="text/html; charset=windows-1252">
  <title>Profile: Aphrodite</title>
  <link rel="stylesheet" type="text/css">
</head>
<body bgcolor="yellow">
  <center>
    <br><br>
    <img src="./Profile_ Aphrodite_files/aphrodite.gif">
    <h2>Name: Aphrodite</h2>
    <br><br>
    Favorite animal: Dove
    <br><br>
    Favorite color: Red
    <br><br>
    Hometown: Mount Olympus
  </center>
</body></html>
```

Grab all html from a web page

```
from urllib.request import urlopen
my_address = "http://www.staff.city.ac.uk/~ddimak/python
html_page = urlopen(my_address)
html_text = html_page.read().decode('windows-1252')
print(html_text)
```

What is the type of object that is returned?

Parsing a web page with a String's method

- You can use the **find()** method
- Example:

trinket Python3 Run Share Remix

main.py

```python
this_is_my_string = 'Programming in python'
string_to_find = input('Enter a string to find in \'%s\': ' % this_is_my_string)
index_found = this_is_my_string.find(string_to_find)
print(index_found)
print(this_is_my_string[index_found])
```

Find a word between 2 other words

trinket | Run | Modules | Share | Remix

main.py

```python
my_string = 'some text with a special word ' \
            '<strong>Equanimity</strong>'
start_tag = "<strong>"
end_tag = "</strong>"
start_index = my_string.find(start_tag) + len(start_tag)
end_index = my_string.find(end_tag)
# We extract the text between
# the last index of the first tag '>'
# and the first index of the second tag '<'
print(my_string[start_index:end_index])
```

Parsing the title with the find() method

```
from urllib.request import urlopen
my_address = "http://www.staff.city.ac.uk/~ddimak/python
             "practice/Profile_Aphrodite.htm"
html_page = urlopen(my_address)
html_text = html_page.read().decode('windows-1252')
start_tag = "<title>"
end_tag = "</title>"
start_index = html_text.find(start_tag) + len(start_tag)
end_index = html_text.find(end_tag)
print(html_text[start_index:end_index])
```

Limitation of the find() method

- Try to use the same script for extracting the title of Profile_Poseidon.htm

```
from urllib.request import urlopen
my_address = "http://www.staff.city.ac.uk/~ddimak/python
             "practice/Profile_Poseidon.htm"
html_page = urlopen(my_address)
html_text = html_page.read().decode('windows-1252')
start_tag = "<title>"
end_tag = "</title>"
start_index = html_text.find(start_tag) + len(start_tag)
end_index = html_text.find(end_tag)
print(html_text[start_index:end_index])
```

Limitation of the find() method

- Do you see the difference? We are not getting what we want now:

```
<head><meta http-equiv="Content-Type" content="text/html
<title >Profile: Poseidon
```

- This is because of the extra space before the closing ">" in <title >
- The html is still rendered by the browser, but we cannot rely on HTML being 100% compliant if we want to parse a web page.

Regular expressions

- They are used to determine whether or not a text matches a particular pattern
- We can use them thanks to the **re** module in python
- They use special characters to represent patterns: ^, $, *, +, ., etc...

re.findall() using *

- The asterisk character * stands for "zero or more" of whatever came just before the asterisk
- **re.findall():**
  - finds any text within a string that matches a given pattern i.e. regex
  - takes 2 arguments, the 1st is the regex, the 2nd is the string to test
  - returns a list of all matches

```
# re.findall(<regular_expression>, <string_to_test>)
```

Interactive example

trinket | Run | Modules | Share | Remix

main.py

```python
import re

print(re.findall("ab*c", "ac"))
print(re.findall("ab*c", "abcd"))
print(re.findall("ab*c", "acc"))
print(re.findall("ab*c", "abcac"))  # 2 found
print(re.findall("ab*c", "abdc"))  # nothing found
```

re.findall() case insensitive

- Note that re.findall() is case sensitive

```
re.findall('ab*c', 'ABC') # nothing found
```

- We can use a 3rd argument **re.IGNORECASE** to ignore the case

```
re.findall('ab*c', 'ABC', re.IGNORECASE) # ABC found
```

re.findall() using . (period)

- the period **.** stands for any single character in a regular expression
- For instance we could find all the strings that contain letters "a" and "c" separated by a single character as follows:

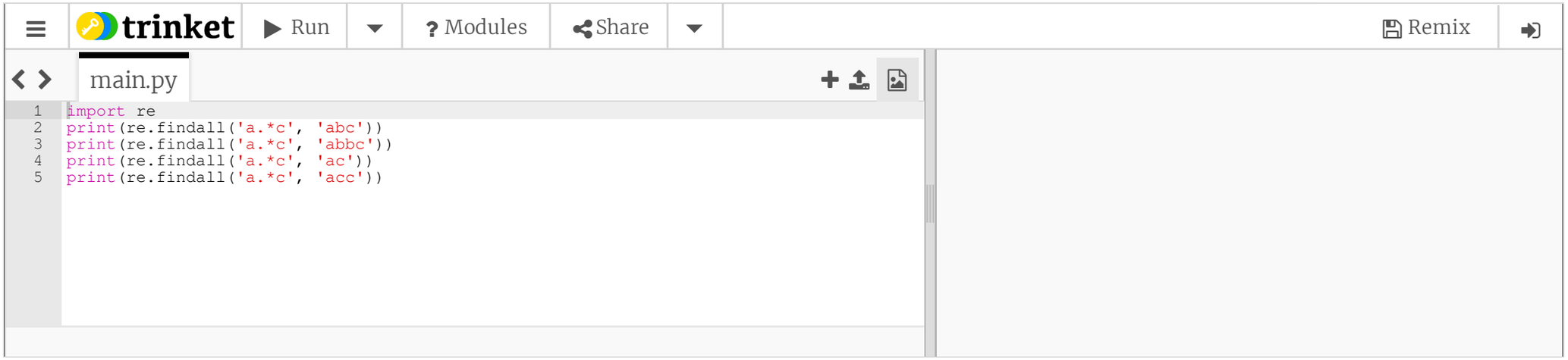
trinket ▶ Run ? Modules Share Remix

main.py

```
import re
print(re.findall('a.c', 'abc'))
print(re.findall('a.c', 'abbc'))
print(re.findall('a.c', 'ac'))
print(re.findall('a.c', 'acC', re.IGNORECASE))
```

re.findall() using .* (period asterisk)

- the term .* stands for any character being repeated any number of times
- for instance we could find all the string that starts with "a" and ends with "c", regardless of what is in between with:



re.search()

- **re.search():**
  - searches for a particular pattern inside a string
  - returns a MatchObject that stores different "groups" of data
  - when we call the group() method on a MatchObject, we get the first and most inclusive result

```
import re
match_results = re.search('ab*c', 'ABC', re.IGNORECASE)
print(match_results.group())  # returns ABC
```

re.sub()

- re.sub()
  - allows to replace a text in a string that matches a pattern with a substitute (like the replace() string method)
  - takes 3 arguments:
    1. regex
    2. replacement text
    3. string to parse

```
my_string = "This is very boring"
print(my_string.replace('boring', 'funny'))
import re
print(re.sub('boring', 'WHAT?', my_string))
```

greedy regex (*)

- **greedy** expressions try to find the longest possible match when character like * are used
- for instance, in this example the regex finds everything between '<' and '>' which is actually the whole '*<replaced> if it is in <tags>*'

```
my_string = 'Everything is <replaced> if it is in <tags>
my_string = re.sub('<.*>', 'BAR', my_string)
print(my_string)  # 'Everything is BAR'
```

non-greedy regex (*?)

- ***?**
  - works the same as * BUT matches the shortest possible string of text

```
my_string = 'Everything is <replaced> if it is in <tags>
my_string = re.sub('<.*?>', 'BAR', my_string)
print(my_string)  # 'Everything is BAR if it is in BAR'
```

Use case: Using regex to parse a webpage

- Profile_Dionysus.htm
- We want to extract the title:

```
<TITLE >Profile: Dionysus</title  / >
```

- We will use the regular expression for this case

Use case: solution

```
import re
from urllib.request import urlopen
my_address = "http://www.staff.city.ac.uk/~ddimak/python
html_page = urlopen(my_address)
html_text = html_page.read().decode('windows-1252')
match_results = re.search("<title .*?>.*</title .*?>", h
title = match_results.group()
title = re.sub("<.*?>", "", title)
print(title)
```

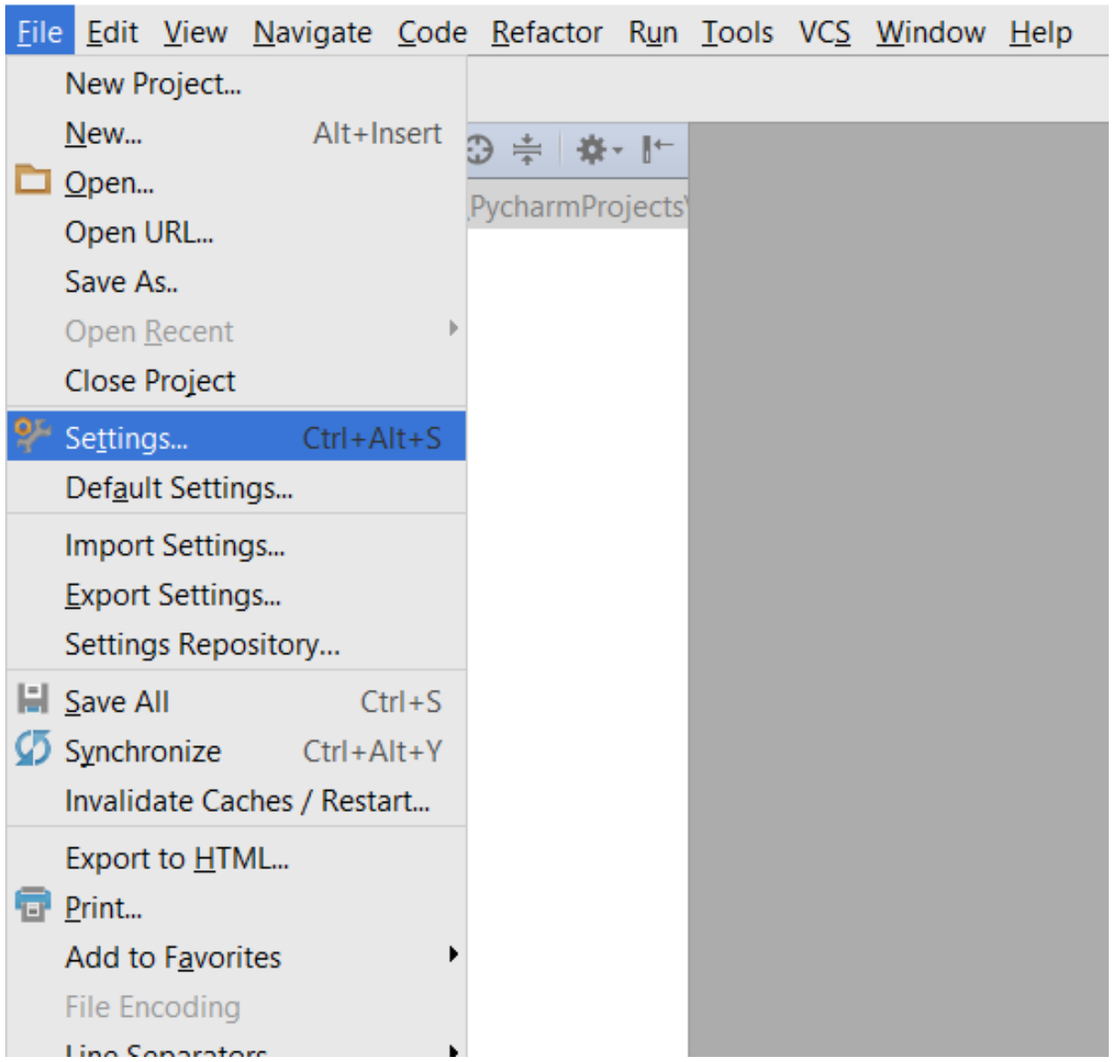
Use case: explanation

- <title .*?> finds the opening tag where there must be a space after the word "title" and the tag must be closed, but any characters can appear in the rest of the tag. We use the non-greedy ***?**, because we want the first closing ">" to match the tag's end
- .* any character can appear in between the <title> tag
- <\title .*?> same expression as the first part but with the forward slash to represent a closing HTML tag
- More on regex: https://docs.python.org/3.5/howto/regex.html

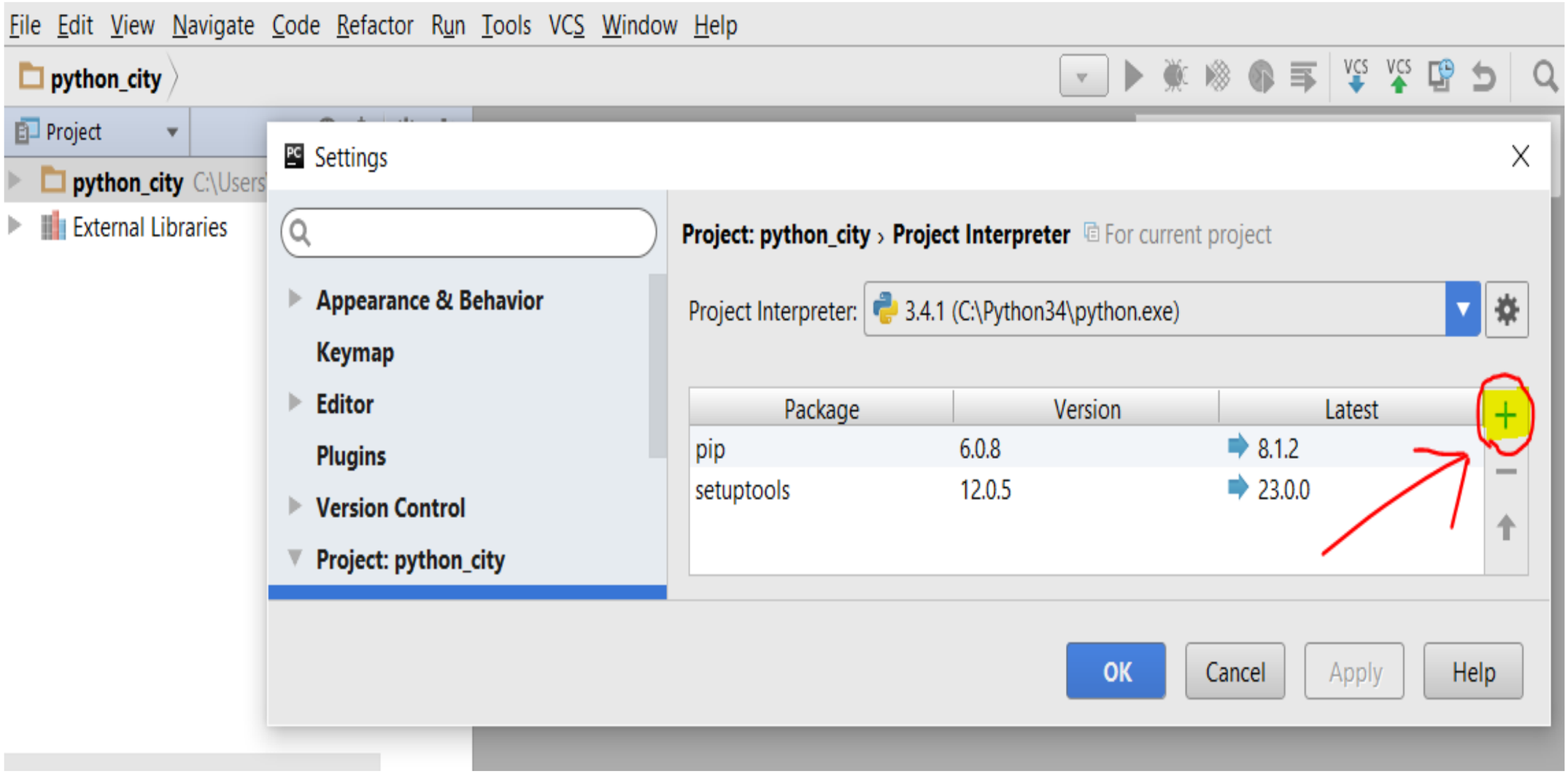
Installing an external library

- Sometimes what you need is not included in the python standard library and you have to install an external library
- You are going to use a python package manager: **pip**
- The packages (libraries) that you can install with pip are listed on https://pypi.python.org/pypi
- If you do not have pip, you can use the command "python setup.py install" from the package you would have downloaded and uncompressed from pypi

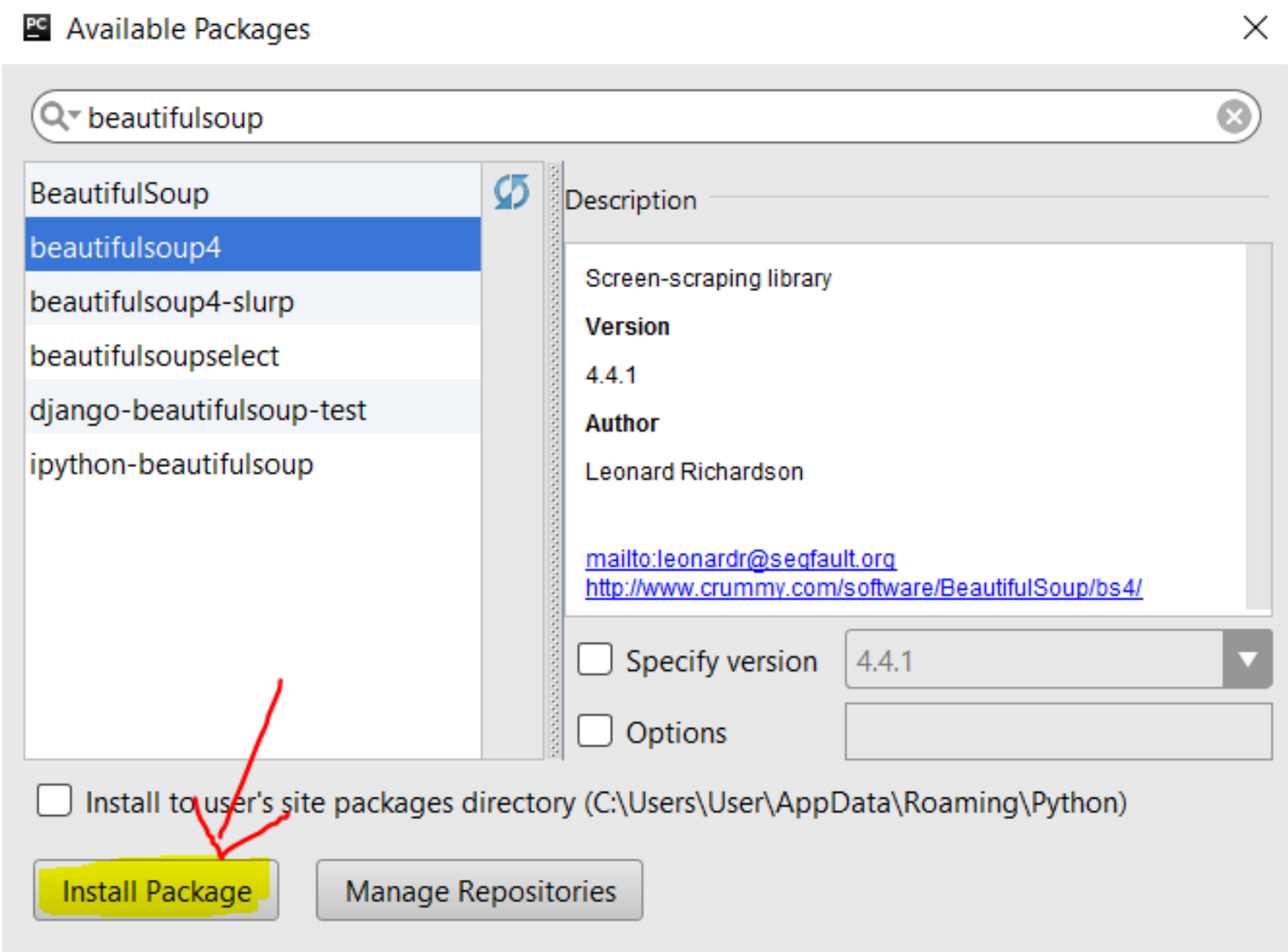
Installing with Pycharm (1)



Installing with Pycharm (2)



Installing with Pycharm (3)



Using Beautiful Soup

```
from bs4 import BeautifulSoup
from urllib.request import urlopen
my_address = "http://www.staff.city.ac.uk/~ddimak/python
             "practice/Profile_Dionysus.htm"
html_page = urlopen(my_address)
html_text = html_page.read().decode('utf-8')
my_soup = BeautifulSoup(html_text, "html.parser")
```

BeautifulSoup: get_text()

- **get_text()**
  - is extracting only the text from an html document

```
print(my_soup.get_text())
```

- there are lot of blank lines left but we can remove them with the method **replace()**

```
print(my_soup.get_text().replace("\n\n\n",""))
```

- Using BeautifulSoup to extract the text first and use the find() method is *sometimes* easier than to use regular expressions

BeautifulSoup: find_all()

- find_all()
  - returns a list of all elements of a particular tag given in argument

```
print(my_soup.find_all("img"))
```

- What if the HTML page is broken?

BeautifulSoup: Tags

```
[<img src="dionysus.jpg"/>, <img src="grapes.png"><br><b
Hometown: Mount Olympus
<br><br>
Favorite animal: Leopard <br>
<br>
Favorite Color: Wine
</br></br></br></br></br></br></img>]
```

- This is not what we were looking for. The <img> is not properly closed therefore BeautifulSoup ends up adding a fair amount of HTML after the image tag before inserting a </img> tag on its own. This is often seen in the wild.
- NB: BeautifulSoup is storing HTML tags as *Tag* objects and we can extract information from each Tag.

BeautifulSoup: Extracting information from Tags

- **Tags**:
  - have a name
  - have attributes
    - Attributes are accessible using keys
    - works similarly to accessing values of a dictionary through its keys

```
for tag in my_soup.find_all("img"):
    print(tag.name)
    print(tag['src'])
```

BeautifulSoup: accessing a Tag through its name

```
print(my_soup.title)
```

- The HTML is cleaned up
- We can use the string attributes stored by the title

```
print(my_soup.title.string)
```

The select method (1)

- ... will return a list of Tag objects, which is how Beautiful Soup represents an HTML element. The list will contain one Tag object for every match in the BeautifulSoup object's HTML

The select method (2)

Selector passed to the select method	Will match...
soup.select('div')	All elements named
soup.select('#author')	The element with an **id** attribute of **author**
soup.select('.notice')	All elements that use CSS class notice
soup.select('div span')	All elements named that are within an element named
soup.select('div > span')	All elements named that are directly within an element named , with no other elements in between
soup.select('input[name]')	All elements named <input> that have a **name** attribute with any value
soup.select('input[type="button"]')	All elements named <input> that have an attribute name **type** with value **button**

Emulating a web browser

- Sometimes we need to submit information to a web page, like a login page
- We need a web browser for that
- **MechanicalSoup** is an alternative to urllib that can do all the same things but has more added functionality that will allow us to talk back to webpages without using a standalone browser. Perfect for fetching web pages, clicking on buttons and links, and filling out and submitting forms

Installing MechanicalSoup

- You can install it with pip: *pip install MechanicalSoup* or within Pycharm (like what we did earlier with BeautifulSoup)
- You might need to restart your IDE for MechanicalSoup to load and be recognised

MechanicalSoup: Opening a web page

- Create a browser
- Get a web page which is a Response object
- Access the HTML content with the *soup* attribute

```
import mechanicalsoup

my_browser = mechanicalsoup.Browser(
                 soup_config={'features':'html.parser'
page = my_browser.get("http://www.staff.city.ac.uk/~dd
           "practice/Profile_Aphrodite.htm")
print(page.soup)
```

MechanicalSoup: Submitting values to a form

- Have a look at this login page
- The important section is the login form
- We can see that there is a submission <form> named "login" that includes two <input> tags, one named *username* and the other one named *password*.
- The third <input> is the actual "Submit" button

MechanicalSoup: script to login

```
import mechanicalsoup

my_browser = mechanicalsoup.Browser(
    soup_config={'features':'html.parser'})
login_page = my_browser.get(
    "https://whispering-reef-69172.herokuapp.com/login")
login_html = login_page.soup

form = login_html.select("form")[0]
form.select("input")[0]["value"] = "admin"
form.select("input")[1]["value"] = "default"

profiles_page = my_browser.submit(form, login_page.url)
print(profiles_page.url)
print(profiles_page.soup)
```

Methods in MechanicalSoup

- We created a Browser object
- We called the method *get* on the Browser object to get a web page
- We used the *select()* method to grab the form and input values in it

Interacting with the Web in Real Time

- We want to get data from a website that is constantly updated
- We actually want to simulate clicking on the "refresh" button
- We can do that with the *get* method of MechanicalSoup

Use case: fetching a stock quote from Nasdaq (1)

- Let us identify what is needed
- ▪ What is the source of the data?
  https://www.nasdaq.com/symbol/ba
  - What do we want to extract from this source?
    The stock price

Use case: fetching a stock quote from Nasdaq (2)

- If we look at the source code, we can see what the tag is for the stock and how to retrieve it:
- `<div id="qwidget_lastsale" class="qwidget-dollar">$367.16<`
- An **id** is unique and should only appear once in the page. However, it is good practice to check that the id appears only once in the webpage.

MechanicalSoup: script to find Boeing current price

```
import mechanicalsoup

my_browser = mechanicalsoup.Browser()
page = my_browser.get("https://www.nasdaq.com/symbol/ba")
html_text = page.soup
# return a list of all the tags where
# the css id is 'qwidget_lastsale'
my_tags = html_text.select("#qwidget_lastsale")

# take the BeautifulSoup string out of the
# first (and only) <div> tag
my_price = my_tags[0].text
print("The current price of "
      "Boeing is: {}".format(my_price))
```

Repeatedly get Boeing's current price

- Now that we know how to get the price of a stock from the Nasdaq web page, we can create a for loop to stay up to date
- Note that we should not overload the Nasdaq website with more requests than we need. And also, we should also have a look at their robots.txt file to be sure that what we do is allowed

Introduction to the *time.sleep()* method

- The *sleep()* method of the module time takes a number of seconds as argument and waits for this number of seconds, it enables to delay the execution of a statement in the program

```
from time import sleep
print "I'm about to wait for five seconds..."
sleep(5)
print "Done waiting!"
```

Repeatedly get the Boeing current price: script

```
from time import sleep
import mechanicalsoup
my_browser = mechanicalsoup.Browser()
# obtain 1 stock quote per minute for the next 3 minutes
for i in range(0, 3):
    page = my_browser.get("https://www.nasdaq.com/symbol/ba")
    html_text = page.soup
    # return a list of all the tags where the css id is 'qwidget_l
    my_tags = html_text.select("#qwidget_lastsale")
    # take the BeautifulSoup string out of the first tag
    my_price = my_tags[0].text
    print("The current price of BA is: {}".format(my_price))
    if i<2: # wait a minute if this isn't the last request
        sleep(60)
```

Exercise: putting it all together

- Install a new library called *requests*
- Using the select method of BeautifulSoup, parse (that is, analyze and identify the parts of) the image of the day of http://xkcd.com/
- Using the *get* method of the *requests* library, download the image
- Complete the following program xkcd_incomplete.py

Using request

- You first have to import it

```
import requests
```

- If you want to download the webpage, use the get() method with a url in parameter, such as:

```
res = requests.get(url)
```

- Stop your program if there is an error with the raise_for_status() method

```
res.raise_for_status()
```

Next? Web crawling!

- From Wikipedia: A Web crawler is an Internet bot which systematically browses the World Wide Web, typically for the purpose of Web indexing.
- How do you navigate a website? For example, for the http://xkcd.com/ website, how could you **retrieve all of its images?**
- Write down how you would design your program
- Write the program

Solution for Web Crawling

Solution

Download the script here: xkcd_downloader.py