Midterm Project Report
Advanced Computer Programming

Course Details

Scraping and Viewing with

Python and HTML

Student Name : Arya Widia Putra
Student ID : 112021200
Teacher : DINH-TRUNG VU

2024-04

Chapter 1 Introduction

1.1 Github

1) **Personal Github Account**: https://github.com/aryawidiap
2) **Group Github Account**: https://github.com/advcomp-earthquake
3) **Group Project Repository**: https://github.com/advcomp-earthquake/earthquake-monitor
4) **List of submitted files**:
    - **main**.py
    - **semCourseSpider**
        1. **spiders**
            - **__init__**.py
            - **semcoursespider**.py
        2. **__init__**.py
        3. **course_data**.csv
        4. **items**.py
        5. **middlewares**.py
        6. **pipelines**.py
        7. **settings**.py
        8. **show_course**.py
    - **course**.html
    - **examplePage**.html
    - **formPage**.html
    - **README**.md
    - **scrapy**.cfg

1.2 Topic

Our group topic is inspired by the recent earthquake that happened on April 3rd, 2024, in Taiwan. When we go to the earthquake report website by the Central Weather Administration (https://www.cwa.gov.tw/V8/E/E/index.html), we can see the recent earthquakes that happened in Taiwan. However, we have to go into the detail pages to know whether a city, for example, Taichung, is affected by the earthquake or not. Therefore, our final project will be extracting the data from the detail page of the recent earthquakes and getting the data on its effect in Taichung.

For the individual midterm project, I am inspired by my responsibility to know and report on which course I can take at Asia University to my home university in Indonesia.

Usually, I would have to read the course detail page of each course offered in the semester (https://webs.asia.edu.tw/course_eng/courselist.asp). Furthermore, I first have to select the query to find the list of courses that are suitable for me on the course query page (https://webs.asia.edu.tw/course_eng/). In this project, I made a program that can automatically query from a set argument declared in the program and then get to each of the detail pages through the link in the course list page. The program then extracts each of the course details, gets some of the details needed, and writes it to a CSV file, as I am usually asked to present the courses in a spreadsheet. Another feature of my project is a program to output the data in HTML so that I can review the course in a web page format.

1.3 Project Overview

This project consists of two smaller programs: a scraper and an HTML writer. The scraper uses the Scrapy, pattern matching, and regular expression. The HTML writer uses data class and input and output functions. The project extracted information from started scraping from https://webs.asia.edu.tw/course_eng/, sending a request for the course list page (https://webs.asia.edu.tw/course_eng/courselist.asp). The code for this project can be found in https://github.com/aryawidiap/semCourseSpider.

Chapter 2 Implementation

2.1 Class CourseItem

Declaration of class CourseItem can be found in items.py. This class is used to store or represent the data scraped from the desired webpage. This class has several fields related to the data we want to extract from the page.

2.1.1 Fields

This class has a lot of fields that are all related to the course and are self-explanatory based on their names. Additional explanations for some of the fields will be written next to the field.

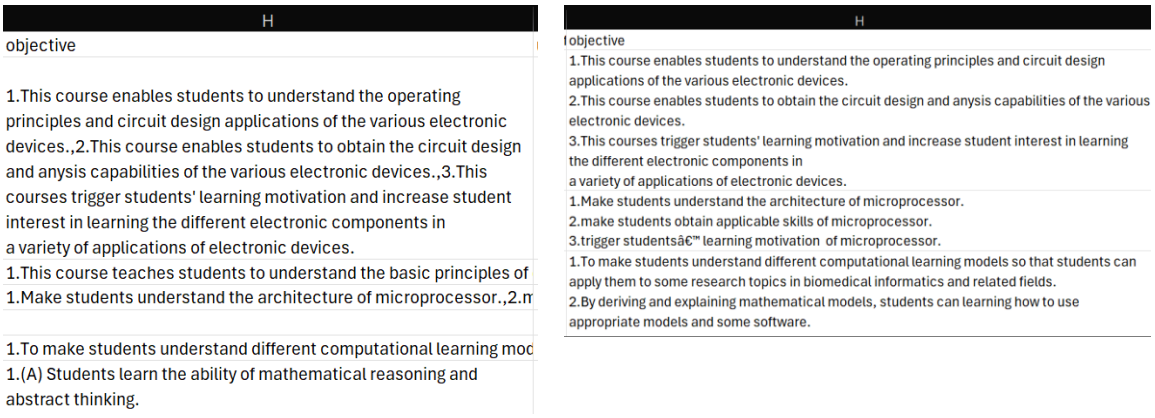
1. course_code
2. class_name
3. credits
4. course_name_in_English
5. link_to_syllabus
6. number_of_students
7. upper_limit: The number of students that can enroll in the class.
8. objective: The class objective, usually gave us a clearer idea about what the course will be about than the course name.
9. materials: Another course/class attribute that describes what the course will be about.
10. year: the grade of the student the course is targeted at.

2.2 Class SemcoursespiderPipeline

This class is used to process the data scraped and yielded using the Item class (in this case CourseItem) before being shown to the terminal or before being written to the output file.

2.2.1 Methods

process_item method is used to process the CourseItem before writing to the output file. In this method, I added functionality to rearrange the ‘objective’ field to make it more readable. I joined all the members of the objective list, separating it with a new line so that it would look more appropriate when opened using applications like Excel.



(A) (B)

Figure 1. Comparison of CSV file output (A) Output before using pipeline, (B) Output after using pipeline

2.3 Class SemcoursespiderSpider

Class Semcoursespider is the driver of the Scrapy scraper. I wanted to access the course list available this semester in English which I have to query first through the form at https://webs.asia.edu.tw/course_eng/. Because of that, I have to send a post request first before getting the course list. After the post request, we will get the list page and we can start extracting the link to the available courses and open the detail page through each link. Then, we extract the details of each course.

2.3.1 Fields

There are some variables that we have or can declare to set up the project. Naming the project is done by assigning the name to the ‘name’ variable. The ‘allowed_domains’ variable is used to restrict the link. The ‘start_urls’ variable is used as the first page to crawl. The ‘custom_settings’ is used to pass some settings to the Scrapy settings without modifying the settings.py directly. I added the ‘FEEDS’ dictionary to declare that I want the scraped data to be written to a CSV file named ‘course_data.csv’

```
name = "semcoursespider"
allowed_domains = ["asia.edu.tw"]
start_urls = ["https://webs.asia.edu.tw/course_eng/"]
custom_settings = {
    'FEEDS': {
        'course_data.csv': {'format':'csv', 'overwrite': True}
    }
}
```

2.3.2 Method parse()

The parse() method is the first method that will be called by the Scrapy. As I have to send the POST request first to the course query page, we have the ‘form_data’ dictionary to store all the arguments or data we want to pass to the form. We use this dictionary in ‘scrapy.FormRequest.from_response’ method call. This method call simulates a submitting action by clicking a button thus we have a ‘clickdata’ argument where we define which button to click. In this case, the button to click has the attribute ‘name’ of ‘Qry’

```
def parse(self, response):
    form_data = {
        'cos_setyear_q': '112',
        'cos_setterm_q': '2',
        'chk_eng': 'E',
        'dept_no_q': 'EE30',
        'Qry': 'Query',
    }
    return scrapy.FormRequest.from_response(
        response,
        formdata=form_data,
        clickdata={'name': 'Qry'},
        callback=self.parse_after_form
    )
```

2.3.3 Method parse_after_form()

This method is used after the post request sends us the response. The response should be the course list page. We collected all the course detail links and passed each link to the ‘parse_table’ method.

```
def parse_after_form(self, response):
    # Get courses' links from the page
    course_link_list =
response.css('a[href^="course_outline.asp?"]::attr(href)').getall()
    # For each link, request the page and process with parse_table
method
    for link in course_link_list:
        yield
scrapy.Request("https://webs.asia.edu.tw/course_eng/"+link,
self.parse_table)
```

2.3.4 Method parse_table()

Method parse_table is used to extract the details of a course from the course’s detail page. The data is presented as a table in the detail page, hence we treat the elements we extracted as table rows. The elements are mostly not unique and have no identifying attributes, thus we run through each of the rows first. Additionally, some rows have 2 columns, which I handled using the conditional if else statement. The code below extracts each row of data to a dictionary, getting the class detail’s name as the key and the class detail’s value as the value.

```
def parse_table(self, response):
    # Get the rows from main table element
    table = response.css('table.standard-table2 > tr')
    # Parse all rows from the main table
    attributeDict = {}
...
```

```
...
    for row in table:
        table_data = row.css('td')
        if(len(table_data) == 4):
            attributeDict[table_data[0].css('::text').get().strip()] =
table_data[1].css('::text').extract()
            attributeDict[table_data[2].css('::text').get().strip()] =
table_data[3].css('::text').extract()
        elif(len(table_data) == 2):
            attributeDict[table_data[0].css('::text').get().strip()] =
table_data[1].css('::text').extract()
...
```

After all the rows were extracted and put into the dictionary, I selected the details I wanted by pattern-matching the keys of the dictionary. I collected the data of a row using an object of CourseItem and then yielded the object. Due to the return and yield chain, the objects data will be returned to the parse class and processed to be written to the CSV file.

```
...
    # Assign the desired values from attributeDict to course_item
    course_item = CourseItem()
    for key in attributeDict:
        match key:
            case 'Course Code / Class':
                code_and_class = attributeDict[key][0].split()
                course_item['course_code'] = code_and_class[0]
                course_item['class_name'] = code_and_class[2]
            case 'Title of Course':
                course_item['course_name_in_English'] =
attributeDict[key]
            case 'Credits':
                course_item['credits'] = attributeDict[key]
            case 'Number of Students':
                if(len(attributeDict[key]) > 0):
                    course_item['number_of_students'] =
attributeDict[key][0].strip()
...
```

```
...
            case 'Upper Limit':
                if(len(attributeDict[key]) > 0):
                    course_item['upper_limit'] =
attributeDict[key][0].strip()
            case '一、教學目標(Objective)':
                course_item['objective'] = attributeDict[key]
            case '三、教材內容(Materials)':
                course_item['materials'] = attributeDict[key]
            case 'Department / Year':
                match_group = re.search("\d", attributeDict[key][0])
                if(match_group):
                    course_item['year'] = match_group.group()
                else:
                    course_item['year'] = 0
    course_item['link_to_syllabus'] = response.request.url


    yield course_item
```

2.4 Data class Course (show_course.py)

This data class is a template for objects that represent the course and its details.

The module used for this is the **dataclass** from **dataclasses**.

2.4.1 Fields

Course data class has 10 fields used to keep data of the course's details.

1. class_name (str): class name for the specific time block (for example class A, B, C)
2. course_code (str)
3. course_name_in_English (str)
4. credits (int)
5. link_to_syllabus (str)
6. materials (str)
7. number_of_students (int): number of students already enrolled in the class.
8. objective (str)
9. upper_limit (int): the limit of the student that can enroll in the class.
10. year (int): the grade of student the course is targeted at.

2.5 Function get_courses (show_course.py)

The function get_course is used to get the course data from the course_data.csv file into the Course objects representation. These objects are contained in a list that is sorted before being returned to the calling instance. First, the file is opened using the **open** function. Then, the CSV is read using **csv.reader()** method. After that, we will read the file line by line. Because the first line is just the column name, we can skip it. The rest of the rows are broken down into each column and constructed into a Course object, then appended to the course_list. Lastly, the list is sorted before being returned to the calling instance.

In this function, I used the **csv module** and the **input capability** of Python.

```
def get_courses():
    course_list = []
    # Open csv file
    with open('semCourseSpider/course_data.csv', 'r', newline='') as
f:
        reader = csv.reader(f)
        is_first_row = True
        for row in reader:
            if is_first_row:
                is_first_row = False
                continue
            course_list.append(Course(
                class_name=row[0],
                course_code=row[1],
                course_name_in_English=row[2],
                credits=row[3],
                link_to_syllabus=row[4],
                materials=row[5],
                number_of_students=row[6],
                objective=row[7],
                upper_limit=row[8],
                year=row[9],
            ))
    return sorted(course_list, key=operator.attrgetter("year"))
```

2.6 Function main (show_course.py)

In this function, I created an HTML file named course.html that will show the course on a webpage, divided by its year, and each course is represented using a card item. For this page, I choose to only show the course name, course code, credits, number of students, upper limit, course objectives, and the link to details. This function is rather long, please refer to the show_course.py

In this function, I used the output and write capability of Python.

Chapter 3 Results

3.1 Result 1: course_data.csv

The SemcoursespiderSpider.py file outputs a CSV file named course_data.csv. This file has all the information I needed of the courses, that is the class name, course code, course name, credits, link to syllabus, materials, number of students and the upper limit, objective, and the year/grade the course is targeted at.

class_nam	course_co	course_na	credits	link_to_syl	materials	number_of	objective	upper_limi	year
C	EE300117	Electronic	3	https://we	1. Semicor	16	1.This	40	3
C	EE300099	Deep learr	3	https://we	Students c	14	1.This	60	3
C	EE300026	Microproc	3	https://we	In recent y	18	1.Make	60	3
C	EE300012	Probability	3	https://we	This cours	22	1.(A)	30	1
C	EE300056	Discrete M	3	https://we	Five impor	19	1.Student	23	1
C	EE300116	Digital Log	3	https://webs.asia.edu		17		20	2
C	EE300098	Machine le	3	https://we	Learning b	16	1.To	20	2

Figure 2. A peek at the course_data.csv

The objective attribute of the course data also has been formatted like I wanted it to. However, there seem to be some special characters that slip into the objective. It would be better to consider preprocessing the text further to make the text even more readable.

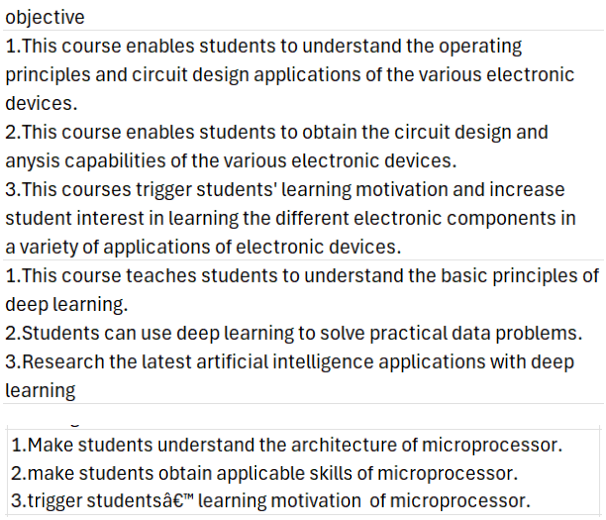


Figure 3. The objectives of the courses

3.2 Result 2: course.html

The show_course.py is used to extend the functionalities of the whole module from only scraping to reconstructing the data into a preferred look in an HTML file. Figure 4 is the resulting code output by the show_course.py. As we can see in Figure 4, the file is quite

decent and working properly, but a bit messy in code form as the indentations are a bit inconsistent.

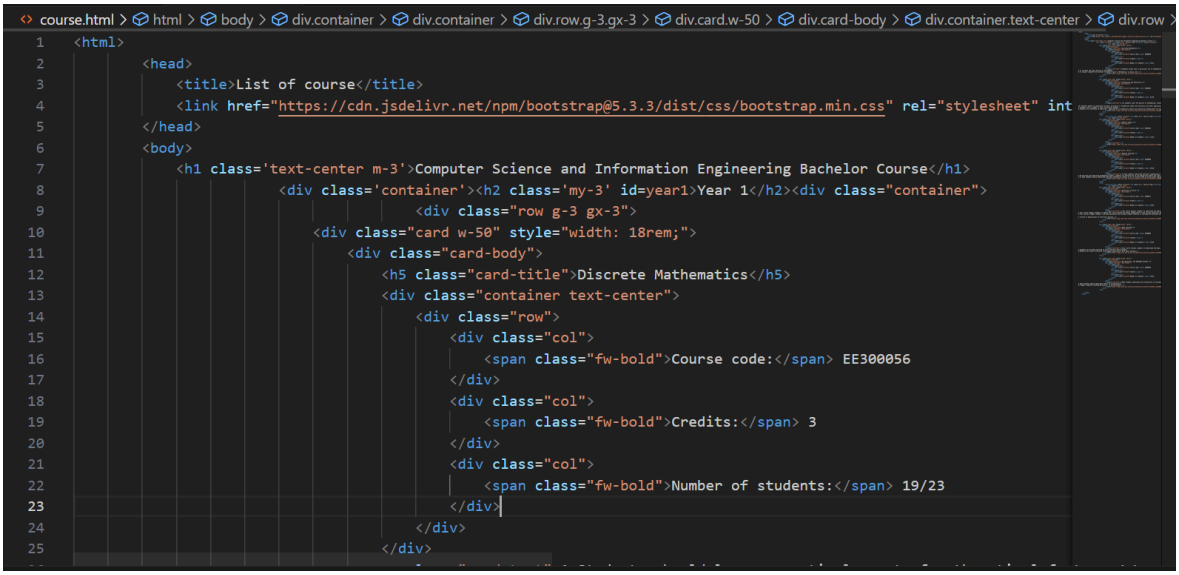


Figure 4. The resulting HTML file of the show_course.py program

In Figure 5, we can see that the functionality of showing the course with the details has been achieved. However, the presentation of the objectives is still difficult to read. Regardless, it saved us time from having to click each link to know the objectives of the courses.

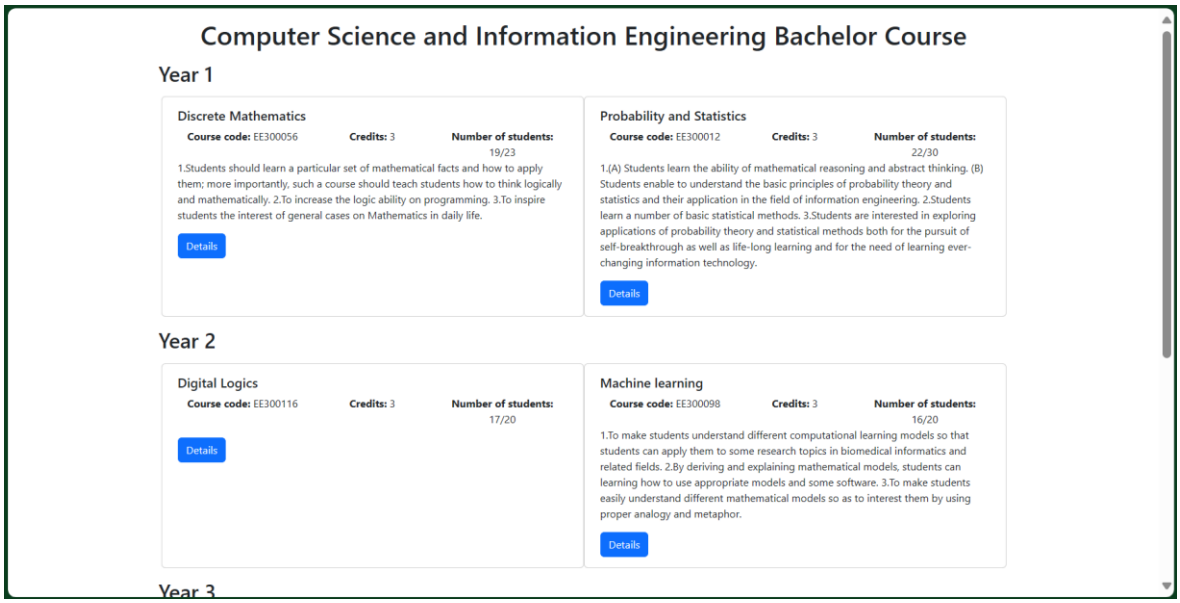


Figure 5. The user interface resulted from the course.html

Chapter 4 Conclusions

This midterm project taught me how to use Scrapy to scrape and extract data from web pages and use the item class template and pipeline to streamline the scraping process. On the side, I also used regular expressions to extract some text and match cases to make conditional statements for the data refinement process. I also learned how to write the data scraped to a CSV file. I also learned more about input/output in Python by using the CSV file to create a web page about course information. I also used Dataclass to represent the data read from the CSV. In the future, I would like to refine the writing of the HTML file via Python and refine the information present on the web page itself.