Discussion 6 Solutions

FA23 Final Problem 7

Alan set up a web page for his DSC 80 notes with the following HTML:

```
<html>
    <body>
        <div id = "hero">DSC 80 NOTES</div>
        <div class="notes">
            <div class="notes">
                <p>Lecture 1: 5/5 stars!</p>
            </div>
            <div class="lecture notes">
                <p>Lecture 2: 6/5 stars!!</p>
            </div>
        </div>
        <div class="lecture">
            <p>Lecture 3: 10/5 stars!!!!</p>
        </div>
    </body>
</html>
```

Assume that the web page is parsed into a BeautifulSoup object named `soup`. Fill in each of the expressions below to evaluate to the desired string. Pay careful attention to the indexes after each call to `find_all`!

Part 1

Desired string: `"Lecture 1: 5/5 stars!"`

`soup.find_all(____)[0].text`

Answer: `soup.find_all('p')[0].text`

`"Lecture 1: 5/5 stars!"` is surrounded by `<p>` tags, and `find_all` will get every instance of these tags as a list. Since `"Lecture 1: 5/5 stars!"` is the first instance, we can get it from its index in the list, `[0]`. Then we grab just the text using `.text`.

Part 2

Desired string: `"Lecture 2: 6/5 stars!!"`

`soup.find_all(____)[3].text`

Answer:

`soup.find_all('div')[3].text`

`"Lecture 2: 6/5 stars!!"` appears as the text surrounded by the fourth `<div>` tag, and since we are grabbing index `[3]` we need to `find_all('div')`. Then `.text` grabs the text portion.

Part 3

Desired string: `"Lecture 3: 10/5 stars!!!!"`

`soup.find_all(____)[1].text`

Answer:

Answer:

`soup.find_all('div', class_='lecture')[1].text`

We need to return a list with `"Lecture 3: 10/5 stars!!!!"` as the second index since we access it with `[1]`. We see that we have two instances of `'lecture'` attributes in `div` tags: one with `class="lecture notes"` and `class="lecture"`.

Note that `class="lecture notes"` actually means that the tag has two `class` attributes, one of `"lecture"` and one of `"notes"`. The `class_='lecture'` optional argument in `find_all` will find all tags that have a `'lecture'` attribute, meaning that it'll find both the `class="lecture notes"` tag and the `class="lecture"` tag. So, `soup.find_all('div', class_='lecture')` will find the two aforementioned `<div>`s, `[1]` will find the second one (which is the one we want), and `.text` will find `"Lecture 3: 10/5 stars!!!!"`.

SP23 Final Problem 1

Consider the following Code Snippet:

`re.findall(r'__(a)__', 'my cat is hungry, concatenate!, catastrophe! What a cat!')`

Part 1

Which regular expression in **__(a)__** will generate the following output? `Output: ['my', 'a']`

- `\b([a-z]*)cat\b`
- `\b[a-z]*\scat\b`
- `([a-z]*)\scat\b`
- `\b([a-z]*\scat)\b`

Answer: C - `([a-z]*)\scat\b`

This regular expression selects the sections of matches that consist of zero or more lowercase letters that are followed by a space and then followed by cat as a whole word (not with cat as a substring of a larger word), essentially selecting words followed by a space and the word cat. Thus this option correctly selects `['my', 'a']`.

Option 1 would select `['', '']`

Option 2 would select `['my cat', 'a cat']`

Option 4 would select `['my cat', 'a cat']`

Part 2

Which regular expression in **__(a)__** will generate the following output?

`Output: ['concatenate']`

- `\b.*cat.*\b`
- `[a-z]*cat[a-z]*`
- `[a-z]+cat[a-z]+`
- `\b[a-z]*cat[a-z]*\b`

Answer: C - `[ a-z]+cat[a-z]+`

This regular expression selects matches where one or more lowercase letters are followed by the substring cat, and then followed by one or more lowercase letters, essentially selecting words with cat as a substring but not a prefix. Thus this option correctly selects

`['concatenate']`.

Option 1 would select

`['my cat is hungry, concatenate!, catastrophe! What a cat']`

Option 2 would select

`['cat', 'concatenate', 'catastrophe', 'cat']`

Option 4 would select

`['cat', 'concatenate', 'catastrophe', 'cat']`

Part 3

Which regular expression in **__(a)__** will generate the following output?

`Output: ['cat', 'concatenate', 'catastrophe', 'cat']`

- `.*cat.*`
- `\b.*cat.*\b`
- `\b[a-z]*cat[a-z]*\b`
- `\b[a-z]+cat[a-z]+\b`

Answer: C - `\b[a-z]*cat[a-z]*\b`

This regular expression selects matches where a word boundary is followed by 0 or more lowercase letters, cat, and then follwoed by 0 or more lowercase letters followed by a word boundary, essentially selecting words containing cat. Thus this option correctly selects

`['cat', 'concatenate', 'catastrophe', 'cat']`.

Option 1 would select

`['my cat is hungry, concatenate!, catastrophe! What a cat!']`

Option 2 would select

`['my cat is hungry, concatenate!, catastrophe! What a cat!']`

Option 4 would select `['concatenate']`

WI23 Final Exam Problem 4

To prepare for the verbal component of the SAT, Nicole decides to read research papers on data science. While reading these papers, she notices that there are many citations interspersed that refer to other research papers, and she'd like to read the cited papers as well.

In the papers that Nicole is reading, citations are formatted in the *verbose numeric* style. An excerpt from one such paper is stored in the string `s` below.

```
s = '''
In DSC 10 [3], you learned about babypandas, a strict subset
of pandas [15][4]. It was designed [5] to provide programming
beginners [3][91] just enough syntax to be able to perform
meaningful tabular data analysis [8] without getting lost in
100s of details.
'''
```

We decide to help Nicole extract citation numbers from papers. Consider the following four extracted lists.

```
list1 = ['10', '100']
list2 = ['3', '15', '4', '5', '3', '91', '8']
list3 = ['10', '3', '15', '4', '5', '3', '91', '8', '100']
list4 = ['[3]', '[15]', '[4]', '[5]', '[3]', '[91]', '[8]']
list5 = ['1', '0', '3', '1', '5', '4', '5', '3',
         '9', '1', '8', '1', '0', '0']
```

For each expression below, select the list it evaluates to, or select “None of the above.”

Part 1

`re.findall(r'\d+', s)`

Answer: `list3`

This regex pattern `\d+` matches one or more digits anywhere in the string. It doesn’t concern itself with the context of the digits, whether they are inside brackets or not. As a result, it extracts all sequences of digits in s, including `'10'`, `'3'`, `'15'`, `'4'`, `'5'`, `'3'`, `'91'`, `'8'`, and `'100'`, which together form `list3`. This is because `\d+` greedily matches all contiguous digits, capturing both the citation numbers and any other numbers present in the
text.

Part 2

`re.findall(r'[\d+]', s)`

Answer: `list5`

This pattern `[\d+]` is slightly misleading because the square brackets are used to define a character class, and the plus sign inside is treated as a literal character, not as a quantifier. However, since there are no plus signs in `s`, this detail does not affect the outcome. The character class `\d` matches any digit, so this pattern effectively matches individual digits throughout the string, resulting in `list5`. This list contains every single digit found in `s`, separated as individual string elements.

Part 3

`re.findall(r'\[(\d+)\]', s)`

Answer: `list2`

This pattern is specifically designed to match digits that are enclosed in square brackets. The `\[(\d+)\]` pattern looks for a sequence of one or more digits `\d+` inside square

brackets `[]`. The parentheses capture the digits as a group, excluding the brackets from the result. Therefore, it extracts just the citation numbers as they appear in `s`, matching `list2` exactly. This method is precise for extracting citation numbers from a text formatted in the verbose numeric style.

WI23 Final Exam Problem 5

After taking the SAT, Nicole wants to check the College Board's website to see her score. However, the College Board recently updated their website to use non-standard HTML tags and Nicole's browser can't render it correctly. As such, she resorts to making a GET request to the site with her scores on it to get back the source HTML and tries to parse it with BeautifulSoup.

Suppose `soup` is a BeautifulSoup object instantiated using the following HTML document.

```
<college>Your score is ready!</college>

<sat verbal="ready" math="ready">
  Your percentiles are as follows:
  <scorelist listtype="percentiles">
    <scorerow kind="verbal" subkind="per">
      Verbal: <scorenum>84</scorenum>
    </scorerow>
    <scorerow kind="math" subkind="per">
      Math: <scorenum>99</scorenum>
    </scorerow>
  </scorelist>
  And your actual scores are as follows:
  <scorelist listtype="scores">
    <scorerow kind="verbal"> Verbal: <scorenum>680</scorenum> </scorer
    <scorerow kind="math"> Math: <scorenum>800</scorenum> </scorerow>
  </scorelist>
</sat>
```

- [ ] `soup.find("scorerow").get("kind")`
- [ ] `soup.find("sat").get("ready")`
- [ ] `soup.find("scorerow").text.split(":")[0].lower()`
- [ ] `[s.get("kind") for s in soup.find_all("scorerow")][-2]`
- [ ] `soup.find("scorelist", attrs={"listtype":"scores"}).get("kind")`

☐ None of the above

Solution

Answer: Option 1, Option 3, Option 4

Correct options:

- Option 1 finds the first `<scorerow>` element and retrieves its `"kind"` attribute, which is `"verbal"` for the first `<scorerow>` encountered in the HTML document.
- Option 2 finds the first `<scorerow>` tag, retrieves its text `("Verbal: 84")`, splits this text by ":", and takes the first element of the resulting list `("Verbal")`, converting it to lowercase to match `"verbal"`.
- Option 3 creates a list of `"kind"` attributes for all `<scorerow>` elements. The second to last (-2) element in this list corresponds to the `"kind"` attribute of the first `<scorerow>` in the second `<scorelist>` tag, which is also `"verbal"`.

Incorrect options:

- Option 2 attempts to get an attribute ready from the `<sat>` tag, which does not exist as an attribute.
- Option 5 tries to retrieve a `"kind"` attribute from a `<scorelist>` tag, but `<scorelist>` does not have a `"kind"` attribute.