In [1]:
```python
# Initialize Otter
import otter
grader = otter.Notebook("lab7.ipynb")
```

Lab 7: Regular Expressions

In [2]:
```python
import pandas as pd
import numpy as np
import re

from pathlib import Path
```

Objectives:

First, read this section on regular expressions: https://docs.python.org/3/library/re.html As you work through the first part of the lab, you may also find the website http://regex101.com helpful.

Part 1: Regular Expressions

We'll start by learning about the simplest possible regular expressions. Since regular expressions are used to operate on strings, we'll start with the most common task: matching characters.

Most letters and characters will simply match themselves. For example, the regular expression `r'test'` will match the string `test` exactly. There are exceptions to this rule; some characters are special, and don't match themselves.

Here is a list of metacharacters that are widely used in regular experssion.

Pattern	Description
`^`	Matches beginning of line.
`$`	Matches end of line.
`.`	Matches any single character except newline.
`*`	Matches 0 or more occurrences of preceding expression.
`+`	Matches 1 or more occurrence of preceding expression.
`?`	Matches 0 or 1 occurrence of preceding expression.

`[...]`	Matches any single character in brackets.
`[^...]`	Matches any single character not in brackets.
`{n}`	Matches exactly n number of occurrences of preceding expression.
`{n,}`	Matches n or more occurrences of preceding expression.
`{n,m}`	Matches at least n and at most m occurrences of preceding expression.
`a|b`	Matches either a or b.
`\1...\9`	Matches n-th grouped subexpression.

Perhaps the most important metacharacter is the backslash, '\'. As in Python string literals, the backslash can be followed by various characters to signal various special sequences. It's also used to escape all the metacharacters so you can still match them in patterns; for example, if you need to match a `[` or `\`, you can precede them with a backslash to remove their special meaning: `\[` or `\\`.

The following predefined special sequences are available:

Pattern	Description
`\d`	Matches any decimal digit; this is equivalent to the class `[0-9]`
`\D`	Matches any non-digit character; this is equivalent to the class `[^0-9]`.
`\s`	Matches any whitespace character; this is equivalent to the class `[ \t\n\r\f\v]`
`\S`	Matches any non-whitespace character; this is equivalent to the class `[^ \t\n\r\f\v]`.
`\w`	Matches any alphanumeric character; this is equivalent to the class `[a-zA-Z0-9_]`
`\W`	Matches any non-alphanumeric character; this is equivalent to the class `[^a-zA-Z0-9_]`.

Question 1

In this question, write patterns that match the given sequences. It may be as simple as the common letters on each line.

Question 1a

Write a single regular expression to match the following strings without using the `|` operator.

1. **Match:** `abcdefg`
2. **Match:** `abcde`
3. **Match:** `abc`
4. **Skip:** `c abc`

```
In [3]: regx1 = r"^abc.*$"
```

```
In [4]: grader.check("q1a")
```

```
Out[4]:
```

q1a passed! 🌈

Question 1b

Write a single regular expression to match the following strings without using the `|` operator.

1. **Match:** `can`
2. **Match:** `man`
3. **Match:** `fan`
4. **Skip:** `dan`
5. **Skip:** `ran`
6. **Skip:** `pan`

```
In [5]: regx2 = r"^[cmf]an$"
```

```
In [6]: grader.check("q1b")
```

```
Out[6]:
```

q1b passed! 🌈

Question 2

Now that we have written a few regular expressions, we are now ready to move beyond matching. In this question, we'll take a look at some methods from the `re` package.

Question 2a:

Write a Python program to extract and print the numbers of a given string.

1. **Hint:** use `re.findall`
2. **Hint:** use `\d` for digits and one of either `*` or `+`.

```
In [7]: text_q2a = "Ten 10, Twenty 20, Thirty 30"

        res_q2a = re.findall(r"\d+", text_q2a)


        res_q2a
```

```
Out[7]: ['10', '20', '30']
```

```
In [8]: grader.check("q2a")
```

Out[8]:

q2a passed! ✨

Question 2b:

Write a Python program to replace at most 2 occurrences of space, comma, or dot with a colon.

Hint: use `re.sub(regex, "newtext", string, number_of_occurences)`

```
In [9]: text_q2b = 'Python Exercises, PHP exercises.'

        res_q2b = re.sub("[ ,.]", ":", text_q2b, 2)

        res_q2b
```

```
Out[9]: 'Python:Exercises: PHP exercises.'
```

```
In [10]: grader.check("q2b")
```

Out[10]:

q2b passed! 🌈

Question 2c:

Write a Python program to extract values between quotation marks of a string.

In [11]:
```python
text_q2c = '"Python", "PHP", "Java"'

res_q2c = re.findall(r'"(.*?)"', text_q2c)

res_q2c
```

Out[11]:
```
['Python', 'PHP', 'Java']
```

In [12]:
```python
grader.check("q2c")
```

Out[12]:

q2c passed! 🍀

Question 2d:

Write a regular expression to extract and print the quantity and type of objects in a string. You may assume that a space separates quantity and type, ie. `"{quantity} {type}"`. See the example string below for more detail.

1. **Hint:** use `re.findall`
2. **Hint:** use `\d` for digits and one of either `*` or `+`.

In [13]:
```python
text_q2d = "I've got 10 eggs that I stole from 20 gooses belonging to 30 gia

res_q2d = re.findall(r"\d+\s[^\s.]+", text_q2d)


res_q2d
```

Out[13]:
```
['10 eggs', '20 gooses', '30 giants']
```

In [14]:
```python
grader.check("q2d")
```

Out[14]:

q2d passed! 🌈

Question 2e (optional):

Write a regular expression to replace all words that are not `"mushroom"` with `"badger"`.

In [15]:
```python
text_qe = 'this is a word mushroom mushroom'
res_qe = ... # Hint: https://www.regextester.com/94017
...
res_qe
```

Out[15]:
```
Ellipsis
```

To double-check your work, the cell below will rerun all of the autograder tests.

In [16]:
```python
grader.check_all()
```

Out[16]:
```
q1a results: All test cases passed!

q1b results: All test cases passed!

q2a results: All test cases passed!

q2b results: All test cases passed!

q2c results: All test cases passed!

q2d results: All test cases passed!
```

Submission

1. Save file to confirm all changes are on disk
2. Run *Kernel > Restart & Run All* to execute all code from top to bottom
3. Save file again to write any new output to disk
4. Select *File > Save and export Notebook as > HTML*.
5. Open in Google Chrome and print to PDF.
6. Submit to Gradescope