CS 357 Lab 1: Regular Expressions
Sept 18, 2023 during class

Due date: Wednesday, Sept 20 at 5pm [for those not checked off during class]

Grading Guidelines: Each exercise is worth 5 points and contributes toward the homework/lab/quizzes portion of your grade in the course. For each exercise, you will receive three points for correct functionality and two points for providing new test cases.

Pair Programming: Work with another student through these activities.

Get Started:
We will be using an online regular expression checking program at pythex.org. Go to pythex.org now in a web browser.

At the top is a text box to enter a regular expression. The next text box allows for text input. We will usually be testing in multiline mode, so that each line is a test string to match the regular expression. The next text box shows which strings match the regular expression – those that turn green are a match.

Click on the regular expression cheatsheet to see how to encode a regex using python. Below is a summary of the syntax you need to solve the activities in this lab. If you want to see the full library, go to:
https://docs.python.org/3/library/re.html

^	matches the beginning of string
[ ]	creates a character class
	[a-z] creates a|b|c|…|z
	[a-zA-Z] creates all lowercase and uppercase characters
	[0-9] creates all digits
|	union
+	one or more copies
*	zero or more copies
?	zero or one copy
( )	grouping of a regular expression for precedence
$	matches the end of the string
.	matches any character
\s	matches space character
\\.	matches period character
{m}	exactly m occurrences

Concatenation happens by simply typing adjacent regular expressions. You should be able to complete the tasks below with only the above operations.

Activity: Warm UP
Let’s try a regular expression that matches an all-lowercase word. Type the following into the regular expression box:

```
^[a-z]+$
```

Type in several words in the test string box, and use multiple lines. Hmm, did it match anything? Probably not… the ^ and $ chars say to match the entire string in the test string box, and you likely have newlines and spaces between your test cases. Instead, we will treat each line separately as a different test string. Click on the MULTILINE button, which will put you into this one-string-per-line mode.

Now, what happens? Which strings are matching lowercase words?

aaa bbb ccc
____

Now, let's try matching for two specific strings – 35 or 12. In the regular expression box, type:
```
^35|12$
```

In the test string box, type:
```
35
10
12
-8
```

Hopefully, you got the results you expected. If not, make sure the MULTILINE button is selected.

You are now ready to create your own regexes to match certain patterns.

Positive numbers (worth 5 pts)

Situation: Validating input from a text form is a common computing task. Suppose we let users enter how much money to transfer from one account to another. It could be dangerous for someone to transfer a negative amount of money.

Task: Create a regular expression for all valid **positive** numbers. Note that the empty string is not a valid number. Valid numbers can have leading zeros and may or may not include the decimal point. Valid numbers have at most one decimal point. For this exercise, we will consider zero "0" a positive number.

Make sure you enter enough test strings to ensure your regex accounts for all versions of valid positive numbers.

Here are example strings that should match (true) and example strings that should not match (false):

True:	**False:**
"1254"	"pilots"
"35.125"	"35.ab"
"50000000"	"-2.1cdef"
"45.3333"	"abc"
"325.333"	"15e2"
"01234"	""
".134"	"-45"
"0"	"35.52.3"

.

Checkpoint 1: Show the regex and match results to the instructor. If you do not want to wait, take a screenshot of your results. Or open a new tab to keep these results for when Tammy is available. Upload the screenshot (positive_numbers.jpg) with your zip file submission for the lab if not checked off in class.

Email Addresses (worth 5 pts)

Situation: Email addresses are often used as unique logins/IDs for online accounts and validating that the email address text is legit (not necessarily that someone can access mail sent to that address) is a common process.

Task: Complete the regular expression for `email addresses` that matches valid text email address. A valid email address is defined as the following (real emails can have other chars, but we will keep this simpler for this exercise), in order from left to right:

- Must start with a character (a – z) or (A – Z)
- Following the first character, contains zero or more characters from this set (alphanumeric, period, hyphen, underscore): {a-z, A-Z, 0-9, ., -, _} before @ symbol
- Must contain a single @ symbol
- Contains a domain or subdomain.domain or (subdomain.)*domain after the @ symbol
- Each domain and subdomain must contain at least one character (a-z or A-Z) and may contain one or more hyphens

Here are example strings that should match (true) and example strings that should not match (false):

True:	**False:**
bill@up.edu	bill@.22
Harry_White07@cs.fla.edu	sara_goodwin.ups.com
Jen.scott@a-a.b.c.e.d	one-hit-wonder@

You should test more strings to ensure that the regex matches a variety of proper email addresses and does not match a variety of improper email addresses.

Checkpoint 2: Show the regex and match results to the instructor. If you do not want to wait, take a screenshot of your results. Or open a new tab to keep these results for when Tammy is available. Upload the screenshot (positive_numbers.jpg) with your zip file submission for the lab if not checked off in class.

If you complete the exercises above during class, here is an extra credit challenge for you to try:

Extra Credit (up to 2 points)

Situation: Email addresses may include IP addresses for the domain part (text after @).

Task: Write a regular expression for `is_email_advanced` that adds the following definition to a valid email (keep the regex from prior checkpoint – this one should include more valid email addresses):

- Could have a valid IP address after the @ symbol, such as 192.144.3.2

- A valid IP address contains 4 numbers, separated by periods. Each number must be between 0 and 255, with 0 represented as a single 0 (not 00 or 000)

These should return true:	**These should return false:**
Jessica@145.210.1.6	joe@256.-4.6.20
Heidi-willis@192.150.10.3	a-b-c@1000.100.101
linda@email.domain.com	tammy@260.240.101.8
a@123.234.21.100	b@10.200

(extra credit – up to two points) Checkpoint 3: Show the regex and match results to the instructor. If you do not want to wait, take a screenshot of your results. Or open a new tab to keep these results for when Tammy is available. Upload the screenshot (positive_numbers.jpg) with your zip file submission for the lab if not checked off in class.