Exam Prep 10: Regular Expressions

Students from past semesters wanted more content and structured time to prepare for exams. Exam Prep sections are a way to solidify your understanding of the week's materials. The problems are typically designed to be a bridge between discussion/lab/homework difficulty and exam difficulty.

Reminder: There is nothing to turn in and there is no credit given for attending Exam Prep Sections.

We try to make these problems **exam level** , so you are not expected to be able to solve them coming straight from lecture without additional practice. To get the most out of Exam Prep, we recommend you **try these problems first on your own** before coming to the Exam Prep section, where we will explain how to solve these problems while giving tips and advice for the exam. Do not worry if you struggle with these problems, **it is okay to struggle while learning**.

You can work with anyone you want, including sharing solutions. We just ask you don't spoil the problems for anyone else in the class. Thanks!

You may only put code where there are underscores for the codewriting questions.

You can test your functions on their doctests by clicking the red test tube in the top right corner after clicking Run in 61A Code. Passing the doctests is not necessarily enough to get the problem fully correct. You must fully solve the stated problem.

A good reference for these problems is lab 13 regex review (https://cs61a.org/lab/lab13/#regular-expressions)

Q1: Phone Number Validator

Difficulty: ⭐⭐

Create a regular expression that matches phone numbers that are 11, 10, or 7 numbers long.

Phone numbers 7 numbers long have a group of 3 numbers followed by a group of 4 numbers, either separated by a space, a dash, or nothing.

Examples: `123-4567`, `1234567`, `123 4567`

Phone numbers 10 numbers long have a group of 3 numbers followed by a group of 3 numbers followed by a group of 4 numbers, either separated by a space, a dash, or nothing.

Examples: `123-456-7890`, `1234567890`, `123 456 7890`

Phone numbers 11 numbers long have a group of 1 number followed by a group 3 numbers followed by a group of 3 numbers followed by a group of 4 numbers, either separated by a space, a dash, or nothing.

Examples: `1-123-456-7890`, `11234567890`, `1 123 456 7890`

It is fine if spacing/dashes/no space mix! So `123 456-7890` is fine.

Note: The skeleton code is just a suggestion; feel free to use your own structure if you prefer.

```
import re
def phone_number(string):
    """
    >>> phone_number("Song by Logic: 1-800-273-8255")
    True
    >>> phone_number("123 456 7890")
    True
    >>> phone_number("1" * 11) and phone_number("1" * 10) and phone_number("1" * 7)
    True
    >>> phone_number("The secret numbers are 4, 8, 15, 16, 23 and 42 (from the TV show Lost)'
    False
    >>> phone_number("Belphegor's Prime is 1000000000000066600000000000001")
    False
    >>> phone_number(" 1122334455 ")
    True
    >>> phone_number(" 11 22 33 44 55 ")
    False
    >>> phone_number("Tommy Tutone's '80s hit 867-5309 /Jenny")
    True
    >>> phone_number("11111111") # 8 digits isn't valid, has to be 11, 10, or 7
    False
    """
    return bool(re.search(_____________________________________________________))

```

Q2: Email Domain Validator

Difficulty: ★★

Create a regular expression that makes sure a given string `email` is a valid email address and that its domain name is in the provided list of `domains`.

An email address is valid if it contains letters, number, or underscores, followed by an @ symbol, then a domain.

All domains will have a 3 letter extension following the period.

Hint: For this problem, you will have to make a regex pattern based on the inputs `domains`. A for loop can help with that.

Extra: There is a particularly elegant solution that utilizes join (https://python-reference.readthedocs.io/en/latest/docs/str/join.html) and replace (https://python-reference.readthedocs.io/en/latest/docs/str/replace.html)

Note: The skeleton code is just a suggestion; feel free to use your own structure if you prefer.

```
import re
def email_validator(email, domains):
    """
    >>> email_validator("oski@berkeley.edu", ["berkeley.edu", "gmail.com"])
    True
    >>> email_validator("oski@gmail.com", ["berkeley.edu", "gmail.com"])
    True
    >>> email_validator("oski@berkeley.com", ["berkeley.edu", "gmail.com"])
    False
    >>> email_validator("oski@berkeley.edu", ["yahoo.com"])
    False
    >>> email_validator("xX123_iii_OSKI_iii_123Xx@berkeley.edu", ["berkeley.edu", "gmail.com'
    True
    >>> email_validator("oski@oski@berkeley.edu", ["berkeley.edu", "gmail.com"])
    False
    >>> email_validator("oski@berkeleysedu", ["berkeley.edu", "gmail.com"])
    False
    """
    pattern = _____________________
    for _____________________:
        'Use as many lines as necessary'
    return bool(re.search(pattern, email))

```

Q3: Reg Extreme

Difficulty: ★★★

Regex is too tricky! Create a function `reg_extreme` that solves regex by receiving strings that should match and shouldn't match and a valid regex pattern.

First, create the generator `all_patterns` which generates all regex patterns of length 0 to `n`.

For efficiency sake, don't generate patterns that have explicit letters, but do generate patterns that have numbers and all special symbols. Make sure to include `\d`, `\w`, `\b`, and `\s`, as well as escaped special symbols like `\\`, `\.`, etc. These are all considered 1 symbol, so `\d\d\d\d\d` is length 5.

You would need to generate `12345` as a length 5 pattern, but not `abcde`, since this problem doesn't ask for explicit letters in the generated patterns.

Then complete `reg_extreme` which returns a pattern that matches all strings in `matches`, but none of the strings in `no_matches`.

Hint: Think about how to generate all patterns of length `n` given all patterns of length `n-1`

Hint: How can a try/except block help?

Note: In case the red test tube icon causes tests to time out, use the interpreter to test with `test(all_patterns)` and `test(reg_extreme)`. Doing things locally is more efficient, and I was able to find patterns length 5/6 as opposed to only 2/3 on code.cs61a.org.

Note: The skeleton code is just a suggestion; feel free to use your own structure if you prefer.

```
import re
def all_patterns(n):
    """
    >>> "12" in all_patterns(2)
    True
    >>> r"\d\d" in all_patterns(2)
    True
    >>> "1." in all_patterns(2)
    True
    >>> "1." in all_patterns(1)
    False
    >>> "a" in all_patterns(1)
    False
    >>> ".*" in all_patterns(3)
    True
    """
    numbers = list("0123456789")
    special = list(r"\()[]{}+*?|$^.")
    rest = ____________________________
    everything = [""] + numbers + special + rest
    if ____________________________:
        ____________________________
    else:
        'Use as many lines as necessary'

def reg_extreme(matches, no_matches, n=3):
    """
    >>> pattern = reg_extreme(["11", "12", "13"], ["1", "a"])
    >>> bool(re.search(pattern, "11"))
    True
    >>> bool(re.search(pattern, "12"))
    True
    >>> bool(re.search(pattern, "a"))
    False
    """
    for ____________________:
        'Use as many lines as necessary'

```