Week 10 - Monday

COMP 1800

Last time

- What did we talk about last time?
- Work day
- Before that:
  - Tuples
  - The `list()` function
  - Customized sorting
  - Cracking the substitution cipher

Questions?

Assignment 7

Regular Expressions

Searching for text

- We've already talked about Python tools to search for a string in another string
- The **`find()`** method will search for a string inside a string and return the index where it starts (or `-1` if it can't be found)

```
word = 'dysfunctional'
location1 = word.find('fun')      # location1 is 3
location2 = word.find('games')    # location2 is -1
```

What if you wanted to do partial matches?

- Maybe you want to search for text that:
  - Ends with "tion"
  - Starts with either "Password:" or "password:"
  - Has exactly five digits, like a zip code
  - Has a number followed by words like "street", "road", "avenue", "boulevard", "court", "way", or a few other possibilities
- The tool you want is called **regular expressions**
- Regular expressions can also be used to verify the formatting of data entered into websites

Regular expressions

- Regular expressions (sometimes shortened to **regexes**) are a way of describing a pattern of characters
- They have their roots in the 1950s as an important idea in theoretical computer science
- In the 1980s, the programming language Perl introduced a syntax for describing regular expressions
- Many languages, including Python, have adopted syntax that is identical or similar to the Perl syntax

Regular expression syntax

- In Python, regular expressions are written as strings, using symbols that have special meanings

Symbols	Meaning	Example	Explanation
`[]`	Set of characters	`'[m-z]'`	Letters m through z
`\`	Special sequence	`'\d'`	Numerical digits
`.`	Any character (except newline)	`'cr.p'`	`'crap'`, `'crip'`, `'cr8p'`, etc.
`^`	Starts with	`'^the'`	Line starts with `'the'`
`$`	Ends with	`'dog$'`	Line ends with `'dog'`
`*`	Zero or more occurrences	`'hi*'`	`'h'`, `'hi'`, `'hii'`, `'hiii'`, etc.
`+`	One or more occurrences	`'hi+'`	`'hi'`, `'hii'`, `'hiii'`, etc.
`?`	Zero or one occurrences	`'team?'`	`'tea'` or `'team'`
`{}`	The specified occurrences	`'he.{2}o'`	`'hello'`, `'helpo'`, `'hemno'`, etc.
`|`	Either/or	`'gray|grey'`	`'gray'` or `'grey'`

Special sequences

- Because there are certain sets of characters used a lot, there are special sequences for those

Sequence	Meaning
`\d`	Numerical digit (0-9)
`\D`	**Not** a numerical digit
`\s`	White space (space, tab, etc.)
`\S`	**Not** white space
`\w`	Alphanumeric (A-Z, a-z, 0-9, and underscore)
`\W`	**Not** alphanumeric

Set syntax

- Sets of characters are used a lot
- There are special rules inside the brackets

Set Example	Meaning
`[amp]`	Either a, m, or p
`[a-n]`	Any lowercase character in the range from a to n
`[^amp]`	Any character except a, m, or p
`[0-9]`	Any digit 0-9
`[a-zA-Z]`	Any lowercase or uppercase letter
`[+]`	The character +, since most special characters have no special meaning **inside** sets

Example: E-mail addresses

- Write a regular expression that will match an e-mail address:
  - Numbers or letters
  - Followed by an @ sign
  - Followed by numbers or letters
  - Followed by a dot
  - Followed by dots or numbers or letters

Example: Dates

- Write a regular expression that will match a date in this format:

```
12/03/2021
```

- Can you extend it so that the months and the days can be either one digit or two digits?

Example: Phone numbers

- Write a regular expression that will match a phone number in this format:

```
(404) 555-6789
```

- Extend it so that it can accept this format as well:

```
404-555-6789
```

Raw strings

- Both regular expressions and Python strings use backslash (****) to mean special things
- For this reason, it's common to use **raw strings** in Python when specifying a regular expression
- Raw strings start with **r** (before the quotes) and don't treat backslashes as special characters
- Raw strings are still normal strings, they just let you type things in differently

```
word1 = '\n'   # contains newline
word2 = '\\n'  # contains \n (two characters)
word3 = r'\n'  # contains \n (two characters)
```

Python functions for regular expressions

- Once you have a string that represents a regular expression, how can you use it?
- First, import **`re`**
- The **`re`** module has a number of functions, but three will be useful for us:

Function	Description
`findall()`	Return a list of all the strings that match
`split()`	Split a string into a list separated by places that match
`sub()`	Replace matches with a string

Regular expression examples

```
import re

text = 'we are the wombat combat warriors'
# get all words that start with w
wWords = re.findall(r'w[a-z]*', text)
# Gets: ['we', 'wombat', 'warriors']

# split up the string by words that start with w
noWWords = re.split(r'w[a-z]*', text)
# Gets: ['', ' are the ', ' combat ', '']

# replace every word that starts with w with goat
newText = re.sub(r'w[a-z]*', 'goat', text)
# Gets: 'goat are the goat combat goat'
```

Explanation of the regex in Assignment 7

```
def parseFile(filename):
    import re
    with open(filename, 'r') as file:
        text = file.read().lower()
        return re.findall("[a-z]+['-]?[a-z]+|[a-z]+", text)
```

- This function:
  - Opens **`filename`**
  - Reads everything into a single string and makes it lowercase
  - Then, it returns a list of words
  - Words are either a sequence of lowercase letters with a single apostrophe or hyphen or they're simply a sequence of lowercase letters

Limitations of regular expressions

- Regular expressions can't count
- Regular expressions can't make sure that matching separators (like left and right parentheses, left and right braces, left and right brackets) match up and are legally nested
- Some patterns that are simple to say have very long regular expressions

Upcoming

Next time…

- Review for Exam 2 on Wednesday
- Work time on Friday

Reminders

- Review chapters 5, 6, 7, and 8
- **Work on Assignment 7**