Class 22: Regular Expressions

April 11, 2019

Regular expressions are an extremely powerful method of searching and extracting information from strings. A good, basic tutorial is available here. (https://developers.google.com/edu/python/regular-expressions)

In the most-common use case, you have a test string that you test against a regular expression string, using the function `search` from the `re` module:

In [1]:
```
import re # we need to import the re module to use it

test_string = "My name is John Doe"

# test whether test_string contains "name"
# (pay attention to the r in front of the string; we need this)
match = re.search(r"name", test_string)
if match: # did we find a match?
    print("Test string matches.")
    print("Match:", match.group()) # print out the part of the string that matc
hed
else:
    print("Test string doesn't match.")
```

```
Test string matches.
Match: name
```

In [2]:
```
test_string = "My email is john@utexas.edu"

# test whether test_string contains "name"
match = re.search(r"name", test_string)

if match: # did we find a match?
    print("Test string matches.")
    print("Match:", match.group())
else:
    print("Test string doesn't match.")
```

```
Test string doesn't match.
```

Much of the power of regular expressions stems from the fact that you can match on general patterns. For example, `\S+` will match an arbitrary number of non-whitespace characters:

In [3]:
```
test_string = "My age is secret."
match = re.search(r"My \S+ is", test_string)
print("Match:", match.group())

test_string = "My mood is good."
match = re.search(r"My \S+ is", test_string)
print("Match:", match.group())
```

```
Match: My age is
Match: My mood is
```

We can also capture substrings using regular expressions, by encapsulating the parts of interest in parentheses `()`:

In [4]:
```
test_string = "My age is secret."
match = re.search(r"My (\S+) is (\S+)", test_string)
print("Match:", match.group(0))
print("Captured group 1:" , match.group(1))
print("Captured group 2:" , match.group(2))
```

```
Match: My age is secret.
Captured group 1: age
Captured group 2: secret.
```

Problems

Problem 1

Use the online python regular expression editor available here: http://pythex.org/ (http://pythex.org/) to explore regular expressions. For each of the given test strings, find the regular expressions that achieves the given goals.

1. Test string: "`my email is: john@utexas.edu`"

- Match on: "`my email is`"
  Solution: `/my email is/`
- Match on any email address
  Solution: `/\S*@\S*/`
- Match on: "`@utexas.edu`"
  Solution: `/@utexas.edu/`
- Capture the entire email address
  Solution: `/(\S*@\S*)/`
- Capture both the part before the @ sign and the part after the @ sign separately
  Solution: `/(\S*)@(\S*)/`
- Capture the username of any `utexas.edu` email address
  Solution: `/(\S*)@utexas.edu/`

2. Test string: "`phone number: 123-456-7890`"

- Match on "`phone number:`" and capture the phone number
  Solution: `/phone number: (\S+)/`
- Match on any string of the form of a phone number, with three digits, a hyphen, three more digits, another hyphen, and four digits
  Solution: `/\d\d\d-\d\d\d-\d\d\d\d/` Or: `/\d{3}-\d{3}-\d{4}/`
- Use the same match as before, but now capture the area code
  Solution: `/(\d{3})-\d{3}-\d{4}/`

3. Invent a few more problems and solutions on your own.

Problem 2:

Write python code that can take a string of the form "`My name is: ...`", extract the name (indicated here by `...`), and then print it. Make sure you get the full name, not just the first name.

In [5]:
```
test_string = "My name is: John Doe"

match = re.search(r'My name is: (.*)', test_string)
if match:
    print(match.group(1))
```

```
John Doe
```

If this was easy

Problem 3:

Write a function that can parse phone numbers in any sort of format and print them out in the standard 123-456-7890 format.

In [6]:
```
def clean_phone_number(input):
    match = re.search(r'(\d{3})\D*(\d{3})\D*(\d{4})', input)
    if match:
        cleaned_number = match.group(1) + "-" + match.group(2) + "-" + match.gr
oup(3)
        print("'" + input + "' contains the phone number " + cleaned_number)
    else:
        print("'" + input + "' is not a valid phone number")

# all these calls should produce the number 123-456-7890
clean_phone_number("1234567890")
clean_phone_number("+1 (123) 456-7890")
clean_phone_number("1 123 456 7890")
clean_phone_number("(123) 4567890")
# the function should realize that this is not a valid phone number
clean_phone_number("123456")
```

```
'1234567890' contains the phone number 123-456-7890
'+1 (123) 456-7890' contains the phone number 123-456-7890
'1 123 456 7890' contains the phone number 123-456-7890
'(123) 4567890' contains the phone number 123-456-7890
'123456' is not a valid phone number
```