Introduction to Python – Part IV

Outline

- Files
- Modules and Packages
- Reading Web Pages
- Regular Expressions
- Python and Regexes

Files

Opening Files

- We can open a file for reading/writing using open function
- open returns a file handle object

```
file = open('test.txt', 'w')

file.write('Hi!\n')
file.write('This is a test.\n')

file.close()
```

Reading Files

- Files handles are iterable
- We can use handles to read files line by line

```
file = open('test.txt', 'r')

for line in file:
    print(line.rstrip())

file.close()
```

Reading Files at Once

- We can read the entire content of a file:
  - as a single string by read, or
  - as a list of strings by readlines

```
>>> open('test.txt').read()
'Hi!\nThis is a test.\n'

>>> open('test.txt').readlines()
['Hi!\n', 'This is a test.\n']
```

Modules and Packages

Modules

- A module is a file containing Python definitions and statements to be used in other Python programs

```
# mymodule.py
def foo():
    pass
bar = 10

# test.py
import mymodule

mymodule.foo()
```

Importing Modules

- There are three different ways to import a module

```
import math
math.pi

from math import pi, cos
cos(pi)

import math as m
m.pi
```

Packages

- We can organize modules inside packages, and access them via dot notation
- A package is simply a directory containing an (empty) __init__.py file

```
App/
  __init__.py
  test.py
  Tools/
    __init__.py
   utils.py
   mytools.py

from App.Tools import utils
```

Reading Web Pages

Retrieve a Page(python 3.x)

- We can use urlretrieve function to download any kind of content from the Internet
- The function is located in request module in urllib package

```
from urllib.request import urlretrieve

url = 'http://google.com'
file_name = 'google.html'

urlretrieve(url, file_name)
```

Retrieve a page (python 2.x)

```
import requests
url= 'http://google.com'
r = requests.get(url)
text = r.content
f = open('a.html','w')
f.write(text)
```

Regular Expressions

Regular Expressions

- A regular expression (aka regex or regexp) is a sequence of characters that forms a search pattern
- Python supports regexes through the standard library re module

```
import re

m = re.match('me', 'meanwhile')
if m is not None:
    print(m.group())
```

Regular Expression Syntax

- Regular expressions are strings containing text and special characters (such as ? and *) that describe a pattern
- The simplest regular expressions are just strings, with no special characters
- The choice | operator creates a regular expression that matches one of two things

```
if re.match('Ali|Hamid', user):
   // user is valid
```

Character Classes

- The character class operator [] allows to match any character within the class
  - [abcd] is equivalent to a|b|c|d
- We can use a range of characters within a class
  - [a-f] is equivalent to [abcdef]
- We can also reverse a class using ^ operator
  - [^0-9] matches any non-digit character

Predefined Classes

- There are a few predefined character classes

\d	any digit [0-9]
\w	any word character [0-9a-zA-Z_]
\s	any whitespace [\t\n\r]
.	any character (except \n)
\D	any non-digit character [^0-9]
\W	any non-word character [^\w]
\S	any non-space character [^\s]

Repetition Operators

- The following operators can be used to match the same expression repeatedly

– *	match 0 or more times
– +	match 1 or more times
– ?	match 1 or 0 times
– {n}	match exactly n times
– {n,}	match at least n times
– {n,m}	match at least n but not more than m times

- These operators are greedy: they match as much text as possible (add ? for minimal fashion)

Special Characters

- There are some important special characters
  - ^ match the beginning of the string
  - $ match the end of the string (or before the newline)
- You can use ^ and $ to make sure your strings don't contain garbage
  - This is good practice for validating user input

```
if re.match(r'^\w*$', filename):
    // this is a safe filename
```

Groups

- We can group parts of the regular expression, mainly for further retrieval
  - (…) indicates the start and end of a group
  - \number matches the content of a group of the same number
- Examples:
  - \d+(\.\d+)? matches a simple float number
  - (.+) \1 matches e.g. "the the"

Named Groups

- We can assign names to matched groups for easier access
  - (?P<name>…) the substring matched by the group is a given a name *name*
  - (?P=name) matches the text matched by earlier group named *name*
- Example:
  - (?P<word>\w+) (?P=word) matches "the the"

Python and Regexes

The re Module

- Useful functions in re module
  - match() match pattern to string from the beginning
  - search() search for first occurrence of pattern in string
  - compile() compile a pattern for faster match
  - findall() find all (non-overlapping) occurrences of pattern
  - finditer() like findall but returns an iterator instead of list
  - split() split string according to pattern delimiter
  - sub() replace all occurrences of pattern by a string

```
>>> re.findall('\w+', 'ali-ha 12!')
['ali', 'ha', '12']
```

Modifiers

- Modifiers that appear after the second / control aspects of the RE matching process
  - re.I performs case-insensitive matching
  - re.M treats string as a multiline string
  - re.S makes . match any character including newline
  - re.X ignores whitespace in the pattern (for readability)

```
>>> re.findall('^a\w+', 'ali\nA12!', re.M | re.I)
['ali', 'ha', '12']
```

Match Objects

- The output of match() and search() functions, if successful, is a match object
- Match objects have three primary methods, group(), groups() and groupdict()

```
>>> re.match('(\w+)-(\w+)', 'ali-ha').group()
'ali-ha'
>>> re.match('(\w+)-(\w+)', 'ali-ha').groups()
('ali', 'ha')
>>> re.match('(?P<k>\w+)', 'ali-ha').groupdict()
{'k': 'ali'}
```

References

- Python Web Development with Django
  - By Jeff Forcier, Paul Bissex, Wesley Chun
- Core Python Applications Programming
  - By Wesley J. Chun
- Internet Programming by Pat Morin
  - http://cg.scs.carleton.ca/~morin/teaching/2405/