CS 38003 PYTHON PROGRAMMING

Ruby Tahboub

FILE PROCESSING

FILE PROCESSING

- The process of opening a file involves associating a file on disk with an object in program memory.
- For reading or writing a file, you need to open the file initially.

```
open(filename, access_mode) # opens a file
```

- Note: if you do not provide a full path the file is assumed to be in the same directory where your program exists.
- `access_mode` specifies the purpose of opening the file.

ACCESS MODES

Mode	Description
'r'	Open file for reading only. Starts reading from beginning of file. This default mode.
'rb'	Open a file for reading only in binary format. Starts reading from beginning of file.
'r+'	Open file for reading and writing. File pointer placed at beginning of the file.
'w'	Open file for writing only. File pointer placed at beginning of the file. Overwrites existing file and creates a new one if it does not exists.
'wb'	Same as w but opens in binary mode.
'w+'	Same as w but also allows to read from file.
'wb+'	Same as wb but also allows to read from file.
'a'	Open a file for appending. Starts writing at the end of file. Creates a new file if file does not exist.
'ab'	Same as a but in binary format. Creates a new file if file does not exist.
'a+'	Same as a but also open for reading.
'ab+'	Same as ab but also open for reading.

READING A FILE

```
myfile = open(“example.txt", "r") # opens a file for reading
```

- Reading a file into a string.

```
content = myfile.read()
```

- Reading a single line (file pointer moves to the next line)

```
singleLine = myfile.readline()
```

- Reading file content into a list of strings, each line ends with `\n`

```
allLines = myfile.readlines()
```

LOOPING OVER FILES

```
infile = open("myFile.txt", "r")
for line in infile:
  # process each line here
  # …
infile.close()
```

```
infile = open("myFile.txt", "r")
for line in infile.readlines():
  # process each line here
  # …
infile.close()
```

```
infile = open("myFile.txt", "r")
lines = infile.read().split('\n'):
for line in lines
  # process each line here
  # …
infile.close()
```

```
infile = open("myFile.txt", "r")
line = infile.readline()
while line:
  # process each line here
  # …
  line = infile.readline()
infile.close()
```

WRITING TO A FILE

```
myfile = open(“example.txt", “w") # opens a file for writing
```

- Open an existing file for writing would erase its contents.
  - If the named file does not exist, a new one is created automatically.

```
myfile.write("some content ...") # writes some content … to myfile
```

- The next string written to myfile will be on the same line (unlike the `print` function).
  - Contents written are not necessarily written back to disk.

```
myfile.flush()
```

- Forces the content written so far to be written back to disk.

CLOSING A FILE

```
myfile.close() # closes myfile
```

- Always remember to close the file
- In case of writing, closing the file forces the written contents to be written back to disk.
- Failing to close the file may risk written data not be written back to disk.
- In case of reading, closing the file allows for other applications to gain access to the file.

REGULAR EXPRESSIONS

REGULAR EXPRESSIONS

- A regular expression is a special string that describes one or more strings.
  - import `re` module
- Used to search for strings with certain content.
  - Example: search for all words starting with a, b and c and ending with ing
- If special characters are included in the expression such as:

```
". ^ $ * + ? { } [ ] \ | ( )
```

then the regular expression may describe 0 or more other strings.

REGULAR EXPRESSIONS

- The square brackets [and] are used for specifying a character class.
- For example, [abc] will match the characters a, b, or c.
- When using ^ as the first character, like in [^abc] then it means all characters that are not (a, b, or c).
- The \ is used as a escape sequence. For example, \[will match the [character.

PREDEFINED CHARACTER CLASSES

`\d`	Matches any digit. This is equivalent to the class [0-9].
`\D`	Matches any non digit. This is equivalent to the class [^0-9].
`\s`	Matches any whitespace character.
`\S`	Matches any non whitespace character.
`\w`	Matches any alphanumeric. This is equivalent to the class [a-z A-Z 0-9 _].
`\W`	Matches any non-alphanumeric.

REPEATING STRINGS WITH * and +

- To represent the repetitions of a string you use the **"*"** operator.
- **"x*"** will represent the repetition of "x" **0 or more** times like "x" or "xx" or ""
- For example "ab*c" will match the strings "ac", "abc", "abbbbc".
- The "+" operator can be used to represent **one or more** repetitions of a string.
- For example "x+" will represent one or more repetitions of "x", like "x", "xx" but not "".

THE ? and {} OPERATORS

- The “?” operators matches a regular expression 0 or 1 time.
- For example, “x?” will match “x” or “”.
  - Also, “ab?c” will match “abc or “ac”.
- The “x{m,n}” qualifier represents that x can be repeated at least m times and at most n times.
- For example x{1, 3} will match “x”, “xx”, “xxx” but it will not match “” or “xxxx”.
- You can omit m or n that means that m=0 or n is infinite.x{2,} matches “xx”, “xxx”, “xxxx….”

COMPILING REGULAR EXPRESSIONS

- Internally processing a regular expression to low-level code for improved performance.

```
>>> p = re.compile ('ab*c') # compiling a pattern

>>> print(p.match("a"))
None

>>> print(p.match("ac"))
<re.Match object; span=(0, 2), match='ac'>

>>> print(p.match("abc"))
<re.Match object; span=(0, 3), match='abc'>

>>> print(p.match("abbc"))
<re.Match object; span=(0, 4), match='abbc'>

>>> print(p.match("abd"))
None
```

USING REGULAR EXPRESSIONS

- `match(pattern)`
  - Determines if the RE matches at the beginning of the string.
- `search(pattern)`
  - Scans through a string, looking for any location where this RE matches.
- `findall(pattern)`
  - Finds all substrings where the RE matches, and returns them as a list.

TEST MATCHING

- The result of pattern matching can be tested in any condition as if and while statements.

```
import re

def testMatch(regexpr, pattern):

  p = re.compile(regexpr)
  if(p.match(pattern)):
    print("Match!")
  else:
    print("doesn't match")

testMatch("ab*c", "abbc")
testMatch("ab*c", "bbc")
```

```
Match!

doesn't match
```

TEST MATCHING

- search looks for a substring match (unlike match that looks for a full match)

```
import re
def checkSearch(regexpr, pattern):

  p = re.compile(regexpr)
  if(p.search(pattern)):
    print("search success!")
  else:
    print("search failed!")

checkSearch("ab*c", "xabbc")
checkSearch("ab*c", "bbc")
```

search success!

search failed!

THE `findall` OPERATION

- `findall` returns a list of all matches

```
>>> p = re.compile('\w*ing')

>>> p.findall('I like swimming and running')
['swimming', 'running']

>>> re.findall('\w*ing', 'I like swimming and running')
['swimming', 'running']
```

THE `split` OPERATION

- `split` splits a string based on a delimiter (a regular expression in this case).
- The delimiter will be removed out of the string.

```
>>> p = re.compile('\w*ing') # compiling a regular expression

# split with one parameter a string
>>> p.split('I like swimming and running.')
['I like ', ' and ', '.']


# split with two parameters: regular expression and string

>>> re.split('\w*ing', 'I like swimming and running.')
['I like ', ' and ', '.']
```

MORE ON REGULAR EXPRESSIONS

- This website is a good practice portal for regular expressions:

  https://www.regexpal.com

PARSING WEB PAGES with BEAUTIFUL SOUP

BEAUTIFUL SOUP: OBTAIN PAGE SOURCES

```
from bs4 import BeautifulSoup
import urllib.request

r = urllib.request.urlopen('https://www.cs.purdue.edu').read()

soup = BeautifulSoup(r)

# print out results
print(soup.prettify())
```


material obtained from CS242

BEAUTIFUL SOUP: PARSE LINKS

```
# find all links
links = soup.find_all('a')


# get text associated with links
for link in links:
  linkText = link.get_text().strip()
  print(linkText)


# get urls
for link in links:
  linkUrl = link["href"]
  # filter for html pages with absolute path
  if 'http' in linkUrl and '.html' in linkUrl:
    print(linkUrl)
```


material obtained from CS242

BEAUTIFUL SOUP: PARSE PARAGRAPH

```
# find all paragraph text without embedded links
paragraphs = soup.find_all('p')
for p in paragraphs:
  if not p.find('a'):
    pText = p.get_text()
    print(pText)
```


material obtained from CS242

THANK YOU!