Version: 2020/06/07 13:04

CSC1015F Assignment 7: Files and Recursion

Assignment Instructions

This assignment involves constructing Python functions that use recursion, and programs that perform file I/O.

There are four questions.

- The first concerns identifying palindromes.
- The second concerns constructing a module containing a *pattern matching* function.
- The third involves using the pattern matching module to build a program to search a file of words.
- The last question involves constructing a 'book cypher' program, a program that encodes a message by mapping each word of the message to a location in a book.

The first two question require that you use recursion. You must not use loop constructs (such as 'for' and 'while') in your solutions.

Assessment

Your code will be automatically marked.

In the case of questions one and two, the first *(N-1)* automarker trials will check that your code functions correctly by executing it on test inputs, while the *N*th is a penalty test. The *N*th scans your code for evidence of the use of iteration or the use of list/string reversal expressions. If it finds any, then it deducts the marks for the question!

NOTE: In some cases, the penalty test will report a false positive. For instance, it thinks you're using loops but you are not, you simply have a variable name containing the word 'for', e.g. 'former', 'afford'.

Pattern matching

Let's say that we are building a 'dictionary' program that people can use to look up the spelling and meaning of words. People don't always know the spelling of the word they're looking for.

For the program to function well, some sort of pattern matching is required. Pattern matching is a process by which we search for words with some given characteristics. The characteristics of words include their length, character content and character order.

A pattern is a description of required characteristics.

A simple way of representing a pattern is by using a sequence of letters and special 'wild card' characters. Question 2 and 3 concern such a scheme.

We'll introduce the scheme with examples:

Pattern	*Possible word matches*
Letter	Letter
l?ad	lead, load
l*ad	lad, launchpad, lead, letterhead, lipread, load, loggerhead, lunkhead, ...
l?*ad	launchpad, lead, letterhead, lipread, load, loggerhead, lunkhead, ...
*action	abreaction, abstraction, action, attraction, benefaction, coaction, ...

CONTINUED

And here are the rules:

- A pattern is a sequence of letters and the wild card characters ‘?’ and ‘*’.
- When a letter appears in a pattern it indicates that precisely that letter must appear at the same position in matching words.
- When the wildcard character ‘?’ appears in a pattern it indicates that any letter may appear at that position.
- When the wildcard character ‘*’ appears in a pattern it indicates that a sequence of zero or more of letters is acceptable at that position.

CONTINUED

Question One [20 marks]

A palindrome is a word, phrase, number, or other sequence of symbols or elements, whose meaning may be interpreted the same way in either forward or reverse direction (see Wikipedia).

Write a Python program called '`palindrome.py`' that uses a recursive function to calculate whether or not a string is a palindrome (reads the same if reversed).

You may NOT use iteration, or a string slice expression (or any other technique) to reverse the string without using recursion!

Sample I/O:

```
Enter a string:
able was I ere I saw elba
Palindrome!
```

Sample I/O:

```
Enter a string:
elba is a noob
Not a palindrome!
```

Question Two [20 marks]

Write a Python module called *simplematch.py* that contains a recursive function called '*match(pattern, word)*', that can be used to determine if a given pattern matches a given word.

A pattern consists of letters and '?' wildcards.

A '?' wildcard matches any letter at the corresponding position in the word.

On the Vula page for this assignment you will find a program called *testsimple.py* that you can use to check your function. (The automatic marker will use this.)

Sample I/O:

```
Enter a pattern (or 'q' to quit):
l?ad
Enter a word:
lead
It's a match.
Enter a pattern (or 'q' to quit):
le?d
Enter a word:
led
They don't match.
Enter a pattern (or 'q' to quit):
l??d
Enter a word:
lend
It's a match.
Enter a pattern (or 'q' to quit):
q
```

HINTS:

- I can think of three base cases. One is when both strings are empty. Another is when one is empty and the other is not. And the third is …
- The palindrome function makes progress by 'eating' the characters at the ends of the string. This function makes progress by eating the first letter of each string.

CHALLENGE (NOT FOR MARKS): Write a module called *advancedmatch.py* recursive function called '*match(pattern, word)*' that can be used to determine if a given pattern matches a given word.

In this case, a pattern consists of letters and '?' and '*' wildcards.

A '?' wildcard matches any letter at the corresponding position in the word.

A '*' wildcard matches zero or more letters at the corresponding position in the word.

HINTS:

- We now have an extra base case: if *word* is *''* and *pattern* is *'*'* then return *True*.
- Generally, when the first character of the pattern is a *'*'* wildcard, try to (i) match the rest of the pattern to the word, or (ii) match the pattern to the rest of the word.

Question Three [30 marks]

Write a program called '`wordsearch.py`' that searches a file of words that match a given pattern.

Sample I/O (User input in bold):

```
***** Word Search *****
Enter the name of the words file:
EnglishWords.txt
Enter a search pattern:
r?v?r
['rover', 'raver', 'river']
```

Sample I/O (User input in bold):

```
***** Word Search *****
Enter the name of the words file:
EnglishWords.txt
Enter a search pattern:
???bron
Sorry, matches for '???bron' could not be found.
```

Sample I/O (User input in bold):

```
***** Word Search *****
Enter the name of the words file:
words.txt
Sorry, could not find file 'words.txt'.
```

The last example of program behaviour shows what the program should do if it does not find the words file.

A sample lexicon (words file) has been provided on the Vula assignment page ('`EnglishWords.txt`').

This file begins with a copyright notice that should not be removed.

- You should assume that any file used by your program will contain such a notice.
- Your program must skip this notice by reading lines until it encounters one consisting of the word "START". (You might consider writing a function to do this.)

NOTE:

- You are expected to complete this question using your solution for question two i.e. the `wordsearch.py` program will import the `match` function from `simplematch.py`.
- If you struggled with question two, you can complete this one using a `simplematch.py` that contains iterative solution to pattern matching (i.e. a match function that uses loops.)

Question Four [30 marks]

This question concerns the use of a book cypher to encode a message.

Write a program called '`bookcypher.py`' that accepts:

1. The file name for a Project Gutenberg book text.
2. A message that is to be encoded.
3. A file name for output of the encoded version of the message.

The program will create an encoded version of the message by replacing its words with the locations of words in the book.

Say, for example, the message is 'meet me at the usual time and place', and the Gutenberg text is 'The Importance of Being Earnest' by Oscar Wilde.

- There is an occurrence of 'meet' as the 5th word on the 493rd line of the book text.
- There is an occurrence of 'me' as the 9th word on the 190th line of the text.
- There is an occurrence of 'at' as the 6th word on the 8th line of the text.
- And so on.

Representing a word position as a conjunction of the line number and the index of the word within the line, we can encode the message as 493-5, 190-9, 8-6, etc.

Sample I/O (user input in bold):

```
Enter the book filename:
ImportanceOfBeingEarnest.txt
Enter the message:
Meet me at the usual time and place
Enter the output filename:
secret.txt

BEGIN
492-5
129-9
4-10
1-1
174-3
71-1
4-13
253-6
END
```

CONTINUED

The program will output results to the given file (‘`secret.txt`’ in the example) AND to the screen.

- The first line of results will consist of the word ‘BEGIN’.
- Subsequent lines will consist of word encodings, one per line.
- A word encoding will consist of a line number, a hyphen, and a word index.
- The last line of results will consist of the word ‘`END`’.

NOTE:

- You can assume the message does not contain punctuation.
- If a word match cannot be found, the program should output the word itself instead of a line number and word index encoding.
- You can use the string ‘`split()`’ method to split a sentence up into words e.g.

```
>>> 'how now, brown cow.'.split()
    ['how', 'now,', 'brown', 'cow.']
>>>
```

  The result is a list of strings, one for each word.
- To clean up the results of `split()`, i.e. to get rid of bits of punctuation stuck to words, you can use the string ‘`strip()`’ method e.g.

```
>>> 'eh?'.strip('?!.,;:)([]{}\'" \n')
    'eh'
>>> "('ello!)".strip('?!.,;:)([]{}\'" \n')
    'ello'
>>>
```

  The parameter to the `strip()` method is a string containing instances of all the characters we want to remove.
- To placate the automatic marker, given a word in the message, when searching for a match, the program should always start from the beginning of the text and work towards the end.
- There is a sample Gutenberg text on the Vula assignment page. Gutenberg texts come in various formats. If you wish to download your own, make sure it is in ‘Plain Text UTF-8’ format.

Submission

Create and submit to the automatic marker a Zip file called ‘`ABCXYZ123.zip`’ (where ABCXYZ123 is YOUR student number) containing `palindrome.py`, `simplematch.py`, `wordsearch.py`, and `bookcypher.py`.

END