Version: 2025/04/22 15:40

CSC1015F Assignment 9

Recursion

Learning Objectives

By the end of this assignment, you should be able to:

- Understand what recursion is
- Identify a recursive definition
- Create a recursive function
- Identify what the base case is
- Formulate the recursive call
- Understand the difference between recursion and iteration
- Understand the need for recursion.

Assignment Instructions

This assignment involves constructing Python functions that use recursion. You must **NOT** use loop constructs (such as '`for`' and '`while`') in your solutions.

There are four questions. The first concerns identifying binary palindrome primes, the next two concern constructing a module containing *pattern matching* functions. The last concerns the construction of a power set.

Assessment

Your code will be automatically marked. Say that there are *N* trials for a question. The first *(N-1)* trials will check that your code functions correctly by executing it on test inputs. The *N*th is a penalty test. It scans the code for evidence of the use of iteration or the use of list/string reversal expressions. If it finds evidence, then it deducts the marks for the question.

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'.

Furthermore**,** your solutions to this assignment will be evaluated for correctness and for the following qualities:

- Documentation
  - Use of comments at the top of your code to identify program purpose, author and date.
  - Use of comments within your code to explain each non-obvious functional unit of code.
- General style/readability
  - The use of meaningful names for variables and functions.
- Algorithmic qualities
  - Efficiency, simplicity

These criteria will be manually assessed by a tutor and commented upon. Up to 10 marks will be deducted for deficiencies.

Question one [30 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 program called '`binary_palindrome_primes.py`' that recursively finds and prints all prime numbers between two integers M and N (inclusive) whose binary representations are also palindromes. The Amathuba page for this assignment has the skeleton of `binary_palindrome_primes.py`. Download it and complete the functions indicated.

You may assume it's always the case that M>*1*, and that *M≤N*.

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

Add the following lines at the top of your program to increase the amount of recursion that Python will allow:

```
import sys
sys.setrecursionlimit (30000)
```

Sample IO *(The input from the user is shown in* ***bold*** *font – do not program this)*:

```
Enter the starting point M:
20
Enter the ending point N:
50
Binary palindrome primes between 20 and 50 are:
31 (binary: 11111)
```

Sample IO *(The input from the user is shown in* ***bold*** *font – do not program this)*:

```
Enter the starting point M:
11
Enter the ending point N:
999
Binary palindrome primes between 11 and 999 are:
17 (binary: 10001)
31 (binary: 11111)
73 (binary: 1001001)
107 (binary: 1101011)
127 (binary: 1111111)
257 (binary: 100000001)
313 (binary: 100111001)
443 (binary: 110111011)
```

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, ...

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.

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 Amathuba 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 IO *(The input from the user is shown in **bold** font – do not program this):*

```
Enter a pattern (or 'q' to quit):
l?ad
Enter a word
lead
It's a match.
Enter a pattern:
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.

Question three [30 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.

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

Sample IO *(The input from the user is shown in **bold** font – do not program this):*

```
Enter a pattern (or 'q' to quit):
l*ad
Enter a word:
launchpad
It's a match.
Enter a pattern (or 'q' to quit):
*ad
Enter a word:
lead
It's a match.
Enter a pattern (or 'q' to quit):
```

```
**ad
Enter a word:
lard
They don't match.
Enter a pattern (or 'q' to quit):
q
```

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 Four [20 marks]

Write a program called '`power_set.py`' that uses recursive functions to find a power set. The power set of a set is the set of all possible subsets, including:

- The empty set ∅
- The set itself

  For a set with n elements, the power set contains 2^n subsets.

For example:

For the set {1, 2}, the power set is:

`[[], [1], [2], [1, 2]]`

You MUST NOT use any form of loop in your program! (Yes, we're pretty obsessive about this.)

***Sample IO** (The input from the user is shown in **bold** font – do not program this):*

```
Enter elements of the set separated by spaces:
3 2 5 4
There are 16 subsets.
The subsets making up the power set are:
[]
['4']
['5']
['5', '4']
['2']
['2', '4']
['2', '5']
['2', '5', '4']
['3']
['3', '4']
['3', '5']
['3', '5', '4']
['3', '2']
['3', '2', '4']
['3', '2', '5']
['3', '2', '5', '4']
```

Submission

Create and submit a Zip file called ‘`ABCXYZ123.zip`’ (where ABCXYZ123 is YOUR student number) containing `simplematch.py, binary_palindromeprime_primes.py, advancedmatch.py` and `power_set.py`.