Homework 6
Recursion

Due Date: Monday, November 9, 2020 by 11:59:59 PM
Value: 50 points

This assignment falls under the standard cmsc201 academic integrity policy. This means you should not discuss/show/copy/distribute your solutions, your code or your main ideas for the solutions to any other student. Also, you should not post these problems on any forum, internet solutions website, etc.

Make sure that you have a complete file header comment at the top of each file, and that all of the information is correctly filled out.

```
"""
File:     FILENAME.py
Author:   YOUR NAME
Date:     THE DATE
Section:  YOUR DISCUSSION SECTION NUMBER
E-mail:   YOUR_EMAIL@umbc.edu
Description:
  DESCRIPTION OF WHAT THE PROGRAM DOES
"""
```

I've rated the problems based on what I think the expected difficulty is. This is not a guarantee about what you'll find hard or easy, but it's basically just my expectation.

Submission Details

Submit the files under the following titles:
(These are case sensitive as usual.)

submit cmsc201 HW6 the_third_degree.py matching_brackets.py down_the_path.py houses_and_people.py project_piece_test.py board_square.py

Problem 1 - The Third Degree	the_third_degree.py
Problem 2 - Matching Brackets	matching_brackets.py
Problem 3 - Down the Path	down_the_path.py
Problem 4 - Houses and People	houses_and_people.py
Problem 5 - A Piece of the Project	project_piece_test.py board_square.py

Note that there are two files you need to submit for problem 5.

Coding Standards

Coding standards can be found here.

We will be looking for:

1. At least one inline comment per program explaining something about your code.
2. Constants above your function definitions, outside of the "if __name__ == '__main__':" block.
    a. A magic value is a string which is outside of a print or input statement, but is used to check a variable, so for instance:
        i. `print(first_creature_name, 'has died in the fight. ')` does not involve magic values.
        ii. However, `if my_string == 'EXIT':` exit is a magic value since it's being used to compare against variables within your code, so it should be:

```
EXIT_STRING = 'EXIT'
...
if my_string == EXIT_STRING:
```

    b. A number is a magic value when it is not 0, 1, and if it is not 2 being used to test parity (even/odd).
    c. A number is magic if it is a position in an array, like my_array[23], where we know that at the 23rd position, there is some special data. Instead it should be
    USERNAME_INDEX = 23
    my_array[USERNAME_INDEX]
    d. Constants in mathematical formulas can either be made into official constants or kept in a formula.
3. Previously checked coding standards involving:
    a. snake_case_variable_names
    b. CAPITAL_SNAKE_CASE_CONSTANT_NAMES
    c. Use of whitespace (2 before and after a function, 1 for readability.)

Allowed Built-ins/Methods/etc

- Declaring and assigning variables, ints, floats, bools, strings, lists, dicts.
- Using +, -, *, /, //, %, **; +=, -=, *=, /=, //=, %=, **= where appropriate
- Comparisons ==, <=, >=, >, <, !=, in
- Logical and, or, not
- if/elif/else, nested if statements
- Casting int(x), str(x), float(x), (technically bool(x))
- For loops, both *for i* and *for each* type.
- While loops
    - sentinel values, boolean flags to terminate while loops
- Lists, list(), indexing, i.e. my_list[i] or my_list[3]
    - 2d-lists if you want them/need them my_2d[i][j]
    - Append, remove
    - **list slicing**
- If you have read this section, then you know the secret word is: createous.
- String operations, concatenation +, +=, split(), strip(), join(), upper(), lower(), isupper(), islower()
    - **string slicing**
- Print, with string formatting, with end= or sep=:
    - '{}'.format(var), '%d' % some_int, f-strings
    - Really the point is that we don't care how you format strings in Python
    - Ord, chr, but you won't need them this time.
- Input, again with string formatting in the prompt, casting the returned value.

- **Dictionaries**
  - creation using dict(), or {}, copying using dict(other_dict)
  - .get(value, not_found_value) method
  - accessing, inserting elements, removing elements.
- Using the functions provided to you in the starter code.
- Using import with libraries and specific functions **as allowed** by the project/homework.
- **Recursion - allowed for HW6**

Forbidden Built-ins/Methods/etc

This is not a complete listing, but it includes:

- break, continue
- methods outside those permitted within allowed types
    - for instance str.endswith
    - list.index, list.count, etc.
- Keywords you definitely don't need: await, as, assert, async, class, except, finally, global, lambda, nonlocal, raise, try, yield
- The *is* keyword is forbidden, not because it's necessarily bad, but because it doesn't behave as you might expect (it's not the same as ==).
- built in functions: any, all, breakpoint, callable, classmethod, compile, exec, delattr, divmod, enumerate, filter, map, max, min, isinstance, issubclass, iter, locals, oct, next, memoryview, property, repr, reversed, round, set, setattr, sorted, staticmethod, sum, super, type, vars, zip
- If you have read this section, then you know the secret word is: argumentative.
- exit() or quit()
- If something is not on the allowed list, not on this list, then it is probably forbidden.
- The forbidden list can always be overridden by a particular problem, so if a problem allows something on this list, then it is allowed for that problem.

Problem 1 - The Third Degree (Easy)

Value: 15 points

Create a recursive function the_third_degree(n) which takes an integer n as an argument and returns the n^{th} term in the sequence:

$$D_n = \begin{cases} 2 & n = 0 \\ 1 & n = 1 \\ 5 & n = 2 \\ D_n = 3D_{n-1} + 2D_{n-2} + D_{n-3} & n > 2 \end{cases}$$

Calculating the sequence means for instance when n = 3:
$D_3 = 3(5) + 2(1) + 2 = 19$
$D_4 = 3(19) + 2(5) + 1 = 68$

Test Driver and Starter Code

The test driver code is:

```
if __name__ == '__main__':
    for i in range(10):
        print(the_third_degree(i))
```

Sample Output

```
linux5[201]% python3 the_third_degree.py
2
1
5
19
68
247
896
3250
11789
42763
```

Problem 2 - Matching Brackets (Hard)

Value: 20 points

Write a *recursive* function:

```
def match_brackets(bracket_string, count=0):
```

This function's goal is to match curly braces in the following way:

1. If there is a start curly brace { you should increment the count when you recurse.
2. If there is an end curly brace } you should decrement the count.
3. Ignore any other characters.

Brackets must match like this:
{{a}{b}{{c}d}}

An example of unmatched brackets is:
1. }a{b{c end brackets before start brackets
2. {a{b} not enough end brackets to close the expression
3. {a}b} too many end brackets.

If the brackets match, then you should return True otherwise return False.

Hints, consider these things:

1. What happens when the count is negative?
2. What happens if the string is empty?
3. There are multiple base cases and multiple recursive steps.

Sample Output

```
linux5[109]% python3 matching_brackets.py
Enter a string with brackets: {{a}{b}}
True
Enter a string with brackets: a{{{}}}{{{{}}}}
True
Enter a string with brackets: }{}}
False
Enter a string with brackets: {}{}{
False
Enter a string with brackets: {abcd}
True
Enter a string with brackets: {a{b{c}}}
True
Enter a string with brackets: quit
```

Problem 3 - Down the Path (Medium)

Value: 15 points

Create a recursive function:

```
def down_the_path(n):
    """
    :param n: an integer
    :return: the number of times that down the path runs
    """
```

This recursive function should simply calculate the number of times which it runs. In order to do something like that, you should play with the function:

```
def count_down(count):
    if count <= 0:
        return 0
    else:
        return 1 + count_down(count - 1)
```

However, down_the_path will count in a different way.

1. If n is divisible by 15, then divide by 15 and go to the next step.
2. If n is divisible by 5 then divide by 5 and go to the next step.
3. If n is divisible by 3 then divide by 3 and go to the next step.
4. If n is greater than 0 but none of the rest, then subtract 1 and go to the next step.
5. If n is negative or zero, then return 0.

Sample Output

The sample output is generated by:

```
if __name__ == '__main__':
  for i in range(20):
    print(i, down_the_path(i))
```

```
linux5[109]% python3 down_the_path.py
0 0
1 1
2 2
3 2
4 3
5 2
6 3
7 4
8 5
9 3
10 3
11 4
12 4
13 5
14 6
15 2
16 3
17 4
18 4
19 5
```