This problem set is due Thursday, November 2 at 11:59pm, KST. Note that that the due date that you see on Blackboard is not accurate since it shows the time in EST. You should go by the due date in this handout.

Add your name and email address as a comment at the top of each file you submit.

What to submit

Hand in your work on Blackboard. Your submission should include the following. Please do not submit other files that I did not ask for.

`ps4.py`

Multiple submissions are allowed before the due date and time. Late submissions will not be graded.

Assignment objectives

After completing this problem set, you should be able to write short functions that solve simple problems in a recursive manner.

Incomplete work

Please read what I said about this in PS 2.

Naming conventions in Python and programming style in general

Please read what I said about this in PS 2.

Problem 1. Recursive f (5 points)

A function $f$ is defined by the following rule:

$$ f(n) = \begin{cases} 
  n & \text{if } n < 3 \\
  f(n-1) + 2f(n-2) + 3f(n-3) & \text{if } n \geq 3
\end{cases} $$

Complete the function $f$ given in `ps4.py`. This function (which must be recursive) takes a single parameter (an integer $n$) and computes and returns the value of the given function.

You can see some sample calls of this function in the testing part (main) of `ps4.py`.

Problem 2. Recursive Power (5 points)
Complete the function `power` given in `ps4.py`. This function (which must be recursive) takes two parameters (a double \( x \) and an integer \( n \)) and that returns \( x^n \).

**Hint:** a recursive definition of this operation is \( \text{power}(x, n) = x \times \text{power}(x, n - 1) \). Also, remember that anything raised to the zeroth power is 1.

You can see some sample calls of this function in the testing part (main) of ps4.py.

---

**Problem 3. Recursive GCD (Euclid’s Algorithm) (5 points)**

This following algorithm is known as Euclid’s Algorithm because it appears in Euclid’s *Elements* (Book 7, ca. 300 B.C.). It may be the oldest nontrivial algorithm.

The algorithm is based on the observation that if \( r \) is the remainder when \( a \) is divided by \( b \), then the common divisors of \( a \) and \( b \) are the same as the common divisors of \( b \) and \( r \). Thus, we can use the equation:

\[
\text{gcd}(a, b) = \text{gcd}(b, r)
\]

to successively reduce the problem of computing a GCD to the problem of computing the GCD of smaller and smaller pairs of integers. For example,

\[
\text{gcd}(36, 20) = \text{gcd}(20, 16) = \text{gcd}(16, 4) = \text{gcd}(4, 0) = 4
\]

implies that the GCD of 36 and 20 is 4. It can be shown that for any two starting numbers, this repeated reduction eventually produces a pair where the second number is 0. Then, the GCD is the other number in the pair.

Complete the function `gcd` given in `ps4.py`. This function (which must be recursive) takes two integer parameters, say \( a \) and \( b \), and uses Euclid’s Algorithm to compute and return the greatest common divisor of the two numbers.

You can see some sample calls of this function in the testing part (main) of ps4.py.

---

**Problem 4. Recursive Print of a List (10 points)**

Complete the function `print_list` given in `ps4.py`. This function (which must be recursive) takes a single list parameter and prints the elements of the list, one element per line.

A sample call:

```
p = [1, 2, 3, 4]
p = print_list(p)  # would produce the output:
1
2
3
4
```

You can see some sample calls of this function in the testing part (main) of ps4.py.

---

**Problem 5. Recursive Print Backward of a List (10 points)**

Complete the function `print_list_backward` given in `ps4.py`. This function (which must be recursive) takes a single list parameter and prints the elements of the list backwards (one element per line).

A sample call:

```
p = [1, 2, 3, 4]
p = print_backward(p)  # would produce the output:
4
3
```

You can see some sample calls of this function in the testing part (main) of ps4.py.
You can see some sample calls of this function in the testing part (main) of ps4.py.

Problem 6. Recursive Reverse of a List (10 points)
Complete the function reverse_list given in ps4.py. This function (which must be recursive) takes a list as a single parameter and returns a new list. The new list should contain the same elements as the parameter, but in reverse order.

Sample calls:

reverse_list([1, 2, 3, 4]) returns [4, 3, 2, 1]
reverse_list([1, [2, 4], [1, 2, 3], 4]) returns [4, [1, 2, 3], [2, 4], 1]

You can see these sample calls in the testing part (main) of ps4.py.

Problem 7. Recursive Salting (10 points)
Complete the function alee_salting given in ps4.py. This function (which must be recursive) takes the following parameters, in this order:

1. text: a string containing alphanumeric characters
2. salt: another string containing alphanumeric characters

Your function should append the salt string after each character of the text, and return the string of interleaved characters. For example, if the text is "wxyz" and salt is "CSE101" the return value is "wCSE101xCSE101yCSE101zCSE101".

Note: If length of text is 0, simply return text. If the length of text is 1, your function should still append the salt string anyway.

A possible pseudocode for the function would be:

if the length of text is zero then
    return text
otherwise
    return the concatenation of (a) the first letter of the text,
        (b) the salt string, and (c) the remainder of the text after
        "salting" the remainder

Here are some example calls:

alee_salting("ILOVECSE101", "|") returns "I|L|O|V|E|C|S|E|1|0|1|
alee_salting("Prof", "aLeE") returns "PaLeEraLeEoaLeEfaLeE"
alee_salting("", "TEST") returns ""
alee_salting("A", "xyz") returns "Axyz"

Problem 8. Sum of Pairs (15 points)
Complete the function sum_pairs given in ps4.py. This function (which must be recursive) takes the following parameters, in this order:

1. u: a list of integers
2. \( v \): another list of integers

The function recursively computes the pairwise sums of all elements from \( u \) and \( v \), inserting each sum into a list that is returned by the function. If one list is shorter than the other, then the extra elements from the longer list are simply appended to the returned list.

A possible **pseudocode** for the recursive call only would be:

```python
let x = the sum of the first two elements of the lists
let y = a list containing the sums of all pairs of elements from the two lists,
       EXCEPT for the sum of the first pair
insert the sum \( x + y \) at the first position of \( y \)
return \( y \)
```

Here are some example calls:

- \( \text{sum_pairs([0, 1, 7], [2, 3]) returns [2, 4, 7]} \)
- \( \text{sum_pairs([1, 2, 3, 4], [5]) returns [1, 2, 3, 4, 5]} \)
- \( \text{sum_pairs([6, 7, 8], [15, 14, 13]) returns [21, 21, 21]} \)

**Problem 9. Recursive GPA Calculator (15 points)**

Complete the function `gpa_calculator` given in `ps4.py`. This function takes a single parameter:

- `grades`: a list containing alternating letter grades and credits for a group of courses taken by a student (see below for further explanation). Only the grades A, B, C, D and F are possible.

A list of grades like `['A', 3, 'B', 4, 'D', 3]` indicates that the student took three courses with the respective credit values (3, 4 and 3). The function computes the weighted GPA and returns it. Note that letter grades can be provided in uppercase or lowercase.

**Note:** Your code should be able to handle invalid grades by returning `None`. Invalid grades include:

1. A negative or zero credit value.
2. A letter grade that is not one of A/a/B/b/C/c/D/d/F/f.

**Hint:** Consider using helper functions to calculate (1) the total number of credits and (2) the GPA. Pseudocode is given for both below:

A possible **pseudocode** for a (recursive) helper function that calculates the credit total:

```python
def credit_sum(grades):
    if the length of grades is zero then
        return 0.0
    otherwise
        let x = the letter grade of the first course in the list
        let y = the # of credits of the first course in the list
        if either x or y is invalid then
            return None
        let z = sum of credits of the courses after current course
        if z is None then
            return None
        otherwise
            return the sum \( y + z \)
```
A possible **pseudocode** for a (recursive) helper function that calculates the total grade points:

```python
def grade_points(grades):
    if the length of grades is zero then
        return 0.0
    otherwise
        let x = the letter grade of the first course in the list
        let y = the # of credits of the first course in the list
        if either x or y is invalid then
            return None
        let g = the grade point associated with x
        let r = sum of grade points of courses after the first course
        if r is None:
            return None
        otherwise:
            return the sum (y * g) + r
```

Grade points are calculated based on the following table:

<table>
<thead>
<tr>
<th>Grade</th>
<th>Grade Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4.0</td>
</tr>
<tr>
<td>B</td>
<td>3.0</td>
</tr>
<tr>
<td>C</td>
<td>2.0</td>
</tr>
<tr>
<td>D</td>
<td>1.0</td>
</tr>
<tr>
<td>F</td>
<td>0.0</td>
</tr>
</tbody>
</table>

A possible **pseudocode** for top-level function:

```python
def gpa_calculator(grades):
    return grade_points(grades) / credit_sum(grades)
```

The pseudocode for `gpa_calculator` is actually incomplete. There are some error cases you will have to incorporate.

Here are some example calls:

```python
gpa_calculator(['A', 4, 'B', 4, 'A' 4]) returns 3.666...
gpa_calculator(['A', 4, 'Q', 3, 'C' 1, 'F', 2]) returns None
gpa_calculator([]) returns 0.0
```

**Problem 10. Recursive Merge (15 points)**

Complete the function `rmerge` given in `ps4.py`. This function takes the following parameters, in this order:

1. u: a list of integers, where integers are in sorted order, in ascending order.
2. v: another list of integers, where integers are in sorted order, in ascending order.

The function (which must be recursive or at least use a recursive helper function) merges the two lists, `u` and `v`, into one list and returns it. The result is also in sorted order, in ascending order.

**Note:** Either or both of the lists `u` and `v` may be empty. Their lengths may not be the same.
**Hint:** Consider using an auxiliary function (helper function), say `rmerge_aux`. With this design, `rmerge` sets things up and calls `rmerge_aux`. `rmerge_aux` then does most of its work. With this plan `rmerge_aux` would be recursive, but `rmerge` itself is not.

Here are some example calls:

- `rmerge([], [2, 3])` returns `[2, 3]`
- `rmerge([1, 2, 3], [])` returns `[1, 2, 3]`
- `rmerge([1, 2, 3], [1, 2, 3])` returns `[1, 1, 2, 2, 3, 3]`
- `rmerge([1, 4, 6], [1, 2, 3, 7])` returns `[1, 1, 2, 3, 4, 6, 7]`