CS 1: Introduction to Computer Programming

Recitation 2: Prepare to Battle Ships (part 2!) Solutions

In this second "problem-solving session", we will take a closer look at functions and cumulative algorithms. We will explore how functions can help you structure code more effectively to make it reusable. We will also see some cases of cumulative algorithms which are essential for solving problems that require maintaining a running sum, product, or result.

If you have any questions, don't hesitate to ask for help! That's what we're here for!

Common Errors

Always Use a `return` to Pass Back a Value

Input/Output Example

```
1  def add_two_numbers(operand1, operand2):
2      sum = operand1 + operand2  # sum is not returned
3
4  print(add_two_numbers(3, 5))
5
6  def add_two_numbers(a, b):
7      sum = operand1 + operand2
8      return sum  # value passed back when function is called
9
10 print(add_two_numbers(3, 5))
```

```
>> None
>> 8
```

Arguments Should be Used to Avoid Hard-Coding Values

Input/Output Example

```
1  def greeting_message_hopper():
2      print("Nice to meet you, Hopper!")
3
4  def greeting_message_adam():
5      print("Nice to meet you, Adam!")
6
7  greeting_message_hopper()
8  greeting_message_adam()
9
10 def greeting_message(user):
11     print(f"Nice to meet you, {user}!")
12
13 greeting_message("Hopper")  # we can use a single function for all users
14 greeting_message("Adam")
```

```
>> Nice to meet you, Hopper!
>> Nice to meet you, Adam!
>> Nice to meet you, Hopper!
>> Nice to meet you, Adam!
```

Understand Variable Scope Within Functions

Input/Output Example

```
1  bank_balance = 5000
2
3  def subtract_from_balance(balance, amount):
4      bank_balance = balance - amount # variables defined in a function are local to that
       function
5
6  subtract_from_balance(bank_balance, 2000)
7  print(bank_balance)
```

```
>> 5000
```

Encapsulate Repeating Code

Input/Output Example

```
1  game_1_result = "win"
2  game_2_result = "lose"
3
4  if game_1_result == "win":
5      print("You won!")
6  else:
7      print("You lost!")
8
9  if game_2_result == "win":
10     print("You won!")
11 else:
12     print("You lost!")  # imagine we have 50 other games...
13
14
15 def game_result(result):
16     if result == "win":
17     print("You won!")
18 else:
19     print("You lost!")  # we can use this function for all games!
20
21 game_result(game_1_result)
22 game_result(game_2_result)
```

```
>> You won!
>> You lost!
>> You won!
>> You lost!
```

Use Descriptive Argument Names

Input/Output Example

```
1 def merch_order(a, b, c):
2     print(f"You ordered {c} {b} {a}.")
3
4 merch_order(50, "blue", "t-shirts")  # I am not even sure what to put where...
5
6 def merch_order(item, color, quantity):
7     print(f"You ordered {quantity} {color} {item}.")
8
9 merch_order("t-shirts", "blue", 50) # now that's much easier to understand
```

```
>> You ordered t-shirts blue 50.
>> You ordered 50 blue t-shirts.
```

Ensure Proper Indentation

Input/Output Example

```
1  def sum_to_n(n):
2      sum = 0
3      for i in range(n+1):
4          sum += i
5          return sum  # sum is returned prematurely
6
7  print(sum_to_n(10))
8
9  def sum_to_n(n):
10     sum = 0
11     for i in range(n+1):
12         sum += i
13     return sum  # sum is returned once all iterations of the for loop have been completed
14
15 print(sum_to_n(10))
```

```
>> 0
>> 55
```

Problem Solutions

Some `randomization`

Worked Example

Mastermind

In the game Mastermind, players must guess a color combination in the least amount of turns. The colors they can guess from are: red (R), green (G), blue (B), yellow (Y), orange (O), pink (P), grey (Gr), and white (W). Write a function `make_combination` that returns a string of 4 random colors.

```
import random
colors = ["R", "G", "B", "Y", "O", "P", "Gr", "W"]

def make_combination():
    combination = ""
    for i in range(4):
        combination += random.choice(colors)
    return combination

print(make_combination()) # Example output: "RGBB"
```

Faded Example

License Plates

In many states, license plates follow a specific format: 3 uppercase letters followed by 4 digits (e.g. "ABC1234"). Write a function `generate_license_plate()` that returns a string in this format.

```
import random
import string

upper_case_letters = string.ascii_uppercase # string containing all ASCII uppercase characters

def generate_license_plate():
    license_plate = ""
    for i in range(3) :
        licence_plate += random.choice(upper_case_letters)
    for i in range(4) :
        licence_plate += random.randint(0,9)
    return license_plate
```

Your Turn

Treasure Map

Create a function `generate_treasure_map` that takes two arguments, `rows` and `cols`, which represent the size of the treasure map. The function should return a list of string rows where one randomly selected cell contains a treasure, marked as `"T"`, and all other cells are represented by `"."`. A good way to structure your code would be to make a 2D grid filled with `"."`, update the value of a random cell to `"T"`, and finally you can `.join` the elements of each row.

Example output of `generate_treasure_map(4, 5)`:

```
["....", ".....", "T....", "....."]
```

```
import random
```

Solution:

```
def generate_treasure_map(rows, cols):
    map = []
    for i in range(rows):
        row = []
        for j in range(cols):
            row.append(".")
        map.append(row)

    map[random.randint(0, rows-1)][random.randint(0, cols-1)] = "T"

    for i in range(len(map)):
        map[i] = "".join(map[i])
    return map
```

Cumulative Algorithms 1 (DNA)

Worked Example

Counting Nucleotides

Write a function `count_nucleotide` that takes in a `dna_sequence` and a `nucleotide` and returns the number of times that `nucleotide` appears in the `dna_sequence` .

```
1    def count_nucleotide(dna_sequence, nucleotide):
2        count = 0
3        for nuc in dna_sequence:
4            if nuc == nucleotide:
5                count += 1
6        return count
```

Faded Example

Mutations

Write a function `count_mutations(orig_seq, synth_seq)` that counts the number of mutations in `synth_seq`, which was synthesized from `orig_seq`. Assume you have a function `matching_sequence(orig_seq)` that returns the perfect synthesized sequence from `orig_seq`.

```
1    def count_mutations(orig_seq, synth_seq):
2        perfect_seq = matching_sequence(orig_seq)
3
4        mutations = [ 0 ]
5        for [ i in range(len(synth_seq) ] :
6            if perfect_seq[i] [ != ] synth_seq[i]:
7                mutations [ += 1 ]
8        return [ mutations ]
```

Your Turn

Longest Consecutive Nucleotide

Write a function `longest_repeating_nucleotide` that takes in a `dna_sequence` and returns the length of the longest consecutive sequence of the same nucleotide in `dna_sequence`.
Hint: think about what you will need to track and initialize the proper variables. In your for loop, what are the different cases you might want to consider?

- `longest_repeating_nucleotide("AATTTGC")` should return 3
- `longest_repeating_nucleotide("AATTCCGG")` should return 2

Solution:

```
1  def longest_repeating_nucleotide(dna_sequence):
2     max = 0
3     current_length = 0
4     current_nucleotide = ''
5     for i in range(len(dna_sequence)):
6        if current_nucleotide != dna_sequence[i]:
7           current_nucleotide = dna_sequence[i]
8           current_length = 0
9        current_length += 1
10       if current_length > max:
11          max = current_length
12    return max
```

Cumulative Algorithms 2 (Rainfall)

Worked Example

Total Rainfall

Write a function `total_rainfall` that takes in a list of daily rainfall amounts, `rainfalls`, and returns the total rainfall on days when it rained more than 1 unit.

```
1  def total_rainfall(rainfalls):
2      total = 0
3      for rain in rainfalls:
4          if rain > 1: # More than 1 unit of rain
5              total += rain
6      return total
```

Faded Example

Weighted Average Rainfall

Write a function `weighted_average_rainfall` that takes in the lists `rainfalls` and `weights` and returns the weighted average of rainfall on days where it rained more than 1 unit. The `weights` list corresponds to the importance of each days rainfall.

```
1  def weighted_average_rainfall(rainfalls, weights):
2      total_weighted_rain = 0
3      total_weight = 0
4      for i in range( len(rainfalls) ):
5          if rain > 1 :
6              total_weighted_rain += rainfalls[i] * weights[i]
7              total_weight += weights[i]
8      if total_weight > 0 :
9          return total_weighted_rain / total_weight
10     else:
11         return 0
```

Your Turn

Cumulative Rainfall

Write a function `cumulative_rainfall` that takes in a list of rainfall amount per day, `rainfalls`, and returns a new list with each element corresponding to the total amount of rainfall from day 0 to day `i`.

- `cumulative_rainfall([0,1,1,2,0,0,3,4])` should return `[0,1,2,4,4,4,7,11]`
- `cumulative_rainfall([2,5,0,2,0,0,0,2])` should return `[2,7,7,9,9,9,9,11]`

Solution:

```
1  def cumulative_rainfall(rainfalls):
2      cumulative = []
3      total = 0
4      for i in range(len(rainfalls)):
5          total += rainfalls[i]
6          cumulative.append(total)
7      return cumulative
```