🤟

Problem Set 3

---

a. string_times

Prompt

Write a function called `string_times` that, given a string `msg` and a non-negative int `n`, returns a larger string that is `n` copies of the original string.

Starter code

```
def string_times(msg, n):
```

Example Input + Output

input	output	notes
string_times(Hi, 2)	HiHi	
string_times(Hi, 3)	HiHiHi	
string_times(Hello, 0)		
string_times(Hello! , 4)	Hello! Hello! Hello! Hello!	

Example Solution

▼ Click here for the solution

```
# pythonic solution
def string_times(msg, n):
  return msg * n

# loop solution
def string_times(msg, n):
  result = ''
  for i in range(n):
    result = result + msg
  return result
```

Notes

▼ Click here for more information

- What's being assessed: Pythonic string manipulation
- What's challenging: pythonic syntax, but can also be completed with a loop
- Source/Inspired by: Item from Matt's NCME 2019 study
- Written by: Benji Xie
- Don't need logical "or" for this item
- From Matt's study

Task Complexity

Branch	NO
Loop	YES
Nested Branch	NO

---

b. sum_target

Prompt

Write a function called `sum_target` that takes in 3 parameters: an integer `n`, an integer `m`, and an integer `target`. Return `True` if the last digit of n and the last digit of m add up to `target`.

For example, `sum_target(325, 124, 9)` returns True since 5 and 4 add up to 9.

You may assume that all numbers passed in are positive integers.

Starter code

```
def sum_target(n, m, target):
```

Example Input + Output

input	output	notes
sum_target(324, 123, 7)	True	
sum_target(9, 28, 17)	True	
sum_target(0, 1, 0)	False	

Example Solution

▼ Click here for the solution

```
# one-line solution
def sum_target(n, m, target):
  return (n % 10) + (m % 10) == target

# refactored, multi-line solution
def sum_target(n, m, target):
  digit_n = n % 10
  digit_m = m % 10
  return digit_n + digit_m == target
```

Notes

▼ Click here for more information

- What's being assessed: conditions,
- What's challenging: accounting for exclusive bound, separating conditions based on is_snowing parameter
- Note: solution can also factor out modulus on n and m to avoid chaining operators
- Source/Inspired by: trying to make another integer manipulation problem
- Written by: Paul Pham

Task Complexity

Branch	NO
Loop	NO
Nested Branch	NO

---

c. monthly_rent

Prompt

Write a function called `monthly_rent(option, credit_score, renters)` where `price`, `credit_score`, and `renters` are integers. The `price` parameter represents the monthly cost of a property, `credit_score` parameter represents the renter(s) average credit score, and `renters` indicates how many people are splitting the rent. This function should return an integer that represents how much each renter will pay per month.

This function should implement the following rules:

1. If the credit score is greater than or equal to 740, apply a 10% discount to the overall monthly rent.
2. Return the amount of money each renter would pay assuming they split the cost of rent equally.

Starter code

```
def monthly_rent(price, credit_score, renters):
```

Example Input + Output

input	output	notes
monthly_rent(1500, 740, 2)	675	trigger discount case
monthly_rent(2200, 750, 3)	660	trigger discount case
monthly_rent(1200, 650, 4)	300	

Example Solution

▼ Click here for the solution

```
def monthly_rent(option, credit_score, renters):
  if credit_score >= 740:
    price *= 0.9

  return price / renters
```

Notes

▼ Click here for more information

Task Complexity

Branch	NO
Loop	NO
Nested Branch	NO

d. can_buy_car

Prompt

Write a function called `can_buy_car(is_sports_car, credit_score, cash)` that will return `True` if you can buy a car with the given parameters, or `False` otherwise.

The parameter `is_sports_car` is a boolean while `credit_score` and `cash` are integers. This function has the following rules:

- A sports car costs 100 while a non-sports car costs 70
- A credit score below 600 (inclusive) will add 10% of the cost to the overall price of the car
- The `cash` amount must be greater than or equal to the cost of the car in order for the purchase to be successful

Starter code

```
def can_buy_car(is_sports_car, credit_score, cash):
```

Example Input + Output

input	output	notes
can_buy_car(True, 700, 50)	False	
can_buy_car(True, 725, 100)	True	
can_buy_car(False, 690, 77)	True	

Example Solution

▼ Click here for the solution

```
def can_buy_car(is_sports_car, credit_score, cash):
  # ver. 1: conditional with ternary operator
  cost = 100 if is_sports_car else cost = 70
```

```
# ver. 2: conditional structure without ternary operator
cost = 0
if is_sports_car:
  cost = 100
else:
  cost = 70

if credit_score <= 600:
  cost *= 1.10

return cash >= cost
```

Notes

▼ Click here for more information

- What's being assessed: conditionals, operators
- What's challenging: multiple branches, separate special case
- Source/Inspired by: previous problem
- Written by: Paul Pham

Task Complexity

Branch	YES
Loop	NO
Nested Branch	NO

---

e. factorial

Prompt

Write a function called `factorial` that takes an integer `n` and returns the factorial of that number. A factorial is defined as the product of an integer and all the integers below it.

For example, the factorial of 3 is 6 because 3 * 2 * 1 = 6.

Note that the factorial of 1 and 0 both evaluate to 1.

Starter code

```
def factorial(n):
```

Example Input + Output

input	output	notes
factorial(5)	120	
factorial(10	3628800	
factorial(0)	1	
factorial(1)	1	

Example Solution

▼ Click here for the solution

```
def factorial(n):
  product = 1
  for i in range(1, n + 1):
    product *= i
  return product
```

Notes

▼ Click here for more information

- What's being assessed: for-loop with inclusive bound, calculating cumulative product
- What's challenging: pulling off the last digit of an integer, using the while loop bound to handle that case
- Source/Inspired by: Item 8: factorial_match
- Written by: Paul Pham

Task Complexity

Branch	NO
Loop	YES
Nested Branch	NO

---

f. unique_nums_in_range

Prompt

Write a function called `get_nums_in_range(nums, low, high)` that accepts a list called `nums`, a lower bound integer (`low`), and an upper bound integer (`high`) and returns a new list containing all the **unique** numbers in `nums` that fall within the range of the low (inclusive) and high (exclusive) bounds.

For example, the call `get_unique_nums_in_range([3, 4, 5], 3, 5)` will return 2 since there are within the given range, which are 3 and 4.

Starter code

```
def unique_nums_in_range(nums, low, high):
```

Example Input + Output

input	output	notes
get_unique_nums_in_range([3, 4, 5], 3, 5)	2	
get_unique_nums_in_range([6, 4, 9], 3, 6)	1	
get_unique_nums_in_range([11, 12, 0], 1, 10)	0	

Example Solution

▼ Click here for the solution

```
def unique_nums_in_range(nums, low, high):
    tracker = []
    for num in nums:
        if num not in result:
            if num >= low and num < high:
                tracker.append(num)

    return len(tracker)
```

Notes

▼ Click here for more information

- What's being assessed: use of multiple conditions, for loops
- What's challenging: getting unique numbers and using nested control structures to achieve that
- Source/Inspired by: similar to sum_within_range
- Written by: Paul Pham

Task Complexity

Branch	YES
Loop	YES
Nested Branch	YES

---

g. compress

Prompt

Write a function called `compress` which takes a string as an argument and returns a new string such that each character is followed by its count, and any adjacent duplicate

characters are removed (replaced by the single character). Assume the string only contains letters and no whitespace.

Example calls and their returns:

- `compress('cooooooooooooooooolkangaroo')` returns `'c1o17l1k1a1n1g1a1r1o2'`.
- `compress('aaa')` returns `'a3'`.
- `compress('')` returns `''`.

Starter code

```
def compress(s):
```

Example Input + Output

input	output	notes
compress('cooooooooooooooooolkangaroo')	'c1o17l1k1a1n1g1a1r1o2'	
compress('aaa')	'a3'	
compress('')	''	

Example Solution

▼ Click here for the solution

```
def compress(s):
  if s == '':
    return ''
  else:
    result = ''
    curr_char = s[0]
    curr_count = 1
    for c in s[1:]:
        if c == curr_char:
            curr_count += 1
        else:
            result += (curr_char + str(curr_count))
            curr_char = c
```

```
            curr_count = 1
    result += (curr_char + str(curr_count))
    return result
```

Notes

▼ Click here for more information

- What's being assessed: for loops with start, stop, step defined, string manipulation, casting
- What's challenging: avoiding out-of-bounds indexing, casting string to integer to multiply, parsing through characters in pairs, everything lol
- Source/Inspired by: taken from CSE 163, Summer 2021 Take-Home Assessment 1: Primer
- Written by: Hunter Schafer

Task Complexity

Branch	YES
Loop	YES
Nested Branch	YES