24 Aug 2022: THIS IS OUTDATED. See
https://pkdpham.notion.site/Problem-Sets-94cd37670117461db04d006ee1c6c48e

For each item include the following:

a. question/prompt (can format w/ markdown)
b. Starter code (usually function header with parameters)
c. 3-5 example inputs and outputs (unit tests)
d. example solution
e. comments or thoughts you may have. Number items as you go.

Item Template

<Question Num>

A. Prompt

<write prompt here>

B. starter code

<write starter code here>

C. Example inputs and outputs

input	output	notes

D. Example Solution

E. Notes

- What's being assessed:
- What's challenging:

- Source/Inspired by:
- Written by:

New Items

1 verify_employee

A. Prompt

Write a function `verify_employee(role_one, role_two, salary_one, salary_two)` where all parameters are numbers and the function has the following rules:

1. return -1 if `role_one` is less than `role_two` and `salary_one` is also less than `salary_two`.
2. return 0 if `role_one` and `role_two` are the same and `salary_one` and `salary_two` are the same
3. return 1 if role_one is greater than role_two and salary_one is also greater than salary_two.
4. return 2 if none of the previous rules could be verified

B. starter code

```
def verify_employee(role_one, role_two, salary_one, salary_two):
```

C. Example inputs and outputs

input	output	notes
verify_employee(1, 1, 0, 0)	0	
verify_employee(1, 2, 10, 100)	-1	
verify_employee(2, 1, 100, 90)	1	
verify_employee(2, 1, 90, 100)	2	
verify_employee(5, 4, 100, 100)	2	
verify_employee(1, 1, 100, 90)	2	
verify_employee(-2, -1, 10, 100)	-1	

D. Example Solution

```
def verify_employee(role_one, role_two, salary_one, salary_two):
   if(role_one < role_two and salary_one < salary_two):
      return -1
```

```
elif(role_one == role_two and salary_one == salary_two):
    return 0
elif(role_one > role_two and salary_one > salary_two):
    return 1
else:
    return 2
```

E. Notes

- What's being assessed: conditions (if, elif, else), operators (<, >, and, ==)
- What's challenging: translating plain english to conditional operators
- Source/Inspired by: Item from Matt's NCME 2019 study
- Written by: Benji
- Don't need logical "or" for this item
- From Matt's study

2 pet_years

A. Prompt

It turns out that dogs and cats age differently from humans and from each other! There's a rule of thumb that many pet owners go by that converts the age of their pet to their equivalent age in human years. Write a function `pet_years(animal, age)` where the parameters have following rules:

1. Animal can only be "dog" or "cat" as a string. If it isn't, return "Animal must be a cat or dog".
2. Age refers to animal age in human-relative years.
3. Convert the age into dog or cat years depending on the animal type.
4. For dog age, the general rule is age 1: 15 dog years, age 2: 24 dog years, age 3+: 24 + 5* (year over 2)
5. For cat age, the rule is age 1: 15 cat years, age 2: 24 cat years, age 3+: 24 + 4 * (year over 2)
6. If the age is less than 0, return "Please enter a positive age."

B. starter code

```
def pet_years(animal, age):
```

C. Example inputs and outputs

input	output	notes
pet_years(dog, 1)	“Your dog is 15 dog years old.”	Should be a set number fixed to 1 human year: 15 dog years
pet_years(dog, 2)	“Your dog is 24 dog years old.”	2 human year: 24 dog years
pet_years(dog, -1)	“Please enter a positive age.”	
pet_years(cat, 4)	“Your cat is 32 cat years old.”	The rule over 2 years old applies here

D. Example Solution

```
def pet_years(animal, age):
  # these are invalid inputs
   If age < 0:
         print(“Please enter a positive age”)
   If animal != “dog” and animal != “cat” :
         print(“Animal must be a dog or a cat.”)

  # handling dog case
   if(animal == “dog”):
         If age == 1:
                  dogAge =  15
         If age == 2:
                  dogAge = 24
         If age >=3:
                  dogAge = 24+(5*(age - 2))
         print(“Your dog is “, str(dogAge), “dog years old.”)

   if(animal == “cat”):
         If age == 1:
                  catAge=  15
         If age == 2:
                  catAge= 24
         If age >=3:
                  catAge= 24+(5*(age - 2))
         print(“Your cat is “, str(catAge), “cat years old.”)
```

E. Notes

- What's being assessed: conditions (if, elif, else), operators (<, ==, >), data types
- What's challenging: dealing with many data types, casting
- Source/Inspired by: Seattle Central College, CSE 110 problem sets
- Written by: Jared

3 monthly_rent

A. Prompt

Write a function `monthly_rent(option, creditScore, renters)` where option is a string; creditScore and renters are integers. The function has the following rules:

1. The option is a string input that contains apartment information stored in letters A and B:
    a. Option “A”: a one bedroom apartment (rent = $1000/month)
    b. Option “B”: a two-bedroom apartment (rent = “$1900/month)
    c. If input is not A or B, return “Please input apartment A or B.”
2. If the credit score is greater than or equal to 740, apply a 10% overall discount to the monthly rent
3. Parameter renters is an integer of the number of renters
4. Calculate the monthly rent per renter.

B. starter code

```
def monthly_rent(option, creditScore, renters):
```

C. Example inputs and outputs

input	output	notes
monthly_rent(“A”, 720, 2)	“Each renter owes $500 every month.”	
monthly_rent(“B”, 750, 3)	“Each renter owes $570 every month.”	
monthly_rent(“B”, 650, 5)	“Every renter owes $380 every month.”	

D. Example Solution

```
def monthly_rent(option, creditScore, renters):
        if option == “A”:
```

```
        price = 1000
elif option == “B”:
        price = 1900
If creditScore >= 740:
        totalPrice = price * (1 - 0.10)
else:
        totalPrice = price

monthlyRent = totalPrice/renters
Return (“Every renter owes $”, monthlyRent, “every month.”)
```

E. Notes

- What's being assessed: conditions (if, elif, else), operators (<, >, and, ==)
- What's challenging: translating plain English to conditional operators
- Source/Inspired by: problem sets from my CSE 110 class at Seattle Central College
- Written by: Jared Lim

4 last_two

A. Prompt

Given a string, return the string made of its last two characters. For example, given the String “Howdy”, this function would return “dy”. If the given String is shorter than 2 characters, return the String itself; an example of this case would be that this function would return “a” with the input String “a”.

B. starter code

```
def last_two(s):
```

C. Example inputs and outputs

input	output	notes
last_two(’howdy’)	'dy’	
last_two(’on’)	'on’	
last_two(’a’)	‘a’	special return case
last_two(’’)	'’	special return case

D. Example Solution

```
def last_two(string):
        return string[-2:]
```

E. Notes

- What's being assessed: conditions (if, else), string manipulation with negative indices
- What's challenging: slicing into strings with negative indices (although this problem could be solved with the slice string[len(string) - 2:]
- Source/Inspired by: CodingBat problem first_two
- Written by: Paul Pham

5 contains_max_diff

A. Prompt

Write a function that takes in a list of integers and returns True if the list contains the number that represents the difference between the largest and smallest numbers in the list and False otherwise.

For example, the call contains_max_diff([10, 9, 5, 3, 4, 7]) would return True since the difference between the largest number (10) and the smallest number (3) is 7, which is in the given list.

If the length of the list is less than two, return None.

B. starter code

```
def contains_max_diff(nums):
```

C. Example inputs and outputs

input	output	notes
contains_max_diff([10, 9, 5, 3, 4, 7])	True	
contains_max_diff([10, 9, 5, 3, 4]))	False	
contains_max_diff([10])	none	

D. Example Solution

```
def contains_max_diff(nums):
    if len(nums) < 2:
        return None

    small = min(nums)
    big = max(nums)

    difference = big - small
    return difference in nums
```

E. Notes

- What's being assessed: use of built-in functions, arithmetic logic
- What's challenging: calculating difference,
- Source/Inspired by: CodingBat problem max_end3
- Written by: Paul Pham

6 expand

A. Prompt

Write a function called expand that takes in a given string that is formatted with alternating letters and numbers. This function should return a new string where each letter is repeated in the resulting string X times, where X is the number that proceeds the letter.

You may assume that the given string will always be in a valid format, so it will always be an even-number in length and it will always contain alternating letters and numbers. Letters will come first and numbers will come second.

If the given String is empty, return an empty String.

[Optional to Include] Hint: you can turn a character of a number into an integer with casting! int('1') → 1

B. starter code

```
def expand(s):
```

C. Example inputs and outputs

input	output	notes
expand('h3e1l3l1o2')	'hhhellllloo'	

expand(’a4’)	‘aaaa’	
expand(’b1b2’)	'bbb’	repeated letters
expand(’o1o1o1o1’)	‘oooo’	repeated letters
expand(’o1r1a1n1g1e1’)	‘orange’	single occurrences of every letter yeild the same input string without numbers
expand(’’)	'’	special return: empty string

D. Example Solution

```
def expand(s):
    if s == '':
        return s
    else:
        result = ''
        for i in range(1, len(s), 2):
            repeat = s[i]
            letter = s[i - 1]
            result += letter * int(repeat)
            print(i, result)
        return result
```

E. Notes

- 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
- Source/Inspired by: compress problem from CSE 163, Summer 2021
- Written by: Paul Pham

7 compress

A. Prompt

Write a function 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.

B. starter code

def compress(s):

C. Example inputs and outputs

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

D. Example Solution

```
def expand(s):
    result = ''
    for i in range(1, len(s), 2):
        repeat = s[i]
        letter = s[i - 1]
        result += letter * int(repeat)
        print(i, result)
    return result
```

E. Notes

- What's being assessed: string manipulation, string indexing, casting
- What's challenging: fenceposting, checking next character in string
- Source/Inspired by: CSE 163, Summer 2021
- Written by: Hunter Schafer(?)

8 factorial_match

A. Prompt

Wrtie a function named factorial_match that accepts two integers: n and target. Return true if the value of n! is equal to target. Factorials are defined as the product of all integers from 1 through that integer, inclusive. For example, the call of factorial_match(4, 24) should return True since 4 * 3 * 2 * 1 = 24.

The factorial of 0 and 1 are defined to be 1. You may assume that both values passed are non-negative and that the factorial of n can fit into the range of type int.

B. starter code

```
def factorial_match(n, target):
```

C. Example inputs and outputs

input	output	note s
factorial_match(4, 24)	true	
factorial(3, 17)	false	
factorial(0, 1)	true	
factorial(1, 5)	false	

D. Example Solution

```
def factorial_match(n, target):
    if n == 0 or n == 1:
        return target == 1

    result = 1
    for i in range(1, n + 1):
        result *= i
    return result == target
```

E. Notes

- What's being assessed: conditionals, for loops, arithmetic logic
- What's challenging: calculating factorial, handling special cases for 0! and 1!
- Source/Inspired by: factorial in CSE 163, Spring 2022
- Written by: Paul Pham

9 sum_within_range

A. Prompt

Write a function called sum_within_range that accepts a list of numbers, a lower bound, and an upper bound. Return a true if the sum of all the numbers in the list falls above the lower bound (exclusive) and below the upper bound (inclusive).

B. starter code

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

C. Example inputs and outputs

input	output	notes
sum_within_range([1, 2, 3], 3, 10)	true	
sum_within_range([3, 5, 6], 0, 13)	false	
sum_within_range([0], 0, 3))	false	

D. Example Solution

```
def sum_within_range(nums, low, high):
    total = 0
    for num in nums:
        total += num

    return total > low and total < high
```

E. Notes

- What's being assessed: for loops, boolean operators, cumulative sum
- What's challenging: comparing a number against a range with specific inclusive/exclusive boundaries
- Source/Inspired by: CodingBat sorta_sum
- Written by: Paul Pham

10 unique_nums_in_range

A. Prompt

Write a function called get_nums_in_range that accepts a list nums, a lower bound, and an upper bound 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.

B. starter code

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

C. Example inputs and outputs

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

D. Example Solution

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

    return result
```

E. Notes

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

11 santa_on_time

A. Prompt

Santa Claus is a punctual person, but sometimes the odds can be stacked against him. He can deliver all of his presents when his speed is at least 70 mph (i.e. 70 or greater). If it’s snowing, he must lower his speed by 10 in order to get around the world safely. Write a function that takes in Santa’s initial speed and return True if he is on time, or False otherwise.

B. starter code

def santa_on_time(speed, is_snowing)

C. Example inputs and outputs

input	output	notes
santa_on_time(80, True)	True	
santa_on_time(79, True)	False	
santa_on_time(75, True	False	
santa_on_time(65, False)	True	

D. Example Solution

```
def santa_on_time(speed, is_snowing):
    if is_snowing:
        return speed - 10 >= 70
    else:
        return speed >= 70
```

E. Notes

- What's being assessed: boolean operators, conditions
- What's challenging: accounting for exclusive bound, separating conditions based on is_snowing parameter
- Source/Inspired by: CodingBat cigar_party
- Written by: Paul Pham

12 can_buy_car

A. 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.

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 700 (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

B. starter code

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

C. Example inputs and outputs

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

D. Example Solution

```
def can_buy_car(is_sports_car, credit_score, cash):
    cost = 100 if is_sports_car else cost = 70

    # conditional structure without ternary operator
    cost = 0
    if is_sports_car:
        cost = 100
    else:
        cost = 70

    if credit_score <= 700:
        cost += cost * 0.1

    return cash >= cost
```

E. Notes

- What's being assessed: conditionals, operators
- What's challenging: multiple branches

- Source/Inspired by: previous problem
- Written by: Paul Pham

13 enough_candy

A. Prompt

Everyone loves candy, but you've only got so much to hand out for Halloween! Kids and teens come to your door asking for candy, so of course you'll give it to them, but teenagers need more candy get through high school. Write a function called enough_candy that will return True if you had enough candy to last a night of trick-or-treaters (kids and teens who come to your door asking for candy), or False otherwise.

This function has the following rules:

- total_candy is an integer that represents the number of pieces of candy that you have
- treaters is a list of booleans that where each value represents someone who comes to your door; true values represent teens and false values represent kids
- teens ask for 3 pieces of candy
- kids ask for 1 piece of candy
- you have enough candy if you have more than 0 (inclusive) pieces of candy after helping everyone

B. starter code

```
def enough_candy(total_candy, is_teen):
```

C. Example inputs and outputs

input	output	notes
enough_candy(10, [True, True, False, True])	True	
enough_candy(3, [False, True])	False	
enough_candy(6, [True, False, False, False])	True	

D. Example Solution

```
def enough_candy(total_candy, treaters):

    for treater in treaters:

        if treater:

            total_candy -= 3

        else:

            total_candy -= 1

    return total_candy >= 0
```

E. Notes

- What's being assessed: conditionals, operators, translating story problem into code
- What's challenging: multiple branches, traversing the data structure correctly
- Source/Inspired by: previous problem
- Written by: Paul Pham

14 wind_chill

A. Prompt

Write a function called wind_chill(temperature, velocity) that will return the wind chill value when you input the integers temperature (in Fahrenheit) and velocity (wind speed, mph). With this function, use loops to print a table of wind chill values where the temperature will start from -20 to 60 degrees, and wind speeds of from 5 to 60.

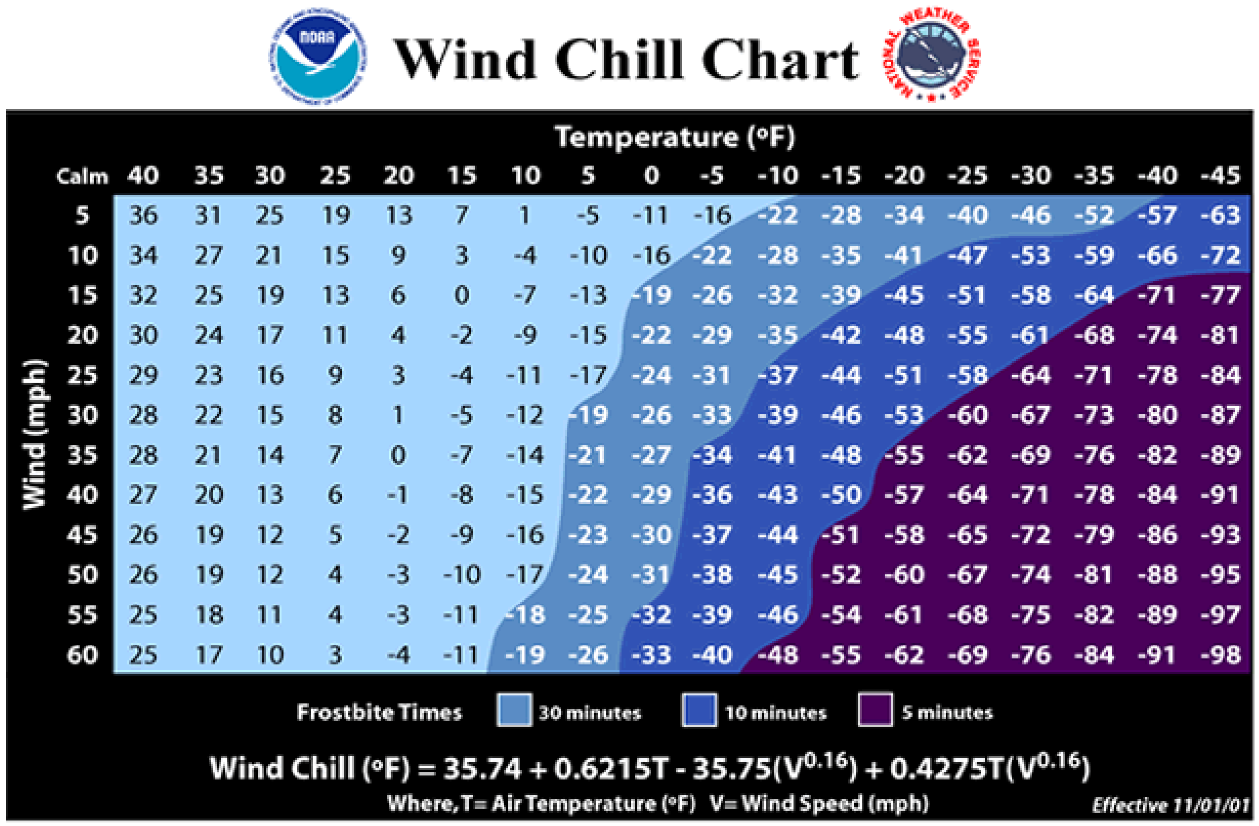
- In the table, the degrees should increase by 10 from -20 to 70.
- The wind speeds should increase by 5 from 5 to 60.
- Wind chill is given by the following formula:
    - $35.74 + 0.6215 \text{ x } T - 35.75 \text{ x } V_{0.16} + 0.4275 \text{ x } T \text{ x } V_{0.16}$

B. starter code

```
Def wind_chill(temperature, velocity):
```

C. Example inputs and outputs

input	output	notes
	(A table of values from -20 to 60 degrees and from 5 mph to 60 mph.)	
	Values should look similar to below:	



Wind Chill Chart

Wind (mph)	Temperature (°F)																	
Calm	40	35	30	25	20	15	10	5	0	-5	-10	-15	-20	-25	-30	-35	-40	-45
5	36	31	25	19	13	7	1	-5	-11	-16	-22	-28	-34	-40	-46	-52	-57	-63
10	34	27	21	15	9	3	-4	-10	-16	-22	-28	-35	-41	-47	-53	-59	-66	-72
15	32	25	19	13	6	0	-7	-13	-19	-26	-32	-39	-45	-51	-58	-64	-71	-77
20	30	24	17	11	4	-2	-9	-15	-22	-29	-35	-42	-48	-55	-61	-68	-74	-81
25	29	23	16	9	3	-4	-11	-17	-24	-31	-37	-44	-51	-58	-64	-71	-78	-84
30	28	22	15	8	1	-5	-12	-19	-26	-33	-39	-46	-53	-60	-67	-73	-80	-87
35	28	21	14	7	0	-7	-14	-21	-27	-34	-41	-48	-55	-62	-69	-76	-82	-89
40	27	20	13	6	-1	-8	-15	-22	-29	-36	-43	-50	-57	-64	-71	-78	-84	-91
45	26	19	12	5	-2	-9	-16	-23	-30	-37	-44	-51	-58	-65	-72	-79	-86	-93
50	26	19	12	4	-3	-10	-17	-24	-31	-38	-45	-52	-60	-67	-74	-81	-88	-95
55	25	18	11	4	-3	-11	-18	-25	-32	-39	-46	-54	-61	-68	-75	-82	-89	-97
60	25	17	10	3	-4	-11	-19	-26	-33	-40	-48	-55	-62	-69	-76	-84	-91	-98

Frostbite Times ▨ 30 minutes ▨ 10 minutes ▨ 5 minutes

Wind Chill (°F) = $35.74 + 0.6215T - 35.75(V^{0.16}) + 0.4275T(V^{0.16})$

Where, T= Air Temperature (°F) V= Wind Speed (mph) *Effective 11/01/01*

D. Example Solution

```
def wind_chill(temperature, velocity):

        Return round(35.74 + 0.6215 * T - 35.75 * V**(0.16) + 0.4275 * T * V**(0.16))

tempList = []

windList = []
```

```
For T in range (-20, 70, 10):

        tempList.append(T)

For V in range(5, 55, 5):

        windList.append(V)


For i in range(len(tempList)):

        print(tempList[i])

print('/n')


# this actually creates the table

For j in range(len(windList)):

        print(windList[j], end=' ')

        For k in range(len(tempList)):

                print(round(wind_chill(tempList[k], windList[j], end=' ')
```

E. Notes

- What's being assessed: for loops, calling function over a loop, lists
- What's challenging: nested for loops
- Source/Inspired by: previous problem in SCC CSE 110
- Written by: Jared Lim

15 contains_name note: flip this to make it easier to test

A. Prompt

Write a list of first names and last names. Then write a function called contains_name. With the function, if the name contains the name given in the function, have it say “me” instead of the person’s full name.

B. starter code

```
Def contains_name(names):
```

C. Example inputs and outputs

input	output	notes
contains_name([“John Doe”,”Thomas James”])	[“Me”,”Thomas James”]	

D. Example Solution

```
namesList = [“John Doe”, “Thomas James”, “Elizabeth Walker”]

contains_name(string_we_want, names):

    For i in len(names):

        If names[i] == string_we_want

            Names[i] = “me”

    Return names

contains_name(namesList)
```

E. Notes

- What's being assessed: for loops, calling function over a loop, lists
- What's challenging: String conditionals, returning the right thing
- Source/Inspired by: previous problem in CSE 110 (similar to wind chill)
- Written by: Jared Lim

16 count_reverse

A. Prompt

Create a function that takes an integer, and prints out numbers starting from the integer to 0. Assume the number can also be negative.

B. starter code

```
Def count_reverse(number):
```

C. Example inputs and outputs

input	output	notes
count_reverse(7)	7,6,5,4,3,2,1,0	

D. Example Solution

```
Def count_reverse(num):

        Seq = -1
```

```
    If num < 0:
        Seq = 1
    For x in range(num, seq, seq):
        print(x)
```

E. Notes

- What's being assessed: for loops
- What's challenging: creating the range of x
- Source/Inspired by: previous problem in CSE 110
- Written by: Jared Lim
- You could also accomplish this with a while loop

17 reverse_word

A. Prompt

Create a function that reverses the given word. It takes a word (String), and returns the word but written backwards.

B. starter code

```
Def reverse_word(word):
```

C. Example inputs and outputs

input	output	notes
reverse_word("Yes")	Sey	

D. Example Solution

```
Def reverse_word(word):

        Reverse = “ “

        For i in range(len(word) -1, -1, -1):

                Reverse += word[i]

        Word[::-1] # short hand python solution

        print(reverse)
```

E. Notes

- What's being assessed: String concat
- What's challenging: concatenating string
- Source/Inspired by: previous problems in CSE 110
- Written by: Jared Lim
- You could also accomplish with just one line: print(word[::-1]), although it may be less intuitive than looping over the word.

18 password checking

A. Prompt

Write a function `check_passcode()` that determines if the variable `inp`, which has a 4 digit integer value, is a valid passcode. `input` is a valid passcode if the sum of the first 3 digits modulus `7` is equal to the last digit. If `input` is valid, the code should print `valid`. If the string is not valid, it should print `NOT valid`.

So if `inp` were set to `5312`, it would be a valid passcode and your code would return `True` because the first 3 digits (`5`, `3`, and `1`) sum to `9` and `9` modulus `7` equals the last digit (`2`). `1234` would not be a valid passcode and your code would return `False`.

Assume a variable `input` is always a 4 digit integer value.

B. starter code

```
check_passcode(input):
```

C. Example inputs and outputs

input	output	notes
check_passcode(5312)	True	
check_passcode(1234)	False	
check_passcode(1236)	True	
check_passcode(9873)	True	
check_passcode(9876)	False	

D. Example Solution

E. Notes

- What's being assessed: variable storage, conditional statement, modulo operator
- What's challenging: processing each digit, modulo operator
- Source/Inspired by: CSE 2019 study post-test
- Written by: Benji
- Probably need to write
- Rubric (out of 9 pts)
  - 2 pts for storing digits (-0.5 for any incorrectly stored digit)
  - 2 pt for using mod to access each digit
  - 2 pt for conditional statement
  - 1 pt for correct print
  - 2 pt for syntax (-1 for minor mistakes, -2 for major)

19 is_profitable

A. Prompt

Write a function called is_profitable that takes in two integers as parameters: revenue and expenses. Return True if your profit, defined as the difference between revenue and expenses, is greater than 0. Otherwise, return False.

B. starter code

Def is_profitable(revenue, expenses):

C. Example inputs and outputs

input	output	notes
is_profitable(2400, 1300)	True	
is_profitable(1350, 1350)	False	0 case, breaks even.
is_profitable(905, 1500)	False	

D. Example Solution

```
def is_profitable(revenue, expenses):
        return revenue > expenses
```

E. Notes

- What's being assessed: booleans
- What's challenging: translating story elements into the solution
- Source/Inspired by: idk
- Written by: Paul Pham

20 sum_target

A. 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,

You may assume that all numbers passed in are positive integers. You may NOT return

B. starter code

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

C. Example inputs and outputs

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

D. Example Solution

```
def sum_target(n, m, target):
        return (n % 10) + (m % 10) == target
```

E. Notes

- 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: Another integer manipulation problem
- Written by: Paul Pham

21 last_digit

A. Prompt

Write a function called last digit that takes in an integer n and returns the last digit in the integer. This function should also treat negative numbers the same as positive numbers. So the calls last_digit(3) and last_digit(-3) should both return 3.

For example, last_digit(1238476) would return 6 since that is the last digit in n.

B. starter code

```
def last_digit(n):
```

C. Example inputs and outputs

input	output	notes
last_digit(1238476)	6	

last_digit(3)	3	
last_digit(-3)	3	Negative special case

D. Example Solution

```
def last_digit(n):
        return abs(n) % 10
```

E. Notes

- What's being assessed: integer manipulation
- What's challenging: remembering that you can pull off the last digit with % operator
- Source/Inspired by: trying to think of an ‘easy’ int manipulation problem without loops
- Written by: Paul Pham

22 digit_sum_match

A. Prompt

Write a function called digit_sum_match that takes two arguments, an integer n and an integer target, that returns True if the sum of the digits in n is the same as target, otherwise returning False.

You are NOT allowed to cast the number into a string to solve this problem. You can assume that the integer n will always be positive.

B. starter code

```
def digit_sum_match(n, target):
```

C. Example inputs and outputs

input	output	notes
digit_sum_match(123, 6)	True	
digit_sum_match(123, 7)	False	

D. Example Solution

```
def digit_sum_match(n, target):
```

```
total = 0
while n > 0:
        digit = n % 10
        total += digit
        n = n // 10
return total == target
```

E. Notes

- What's being assessed: integer manipulation, while loops
- What's challenging: pulling off the last digit of an integer, using the while loop bound to handle that case
- Source/Inspired by: From CSE 163 Week 2 section handout
- Written by: Paul Pham

23 factorial

A. 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.

B. starter code

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

C. Example inputs and outputs

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

D. Example Solution

```
def factorial(n):
    product = 0
    for i in range(target + 1):
        product *= i
    return i
```

E. Notes

- 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, listed above
- Written by: Paul Pham

24 has_word

A. Prompt

Write a function called has_word that takes in two strings as parameters. Return True if the first string parameter contains any instance of the second string parameter.

For example, the call has_word(‘This is a sentence’, ‘sentence’) would return True since ‘sentence’ is a substring of ‘This is a sentence’.

An example that returns False is the call has_word(‘asdfghjkl’, ‘hello’) because ‘hello’ is not in the string ‘asdfghjkl’.

B. starter code

```
def has_word(s, word):
```

C. Example inputs and outputs

input	output	notes
has_word(‘this is a sentence’, ‘sentence’)	True	
has_word(‘asdfghjkl’, ‘hello’)	False	
has_word(‘Thank you!’, ‘k you’)	True	

D. Example Solution

```
def has_word(s, word):
        return word in s
```

E. Notes

- What's being assessed: use of the `in` keyword
- What's challenging: remembering that you can use `in` as a `contains` for string
- Source/Inspired by: trying to think of an ‘easy’ string problem
- Written by: Paul Pham

Other items to incorporate

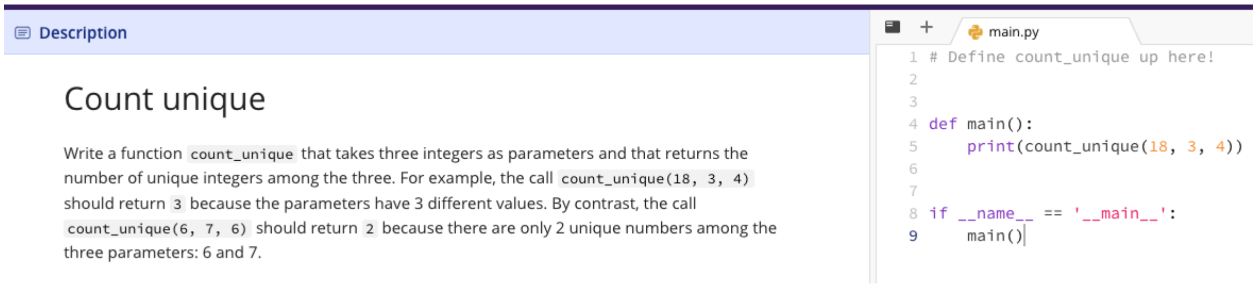
- Practice items from Codeitz study
- Codeitz post-test

Example Items

Coding Bat: Python

CSE 163

Control structures



Description

Count unique

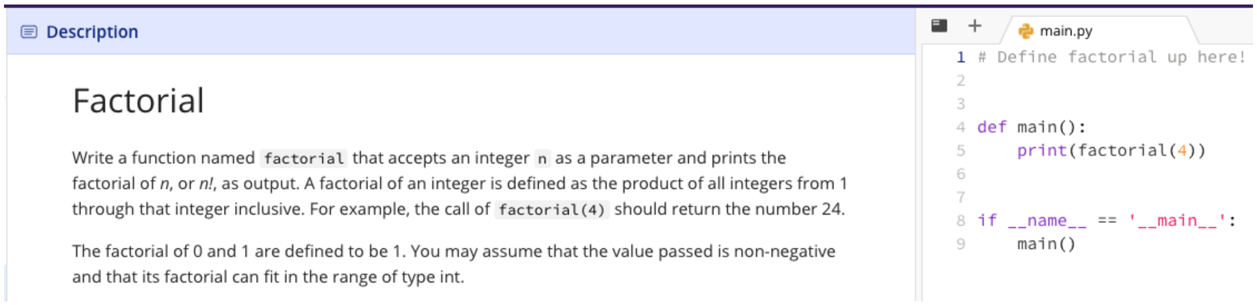
Write a function `count_unique` that takes three integers as parameters and that returns the number of unique integers among the three. For example, the call `count_unique(18, 3, 4)` should return `3` because the parameters have 3 different values. By contrast, the call `count_unique(6, 7, 6)` should return `2` because there are only 2 unique numbers among the three parameters: 6 and 7.

main.py

```python
# Define count_unique up here!


def main():
    print(count_unique(18, 3, 4))


if __name__ == '__main__':
    main()
```



Description

Factorial

Write a function named `factorial` that accepts an integer `n` as a parameter and prints the factorial of *n*, or *n!*, as output. A factorial of an integer is defined as the product of all integers from 1 through that integer inclusive. For example, the call of `factorial(4)` should return the number 24.

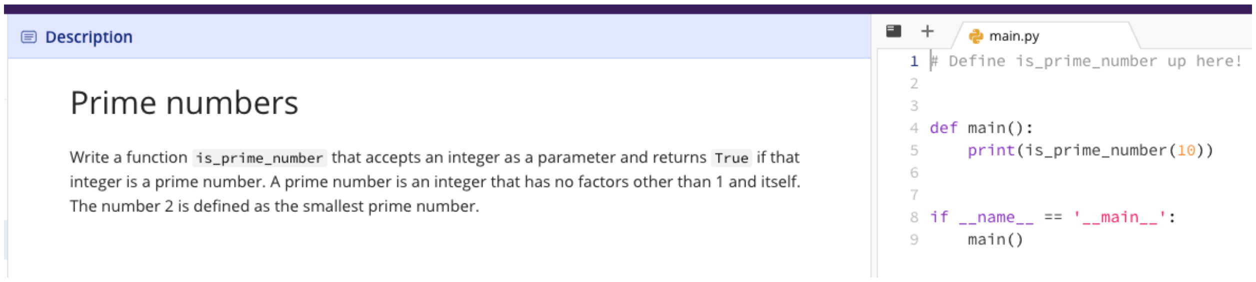
The factorial of 0 and 1 are defined to be 1. You may assume that the value passed is non-negative and that its factorial can fit in the range of type int.

main.py

```python
# Define factorial up here!


def main():
    print(factorial(4))


if __name__ == '__main__':
    main()
```



Description

Prime numbers

Write a function `is_prime_number` that accepts an integer as a parameter and returns `True` if that integer is a prime number. A prime number is an integer that has no factors other than 1 and itself. The number 2 is defined as the smallest prime number.

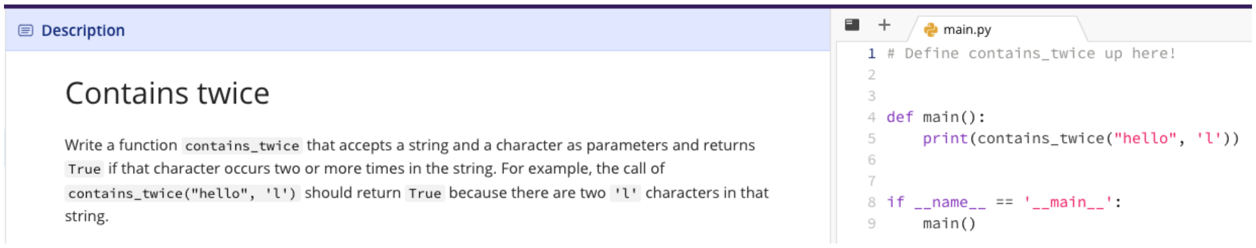
main.py

```python
# Define is_prime_number up here!


def main():
    print(is_prime_number(10))


if __name__ == '__main__':
    main()
```

Strings & Lists



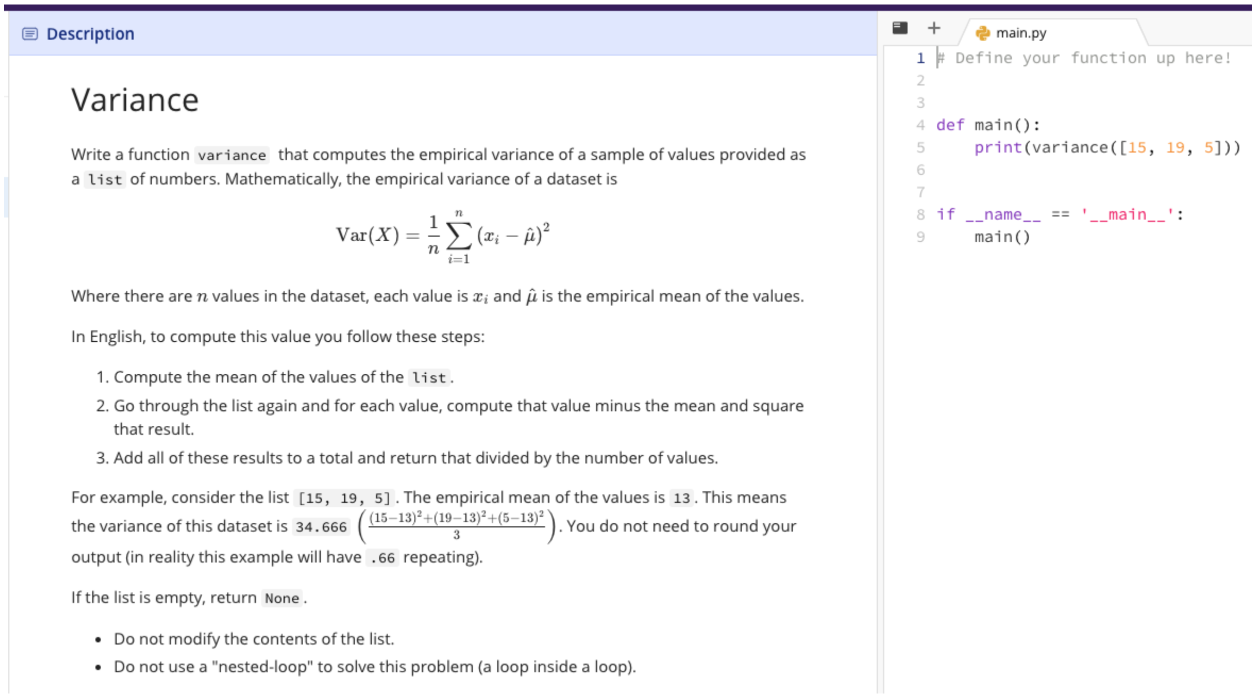
Contains twice

Write a function `contains_twice` that accepts a string and a character as parameters and returns `True` if that character occurs two or more times in the string. For example, the call of `contains_twice("hello", 'l')` should return `True` because there are two `'l'` characters in that string.

```python
# Define contains_twice up here!


def main():
    print(contains_twice("hello", 'l'))


if __name__ == '__main__':
    main()
```



Variance

Write a function `variance` that computes the empirical variance of a sample of values provided as a `list` of numbers. Mathematically, the empirical variance of a dataset is

$$\mathrm{Var}(X) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{\mu})^2$$

Where there are n values in the dataset, each value is x_i and $\hat{\mu}$ is the empirical mean of the values.

In English, to compute this value you follow these steps:

1. Compute the mean of the values of the `list`.
2. Go through the list again and for each value, compute that value minus the mean and square that result.
3. Add all of these results to a total and return that divided by the number of values.

For example, consider the list `[15, 19, 5]`. The empirical mean of the values is `13`. This means the variance of this dataset is `34.666` $\left(\frac{(15-13)^2+(19-13)^2+(5-13)^2}{3}\right)$. You do not need to round your output (in reality this example will have `.66` repeating).

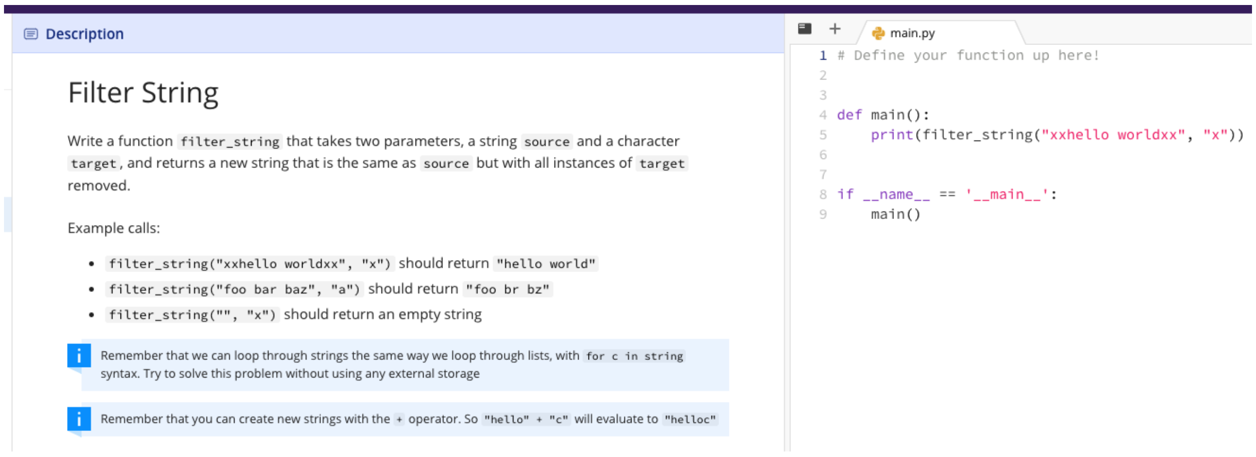
If the list is empty, return `None`.

- Do not modify the contents of the list.
- Do not use a "nested-loop" to solve this problem (a loop inside a loop).

```python
# Define your function up here!


def main():
    print(variance([15, 19, 5]))


if __name__ == '__main__':
    main()
```



Filter String

Write a function `filter_string` that takes two parameters, a string `source` and a character `target`, and returns a new string that is the same as `source` but with all instances of `target` removed.

Example calls:

- `filter_string("xxhello worldxx", "x")` should return `"hello world"`
- `filter_string("foo bar baz", "a")` should return `"foo br bz"`
- `filter_string("", "x")` should return an empty string

Remember that we can loop through strings the same way we loop through lists, with `for c in string` syntax. Try to solve this problem without using any external storage

Remember that you can create new strings with the `+` operator. So `"hello" + "c"` will evaluate to `"helloc"`

```python
# Define your function up here!


def main():
    print(filter_string("xxhello worldxx", "x"))


if __name__ == '__main__':
    main()
```