PA3

PA 3

CSE 8A Fall 2021 PA3

Due date: Friday 10/22/2021 @ 11:59PM PDT

(No Late Submission is Allowed.)

Provided Files

None

File(s) to Submit

- `dining_hall_menus.py`
- `count_exact_match.py`
- `validate_email.py`
- `compute_string_distance_ec.py` **(Optional Extra Credit)**

Part 1: Implementation

`dining_hall_menus.py`

Create a file called `dining_hall_menus.py`, where you will implement the function:

```
def dining_hall_menus(ovt_menu, pines_menu)
```

The function will take in 2 lists, `ovt_menu` and `pines_menu` and returns the difference between the two menus offered that day. We define the "difference" between two menus as a list of unique items that one menu has but the other menu doesn't. For example, the function call `dining_hall_menus(["pizza", "spaghetti", "salad", "fries"], ["burgers", "sushi", "fries", "pizza"])` would return `["spaghetti", "salad", "burgers", "sushi"]`.

Note the following:

- Items in the menus must match the exact capitalization to be considered as the same. For

example, the function call `dining_hall_menus(["pizza"], ["Pizza"])` would return `["pizza", "Pizza"]`.

- `ovt_menu` and `pines_menu` may have duplicates within the list, but will be treated as one item. For example, the function call `dining_hall_menus(["pizza", "salad", "salad"], ["pizza"])` would return the list `["salad"]`.
- Empty lists are valid arguments, passing in a single empty list and a non-empty list should return a list of the unique items in the non-empty list. For example, the function call `dining_hall_menus([], ["fries", "fries"])` would return `["fries"]`. Passing in two empty lists would return an empty list.
- Order of the items in the returned list does not matter.

`count_exact_match.py`

Create a file called `count_exact_match.py`. In this file, you would need to implement the following function:

`def count_exact_match(text, query)`

This function takes in two strings `text` and `query`, and returns the number of exact matches between them. An exact match is the case where the same character appears at exactly the same index of both strings. For example, `count_exact_match("severussnape", "alanrickman")` returns 2, because there are two exact matches -- in both strings, character 'r' appears at index 4, and character 'a' appears at index 9.

Index	0	1	2	3	4	5	6	7	8	9	10	11
Text String	s	e	v	e	r	u	s	s	n	a	p	e
Query String	a	l	a	n	r	i	c	k	m	a	n	

Note the following:

- `text` and `query` may have different lengths
- In the case where a string is an empty string `""` , the returning value should be 0
- The function is case-sensitive, so 'r' and 'R' are not considered as the same character

`validate_email.py`

Create a file called `validate_email.py`. In this file, you would need to implement the following function:

`def validate_email(email_address,email_suffix)`

This function takes in two strings `email_address` and `email_suffix` and returns `True` if the email address is valid and `False` otherwise. We define `email_address` as valid if the substring of what lies

right of `"@"` exactly matches `email_suffix`. You should check the validity of the email address with the following rules:

- `email_address` should contain exactly one `"@"`. For example, for `email_suffix = "gmail.com"`, `email_address = "hello@world@gmail.com"` is not valid, but `email_address="helloworld@gmail.com"` is.
- Other than leading and trailing spaces, `email_address` should not contain any space. For example, for `email_suffix = "gmail.com"`, `email_address = "  hello world@gmail.com  "` is not valid because of the spaces between "hello" and "world" but `email_address = "  helloworld@gmail.com  "` is valid.
- The substring between `"@"` (not including `"@"`) and the first trailing space (not including the space) or the end of the string if there is no trailing space should match the `email_suffix`. For example, for `email_address = "  helloworld@gmail.com  "`, it is valid if `email_suffix = "gmail.com"`, but not valid if `email_suffix = "ucsd.edu"`.
- `email_suffix` can be any string, which means it may contains leading/trailing spaces itself. For example, `email_address = "  hello@ .  "` and `email_suffix = " ."` should return `False` because of the leading space in the suffix.

Part 3: Style

Coding style is an important part of ensuring readability and maintainability of your code. Starting from this assignment on, we will grade your coding style in all submitted code files (**including the optional extra credit file**) according to the style guidelines. You can keep these guidelines in mind as you write your code or go back and fix your code at the end. For now, we will mainly be grading the following:

- **Use of descriptive variable names**: The names of your variables should describe the data they hold. Your variable names should be words (or abbreviations), not single letters.
  - e.g. `a` --> `index_of_apple`; `letter1` and `letter2` --> `lower_case_letter` and `upper_case_letter`
  - Exception: If it is a loop index like `i`, `j`, `k`, one char is OK and sometimes preferred.
- **Inline Comments**: If there is a length of code that is left unexplained, take the time to type a non-redundant line summarizing this length of code (e.g. `#initialize an int` is redundant, vs. `#set initial length to 10 inches` is not). It will let others who look at your code understand what's going on without having to spend time understanding your logic first. But don't be too descriptive, as too many comments reduce readability.

Part 4: Conceptual Questions

You are required to complete the Conceptual Questions in PA lesson. There is no time limit but you must submit by the PA deadline. You can submit multiple times before the deadline. Your latest

submission will be graded.

Submission

Turning in your code

Submit all of the following files to PA3 Lesson on EdStem via the “Mark" Button by **Friday, October 22nd @ 11:59PM PDT (No Late Submission is Allowed)**:

- `dining_hall_menus.py`
- `count_exact_match.py`
- `validate_email.py`
- `compute_string_distance_ec.py` **(Optional Extra Credit)**

Evaluation

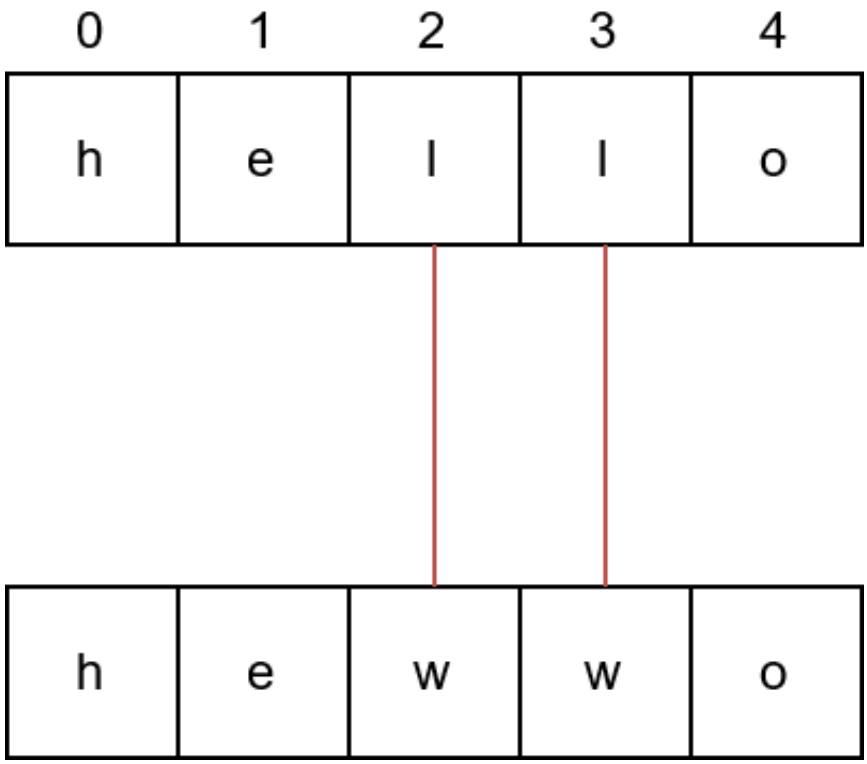
- **Correctness** (**80 points + 5 EC points**) You will earn points based on the autograder tests that your code passes. If the autograder tests are not able to run (e.g., your code does not compile or it does not match the specifications in this writeup), you may not earn credit. Your latest submission will be graded.
- **Style** (5 points)
- **Conceptual Questions** (15 points)

PA 3 EC

In the PA 3 EC slide, create a file called `compute_string_distance_ec.py`. In this file, you would need to implement the following two functions:

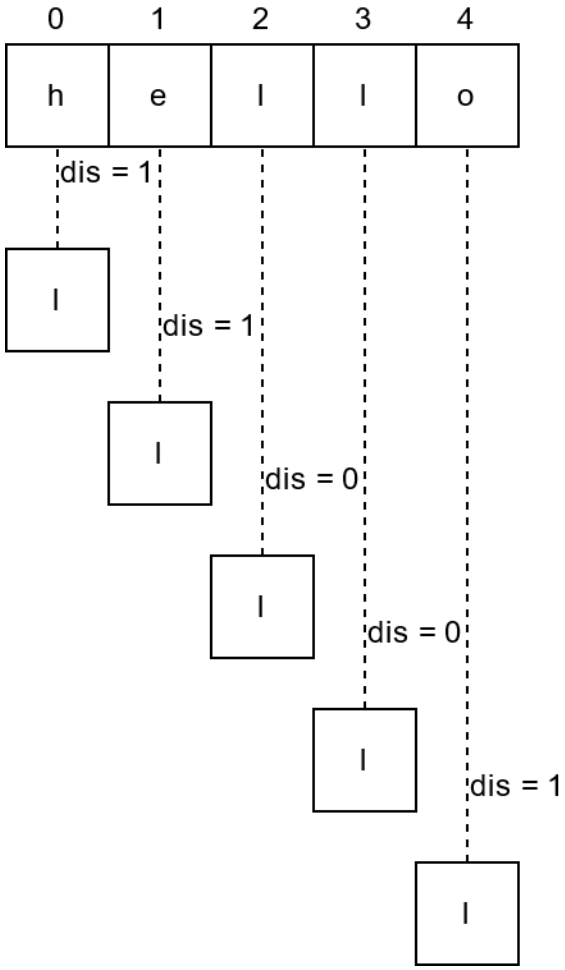
- `def compute_string_distance(text,query)` This function takes in two strings of equal length and returns the distance between the two strings. Distance here is defined as the number of characters at the corresponding indices that are different (known as the hamming distance). You may assume the input strings have the same length.

In the example below, the string "Hello" and "Hewwo" has a distance of 2 because the characters at position 2 and 3 are different as indicated by the red line, so the function would return 2 for the function call `compute_string_distance("hello","hewwo")`.



- `def min_distance_substring(text,query)` This function takes in two strings `text` and `query` where `len(text) >= len(query)` and returns the starting position of the first substring of `text` with same length as `query` that has the smallest distance between the substring and the `query`.
    - **Note**: the `compute_string_distance` function you just defined might be useful and you may call the function here.

Consider the example below with `text = "hello"` and `query = "l"`. There are 5 substrings with length 1 in the text (`"h","e","l","l","o"`). When comparing the distance between the substrings and the `query`, there are two string substrings with distance 0 at index 2 and 3. In this case your function should return 2 as that's the first occurrence of the substring with the smallest distance.


0
1
2
3
4
h
e
l
l
o
dis = 1
l
dis = 1
l
dis = 0
l
dis = 0
l
dis = 1
l

PA 3 Conceptual Questions

Question 1 *Submitted Oct 13th 2021 at 11:59:09 am*

Slicing and indexing into a string are basically the same actions with literally no difference

- ○ True
- ◉ False

Question 2 *Submitted Oct 20th 2021 at 4:50:25 pm*

What is the correct order to tackle a programming problem based on our lecture on Monday of week 3?

Design test cases on paper

Design an algorithm on paper and test it using existing test cases

translate the algorithm on paper to a piece of code

Test and debug the code

Question 3 *Submitted Oct 20th 2021 at 4:48:32 pm*

Which of the functions will remove the leading and trailing spaces of a string?

- ○ count
- ○ index
- ◉ strip
- ○ split

Question 4 *Submitted Oct 20th 2021 at 4:50:07 pm*

Given the following list, select the correct way(s) to slice the list so result will have the value of [1, 3, 5, 6].

```
num_list = [1, 3, 5, 6, 9, -1, 2, 10]
result = _____________
```

- [x] `num_list[:4]`
- [ ] `num_list[4:]`
- [x] `num_list[0:4:1]`
- [ ] `num_list[1:5]`
- [x] `num_list[0:4]`
- [ ] `num_list[1:4]`
- [x] `num_list[:4:1]`
- [ ] `num_list[1:5]`

Question 5 *Submitted Oct 20th 2021 at 4:50:50 pm*

To change some elements of a list in a loop, we normally use indexing into the list.

- (•) True
- ( ) False

Question 6 *Submitted Oct 20th 2021 at 4:54:24 pm*

Which of the following ranges produce the list [1, 2, 3, 4, 5]?

```
1. list(range(0, 5, 1))
2. list(range(5))
3. list(range(1, 6))
4. list(range(6))
```

- (●) 1 only
- ( ) 3 only
- ( ) 1 and 2 only
- ( ) 3 and 4 only