CMPT 103 – Lab #1

General Information

Python Version and IDE:	Python 3 / WingIDE 101
Allocated lab time:	2 hrs and 50 min
Due date:	At the end of lab period
Lab weight:	3%

Topics

- ✓ Functions and Lists

Submission

- ✓ **All the code files (.py) should be submitted electronically** to your Lab Blackboard site.
- ✓ A portion of the total marks (20%) will be allocated for the programming style. For example, functions should be small; names should be meaningful and descriptive; naming convention should be followed consistently; code should be formatted properly; and comments should be accurate and well written.
- ✓ Comments are **required** for:
  - EACH program indicating the student name and program name.
  - EACH function indicating the function purpose, syntax (example usage of the function), parameters, and return value.
  - Any block of code for which the purpose may be unclear (Note: you should always try to write clean code that can be understood easily without comments).

Assignment

For this lab, please put all functions into a file called `Lab1your_initials.py` (e.g., `Lab1FL.py` where F and L are the first letter of your first name and last name).

1) [35 marks] Write a function named **`print_right_aligned`** that *takes* a list of strings and *prints* the strings in a single column right-justified. The width of the column must be equal to the length of the longest string in the list. To help you implement this function, write a helper function named **`max_length`** that *takes* a list of strings and *returns* the length of the longest string in the list. Use **`max_length`** to implement **`print_right_aligned`**. (This is an example of how we can break down the complexity of our code by writing small functions.)

   Some example of test runs are shown below.

```
>>> length = max_length(['abc', 'de', 'fghijklmn'])
>>> print(length)
9
>>>
>>> print_right_aligned(['abc', 'de', 'fghijklmn'])
      abc
       de
fghijklmn
>>>
>>> print_right_aligned(['First Name', 'Surname'])
First Name
   Surname
```

2) [35 marks] Write a function named **`percent_odd`** that *takes* a list of integers and *returns* the percentage of odd numbers in the list. If given an empty list, this function should return 0. Write a helper function named **`count_odd`** that *takes* a list of integers and *returns* the number of odd numbers in the list. Use **`count_odd`** to implement **`percent_odd`**.

Some example of test runs are shown below.

```
>>> total_odd = count_odd([1, 2, 3, 4, 5])
>>> print(total_odd)
3
>>>
>>> percent = percent_odd([1, 2, 3, 4, 5])
>>> print(percent)
60.0
>>>
>>> percent = percent_odd([])
>>> print(percent)
0.0
```

3) [30 marks] Write a function named **`alternate`** that *takes* two lists and *returns* a new list containing alternating elements of the given lists. For example, a call to `alternate([1, 2], ['a', 'b'])` should return `[1, 'a', 2, 'b']`. If the given lists have different lengths, simply append the remaining elements of the longer list to the new list (the result). Please see the examples below.

```
>>> result = alternate([1, 2], ['a', 'b'])
>>> print(result)
[1, 'a', 2, 'b']
>>>
>>> result = alternate([1, 2, 3, 4], ['a', 'b'])
>>> print(result)
[1, 'a', 2, 'b', 3, 4]
>>>
>>> result = alternate([1, 2], ['a', 'b', 'c', 'd'])
>>> print(result)
[1, 'a', 2, 'b', 'c', 'd']
>>>
>>> result = alternate([], ['a', 'b', 'c', 'd'])
>>> print(result)
['a', 'b', 'c', 'd']
```