FM211: Data Science and Artificial Intelligence: Week 02 Lab Assignments

Subhasis Ray*

NOTE: For printing variable within a string, Python 3 introduced something called f-string. This takes the form `f'Literal string {variable} literal {variable} ...'`. In this the `variable` will be substituted by its value. You can also place function calls within braces (`{}`) and they will be replaced by the return value of the function. Search online to find out more, as this is a very convenient feature of Python.

Assignments for Lab, week 2

1. Write a function called `count_letters` that takes a string as a parameter and returns the number of English letters (not punctuation, space, or digit) in it. Save this in a file called `count_letters.py` and upload to codePost page of this course.

   An example test code [in colored syntax highlighting] and its output [black, typewriter font]:

   ```
   mystr = 'Ne rien de rien!'
   print('Letters in this string:', count_letters(mystr))

   Letters in this string: 12
   ```

2. Write a function `split_names` that takes a list of names, and returns two lists, one with first names and the other with the corresponding last names. If a name has a single component assume it to be the first name, and put `''` (the empty string, written as two single quotes with no space in between) for last name. *BONUS POINT: If your function can handle middle name, names with more than 3 components, and incorrect entries like the empty string.* Upload as `split_names.py`. Example usage:

*subhasis.ray@plaksha.edu.in

```
names = ['Betrtrand Russel', 'Kurt Godel', 'Peter Norvig',
         'Srinivas Ramanujan', 'Kanad', 'Demosthenes']
first, last = split_names(names)
print('First names', first)
print('Last names', last)
```

```
First names ['Betrtrand', 'Kurt', 'Peter', 'Srinivas', 'Kanad', 'Demosthenes']
Last names ['Russel', 'Godel', 'Norvig', 'Ramanujan', '', '']
```

3. Similar to above, write a function `find_lastname` which takes a string `firstname` and a list of names `names`, and return the last name of the first entry in `names` whose first name is `firstname`. Think of the cases where name has only one component, and where first name does not appear in the names in the list. To distinguish the two scenarios, your function can return `None` in the latter case. Upload as `find_lastname.py`. Example usage:

```
names = ['Betrtrand Russel', 'Kurt Godel', 'Peter Norvig',
         'Srinivas Ramanujan', 'Kanad', 'Demosthenes']
first = 'Srinivas'
print(f'Lastname of {first}: "{find_lastname(first, names)}"')
first = 'Kanad'
print(f'Last name of {first}: "{find_lastname(first, names)}"')
first = 'Rabi'
print(f'Last name of {first}: "{find_lastname(first, names)}"')
```

```
Lastname of Srinivas: "Ramanujan"
Last name of Kanad: ""
Last name of Rabi: "None"
```

4. Write a function `summary_stats` that takes a `list` of numbers and returns a 3-tuple `(mean, median, sd)` where `mean` is the average of the numbers in the list, `median` is the median value, and `sd` is the standard deviation. Upload with filename `summary_stats.py`.

   Example usage:

```
data = [1, 1, 3, 2, 3, 5, 7]
print('My results:', summary_stats(data))
print('* numpy results:', np.mean(data), np.median(data), np.std(data))
print('=== Another list:')
```

```
data = [-1, -1, -1, -1, 1, 1, 3, 2, 3, 5, 7]
print('My results:', summary_stats(data))
print('* numpy results:', np.mean(data), np.median(data), np.std(data))
```

```
My results: (3.142857142857143, 3, 2.0303814862216996)
* numpy results: 3.142857142857143 3.0 2.0303814862216996
=== Another list:
My results: (1.6363636363636365, 1, 2.568081253059188)
* numpy results: 1.6363636363636365 1.0 2.568081253059188
```

5. The Indian poet-mathematician Pingala first described the numbers now known as Fibonacci series. It is defined as: $F(0) = 0$. $F(1) = 1$. $F(n) = F(n-1)+F(n-2)$. Write a function to compute the Fibonacci number $F(n)$ using a loop. Upload with filename `fibo_iterative.py`.

   Example usage:

```
for ii in range(6):
    print(f'F({ii}) = {fibo_iterative(ii)}')
```

```
F(0) = 0
F(1) = 1
F(2) = 1
F(3) = 2
F(4) = 3
F(5) = 5
```

6. Do you notice that F is defined in terms of itself? This is called a recursive definition. Can you exploit this to write another function which calls itself with `(n - 1)` and `(n - 2)` as parameters? Upload as `fibo_recursive.py`. Example usage:

```
for ii in range(6):
    print(f'F({ii}) = {fibo_recursive(ii)}')
```

```
F(0) = 0
F(1) = 1
F(2) = 1
F(3) = 2
F(4) = 3
F(5) = 5
```

7. (SV) Write a function `matrix_product` that takes two 2D matrices as arguments and returns the product. Upload your code as `matrix_product.py`.

   (a) Try 2x2 and 3x3 matrices for testing by hand.

   (b) Keep doubling the matrix size, as large as you can go and call it multiple times in a loop. Measure the average time (the function `perf_counter` in `time` module can be useful for this).

   (c) Do the same for the `matmul` function in `numpy` module and compare the speed.

   (d) Plot the time of execution against the matrix size.

   Example usage:

```
a = np.array([[1, 2], [3, 4]])
print(a)
print('squared')
print(matrix_product(a, a))
print('* numpy:\n', a @ a)
print('===')
b = np.array([[2, 0], [1, 3]])
print(a, '\nx\n', b)
print(matrix_product(a, b))
print('* numpy:\n', a @ b)
```

```
[[1 2]
 [3 4]]
squared
[[ 7. 10.]
 [15. 22.]]
* numpy:
 [[ 7 10]
 [15 22]]
===
[[1 2]
 [3 4]]
x
 [[2 0]
 [1 3]]
[[ 4.  6.]
```

```
 [10. 12.]]
* numpy:
 [[ 4  6]
 [10 12]]
```