Mini Project 1

Welcome to your first assignment! Look for "TODO" inside the code block and complete the required exercises.

Task 1: Python Basics Exercise

The "Learn the basics" and "Data science tutorial" sections from Online Resource (https://www.learnpython.org/en/Welcome) is a good material to learn Python. After going through the crash course and relevant materials, you should complete the following exercise of each section in this jupyter notebook.

1.1 Print function

Use the `print` function to print the following sentence "I'm a student at CU."

In [ ]:
```
# TODO: print out the line
```

1.2 Variables and Types

The target of this exercise is to create a string, an integer, and a floating point number.

- The string variable should be named `mystring` and initialized to "Python".
- The floating variable should be named `myfloat` and initialized to 5.0.
- The integer variable should be named `myint` and initialized to 10.

In [ ]:
```
# TODO: define your variables


# test code
if mystring == "Python":
    print("String: %s" % mystring)
if isinstance(myfloat, float) and myfloat == 5.0:
    print("Float: %f" % myfloat)
if isinstance(myint, int) and myint == 10:
    print("Integer: %d" % myint)
```

1.3 Lists

In this exercise, you will need to add numbers and strings to the correct lists using the "append" list method.

You must add the numbers 1, 2, and 3 to the "numbers" list and the words 'Hello' and 'World' to the strings variable.

In [ ]:
```
numbers = []
strings = []
names = ["John", "Eric", "Jessica"]

# TODO: append 1, 2, and 3 to numbers respectively


# TODO: append "Hello" and "World" to strings respectively


# TODO: modify the following line to get the second item of names
second_name = None


# test code
if all(i == j for i, j in zip(numbers, [1, 2, 3])):
    print(numbers)
if all(i == j for i, j in zip(strings, ['Hello', 'World'])):
    print(strings)
if second_name == 'Eric':
    print("The second name on the names list is %s" % second_name)
```

1.4 Basic Operators

Create two lists called `x_list` and `y_list`, which contain 10 instances of the variables `x` and `y`, respectively. You are also required to create a list called `big_list`, which contains the variables `x` and `y`, 10 times each, by concatenating the two lists you have created.

In [ ]:
```
x = object()
y = object()

# TODO: append 10 instances of x to x_list using the operator *
x_list = [x]

# TODO: append 10 instances of y to y_list using the operator *
y_list = [y]

# TODO: concatenate y_list into x_list using the operator +
big_list = []

print("x_list contains %d objects" % len(x_list))
print("y_list contains %d objects" % len(y_list))
print("big_list contains %d objects" % len(big_list))

# test code
if x_list.count(x) == 10 and y_list.count(y) == 10:
    if big_list.count(x) == 10 and big_list.count(y) == 10:
        print("Great!")
```

1.5 String Formatting

Write a format string which prints out the data using the following syntax: Hello John Doe. Your current balance is $53.

In [ ]:

```
data = ("John", "Doe", 53.00)
format_string = "Hello"

# TODO: print out the data with required syntax
```

1.6 Basic String Operations

Try to fix the code to print out the correct information by changing the string.

In [ ]:

```
s = "Hey there! what should this string be?"

# TODO: print out the length of s


# TODO: print out the index of the letter a


# TODO: print out how many times that the letter e occurs in the string


# TODO: print out the first five characters of s


# TODO: print out the last five characters of s


# TODO: print out the characters with even index


# TODO: reverse s and print it out


# TODO: convert everything to uppercase and print it out


# TODO: convert everything to lowercase and print it out


# TODO: check if the string starts with "Hey"


# TODO: check if the string ends with "be?"


# TODO: split the string into several separate strings, each containing only
```

1.7 Conditions

Change the variables, so that each if statement resolves as True.

In [ ]:

```
# TODO: change this code
number = 20
second_number = 10
first_array = ['a', 'b', 'c']
second_array = [1, 2, 3]

if number < 15:
    print("1")

if first_array:
    print("2")

if len(second_array) >= 4:
    print("3")

if len(first_array) + len(second_array) == 5:
    print("4")

if first_array and first_array[0] == 1:
    print("5")

if not second_number:
    print("6")
```

1.8 Loops

Loop through and print out all even numbers from the numbers list in the same order they are received.

Don't print any numbers that are **larger than** 500 in the sequence.

In [ ]:
```
numbers = [
    951, 402, 984, 651, 360, 69, 408, 319, 601, 485, 980, 507, 725, 547, 544,
    615, 83, 165, 141, 501, 263, 617, 865, 575, 219, 390, 984, 592, 236, 105,
    942, 941, 386, 462, 47, 418, 907, 344, 236, 375, 823, 566, 597, 978, 328,
    615, 953, 345, 399, 162, 758, 219, 918, 237, 412, 566, 826, 248, 866, 950
    626, 949, 687, 217, 815, 67, 104, 58, 512, 24, 892, 894, 767, 553, 81, 37
    843, 831, 445, 742, 717, 958, 609, 842, 451, 688, 753, 854, 685, 93, 857,
    440, 380, 126, 721, 328, 753, 470, 743, 527
]

even_numbers = []
# TODO: pick even numbers and save them to even_numbers


# test code
result = [402, 360, 408, 390, 236, 386, 462, 418, 344, 236, 328, 162, 412, 24
          104, 58, 24, 440, 380, 126, 328, 470]
if all(i == j for i, j in zip(even_numbers, result)):
    print('Great')
```

1.9 Functions

Add your own codes to the existing function so that you can create a fully functional program.

- Modify `list_benefits()` to return the following list of strings: "More organized code", "More readable code", "Easier code reuse", "Allowing programmers to share and connect code together"
- Modify `build_sentence(info)` to return a sentence starting with the given argument and ending with the string " is a benefit of functions!"
- Run and see all the functions work together!

In [ ]:
```
# TODO: modify this function
def list_benefits():
    pass

# TODO: modify this function
def build_sentence(benefit):
    pass

def name_the_benefits_of_functions():
    # TODO: call the modified function to create a list of benefits


    for benefit in list_of_benefits:
        # TODO: build a sentence according to the given benefit and print it

# test code
name_the_benefits_of_functions()
```

1.10 Classes and Objects

We have a class defined for vehicles. Create two new vehicles called `car1` and `car2`. Set `car1` to be a red convertible worth $60,000.00 with a name of Fer, and `car2` to be a blue van named Jump worth $10,000.00.

In [ ]:
```
# define the Vehicle class
class Vehicle:
    def __init__(self, name, kind, color, value):
        self.name = name
        self.kind = kind
        self.color = color
        self.value = value

    def description(self):
        desc_str = "%s is a %s %s worth $%.2f." % (self.name, self.color, sel
        return desc_str


# TODO: create two instances of Vehicle


# test code
print(car1.description())
print(car2.description())
```

1.11 Dictionaries

Add "Jake" to the `phonebook` with the phone number 938273443, and remove Jill from the `phonebook`.

In [ ]:
```
phonebook = {
    "John" : 938477566,
    "Jack" : 938377264,
    "Jill" : 947662781
}

# TODO: add the given item


# TODO: remove the specific item


# TODO: loop through the modifed dict and print out each key-value pair

```

1.12 Modules and Packages

Find out all functions in the `re` module, but only print out functions starting with `find` in reverse alphabetical order.

In [ ]:

```
import re


# TODO: write your code here
```

1.13 Numpy

1. Convert the list of lengths from a list to a Numpy array.
2. Convert all of the lenghts from inches to centimeters. Use the scalar conversion of 2.54 centimeters per inch to make your conversion.
3. Loop through the converted lengths and print out each inch-cm pair following the format "81.65 inches = 207.391 cm".

In [ ]:

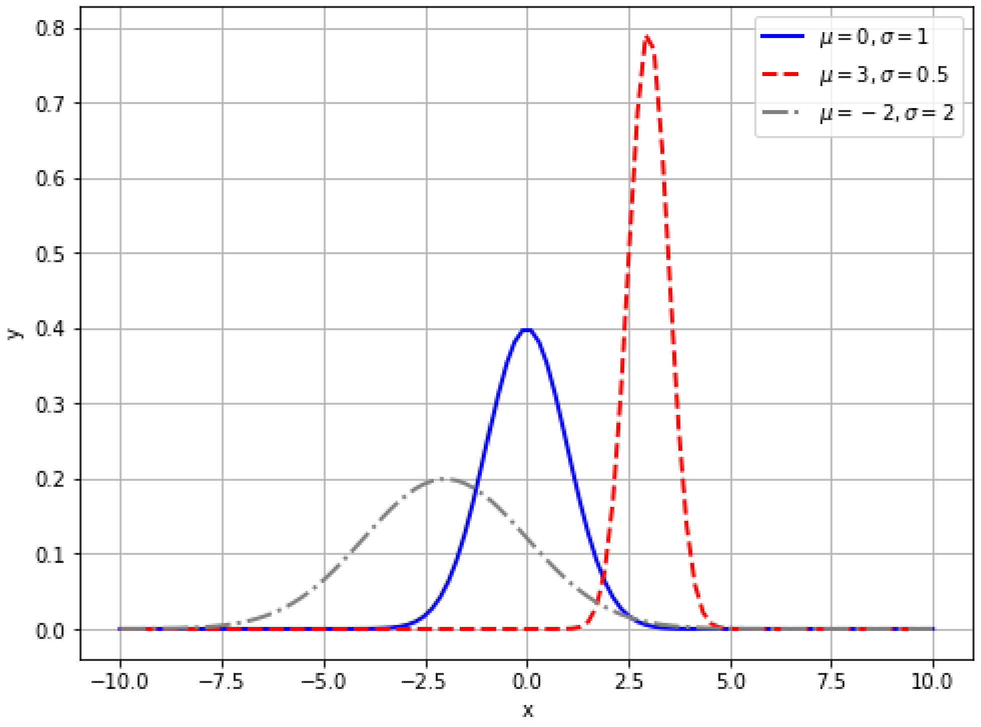
```
import numpy as np


length_inch = [81.65, 97.52, 95.25, 92.98, 86.18, 88.45]

# TODO: create a numpy array np_length_inch from length_inch


# TODO: create np_length_cm from np_length_inch


# TODO: print out np_length_cm
```

1.14 Matplotlib

In this exercise, you will practice using the package of matplotlib to visualize your data in Python. Please draw the same plot like the following one.

Requirements:

1. Each bell-shaped curve depicts the PDF of a Gaussian distribution parameterized by μ and σ.
2. The size of the figure should be 8 inches by 6 inches.
3. Curves have the same width, which is 2 in points.



In [ ]:

```
import matplotlib.pyplot as plt
import numpy as np

%matplotlib inline


# Init
x = np.linspace(-10, 10, 100)

# TODO: write your code here
```

Task 2: Matrix

2.1 Matrix basics

Given:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

$$B = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

$$C_1 = [1, 2, 3], C_2 = [4, 5, 6], C_3 = [7, 8, 9]$$

1. Compute Hadamard product (element-wise product) of A and B
2. Compute dot product of A and B

3. Compute Frobenius Norm of A (use loop)
4. Compute the sum of all the elements of A by using np.sum()
5. Concatenate C_1, C_2, C_3 as a new matrix C with shape of 3 by 3 by using np.concatenate() and np.vstack()
6. Reshape C into a 1 by 9 vector by using np.reshape()

In [ ]:
```
A = np.arange(1, 10, 1).reshape(3, 3)
B = np.arange(1, 10, 1).reshape(3, 3)
C1 = np.copy(A[0])
C2 = np.copy(A[1])
C3 = np.copy(A[2])

# TODO: element-wise product of A and B


# TODO: dot product of A and B


# TODO: Frobenius norm of A (use loop)


# TODO: compute the sum of all entries of A


# TODO: concatenate


# TODO: reshape
```

2.2 Numpy function vs. Loop

In this section you need to generate two 100 by 100 matrix D and E with random integer elements within 1~9.

1. Compute Hadamard product (element-wise product) of D and E by using numpy embeded function and print the computation time.
2. Compute Hadamard product (element-wise product) of D and E by using for loop and print the computation time.
3. Describe what do you find by checking the computational time of two methods. Which of the methods takes more time?

In [ ]:
```
import time


# TODO: generate D and E


# TODO: element-wise product of D and E


# TODO: print out the computation time
```

2.3 Dropout Matrix

Write a function `dropout()` that will randomly set matrix entries to 0 with a given probability. For example, given a 14 by 14 matrix and p=0.2, then 39 entries should be set to 0 at random locations. All other entries will not change.

Hint: $n * m * p$ might not be a integer, you should convert it to an integer using `int()` .

In [ ]:
```
def dropout(X, p):
    """
        Given an n by m matrix X and a probability p, pick n*m*p entries rand

        Parameters
        ----------
        X: ndarray
            Input of the given matrix.
        p: float
            Fraction of the matrix entries to drop.

        Return
        ----------
        res: ndarray
            Output of dropout
    """
    # TODO: write your code here



# test code
X = np.random.randint(2, 8, (15, 15))
p = 0.15

res = dropout(X, p)
if np.sum(res == 0) == int(n*m*p):
    print("Great")
```

2.4 Mini-batch

Creating small batches of training data is a critical step for mini-batch gradient descent. In this exercise, you should implement the function `create_batches(X, y, batch_size)` to generate mini-batches according to a specified batch size. **Note**: the given data should be shuffled in your function before splitting.

For example, the given training data X is a $n \times m$ matrix where n is the number of features and m is the number of samples, and y is a $k \times m$ matrix where k is the dimension of the label and m is the number of samples. The function you created should return two lists to save the splitted X and y respectively. The number of items for each returned list should be the ceiling of $\frac{m}{\text{batch\_size}}$.

In [ ]:

```
def create_batches(X, y, batch_size):
    """
        Split the data X and y with given batch size.

        Parameters
        ----------
        X: ndarray
            Features of the training data with shape n by m.
        y: ndarray
            Labels of the training data with shape k by m.
        batch_size: int
            Batch size.

        Return
        ----------
        batches_X: list
            Created small batches of X.
        batches_y: list
            Created small batches of y.
    """
    # TODO: write your code here



# test code
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical

# Load mnist dataset
(X, y), _ = mnist.load_data()

# Reshape inputs
X = X.reshape(X.shape[0], X.shape[1] * X.shape[2]).T

# Convert labels to one-hot vectors
y = to_categorical(y, 10).T

batch_size = 128
batches_X, batches_y = create_batches(X, y, batch_size)

if isinstance(batches_X, list) and isinstance(batches_y, list):
    if len(batches_X) == 469 and len(batches_y) == 469:
        if batches_X[-1].shape[0] == X.shape[0] and batches_X[0].shape[1] ==
            print('Great')
```

Task 3: Probability and Statistics

3.1 Basic

1. Generate a vector of size 10 with entries drawing from a standard Gaussian distribution $\mathcal{N}(\mu = 0, \sigma^2 = 1)$.
2. Generate a matrix of size 10 by 5 with entries drawing from a Gaussian distribution $\mathcal{N}(\mu = 2, \sigma^2 = 6)$.

In [ ]:
```
# TODO: generate a random Gaussian vector


# TODO: generate a random Gaussian matrix
```

3.2 Covid-19

Given the covid-19 cumulative increase cases of each county in South Carolina from Jan. 22nd, 2020 to Jan. 18th, 2022, you need to calculate the total daily increase cases of SC and complete the following tasks.

1. Read the dataset from the given csv file and calculate the total daily increase data.
2. Generate a bar plot of the positive increase number.
3. Find mean and variance of covid-19 dataset.
4. Define a function `moving_avg(data, n)` for the Moving Average calculation, where n is the past number of days. Use the function to generate the results for n=7 on covid-19 dataset and plot the curve.
5. Define a function `EWMA(data, alpha)` for the Exponential Weighted Moving Average calculation, where alpha is the degree of weighting decrease coefficient. Use the function to generate the results for alpha=0.9 on covid-19 dataset and plot the curve.

Requirements for the plot:

1. Add the title, axis labels, legend for the figures.
2. x axis is the date (simply use integers start from 1), y axis is the daily increase cases.
3. You should have two figures:
    - Bar plot only
    - Bar plot, moving average curve and exponential weighted moving average curve together

Hint: Please find detail of Moving Average and Exponential Weighted Moving Average in the link: https://en.wikipedia.org/wiki/Moving_average (https://en.wikipedia.org/wiki/Moving_average)

In [ ]:

```
import csv
import numpy as np
import matplotlib.pyplot as plt

%matplotlib inline

# TODO: read dataset


# TODO: data preprocessing


# TODO: generate the positive increase number bar plot


# TODO: find mean and variance


# TODO: define the function moving_avg(data, n) and find the moving average f


# TODO: define the function EWMA(data, alpha) and calculate the exponential w


# TODO: plot the daily increase cases curve for moving average results and ex
```