🧪 Exercise 1:

Introduction to Python, iPython Notebooks and basic packages (numpy and matplotlib)

Instructions:

- You will be using Python 3.
- The notebook is a mixture of codes with explaination and tasks to solve.
- Solve all taks that are marked with ✅ **Task for students** tag.
- If you are unable to solve the task, move ahead and try to solve it at home.
- Avoid using for-loops and while-loops, unless you are explicitly told to do so.
- iPython Notebooks are interactive coding environments embedded in a webpage. You will be using iPython notebooks in this class.
- After writing your code, you can run the cell by either pressing "SHIFT"+"ENTER" or by clicking on "Run Cell" (denoted by a play symbol) in the upper bar of the notebook.
- You only need to write code in the cells or specify that they are used for documentation in Markdown format using "Esc" followed by "m" to markdown or "Esc" followed by "y" for code.

After this assignment you will:

- Be able to use iPython Notebooks
- Be able to code in Python
- Be able to use numpy functions and numpy matrix/vector operations
- Be able to plot some simple functions using matplotlib
- Be able to vectorize code

1. Shell Commands in IPython

The shell is a way to interact textually with your computer. Any command that works at the command-line can be used in IPython by prefixing it with the ! character on MacOs or Linux or % on Windows. For example, the ls, pwd, and echo commands can be run as follows:

In [ ]:
```
%pwd
```

In [ ]:
```
contents = !ls
print(contents)
```

2. Import packages section

Here you import the packages you need in your code but you can also do this later on when you need the packages

In [ ]:
```
import numpy as np
import pandas as pd
```

3. Accessing Documentation

- **with shift + Tab:**

  move the cursor into the function and press shift + Tab. A text box with function's signature will pop up showing all options.

- **with ?**

  The Python language and its data science ecosystem is built with the user in mind, and one big part of that is access to documentation. Every Python object contains the reference to a string, known as a doc string, which in most cases will contain a concise summary of the object and how to use it. Python has a built-in help() function that can access this information and prints the results. For example, to see the documentation of the built-in len function, you can do the following:

```
In [1]: help(len)
Help on built-in function len in module builtins:

len(...)
    len(object) -> integer
```

Return the number of items of a sequence or mapping.

In [ ]:
```
# ✅ Task for students: print the signature of the DataFrame object of Pandas

print(pd.DataFrame)

# Print the function/method signature of the DataFrame class

# print(inspect.signature(pd.DataFrame))


# pd.DataFrame?
```

Accessing Source Code with ??

Another usefull tool is reading the source code of the object you're curious about. IPython provides a shortcut to the source code with the double question mark (??):

In [ ]:
```
# ✅ Task for students: print the  source code of numpy mean function

#np.mean

# print(inspect.getsource(np.mean))
```

```
# np.mean?

#help(np.mean)
```

Tab-completion of object contents

Handy tool to see a list of all available attributes of an object, or to find all possible imports in a package ..etc

In [ ]:
```
# ✅ Task for students:  create a array with zeros of shape 4,6 in numpy and lis


arr = np.zeros((4, 6))

arr
```

4. Immutable and mutable Objects in Python

4.1 Immutable

Immutable objects are built-in data types such as int, float, bool, string, Unicode, and tuple . Simply put, an immutable object cannot be changed after it is created.

Example 1: In this example, we take a tuple and try to change its value at a certain index and print it. Since a tuple is an immutable object, if we try to change it, an error will be thrown.

In [ ]:
```
# Python code to test that (outputs error)
# tuples are immutable
tuple1 = (0, 1, 2, 3)
#tuple1[0] = 4
print(tuple1)
```

In [ ]:
```
# strings are immutable  (outputs error)
message = "Welcome to Jupyter for AI & Data Science"
#message[0] = 'p'
print(message)
```

4.2 Mutable Objects in Python

Mutable objects are of type Python list, Python dict , or Python set . User-defined classes are generally mutable.

In [ ]:
```
my_list = [1, 2, 3]
my_list.append(4)
print(my_list)

my_list.insert(5,7)
print(my_list)

my_list.remove(2)
print(my_list)
```

```
popped_element = my_list.pop(0)
print(my_list)
print(popped_element)
```

In [ ]:
```
my_dict = {"name": "Ram", "age": 25}
new_dict = my_dict
new_dict["age"] = 37

print(my_dict)
print(new_dict)
```

In [ ]:
```
my_set = {1, 2, 3}
new_set = my_set
new_set.add(4)

print(my_set)
print(new_set)
```

5. List comprehension

It's a concise way to create a list by iterating over a sequence or range and applying an expression to each element in the iteration.

A list comprehension produces -- a list!

In [ ]:
```
[i for i in range(5)]

[j for j in range(10)]
```

In [ ]:
```
# ✅ Task for students: Use an if condition in a list comprehension to filter nu
# The resulting list should include only the numbers that are divisible by 2 (i.
[i for i in range(9) if i%2==0]
```

6. Assert statements

The assert statement exists in almost every programming language. It has two main uses:

- It helps detect problems early in your program, where the cause is clear, rather than later when some other operation fails. A type error in Python, for example, can go through several layers of code before actually raising an Exception if not caught early on.
- It works as documentation for other developers reading the code, who see the assert and can confidently say that its condition holds from now on.

In [ ]:
```
assert (1+5)==6
```

In [ ]:
```
# assert (1+6)== 8

1+6 == 8
```

7. Functions in Python

In [ ]:
```python
# ✅ Task for students: implent the following function
def add_values(a, b):
    """Calculate the sum a + b"""

    ### BEGIN SOLUTION
    raise NotImplementedError()

    ### END SOLUTION
    return a + b
```

In [ ]:
```python
# assert add_values(1,3) == 4
# assert add_values(-4, 2) == -2
```

In [ ]:
```python
# ✅ Task for students: implement the following function
def cumulative_sum(n):
    """Calculate the sum 1 + 2 + 3 + ... + n"""

    ### BEGIN SOLUTION
    raise NotImplementedError()
    ### END SOLUTION

    return n * (n + 1) // 2
```

In [ ]:
```python
# assert cumulative_sum(1) == 1
# assert cumulative_sum(5) == 1 + 2 + 3 + 4 + 5
# assert cumulative_sum(100) == 5050
```

In [ ]:
```python
# ✅ Task for students: implement the following function
# In Python, [::-1] is called slicing with a negative step.
# This particular syntax is used to reverse a list or sequence (like a string) b
def is_palindrome(s):
    """Returns True, when s does not change when being reversed"""

    ### BEGIN SOLUTION
#    raise NotImplementedError()
    ### END SOLUTION

    return s == s[::-1]
```

In [ ]:
```python
assert is_palindrome("racecar") == True
assert is_palindrome("foo") == False
assert is_palindrome("oof") == False
assert is_palindrome("a") == True
assert is_palindrome("ab") == False
assert is_palindrome("aba") == True
assert is_palindrome("abba") == True
```

In [ ]:
```python
# ✅ Task for students: implement the following function
def find_the_a(s):
    """Returns the position of the first "a" in s or False if none exists"""

    ### BEGIN SOLUTION
    #raise NotImplementedError()
    ### END SOLUTION

    if "a" in s:
```

```
        return s.index("a")  # Return the index of the first occurrence of "a"
    return False  # Return False if "a" is not found
```

In [ ]:
```
assert find_the_a("Hallo Welt!") == 1
assert find_the_a("Bacon and Spam") == 1
assert find_the_a("edcba") == 4
assert find_the_a("Hello world!") == False
assert find_the_a("") == False
```

In [ ]:
```
# ✅ Task for students: implement the following function
def fibonacci(n):
    """Returns the fibonacci sequence [1,1,2,3,5,8,...,m], with m being the last
        see https://en.wikipedia.org/wiki/Fibonacci_number"""

    ### BEGIN SOLUTION
#    raise NotImplementedError()
    ### END SOLUTION

    fibs = [fib(i) for i in range(n)]
    return fibs
```

In [ ]:
```
assert fibonacci(1) == [1,1]
assert fibonacci(2) == [1,1,2]
assert fibonacci(3) == [1,1,2,3]
assert fibonacci(20) == [1,1,2,3,5,8,13]
```

In [ ]:
```
# ✅ Task for students: implement the following function
def is_prime(x):
    """Returns True if x is a prime number"""

    ### BEGIN SOLUTION
    raise NotImplementedError()
    ### END SOLUTION
```

In [ ]:
```
assert is_prime(2) == True
assert is_prime(3) == True
assert is_prime(4) == False
assert is_prime(5) == True
assert is_prime(6) == False
assert is_prime(10) == False
assert is_prime(101) == True
```

8. Object-Oriented Programming (OOP) in Python

In [ ]:
```
class Dog:
    # Constructor to initialize the dog's name and age
    def __init__(self, name, age):
        self.name = name  # attribute
        self.age = age    # attribute

    # Method to make the dog bark
    def bark(self):
        return f"{self.name} says woof!"

    # Method to describe the dog
    def describe(self):
        return f"{self.name} is {self.age} years old."
```

```python
# Create instances (objects) of the Dog class
dog1 = Dog("Buddy", 4)
dog2 = Dog("Lucy", 2)

# Interact with the objects
print(dog1.bark())        # Output: Buddy says woof!
print(dog2.describe())    # Output: Lucy is 2 years old.
print(dog1.describe())    # Output: Buddy is 4 years old.
```

9. Plotting in Ipython

The two main libraries we will be using throughout this course, are searborn and matplotlib.

✅ Task for students: try to create a plot of the following function using matplotlib:

$$f(x) = \begin{cases} x & \text{if } x >= 0 \\ 0 & \text{else} \end{cases}$$

In [ ]:
```python
import matplotlib.pyplot as plt

# Create the plot of f from -3 <= x <= 3

### BEGIN SOLUTION
raise NotImplementedError()
### END SOLUTION
```

🚀 10. NumPy

NumPy is an open-source Python library that facilitates efficient numerical operations on large quantities of data. When coding in numpy remeber to leverage vectorization.

Vectorization allows you to operate on entire arrays without using explicit loops. This leads to faster and more readable code.

In [ ]:
```python
import numpy as np

# Example: Multiply each element in an array by 2

# Without vectorization (Python loop)
a_list = [1, 2, 3, 4, 5]
result_loop = []
for x in a_list:
    result_loop.append(x * 2)
print("Using loop:", result_loop)

# With NumPy vectorization
a_array = np.array([1, 2, 3, 4, 5])
result_vectorized = a_array * 2
print("Using vectorization:", result_vectorized)
```

✅ Task for students:

1. Create a NumPy array from 0 to 9.

2. Compute the square of each number using vectorization.
3. Then, compare it to a loop-based version (you can use list comperhension)
4. use %timeit to measure how much time does it take in both cases

In [ ]:
```
# write your code here
```

✅ Task for students: Build a vectorized function that returns the sigmoid of a real number x.

Use math.exp(x) for the exponential function. The sigmoid is the function $\sigma(x) = \frac{1}{1+e^{-x}}$.

In [ ]:
```
def sigmoid(x):
    """
    Compute the sigmoid of x, with x being a scalar or a numpy array
    """

    ### BEGIN SOLUTION
    raise NotImplementedError()
    ### END SOLUTION
```

In [ ]:
```
assert np.allclose(sigmoid(np.array([1,2,3])), [0.73105858, 0.88079708, 0.952574
```

✅ Task for students:

Implement the numpy vectorized version of the L1 loss. You may find the function abs(x) (absolute value of x) useful.

Reminder:

- The loss is used to evaluate the performance of some ML models. The bigger your loss is, the more different your predictions ($\hat{y}$) are from the true values (y). In Machine learning, you use optimization algorithms like Gradient Descent to train your model and to minimize the cost.
- L1 loss is defined as:

$$L_1(\hat{y}, y) = \sum_{i=0}^{m-1} |y^{(i)} - \hat{y}^{(i)}| \tag{6}$$

In [ ]:
```
def L1(yhat, y):
    """
    Arguments:
    yhat -- vector of size m (predicted labels)
    y -- vector of size m (true labels)

    Returns:
    loss -- the value of the L1 loss function defined above
    """
    ### BEGIN SOLUTION
    raise NotImplementedError()
    ### END SOLUTION
```

In [ ]:
```
yhat = np.array([.9, 0.2, 0.1, .4, .9])
y = np.array([1, 0, 0, 1, 1])
```

```
assert np.allclose(L1(yhat, y), 1.1)
```

✅ Task for students:

Implement the numpy vectorized version of the L2 loss. There are several way of implementing the L2 loss but you may find the function np.dot() useful.

As a reminder, if $x = [x_1, x_2, \ldots, x_n]$, then `np.dot(x,x)` = $\sum_{j=0}^{n} x_j^2$.

L2 loss is defined as:

$$L_2(\hat{y}, y) = \sum_{i=0}^{m-1} (y^{(i)} - \hat{y}^{(i)})^2 \tag{7}$$

In [ ]:
```
def L2(yhat, y):
    """
    Arguments:
    yhat -- vector of size m (predicted labels)
    y -- vector of size m (true labels)

    Returns:
    loss -- the value of the L2 loss function defined above
    """
    # YOUR CODE STARTS HERE
    raise NotImplementedError()
    ### END SOLUTION
```

In [ ]:
```
yhat = np.array([.9, 0.2, 0.1, .4, .9])
y = np.array([1, 0, 0, 1, 1])

assert np.allclose(L2(yhat, y), 0.43)
```

What to remember

- learn the most important shortcuts of jupter notebook (add and remove a cell, write documentation, use shell...etc.)
- Vectorization is very important in Data Analysis. It provides computational efficiency and clarity.
- Familiarize yourself with numpy functions such as, np.shape, np.reshape, np.sum, np.dot, np.multiply, np.maximum, etc...

Resources

- Python Data Science Handbook by Jake VanderPlas; the content is available on GitHub.
- The IPython website: links to documentation, examples, tutorials, and a variety of other resources.
- The nbviewer website: static renderings of any IPython notebook available on the internet.

- A gallery of interesting Jupyter Notebooks: This ever-growing list of notebooks, powered by nbviewer, shows the depth and breadth of numerical analysis you can do with IPython. It includes everything from short examples and tutorials to full-blown courses and books composed in the notebook format!

In [ ]: