PracticalNotebook1

January 22, 2023

1 Practical Notebook 1

PLEASE NOTE, the assignments throughout the course are designed to be solved by searching online.

If you have questions about a function, try to Google it or run the command you want more information from with the question mark, `?` in front of it

1.1 Introduction to Python and Jupyter Notebook

The purpose of this lab is to familiarize oneself with the basics of Python, working with the interactive environment Jupyter Notebooks, as well as some introductory problems with regards to matrix operations, loading data, and plotting. Python is extensively used in machine learning applications.

Jupyter Notebook is an interactive environment for executing Python code (among other languages). These notebooks are executed in sequential order and consist of text and code blocks. Pressing `shift + enter` or the `play` arrow to the left will execute one segment at a time and move one down to the next segment. If you are running a notebook with Google Colab, the notebook will be inactive if no work is executed.

Below are presented some examples of Python code some commonly applicable operations.

1.1.1 Print statements

Print statements in pythons can be done in multiple ways. Below are listed some of the common methods for printing variables in Python

```python
# The various ways for printing values in Python
answer = 42
print("The answer is", answer)
print("The answer is " + str(answer))
print("The answer is %s" % answer)
print(f"The answer is {answer}")

# Printing special character
print("\nItem 1 \nItem 2\n")
```

```
The answer is 42
The answer is 42
```

```
The answer is 42
The answer is 42

Item 1
Item 2
```

1.1.2 Math functions

```python
# Math functions
answer = 42.62
print("Answer = %.2f with two decimals" % answer)
print(f"Answer = {answer:.2f} with two decimals\n")

# Division
print("5/2 =", 5/2)
print("5//2 =", 5//2)

# Rounding
print("\nRounding numbers", int(answer))
print("Rounding numbers", round(answer))
```

```
Answer = 42.62 with two decimals
Answer = 42.62 with two decimals

5/2 = 2.5
5//2 = 2

Rounding numbers 42
Rounding numbers 43
```

1.1.3 Lists

One of the most common data types used in Python are lists. Lists can expand or shrink dynamically and contain any data type.

```python
# Inserting into an empty list
generic_list = []
generic_list.append((42, "Answer", True))
print(generic_list)

# Generating a list up to a range
ranged_list = list(range(1, 5))
print("\nRanged based list:", ranged_list)
```

```
[(42, 'Answer', True)]

Ranged based list: [1, 2, 3, 4]
```

Indexing of a list starts at 0 in python

```
[ ]: # Retreving elements from a list
     generic_list = [0, 1, 2, 3, 4, 5, 6]
     print("generic_list[0] =", generic_list[0])
```

```
generic_list[0] = 0
```

... and can be done in reverse order

```
[ ]: print("generic_list[-1] =", generic_list[-1])
```

```
generic_list[-1] = 6
```

We can also extract slices of a list

```
[ ]: print("Print elements in index 1-5:", generic_list[1:5])
     print("Print all elements up to index 5:", generic_list[:5])
```

```
Print elements in index 1-5: [1, 2, 3, 4]
Print all elements up to index 5: [0, 1, 2, 3, 4]
```

```
[ ]: print("Every other element in a list:", generic_list[::2])
```

```
Every other element in a list: [0, 2, 4, 6]
```

Assignment 1a)

```
[ ]: # ASSIGNMENT
     # print every odd number of the list
     print("Every odd element in a list:", generic_list[1::2])
```

```
Every odd element in a list: [1, 3, 5]
```

Lists can also be nested in other lists

```
[ ]: double_list = [[1, 2, 3], ["Bert", "Elmo", "Big Bird"]]
     print(double_list)
     print(double_list[1])
     print(double_list[1][1])
```

```
[[1, 2, 3], ['Bert', 'Elmo', 'Big Bird']]
['Bert', 'Elmo', 'Big Bird']
Elmo
```

Built in list functions   min, max, concatinating lists, insertion, removal

```
[ ]: generic_list = [1, 932, 77, 52, 2]
     print("min(generic_list) =", min(generic_list))
     print("max(generic_list) =", max(generic_list))
     print("sum(generic_list) =", sum(generic_list))
```

```python
print()

concaticated_list = ["Bert", "Elmo"] + ["Big Bird"]
print(concaticated_list)

# append
concaticated_list.append("Erni")
print(concaticated_list)

# insert at index
concaticated_list.insert(1, "The Count")
print(concaticated_list)

# remove based on index
concaticated_list.pop(3)
print(concaticated_list)

# remove value from list
concaticated_list.remove("Erni")
print(concaticated_list)

# verify if an element is in a list
Bert_in_list = "Bert" in concaticated_list
print("\nBert is in the list? ", Bert_in_list)
```

```
min(generic_list) = 1
max(generic_list) = 932
sum(generic_list) = 1064

['Bert', 'Elmo', 'Big Bird']
['Bert', 'Elmo', 'Big Bird', 'Erni']
['Bert', 'The Count', 'Elmo', 'Big Bird', 'Erni']
['Bert', 'The Count', 'Elmo', 'Erni']
['Bert', 'The Count', 'Elmo']

Bert is in the list?  True
```

Ranged based loops on lists

```python
# Range based for loop
length = 5
for i in range(length):
    print(f"{i} x 2 = {i*2}")
```

```
0 x 2 = 0
1 x 2 = 2
2 x 2 = 4
3 x 2 = 6
```

```
4 x 2 = 8
```

```python
# Non pythonic way of printing a list
name_list = ["Ada Lovelace", "Alan Turing", "Grace Hopper"]
for i in range(len(name_list)):
    print(name_list[i], "a pioneer in computer science")
```

```
Ada Lovelace a pioneer in computer science
Alan Turing a pioneer in computer science
Grace Hopper a pioneer in computer science
```

```python
# Printing a list
# Assignment: print content of the list with out the `range` function
name_list = ["Ada Lovelace", "Alan Turing", "Grace Hopper"]
for i in name_list:
    print(i, "a pioneer in computer science")
```

```
Ada Lovelace a pioneer in computer science
Alan Turing a pioneer in computer science
Grace Hopper a pioneer in computer science
```

```python
# There are multiple ways to do a ranged based for loop in Python
name_list = ["Ada Lovelace", "Alan Turing", "Grace Hopper"]
for i in range(1, (len(name_list)+1)):
    print(i, name_list[i-1])
print()

name_list = ["Ada Lovelace", "Alan Turing", "Grace Hopper"]
index_list = [1, 2, 3]
for i, name in zip(index_list, name_list):
    print(i, name)
print()

name_list = ["Ada Lovelace", "Alan Turing", "Grace Hopper"]
for i, name in enumerate(name_list):
    print(i+1, name)
print()
```

```
1 Ada Lovelace
2 Alan Turing
3 Grace Hopper

1 Ada Lovelace
2 Alan Turing
3 Grace Hopper

1 Ada Lovelace
2 Alan Turing
```

Assignment 1b)

```
[ ]:  # ASSIGNMENT:
      # sort the "generic_list" in ascending order and print it.

      generic_list = [1, 932, 77, 52, 2]
      # YOUR CODE HERE
      # Hint: There is a sorting function in python that will do this. Google it.
      generic_list.sort()
      print("Sorted list: ", generic_list)
```

```
Sorted list:  [1, 2, 52, 77, 932]
```

1.1.4 Sets

Sets are effective methods for filtering a collection of duplicate values

Assignment 1c)

```
[ ]:  lst1 = [0, 1, 3, 5, 6, 7, 5, 8, 9, 10, 12]
      lst2 = [1, 2, 2, 1, 2, 4, 5, 8, 2, 13, 9, 11]

      # ASSIGNMENT:
      # a) print the union of elements between the two lists,
      #    i.e. all the elements should be included once.
      # b) print the intersection of elements between the two lists,
      #    i.e. the elements that appear in both lists.
      #
      # HINT: look into `sets` in Python

      # YOUR CODE HERE
      set1 = set(lst1)
      set2 = set(lst2)
      print("Union of the lists: ", set1.union(set2))

      print("Intersection of the lists: ", set1.intersection(set2))
```

```
Union of the lists:  {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13}
Intersection of the lists:  {8, 1, 5, 9}
```

1.1.5 functions

Functions in can return multiple objects from a function. The results can either be retrieved as a tuple and specify an index or as individual values

Assignment 1d)

```
# ASSIGNMENT:
# Write a function that will return from a list:
#    1.  The highest
#    2.  The lowest
#    3.  The first
#    4.  The last elements from a list

def our_custom_function(lst):
    # YOUR CODE HERE
    return max(lst), min(lst), lst[0], lst[-1]


lst = [6, 90, 42, -1, 45]

highest, lowest, first, last = our_custom_function(lst)

assert highest == 90
assert lowest == -1
assert first == 6
assert last == 45
```

Assignment 1e)

```
# List comprehention
squared = [x**2 for x in range(10)]
print("Squared list:  \t\t\t ", squared)

# ASSIGNMENT:
# Generate a list of the 20 Fibbonacci numbers,
#    BUT exclude all odd numbers Fibbonacci numbers

def fib(n):
    if n <= 1:
        return 1
    else:
        return fib(n-1) + fib(n-2)

# Make sure here you use list comprehension. Find out more about list
  ↪comprehension from: https://www.w3schools.com/python/
  ↪python_lists_comprehension.asp
# TODO YOUR CODE HERE.

lst = [fib(x) for x in range(20) if fib(x)%2==0]

assert lst == [2, 8, 34, 144, 610, 2584]
```

```
Squared list:                     [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

1.2   Python and Libraries

Much like Node, Ruby, and similar languages; Python as language attempts to have a small and extendable library of core functions, where extra functionality can be extended with core libraries or external libraries.

IF are executing a Jupyter Notebook or Python code locally on your computer, we strongly suggest you create and activate a virtual environment for installing your packages such as NumPy, pandas, matplotlib, or similar. This is to ensure that the packages installed to execute this lab do not override previously installed Python libraries. To install packages, we recommend using the Anaconda distribution or Python pip. See this small python guide about how to set up python notebooks locally.

1.3   NumPy

NumPy is an external Python library for matrix and vector operations. This means that the library needs to be installed and imported. If you are using a Jupyter Notebook or a plain Python file you may need to install NumPy. In Google Colab, NumPy and other commonly used libraries are pre-installed. Please note that a Python list is not the same as a vector or matrix.

Below is an illustration of how NumPy can be used to illustrate the mathematics of the Mandelbrot set. We use it as a motivating example of what is possible to do with NumPy and other libraries.

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

def mandelbrot(h=400, w=400, max_iter=20):
    """Returns an image of the Mandelbrot fractal of size (h,w)."""
    y, x = np.ogrid[-1.4:1.4:h*1j, -2:0.8:w*1j]
    c = x + y * 1j
    z = c
    divtime = max_iter + np.zeros(z.shape, dtype=int)

    for i in range(max_iter):
        z = z**2 + c
        diverge = z * np.conj(z) > 2**2          # who is diverging
        div_now = diverge & (divtime == max_iter)  # who is diverging now
        divtime[div_now] = i                      # note when
        z[diverge] = 2                            # avoid diverging too much

    return divtime

plt.figure(figsize=(8,8)) # change the sizes to view the figure more easily
plt.imshow(mandelbrot(400, 400, 100), cmap='twilight_shifted')
```
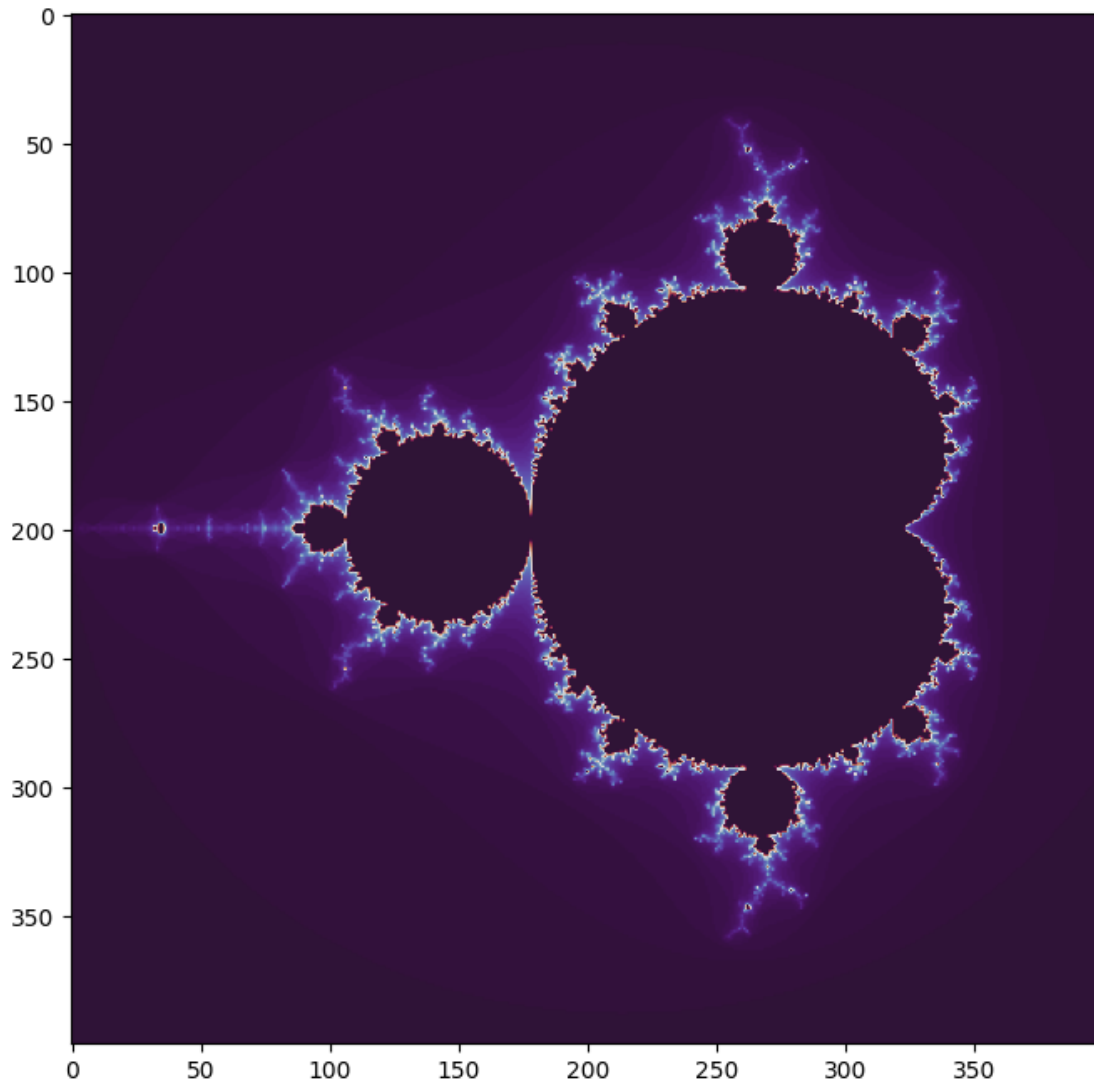
```
[ ]:  <matplotlib.image.AxesImage at 0x7feb385fc6a0>
```

An installed library can be imported
`import numpy`.

A library can be imported with an alias
`import numpy as np`

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

The NumPy library is mainly written in C and wrapped in Python for ease of use.

```
[ ]: # Creates a vector of ones
     vec = np.array([1, 2, 3])

     # Vectors and lists support mathematical operations differently!
```

```python
lst = [1, 2, 3]
print("2*lst:", 2*lst)
assert len(2*lst) == 6

print("2*vec:", 2*vec)
assert len(2*vec) == 3
```

```
2*lst: [1, 2, 3, 1, 2, 3]
2*vec: [2 4 6]
```

```python
# One dimentional arrays
zeros = np.zeros(4)
print(zeros)

ranged = np.arange(4)
print(ranged)

ranged = np.linspace(0, 1, 4)
print(ranged)

custom = np.array([1, 7, 9, 3])
print(custom)

# Creating arrays from lists
custom = np.array([5, 42, 82])
print(custom)
```

```
[0. 0. 0. 0.]
[0 1 2 3]
[0.         0.33333333 0.66666667 1.        ]
[1 7 9 3]
[ 5 42 82]
```

Making multidimensional arrays

```python
# From the start
multi_arr = np.ones([3, 2])
print(multi_arr)
print()


# Or manually assign the shape
multi_arr = np.array([[1,1],
                      [1,1],
                      [1,1]]).astype(np.float32)
print(multi_arr)
```

```
[[1. 1.]
```

```
 [1. 1.]
 [1. 1.]]

[[1. 1.]
 [1. 1.]
 [1. 1.]]
```

Arrays can also be stacked together

```python
[ ]: arr = np.array([[1, 2, 3],
                     [4, 5, 6]])
     stacked_horizontal = np.hstack((arr, arr))
     print(stacked_horizontal)
     print()

     stacked_verticaly = np.vstack((arr, arr))
     print(stacked_verticaly)
```

```
[[1 2 3 1 2 3]
 [4 5 6 4 5 6]]

[[1 2 3]
 [4 5 6]
 [1 2 3]
 [4 5 6]]
```

Assignment 1f)

```python
[ ]: import numpy as np

     arr = np.array([1, 2, 3, 4, 5, 6])
     print(f'shape before: {arr.shape}')
     print(f'dimension before: {arr.ndim}')

     # ASSIGNMENT
     # extend or reshape the 1D array into a 2D array - LOOKUP ONLINE
     # i.e. it should have the shape (1,6).

     # YOUR CODE HERE
     arr = arr[np.newaxis]

     print(f'\nshape after: {arr.shape}')
     print(f'dimension after: {arr.ndim}')

     assert arr.ndim == 2
     assert arr.shape[0] == 1 and arr.shape[1] == 6

     # Feel free to reshape the array to other dimensions as well
```

```
shape before: (6,)
dimension before: 1

shape after: (1, 6)
dimension after: 2
```

Transposing an array

```python
# Transposing a matrix
multi_arr = np.array([[1, 2, 3],
                      [4, 5, 6]])
print("A =\n", multi_arr)


multi_arr_transp = multi_arr.T
multi_arr_transp = np.transpose(multi_arr)

print("\nA.T =\n", multi_arr_transp)
```

```
A =
 [[1 2 3]
 [4 5 6]]

A.T =
 [[1 4]
 [2 5]
 [3 6]]
```

We can also perform the expected mathematical operation on arrays

```python
arr = np.array([1, 2, 3, 4])

# Addition
print("Add one to array", arr + 1)

# Multiplication
print("Multiply array by 2", arr*2)

# Division
print("Divide array by 2", arr/2)

# Raised to some power
print("Log of an array", np.power(arr, 2))

# Logarithm
print("Log of an array", np.log(arr))
print("Log2 of an array", np.log2(arr))
print("Log2 of an array", np.log(arr)/np.log(2))
```

```
Add one to array [2 3 4 5]
Multiply array by 2 [2 4 6 8]
Divide array by 2 [0.5 1.  1.5 2. ]
Log of an array [ 1  4  9 16]
Log of an array [0.         0.69314718 1.09861229 1.38629436]
Log2 of an array [0.         1.         1.5849625 2.         ]
Log2 of an array [0.         1.         1.5849625 2.         ]
```

As well as commonly used matrix operations

```python
A = np.array([[1, 0, 0],
              [3, 1, 0],
              [4, 0, 1]])
B = np.array([[23, 10, 12],
              [0, 0, 0],
              [0, 0, 0]])

print("A =\n", A, "\n")
print("b =\n", B, "\n")

# Matrix multiplication
arr_mult = A @ B     # or
arr_mult = np.matmul(A, B)
print("A*B\n", arr_mult)

# fun fact... The matrix A*B is known as a vampire matrix
```

```
A =
 [[1 0 0]
 [3 1 0]
 [4 0 1]]

b =
 [[23 10 12]
 [ 0  0  0]
 [ 0  0  0]]

A*B
 [[23 10 12]
 [69 30 36]
 [92 40 48]]
```

Assignment 1g)

```python
M = np.array([[23, 10, 12],
              [69, 30, 36],
              [92, 40, 48]])
```

```
# ASSIGNMENT:
# a) Square an matrix M
# b) Square the values in a Matrix M


# Squaring the matrix

# YOUR CODE HERE
sqr_arr = M@M

assert np.all(sqr_arr == np.array([[2323, 1010, 1212],
                                   [6969, 3030, 3636],
                                   [9292, 4040, 4848]]))

# Square the values in an array

# YOUR CODE HERE
sqr_val = M**2

assert np.all(sqr_val == np.array([[529,  100,  144],
                                   [4761, 900, 1296],
                                   [8464, 1600, 2304]]))
```

Assignment 1h)

```
[ ]: rand = np.random.RandomState(42)

# ASSIGNMENT
# Generate a random array of whole numbers between (0, 10)
# with the size of 3x5. Then sort the array along the x-axis

# YOUR CODE HERE
arr = rand.randint(10, size=(3, 5))
arr = np.sort(arr)



assert np.all(arr == np.array([[3, 4, 6, 6, 7],
                               [2, 4, 6, 7, 9],
                               [2, 3, 5, 7, 7]]))
```

```
[ ]: # Logical operations
arr = np.array([1,2,3,4,5,6])
filtered_arr = arr[(arr > 2) & (arr < 6)]
print(filtered_arr)

filtered_arr = arr[arr%2==0]
```

```
print(filtered_arr)
```

```
[3 4 5]
[2 4 6]
```

Assignment 1i)

```
[ ]: # ASSIGNMENT:
     # Filter out duplicate AND odd numbers

     arr = np.array([1, 4, 2, 2, 3, 4, 4, 4, 2, 3, 4, 5, 6])

     # YOUR CODE HERE
     arr = list(set([x for x in arr if x%2 == 0]))

     assert np.all(arr == np.array([2, 4, 6]))
```

Assignment 1i) What is the difference between the following two statements?
a)

```
arr1 = np.array([1, 2, 3])
arr2 = arr1
arr2[0] = 100
```

   b)

```
arr1 = np.array([1, 2, 3])
arr2 = arr1.copy()
arr2[0] = 100
```

YOUR ANSWER:

```
[ ]: """
     ASSIGNMENT:
     Your answer here...

     The first statement simply passes arr1 by reference to arr2, so that editing␣
      ↪arr2, will also edit arr1, as they are technically the same array.
     In this case, making the first element of arr2 = 100, also does the same thing␣
      ↪for arr1

     Comperatively, the second statement copies the array into arr2, seperating the␣
      ↪arrays, and so when changing the first element of arr2 to 100, the first␣
      ↪element of arr1 remains = 1
     """
```

```
[ ]: '\nASSIGNMENT:\nYour answer here…\n\nThe first statement simply passes arr1 by
     reference to arr2, so that editing arr2, will also edit arr1, as they are
     technically the same array.\nIn this case, making the first element of arr2 =
```

100, also does the same thing for arr1\n\nComperatively, the second statement
copies the array into arr2, seperating the arrays, and so when changing the
first element of arr2 to 100, the first element of arr1 remains = 1\n'

1.3.1 Random Walk

The task here is to create a random walk function and plot the result.
You will probably not find all the information of how to solve this task in the lab instructons
entirely, so you will need to find the answer online.
A random walk is categorized as taking a random step in any number of directions and then
accumumalating the distance one has taken.

We want you to make a function which either steps up (+1) or down (-1) in each step (1D random
walk) for a 1000 steps and aggrigate the result. If you for example walk [up, up, down, up, up,
down, down, down], this corresponds to [1,1,-1,1,1,-1,-1,-1]. With the starting position 0 this will
put you in position [1,2,1,2,3,2,1,0] for the different timesteps.

Assignment 1k)

```python
# ASSIGNMENT:
# Complete the random walk function

def random_walk(n=1000):
    res = [0]
    for x in range(n-1):
        res.append(res[-1] + np.random.choice([-1, 1]))
    return res


np.random.seed(7)
steps = np.arange(1000)
walk = random_walk(1000)
```

To better illustrate what a random walk does, we can plot the results with a plotting library. For
Python, the most commonly used plotting library is Matplotlib. We will discuss Matplotlib in
greater detail in a section further on in the lab.

```python
import matplotlib.pyplot as plt

plt.figure()
plt.plot(steps, walk)
plt.title("Random walks")
plt.ylabel("distance (d)")
plt.xlabel("steps (s)")
```

```
Text(0.5, 0, 'steps (s)')
```

Random walks