Problem set 1

Name: [TODO]

Link to your PS1 github repo: [TODO]

Create your DATA1030 environment (10 points)

Before you start this homework assignment, please watch Isabel Restrepo's guest lecture on reproducable data science (available here). She is the Assistant Director of Research Software Engineering and Data Science at the Center for Computation and Visualization at Brown. She covers state of-the-art and industry-standard techniques to make your software, your data, and your workflows reproducable. She discusses the importance of github and conda, two tools we will use in DATA1030 but she covers additional tools that you might use later on during your internships.

Please follow the instructions outlined in this google doc and create your DATA1030 coding environment. We recommend that you use conda but if you are more familiar with other package managers (like docker, homebrew, poetry), feel free to use those. However, please note that the TAs might not be able to help if you do not use conda. The most important thing is to install the packages with their versions as shown in the data1030.yml file of the course's github repository.

Once you are done, run the cell below. If your environment is correctly set up, you'll see 9 green OK signs.

Once you solve the python coding exercises below, please follow the submission instructions in the google doc to submit your problem set solution.

```
from __future__ import print_function
from packaging.version import parse as Version
from platform import python_version

OK = '\x1b[42m[ OK ]\x1b[0m'
FAIL = "\x1b[41m[FAIL]\x1b[0m"

try:
    import importlib
except ImportError:
    print(FAIL, "Python version 3.12.10 is required,"
                " but %s is installed." % sys.version)

def import_version(pkg, min_ver, fail_msg=""):
    mod = None
    try:
        mod = importlib.import_module(pkg)
```

```
        if pkg in {'PIL'}:
            ver = mod.VERSION
        else:
            ver = mod.__version__
        if Version(ver) == Version(min_ver):
            print(OK, "%s version %s is installed."
                  % (lib, min_ver))
        else:
            print(FAIL, "%s version %s is required, but %s installed."
                  % (lib, min_ver, ver))
    except ImportError:
        print(FAIL, '%s not installed. %s' % (pkg, fail_msg))
    return mod


# first check the python version
pyversion = Version(python_version())

if pyversion >= Version("3.12.10"):
    print(OK, "Python version is %s" % pyversion)
elif pyversion < Version("3.12.10"):
    print(FAIL, "Python version 3.12.10 is required,"
                " but %s is installed." % pyversion)
else:
    print(FAIL, "Unknown Python version: %s" % pyversion)


print()
requirements = {'numpy': "2.2.5", 'matplotlib': "3.10.1",'sklearn':
"1.6.1",
                'pandas': "2.2.3",'xgboost': "3.0.0", 'shap':
"0.47.2",
                'polars': "1.27.1", 'seaborn': "0.13.2"}

# now the dependencies
for lib, required_version in list(requirements.items()):
    import_version(lib, required_version)
```

```
[ OK ] Python version is 3.12.10

[ OK ] numpy version 2.2.5 is installed.
[ OK ] matplotlib version 3.10.1 is installed.
[ OK ] sklearn version 1.6.1 is installed.
[ OK ] pandas version 2.2.3 is installed.
[ OK ] xgboost version 3.0.0 is installed.
[ OK ] shap version 0.47.2 is installed.
[ OK ] polars version 1.27.1 is installed.
[ OK ] seaborn version 0.13.2 is installed.
```

I am so happy I finally works!

Python coding questions (30 points total)

Problem 1a (5 points)

This is a live coding interview question I got during a job interview.

Write a function which takes a number as input, and it returns the number of unique digits in it.

If the input is 1, the output is 1.

If the input is 10, the output is 2.

If the input is 11, the output is 1.

If the input is 123, the output is 3.

If the unpit is 555, the output is 1.

We'll be asking you to create tests throughout this class. This is the first time we use functions and tests, so a starter code is provided below.

```
# function
def count_unique_digits(number):
    # count the number of unique digits and update the unique_digits
integer
    unique_digits = 0
    # ADD YOUR CODE BELOW
    # so "-" is not a digit as far as I know, but with method I come
up (making number a string and finding its lengh, it WILL count "-")4
    # so to solve this I will use abs() to get rig of minuses
    # ow number can also be float with "." then
    #I think another approach is better (I got stuck in previous one)
    text = str(number)
    digits = [i for i in text if i.isdigit()]
    unique_digits = len(set(digits))
    #why set? it gas a feature of leaving just unique elements so it
is handy
    return unique_digits

count_unique_digits(-555.584)
```

```
3
```

```
# tests
tests = { 1:1, 10:2, 11:1, 123:3, 555:1 }

for test in tests.items():

    if count_unique_digits(test[0]) == test[1]: #We go through each
test and check if the count_unique_digits function returns the correct
output
```

```
        print('correct!')
    else:
        print('incorrect!')
        print('if the input is '+str(test[0])+', the correct output is
'+str(test[1]))
```

```
correct!
correct!
correct!
correct!
correct!
```

Problem 1b (10 points)

Most people become biased when they see the test cases in 1a and they only consider non-negative integers while writing the code. However numbers can be negative and/or floats as well. This is perfectly fine and the interviewer will bring up the special cases and you will be asked to revise your solution. Keep in mind, we are still only counting unique *digits*.

Generate additional tests that contain at least one example of all special cases. Revise your function and apply it to the 1a and 1b test cases.

```
#code taken from above
tests = { -5:1, 5.159:3,-111.5697852:7, 89:2 }

for test in tests.items():

    if count_unique_digits(test[0]) == test[1]: #We go through each
test and check if the count_unique_digits function returns the correct
output
        print('correct!')
    else:
        print('incorrect!')
        print('if the input is '+str(test[0])+', the correct output is
'+str(test[1]))
```

```
correct!
correct!
correct!
correct!
```

Problem 2

Here is another typical live coding interview problem.

You are climbing a staircase with n steps.

Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

If n = 2, there are two ways to climb to the top:

- 1 step + 1 step
- 2 steps

If n = 3, there are three ways to climb to the top:

- 1 step + 1 step + 1 step
- 2 steps + 1 step
- 1 step + 2 steps

Let's assume that n is not too large: it is less than or equal to 30.

Problem 2a (5 points)

Work it out below in the markdown cell what the correct solution is for n = 4, 5, and 6. Follow the format of the problem 2 description above. What do you notice about the number of steps and the number of distinct ways?

Well, alrighjt for n=4

1. 1+1+1+1
2. 2+1+1
3. 1+2+1
4. 1+1+2
5. 2+2

for n=5

1. 1+1+1+1+1
2. 2+1+1+1
3. 1+2+1+1
4. 1+1+2+1
5. 1+1+1+2
6. 2+2+1
7. 2+1+2
8. 1+2+2

for n=6

1. 1+1+1+1+1+1
2. 2+1+1+1+1
3. 1+2+1+1+1
4. 1+1+2+1+1
5. 1+1+1+2+1
6. 1+1+1+1+2
7. 2+2+1+1
8. 2+1+2+1
9. 2+1+1+2
10. 1+2+2+1

11. 1+2+1+2
12. 1+1+2+2
13. 2+2+2

So what I hAVE n=1 - 1 n=2 - 2 n=3 - 3 n=4 - 5 n=5 - 8 n=6 - 13 Looks like a Fibonacci thing Where each next number equals to sum of two previous ones

Problem 2b (10 points)

Write a function and test it for n = 2 to 6. Follow the code format of Problem 1a (function at the top, iterate through the test cases below). Additionally, print out the solutions for n = 10, 15, and 30.

```
# it. was. hard. maybe my variant is not elegant, and certainly not
fast when we have big number for stairs but for 30 it will do
def n_stairs_ways(n):
    #n will be the number fo stairs, function will (hopefully) return
the number of unique ways to climb them, by 1 and 2-stair steps
    # ow user can write wrong number of stairs
    if not isinstance(n, int) or n<0:
        print("Only natural numbers are accepted")
        return None
    #IT DOESNT WORK!! Unless I start the range from 2, Why well
sequence begis with 1, 1, 2...
    # but our "ways sequence" has only one "1"
    # so it shifts by two places if we put 0 as a start of range,
giving n=5 -21
    #if i start range with 1, n=5 - 13
    # so I do 2, to have correct variant n=5 - 8
    #  i will write cases of n=1 or 0 manually:
    if n == 0:
        return 1
    # why 1 well there is only one way to do nothing
    if n == 1:
        return 1
    #here we go
    a, b = 1, 1
    for i in range(2, n+1):
        a, b = b, a+b
        #so a like "jumps" to be next b, and b is sum of a and b, just
like Fibonacci wanted
    return b


n_stairs_ways(9)
```

```
55
```

```
#i took test code from above
tests = { 0:1, 1:1, 2:2, 3:3, 4:5, 5:8, 6:13, 7:21, 8:34, 9:55, 10:89,
```

```
-234:None}
for test in tests.items():

    if n_stairs_ways(test[0]) == test[1]: #We go through each test and
check if the count_unique_digits function returns the correct output
        print('correct!')
    else:
        print('incorrect!')
        print('if the input is '+str(test[0])+', the correct output is
'+str(test[1]))
```

```
correct!
correct!
correct!
correct!
correct!
correct!
correct!
correct!
correct!
correct!
correct!
Only natural numbers are accepted
correct!
```