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 (https://brown.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=728c2a46-af2e-471c-a97f-ac55010de8c3)). She is the Assistant Director of Research Software Engineering and Data Science at the Center for Computation and Visualization (https://ccv.brown.edu/) 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 (https://docs.google.com/document/d/1Q9qZLIU2ePRiatnLSWY_BNKaLg7kwPkkkGQnErviokM/edit?usp=sharing) 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 (https://github.com/BrownDSI/DATA1030-Fall2023).

Once you are done, run the cell below. If your environment is correctly set up, you'll see 6 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.

In [2]:

```
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 3.10 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("3.11.4"):
    print(OK, "Python version is %s" % pyversion)
elif pyversion < Version("3.11"):
    print(FAIL, "Python version 3.11 is required,"
                " but %s is installed." % pyversion)
else:
    print(FAIL, "Unknown Python version: %s" % pyversion)


print()
requirements = {'numpy': "1.24.4", 'matplotlib': "3.7.2",'sklearn': "1.3.0",
                'pandas': "2.0.3",'xgboost': "1.7.6", 'shap': "0.42.1", 'seaborn': "0.12.2"}

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

```
[ OK ] Python version is 3.11.4
[ OK ] Python version is 3.11.4

[ OK ] numpy version 1.24.4 is installed.
[ OK ] matplotlib version 3.7.2 is installed.
[ OK ] sklearn version 1.3.0 is installed.
[ OK ] pandas version 2.0.3 is installed.
[ OK ] xgboost version 1.7.6 is installed.
[ OK ] shap version 0.42.1 is installed.
[ OK ] seaborn version 0.12.2 is installed.
```

Python coding questions (30 points)

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.

This is the first time we use functions and tests so a starter code is provided below.

In [3]:

```
import numpy as np
# function
def count_unique_digits(number):
    # count the number of unique digits and update the unique_digits integer
    # unique_digits = 0
    n =len(str(number))
    digits = np.zeros(n)
    for i in range(0,n):
      digits[i] = str(number)[i]
    unique_digits = len(np.unique(digits))
    return unique_digits

# 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]:
        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.

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.

In [4]:

```
import numpy as np
# function
def count_unique_digits(number):
    # count the number of unique digits and update the unique_digits integer
    # unique_digits = 0
    str_1 = str(number).replace('-','') #remove minus sign --(Notice: It also applies to positive number
    str_2 = str_1.replace('.','') #remove decimal point --(Notice: It also applies to integers)
    n =len(str_2)
    digits = np.zeros(n)
    for i in range(0,n):
      digits[i] = str_2[i]
    unique_digits = len(np.unique(digits))
    return unique_digits
```

In [5]:

```python
# tests
tests = { 1:1, 10:2, 11:1, 123:3, 555:1, -11:1, 11.1:1, -11.1:1, -123.45:5, -50.5:2 }

for test in tests.items():

    if count_unique_digits(test[0]) == test[1]:
        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!
```

Problem 2

Here is another typical live coding interview problem.

You are climbing a staircase. It takes n steps to reach the top.

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 a 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?

If n = 4, there are five ways to climb to the top:

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

If n = 5, there are eight ways to climb to the top:

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

If n = 6, there are thirteen ways to climb to the top:

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

Actually: when n = 2, the number of ways to climb to the top equals to:

$$C_2^0 + C_1^1 = 2$$

when n = 3, the number of ways to climb to the top equals to:

$$C_3^0 + C_2^1 = 3$$

when n = 4, the number of ways to climb to the top equals to:

$$C_4^0 + C_3^1 + C_2^2 = 5$$

when n = 5, the number of ways to climb to the top equals to:

$$C_5^0 + C_4^1 + C_3^2 = 8$$

when n = 6, the number of ways to climb to the top equals to:

$$C_6^0 + C_5^1 + C_4^2 + C_3^3 = 13$$

So it is not difficult to find out that:

If n is an even number, then the number of ways to climb to the top should subject to:

$$M = C_n^0 + C_{n-1}^1 + C_{n-2}^2 + \cdots + C_{{}^{n}/_{2}}^{{}^{n}/_{2}} = \sum_{i=0}^{{}^{n}/_{2}} C_{n-i}^i$$

If n is an odd number, then the number of ways to climb to the top should subject to:

$$M = C_n^0 + C_{n-1}^1 + C_{n-2}^2 + \cdots + C_{{}^{n+1}/_{2}}^{{}^{n-1}/_{2}} = \sum_{i=0}^{{}^{n-1}/_{2}} C_{n-i}^i$$

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.

In [6]:

```
import math
# function
def count_ways(n):
    # count how many ways we could use to climb up to the top
    # if we could only climb either 1 step or 2 steps each time
    if type(n) != int:
        print("The input number 'n' is not a integer. Please input an integer!")
    else: # The function only works in situation where input number 'n' is an integer
        ways = 0
        if n% 2 == 0: # if n is an even number
            m = int(n/2)
            for i in range(0,m+1):
                ways = ways + math.comb(n-i,i)
        else: # if n is an odd number
            m = int((n-1)/2)
            for i in range(0,m+1):
                ways = ways + math.comb(n-i,i)
        return ways
```

In [7]:

```
# tests
tests = { 2:2, 3:3, 4:5, 5:8, 6:13 }

for test in tests.items():

    if count_ways(test[0]) == test[1]:
        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!
```

In [8]:

```
for i in (10,15,30):
    print('If n = %d, there are %d ways to climb to the top' % (i,count_ways(i)))
```

```
If n = 10, there are 89 ways to climb to the top
If n = 15, there are 987 ways to climb to the top
If n = 30, there are 1346269 ways to climb to the top
```