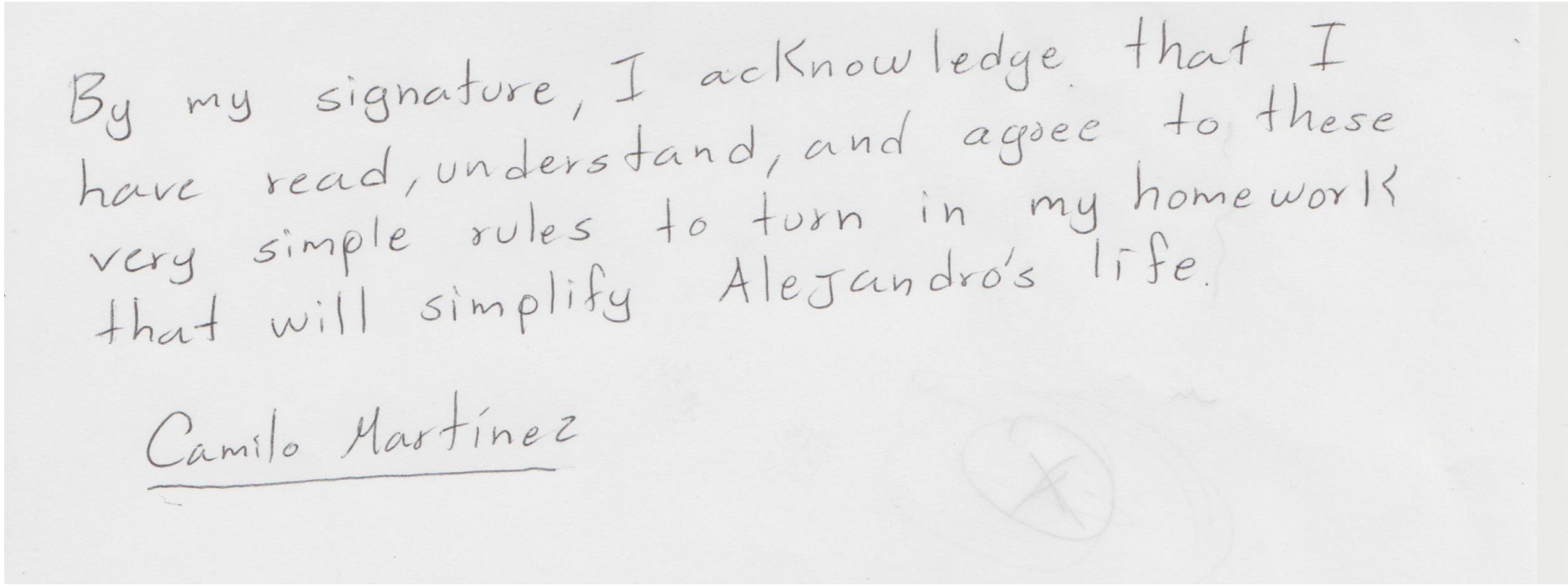
Programming

In [57]:
```
from IPython.display import Image
Image("signature.jpg")
```

Out[57]:



1.

In [47]:
```
import numpy as np

def obtain_lucas_numbers_by_an_n(n):
    L = np.zeros(n)
    L[0], L[1] = 1, 3

    for i in range(2, n):
        L[i] = L[i-1] + L[i-2]

    return L
```

2.

Answer:

If we take into account that around 10^8 operations last about 1 second, and that **obtain_lucas_numbers_by_an_n** is O(n), then we could make some aproximation that in one hour it can calculare about $3 * 10^{10}$ lucas numbers. That's just taking into account the time, however if it reach the maximum memory before completing the hour it would be less. For the purpose of the point, I built another funtion that doesn´t store the lucas numbers, instead it just return the last lucas number it could calculate along with the index of that number for a given amount of minutes. Also, because of my memory is not so big then I just can store about 10^4 numbers. Those are the ones of the plot I show below.

Github link of the repository (https://github.com/Andresmps/Simulation-class)

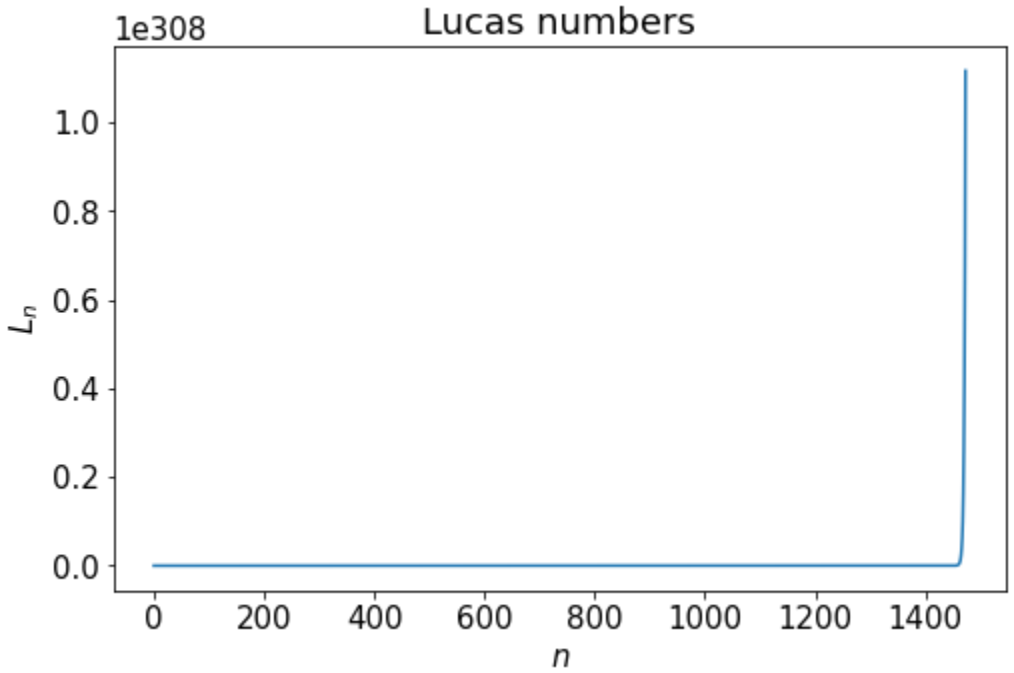
In [55]:
```
n = 10**5
lucas_numbers = obtain_lucas_numbers_by_an_n(n)
```

```
C:\Users\camil\anaconda3\lib\site-packages\ipykernel_launcher.py:8: RuntimeWarning: overflow encountered in double_scala
rs
```

In [56]:
```
plt.figure(figsize=(8, 5))
plt.rc('font', size=15)
plt.title('Lucas numbers')
plt.xlabel('$n$')
plt.ylabel('$L_n$')
plt.plot(lucas_numbers)
```

Out[56]:
```
[<matplotlib.lines.Line2D at 0x2c4ddd31348>]

C:\Users\camil\AppData\Roaming\Python\Python37\site-packages\matplotlib\ticker.py:2234: RuntimeWarning: overflow encount
ered in multiply
  steps = self._extended_steps * scale
```



In [34]:
```python
from datetime import datetime
import matplotlib.pyplot as plt


def obtain_lucas_numbers_by_given_minutes(minute):
    try:
        flag = True
        previos_lucas_number_1, previos_lucas_number_2 = [1, 3]
        start_time = get_minutes()
        i = 0

        while flag == True:
            temp = previos_lucas_number_1 + previos_lucas_number_2
            previos_lucas_number_1 = previos_lucas_number_2
            previos_lucas_number_2 = temp
            i += 1
            time = get_minutes()
            if time - start_time >= minute: break

        return previos_lucas_number_1, i+1
    except Exception as e:
        print(f"Error: {e}")
        print(f"Amount of Lucas numbers calculate: {len(L)}")
        return L


def get_minutes():
    time = datetime.now()
    return (time.hour * 60) + time.minute
```

In [35]:
```
%time

lucas_numbers = obtain_lucas_numbers_by_given_minutes(1) # Last Lucas number in one minute
lucas_numbers # (Lucas number, index)
```

```
623842916424530364704803374054002772769967499191453894099964303664164381851329080652290690447266432599738027843581412474207603465068391503521633545696943975708486330202773673510708364844054347945282876473969869854790320295644857386969794216868199784934404380613419093178572619229363598006147775648907028921859850617068440954569185586616474627397417045669216312241847863291387621950880759558965198830892816763323264959156879252187842943314434739559193254611754014768784263092118776979351543146099290437808188649230735680634441521935311824359363921647632418615393638462917132065076685185279817751068173726935259773662101525127769965169485066609450698755426404590373712328772416679774845331398629568453573694376058890959135734229258804505176221887528339490303863670304147625206978465177395884561458372414968748374374279746244478801466063970550608592170606492538551189952823930601468747275905975384695770431368181938418646416330113210115068305113119224231197531586174957467639925024169002451392993481828414741537510438946963201303975898936317910931954284778563769392235291010498697665541039596511283852997471251190076459101677507341850607654375476757490524719936876594060175570468724895166906873508848204786833341254994865325447187993442131293308352981877471205851282784580175434868338630078245117340450572098684842320251537683942796298161981904904046757595048944472698098753560355337403123841506752865677271561453740823181901968511875820813003024018793334922579567886937101071930934446824423405448677585400073621105080909250747338738261630169291401053727389151933701597318835540263154048723742719663596068779461421712946338978134199963534110212511640271137208609684394216642271917009859603219496703556587117933942203012036644605024840885802211416949054837221690914788082440992182042145850947406956821801672377898128034691378443148293556409733343195454282379724952643704083259410738164346071578709449011055346683381548006088098134964329621894454945964872395424214632577810612258615222533399253110192231430339493408497622167200125677350129024487942210793753463426120791702815960138211816303578817200451451539017115623875900299474790490463331564415697603520954522444045514955866877867364259737304911771057995081813873485446801331565704581767880
```

In [ ]:
```
```