Before you turn this problem in, make sure everything runs as expected. First, **restart the kernel** (in the menubar, select Kernel→Restart) and then **run all cells** (in the menubar, select Cell→Run All).

Make sure you fill in any place that says `YOUR CODE HERE` or "YOUR ANSWER HERE", as well as your name and collaborators below:

```
In [ ]: NAME = ""
        COLLABORATORS = ""
```

CSE 30 Spring 2022 - Homework 4

Instructions

Please disregard the YOUR NAME and COLLABORATORS above. They are put there atomatically by the grading tool. You can find instructions on how to work on a homework on Canvas. Here is a short summary:

Submitting your work

To submit your work:

- First, click on "Runtime > Restart and run all", and check that you get no errors. This enables you to catch any error you might have introduced, and not noticed, due to your running cells out of order.
- Second, download the notebook in .ipynb format (File > Download .ipynb) and upload the .ipynb file to this form. This homework is due by **11:59pm on Thursday 14 April 2022**.

You can submit multiple times; the last submission before the deadline is the one that counts.

Homework format

For each question in this notebook, there is:

- A text description of the problem.
- One or more places where you have to insert your solution. You need to complete every place marked:

  ```
  # YOUR CODE HERE
  ```

and you should not modify any other place.

- One or more test cells. Each cell is worth some number of points, marked at the top. You should not modify these tests cells. The tests pass if no error is printed out: when there is a statement that says, for instance:

  ```
  assert x == 2
  ```

  then the test passes if `x` has value 2, and fails otherwise. You can insert a `print(x)` (for this case!) somewhere if you want to debug your work; it is up to you.

Notes:

- Your code will be tested both according to the tests you can see (the `assert` statements you can see), *and* additional tests. This prevents you from hard-coding the answer to the particular questions posed. Your code should solve the *general* intended case, not hard-code the particular answer for the values used in the tests.

- **Please do not delete or add cells!** The test is autograded, and if you modify the test by adding or deleting cells, even if you re-add cells you delete, you may not receive credit.

- **Please do not import modules that are not part of the standard library.** You do not need any, and they will likely not available in the grading environment, leading your code to fail.

- **If you are inactive too long, your notebook might get disconnected from the back-end.** Your work is never lost, but you have to re-run all the cells before you continue.

- You can write out print statements in your code, to help you test/debug it. But remember: the code is graded on the basis of what it outputs or returns, not on the basis of what it prints.

- **TAs and tutors have access to this notebook,** so if you let them know you need their help, they can look at your work and give you advice.

Grading

Each cell where there are tests is worth a certain number of points. You get the points allocated to a cell only if you pass *all* the tests in the cell.

The tests in a cell include both the tests you can see, and other, similar, tests that are used for grading only. Therefore, you cannot hard-code the solutions: you really have to solve the essence of the problem, to receive the points in a cell.

Code of Conduct

- Work on the test yourself, alone.

- You can search documentation on the web, on sites such as the Python documentation sites, Stackoverflow, and similar, and you can use the results.
- You cannot share your work with others or solicit their help.

Problem 1

Write a generator for the Fibonacci Numbers

Build a generator that returns the Fibonacci numbers: 0, 1, 1, 2, 3, 5, and so on.

In [4]:
```
def fibonacci_generator():
    """Generates all Fibonacci numbers."""
    # YOUR CODE HERE
    num1 = 0
    num2 = 1
    while True:
      yield num1
      num3 = num1
      num1 = num2
      num2 = num3 + num2
```

In [5]:
```
## Here you can test your code.
r = []
for n in fibonacci_generator():
    r.append(n)
    if n > 100:
        break
print(r)
```

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144]
```

In [6]:
```
# Visible tests

r = []
for n in fibonacci_generator():
    r.append(n)
    if n > 100:
        break
assert r == [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144]
```

In [7]:
```
# Hidden tests, 10 points

# These tests check basically that you generate Fibonacci numbers, forever -- witho
# It is the reader (the user) of the function that decides when to stop.
```

Problem 2

Write a prime number generator

Write a generator that returns all the prime numbers. The idea is to loop over all positive integers, test each one to see if it is prime, and if it is, `yield` it.

In [21]:
```
# My solution is simple and not particularly optimized,
# and it is 12 lines long.

def prime_number_generator():
    """This generator returns all prime numbers."""
    # YOUR CODE HERE
    num = 2
    while True:
        for i in range(2, num):
          if num % i == 0:
            break
        else:
          yield num
        num += 1
```

In [22]:
```
## Here you can test your code.
## What you write here will be removed.
prime_number_generator()
```

Out[22]:
```
<generator object prime_number_generator at 0x7f3b125792d0>
```

In [23]:
```
## Visible tests

initial_primes = [2, 3, 5, 7, 11, 13, 17, 19, 23]

idx = 0
for p in prime_number_generator():
    assert p == initial_primes[idx]
    idx += 1
    if idx == len(initial_primes):
        break
```

In [24]:
```
## Hidden tests: 10 points

# We test that the generator goes on and on and on.
```