2. **SPARC to Python**

Consider the following SPARC code of the Fibonacci sequence, which is the series of numbers where each number is the sum of the two preceding numbers. For example, 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610 ...

$$foo\ x =$$
$$\text{if } x \leq 1 \text{ then}$$
$$x$$
$$\text{else}$$
$$\text{let } (ra, rb) = (foo\ (x-1))\ ,\ (foo\ (x-2))\ \text{ in}$$
$$ra + rb$$
$$\text{end}.$$

- 2a. (6 pts) Translate this to Python code – fill in the `def foo` method in `main.py`

- 2b. (6 pts) What does this function do, in your own words?

.

.

.

.

.

.

.

.

3. **Parallelism and recursion**

Consider the following function:

```
def longest_run(myarray, key)
    """
    Input:
       `myarray`: a list of ints
       `key`: an int
    Return:
       the longest continuous sequence of `key` in `myarray`
    """
```

E.g., `longest_run([2,12,12,8,12,12,12,0,12,1], 12) == 3`

- 3a. (7 pts) First, implement an iterative, sequential version of `longest_run` in `main.py`.

- 3b. (4 pts) What is the Work and Span of this implementation?

.

.

The work is O(n) and the span is also O(n).

.

- 3c. (7 pts) Next, implement a `longest_run_recursive`, a recursive, divide and conquer implementation. This is analogous to our implementation of `sum_list_recursive`. To do so, you will need to think about how to combine partial solutions from each recursive call. Make use of the provided class `Result`.

- 3d. (4 pts) What is the Work and Span of this sequential algorithm?

  Work recurrence: $W(n) = 2W(n/2) + 1$

  Span recurrence: $S(n) = 2S(n/2) + 1$

  Span and Work are both $O(n)$.

- 3e. (4 pts) Assume that we parallelize in a similar way we did with `sum_list_recursive`. That is, each recursive call spawns a new thread. What is the Work and Span of this algorithm?

  For parallel,
  Work recurrence: $W(n) = 2W(n/2) + 1$
  Span recurrence: $S(n) = S(n/2) + 1$
  Work is $O(n)$ and span is $O(\log n)$.