

ENGG1003 - Lab 3

Brenton Schulz

- arrays, indexing, slicing
- for loops
- boolean expressions / relational operators
- if statements
- while loops

Task 1: Pre-Lab Reading

Read Sections 2.1 and 2.2 of the textbook: https://link.springer.com/chapter/10.1007/978-3-030-16877-3_2

These sections provide general background information which will help you write Python scripts with confidence. The content is best learned “by immersion”. All the details covered in these sections will be constantly used throughout your programming career.

1 Arrays

Task 2: Array Background Reading

Read Section 2.3 of the textbook, stopping at 2.3.6. Execute examples as you go.

Direct link: https://link.springer.com/chapter/10.1007/978-3-030-16877-3_2#Sec16

You are welcome to read 2.3.6 (regarding 2D arrays) but that content will be covered later.

Task 3: Fibonacci Sequence - Naive Implementation

The Fibonacci sequence is a sequence of numbers, x_0, x_1, x_2, \dots etc, with the following equation used to calculate x_n given x_{n-1} and x_{n-2} :

$$x_n = x_{n-1} + x_{n-2} \quad (1)$$

Write a Python script which, given $x_0 = 1$ and $x_1 = 1$, calculates and prints the next 8 values of the Fibonacci sequence.

To do this, create a NumPy array, `fib[]` containing 10 zeros, manually assign the above 1's to `fib[0]` and `fib[1]`, then write out the equation as follows for the next 8 values:

```
fib[2] = fib[1] + fib[0]
fib[3] = fib[2] + fib[1]
... etc
print(fib)
```

Note that there is a far more efficient method using *loops*. This will be explored later.

2 for Loops

As seen in the previous section computers are able to perform the same calculation repeatedly but doing so with multiple copies of the expression is tedious and error prone.

The `for` loop is a programming *flow control* concept where a block of code is repeated a set number of times. Typically the number of repeats must be known at the time the loop executes.

Task 4: for Loops - Reading

Read Section 3.1 of the textbook: https://link.springer.com/chapter/10.1007/978-3-030-16877-3_3#Sec1

Task 5: Fibonacci Sequence with a for Loop

Modify your Fibonacci sequence script to utilise a `for` loop and the `range()` function. Note that by utilising a `for` loop you now only need to write out Equation 1 *once*, irrelevant of how many values you wish to calculate.

A few notes & tips:

- Try to use a single variable `N` which specifies how many values to calculate
- Since the first calculation is giving the 3rd value the `range()` function needs to be called as `range(2, N)`.
- If `N` is large (more than about 90) care must be taken with the choice of data type. `np.zeros()` will, by default, create `np.float64s` but the Fibonacci sequence is intrinsically an *integer* sequence. Experiment with different datatypes specified in the call to `np.zeros()`. eg: `fib = np.zeros(N, dtype=np.uint64)`

The full list of NumPy datatypes is here: <https://numpy.org/devdocs/user/basics.types.html>. How many terms can you calculate before an “overflow” error with `uint8`, `uint32`, and `uint64`?

- `print(fib)` should print the entire array but you can call `print()` from within the loop so that only a single value is printed on each line.

3 while Loops

Contrasting with a `for` loop the `while` loop allows a block of code to be executed multiple times without the need for the repetition count to be known when the loop starts.

Task 6: while Loops - Reading

Read Section 3.2 of the textbook: https://link.springer.com/chapter/10.1007/978-3-030-16877-3_3#Sec7

Task 7: Fibonacci Sequence with while Loops

Fork your Fibonacci sequence code (ie: save a copy of it so it can be loaded later).

Using a `while` loop, implement a Fibonacci sequence generator which prints the Fibonacci sequence until the printed value exceeds 1 million.

You will need to compare the last printed value with the constant 1000000. Assuming arrays are still used, you will also need to manually increment (and initialise) the variable used to index the array.

Task 8: Fibonacci Sequence Without Arrays

Modify your code so that instead of using an array it calculates the sequence using only 3 variables:

- `xn` - The current value
- `xnm1` - The previous value, x_{n-1}
- `xnm2` - The value of x_{n-2}

The calculation will have to happen in two steps:

1. Calculate the current value, x_n
2. “Move forward in time” by executing `xnm2 = xnm1` and `xnm1 = xn`.

You may use `for` or `while` loops to complete this task.

Note that this implementation has the advantage of using *significantly* less RAM than the array-based version. The disadvantages are that you must print each value as it is calculated and the code is somewhat less “readable” - it looks less like Equation 1 than the array based versions.