CMPT 423/820

Assignment 1 Question 3 -- Solutions and Grading

In this question, we'll introduce Numpy, a very powerful and popular Python module for numerical computation.

Numpy adds a very fast, very flexible data structure called NDArray to our tool-box. A NDArray stores numerical data much more efficiently than Python lists, and its methods are optimized for array-level tasks. The powerful thing about Numpy is that you can express array-level calculations with simple algebraic expressions, and usually without loops. The Numpy module also provides very useful libraries for array-based and matric-based computations.

In [1]:
```
# almost always imported like this
import numpy as np
```

Task 1

The Numpy module includes libraries for numerical calculations. There are array-based calculations, and matrix-based calculations. For example, the cell below defines a small 2×2 array, and then does two kinds of multiplication.

In [2]:
```
# a small array
twobytwo = np.array([[1, 2], [3, 4]])

# array-based
array_product = 2 * twobytwo * twobytwo

# matrix-based
matrix_product = 2 * np.matmul(twobytwo, twobytwo)

# print them both
print(array_product)
print(matrix_product)
```

```
[[ 2  8]
 [18 32]]
[[14 20]
 [30 44]]
```

Task 1 Model Solution:

1. (Easy) Explain the difference between the two products.

> The `matmul()` function does matrix multiplication, but the `*` operator does element-wise multiplication. A deeper answer might define the difference using mathematics, or an example. The point is to recognize that there is a difference, and to be careful to choose the one you need.

2. (Deeper) Python uses the multiplication operator `*` for normal Python numbers, but here we see it being used for NDArrays. How does Python know which kind of multiplication to do?

> All data values in Python are *objects*, even numbers! So every kind of operation we perform on some data is transformed to a method call from the appropriate class. When Python sees `a * b`, it simply calls a multiply method on `a`. Python doesn't even care what class `a` is. It better have a multiply method, or else a runtime error results. So, all numbers have a multiply method, and Numpy implements a multiply method for NDArrays too. Each class knows what `*` means for it. In Python, the mu;tiply method is named `__mult__()`.

Task 2. Scripts using NDArrays

Here's a cute little result. It turns out you can calculate Fibonacci numbers using matrix multiplication. Suppose we want the Nth Fibonacci Number, F_N:

$$\begin{pmatrix} F_{N+1} \\ F_N \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^N \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

Task 2 Model Solutions

1. A model solution is below.
2. In Question 2, we could calculate F_N for very large N, and it would be exact. Explain what happens if we try to do the same thing here. Answer by discussing what your script produces for large N, and then explain why it's wrong.

> For relatively small `n`, we obtain the same answer as in A1Q2. However, for larger `n`, the code below returns the wrong answer. For example, using Numpy for `n=111` returns a value that is 5 orders of magnitude too small. i.e., about 10^5 too small. The reason is that Numpy does not use Python integers. It uses standard 64-bit integers. If a calcualtion would result in an integer requiring more than 64-bits, the result is squeezed into 64 bits by *chopping off the most significant digits*. Because of the way 64-bit integers work, chopping the value like this sometimes leaves the data looking like a negative number. In summary, Numpy integers are finite precision, even though Python integers are infinite precision.

In [8]:
```
n = 30

# a really cute result: This fibarray can calculate Fibonacci numbers!
fibarray = np.array([[1, 1], [1, 0]])

# but we need two copies
u = fibarray.copy()

# now raise fibarray to the nth power
for i in range(n-1):
    u = np.matmul(u, fibarray)

# ta-daa!
print(u)
print('The', n, 'th Fibonacci number is', u[0,1])
```

```
[[1346269  832040]
 [ 832040  514229]]
The 30 th Fibonacci number is 832040
```

What to hand in

Your version of this notebook named A1Q3.pdf, containing completed work above, and your name and student number at the top.

Evaluation:

- 1 mark: Your answer two Task 1 Part 1 was correct.
- 2 marks: Your answer two Task 1 Part 2 was correct.
- 4 marks: Your code cell for Task 2 Part uses Python and Numpy to calculate the 30th Fibonacci number correctly.
- 3 marks: Your explanation for the behaviour of this script for large N is correct.

Grading:

- **Task 1 Part 1**: 1 mark.
  - Any answer that distinguishes between matrix and array multiplication is fine.
  - Deduct the mark only if the answer is very wrong, or missing.
- **Task 1 Part 2**: 2 marks.
  - Any answer that suggests that different classes have different implementations for the same syntactic operator is acceptable.
    - Describing Python operator over-loading is optional
    - Saying that the objects know how to multiply is fine.
    - This is not a course in OOP concepts; the value here is in noticing that the same operator is used differently.
  - Deduct the marks only if the answer is very wrong, or missing.
- **Task 2**: 4 marks. The script needs to:
  - Define an array with the right values in it
  - Use `matmul()` `n` times to multiply obtain the correct result.
  - No part marks here.
- **Task 3**: 3 marks. For full marks:
  - The explanation has to mention that Numpy uses integers that are limited in range (finite precision)
  - No need to explain how they work, only that large values get chopped off.
  - Give only one mark if the answer is not complete.