Algorithms

The contemporary term "algorithm" derives from the proper name of a Persian polymath who worked out of the Wisdom School in Baghdad over a thousand years ago. Iraq had not yet been invented as a country back then.

His name in Persian was: محمد بن موسی خوارزمی, which gets Latinized as Muḥammad ibn Mūsā al-Khwārizmī (c. 780 – c.850), or Al Khwarizmi for short, which sounds a little bit like "Al Gorithm" (but not a lot). "Algorithm" sounds a lot more like "Al Gore" (a joke, why? See note [1] below).

But what is an "algorithm" anyway?

An algorithm is a step-by-step numerical recipe, something to follow.

The idea of "numerical recipes" as at the basis of arithmetic and is what we learn in school as addition, subtraction, multiplication and division. When school teachers talk about elementary school math, they talk about teaching algorithms, such as how to perform long division, or how to find a square root.

This idea of "numerical recipes" spread to the Italian peninsula and to the rest of Europe, through the Republic of Pisa, where the famous Leonardo Fibonacci (c.1170 - c. 1250) wrote *Liber Abaci*, based on Al Khwarizmi's work.

If you learned to use Arabic numerals, Fibonacci told his readers, you could get free of relatively clunky Roman numbers and the abacus. You could learn to compute on paper or on a chalkboard.

You might be thinking "if algorithm means step-by-step number crunching then isn't any computer program an algorithm?"

That's pretty close to how people use the word nowadays. People talk about the "recommendation algorithm" used by online sellers to suggest other, related products you might want ot buy.

Fibonacci Numbers

You may have heard of this famous sequence of numbers, named for Fibonacci. Below is some Python code for generating the first ten terms in the sequence, starting from 0.

Don't worry if you're not familiar with the idea of a "generator function" in Python, which uses the `yield` keyword instead of `return` . We will be introducing features of the Python language as we go.

You could call this short program "an algorithm for generating the Fibonacci numbers". See Note [2] below.

In [1]:
```
# Fibonacci Numbers
# https://oeis.org/A000045

def fibo(a=0, b=1):
    while True:
        yield a
        b, a = a + b, b

gen = fibo()
fibs = [next(gen) for _ in range(10)]
print(fibs)
```

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

Data Structures

When you cook, following a recipe, you need ways to store you ingredients, as well as places to mix them. Containers of various types come to mind: liquid measuring spoons, sized cups, bags and jars. We also need cutting boards and other work surfaces.

Algorithms likewise require lots of data containers, named amounts, for organizing all the ingredients in a numerical recipe.

For example, in the Fibonacci Numbers code, we stored the sequence in a Python list named fibs. The Python list is a type of data structure. What are some other data structures in Python?

We will begin our study of data structures by using them within algorithms.

Python's data structures may be classified into two major categories: *sequences* and *mappings*.

Sequences

In [2]:
```
# List

L = [5, 2, 7, 0, 10]
```

In [3]:
```
L.reverse()  # some algorithm knows how to reverse any list
L
```

Out[3]:
```
[10, 0, 7, 2, 5]
```

In [4]:
```
L.sort()  # some algorithm knows how to sort some lists
L
```

Out[4]:
```
[0, 2, 5, 7, 10]
```

In [5]:
```
M = ['a', 10, True]  # but not all lists can be sorted
try:
    M.sort()
except Exception as e:
    print(e)
```

```
'<' not supported between instances of 'int' and 'str'
```

Any data structure that has a first-to-last ordering is a sequence. A list is a sequence. So is a string.

In [6]:
```
# all of these may be indexed
a_tuple = (1, 7, 9, 12)
a_list = [1, 7, 9, 12]
a_string = "1 7 9 12"
```

In [7]:
```
a_tuple[1] # tuples are like lists but fight being changed
```

Out[7]:
```
7
```

In [8]:
```
a_list[-1] # negative index:  coming from the right
```

Out[8]:
```
12
```

In [9]:
```
a_string[0:5]  # slice notation, still indexing
```

Out[9]:
```
'1 7 9'
```

Mappings

A mapping holds data, perhaps key:value pairs, as in a dict (dictionary), but has no special first-to-last order such that indexes might be used.

In [10]:
```
d = {"Fibonacci": (1170, 1250),
     "Al Khwarizmi": (780, 850),
     "Al Gore": (1948,)}
```

In [11]:
```
d.keys()
```

Out[11]:
```
dict_keys(['Fibonacci', 'Al Khwarizmi', 'Al Gore'])
```

In [12]:
```
d.values()
```

Out[12]:
```
dict_values([(1170, 1250), (780, 850), (1948,)])
```

In [13]:
```
d.items()
```

Out[13]:
```
dict_items([('Fibonacci', (1170, 1250)), ('Al Khwarizmi', (780, 850)), ('Al Gore', (194
8,))])
```

In [14]:
```
default = "Not Found"
d.get("Fibonacci", default)  # get method allows a default
```

Out[14]:
```
(1170, 1250)
```

In [15]:
```
d.get("Joe", default)
```

Out[15]:
```
'Not Found'
```

In [16]:
```
d["Al Khwarizmi"]  # direct access using the key
```

Out[16]:
```
(780, 850)
```

We will talk a lot more about the fine points of Python's data structures as the class progresses.

We will also look at data structures we need to build ourselves, but from built-in data structures.

Notes:

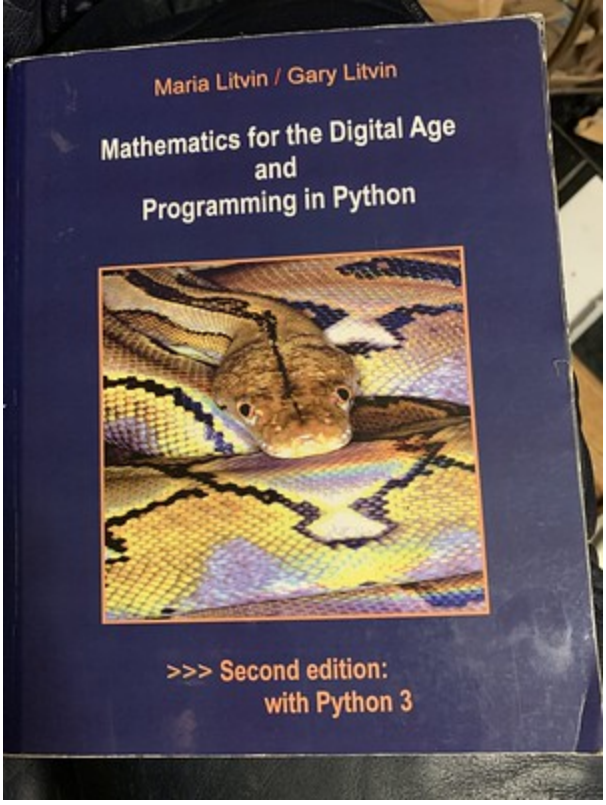
1. Politician Al Gore took credit for "inventing the internet" which many considered too much of an exaggeration, leading to many jokes at Al's expense.

2. Fibonacci Numbers are derived by adding the previous two terms to get the next term, starting from 0 and 1. The closely related Lucas Numbers start off with 2 and 1 instead of 0 and 1.

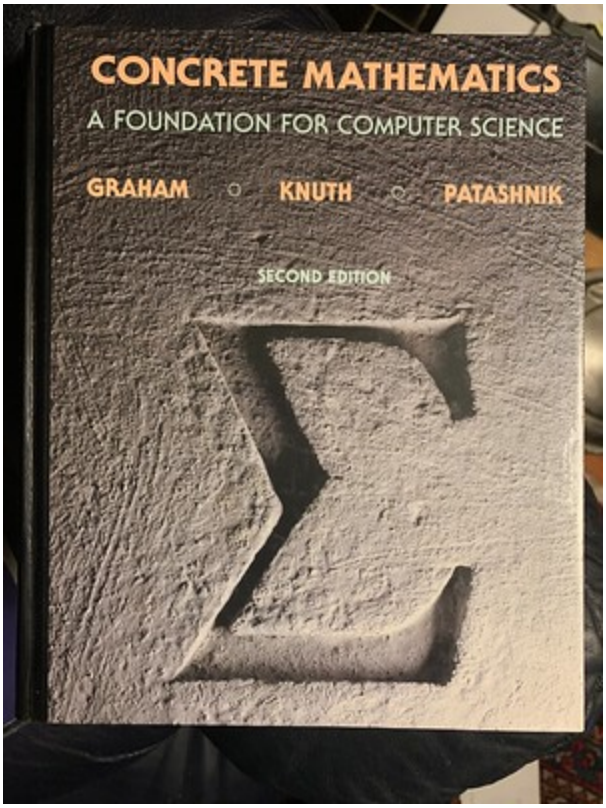
In [17]:
```
gen = fibo(2, 1)  # reusing the same generator, changing defaults
lucas = [next(gen) for _ in range(10)] # OEIS A000032
print(lucas)
```

```
[2, 1, 3, 4, 7, 11, 18, 29, 47, 76]
```

Collectible / Recommended Books:



Used at Phillips Academy / Andover. Introduces the cryptographic algorithm named RSA.



Used at Stanford University to help make *The Art of Computer Programming* more understandable.