An evil-genius guide to computer programming

Daniel Lemire

Montreal 🇨🇦

blog: https://lemire.me

twitter: @lemire

GitHub: https://github.com/lemire/

About me

- I have been programming for nearly 40 years.
- I have been bloging about programming for 20 years at https://lemire.me
- In the top 500 of most followed programmers on GitHub (out of millions).
- Editor of Software: Practice and Experience journal (founded in 1971, Wiley)

About the course

- Not a technical class
- Not about a programming language in partical
- Make you curious. Encourage you to dig deeper.
- Motivate you !

Slides/notes

You can simply request a copy of the slides. They will be available online.

Caveats

- If you have never programmed, and you merely follow six weeks of classes, you won't know much about programming.
- If you are an expert programmer, it is possible that much of what I cover will be known to you.

Cognitive stretch

- To get better, you have to be unconfortable.
- It pushes you to get better…
- … as long as you do not let yourself become demoralized.

Fun

Let us try to have fun !

Keep your mind open

Week 1

Learning or re-learning programming in 60 minutes.

What is programming?

- Not the use of a programming language.

What is programming?

- Processing information.

What is programming?

- The world is too complex.
- Programmers must 'manage complexity'.

Main tool of the programmer

- Abstraction

Caveat

All non-trivial abstractions, to some degree, are leaky.

(The Law of Leaky Abstractions, Spolsky)

Making matters worse

- Programming is social.

Python

- Simple and powerful
- Readable
- Not *very* fast (no SIMD, so-so multithreading)
- Python 3 (2008)

Python online

- https://replit.com

Python values

- numbers, strings, sets, lists...

Arithmetic

```
>>> 0.1+0.2+0.3+0.4
1.0
```

Comparing values

```
>>> 0.1 == 0.1
True
```

In most programming languages, `==` will compare two values and return true/false.

```
>>> 0.1 + 0.2 == 0.3
False
```

```
>>> 0.1+0.2
0.30000000000000004
```

$$0.1 \rightarrow 7205759403792793 \times 2^{-56}$$

0.1000000000000000555

$$0.2 \rightarrow 7205759403792794 \times 2^{-55}$$

0.2000000000000000111

$$0.3 \rightarrow 5404319552844595 \times 2^{-54}$$

0.2999999999999999888977697 5

At the beginning

- bits
- bytes
- machine words

Base ten (decimal)

0-9, 10-19, …

Binary notation

```
0 0b0
1 0b1
2 0b10
3 0b11
4 0b100
5 0b101
6 0b110
7 0b111
8 0b1000
9 0b1001
10 0b1010
11 0b1011
12 0b1100
13 0b1101
14 0b1110
15 0b1111
16 0b10000
17 0b10001
18 0b10010
19 0b10011
```

Binary notation

```
>>> bin(233)
'0b11101001'
```

Hexadecimal notation

```
0 0x0
1 0x1
2 0x2
3 0x3
4 0x4
5 0x5
6 0x6
7 0x7
8 0x8
9 0x9
10 0xa
11 0xb
12 0xc
13 0xd
14 0xe
15 0xf
16 0x10
17 0x11
18 0x12
19 0x13
```

Hexadecimal notation

```
>>> hex(233)
'0xe9'
```

Variables

It is useful to name values:

```
>>> x = 0.1
>>> print(x)
0.1
```

In most programming languages, `=` will assign a value to a variable.

Comparing variable values

```
>>> x = 0.1
>>> y = 0.1
>>> x == y
True
```

Comparing variable values

```
>>> x = "é"
>>> y = "é"
>>> x == y
True
```

Comparing variable values

```
>>> print(x)
é
>>> print(y)
é
>>> x == y
False
```

Comparing variable values

```
>>> "\u00e9"
'é'
>>> "\u0065\u0301"
'é'
```

Unicode

- Code point between 0 and 1114112.
- Each code point is stored using between 1 and 4 bytes (depending on the format, UTF-8, UTF-16, UTF-32).
- Multiple code points can be combined to create the same visual character.

Arrays

```
>>> x= [1,2,3]
```

Indexing

```
>>> x = [1,2,3]
>>> x[1]
2
```

Computers do not know about variables!

Computers only move bytes around.

The concept of a variable is an abstraction.

Two types of variables: values and references

```
>>> x = 0.2
>>> y = x
>>> x = 0.1
```

Simpler types are typically by value (copy)

```
>>> x = 0.2
>>> y = x
>>> x = 0.1
>>> y
0.2
```

Array variables are...

```
>>> x = [1,2,3]
>>> y = x
>>> x[0] = 0
```

Array variables are often references

```
>>> x = [1,2,3]
>>> y = x
>>> x[0] = 0
>>> x
[0, 2, 3]
>>> y
[0, 2, 3]
```

Maps and/or dictionaries

Key-value pair

```
>>> x = {}
>>> x[1] = 2
>>> x["abc"] = 3
>>> x
{1: 2, 'abc': 3}
```

Combining arrays and maps for fun

```
daniel = {
  "name": "Daniel Lemire",
  "age" : 51,
  "hobbies" : ["programming", "gaming", "netflix"]
}
```

Code as values: functions

A function takes some input and returns and output.

Code as values: function example

```
>>> def f(x):
...     return x + 1
...
>>> f(1)
2
```

Functions are like magical values

```
>>> g = f
>>> g(2)
3
```

Where is the data

- network
- disk
- memory
- cache
- register
- all of the above

Sending data (JSON)

```
daniel = {
  "name": "Daniel Lemire",
  "age" : 51,
  "hobbies" : ["programming", "gaming", "netflix"]
}
import json
with open( "datafile.json" , "w" ) as write:
  json.dump( daniel , write )
```

Reading data (JSON)

```
>>> with open( "datafile.json" , "r" ) as read:
...   json.load( read )
...
{'name': 'Daniel Lemire', 'age': 51, 'hobbies': ['programming', 'gaming', 'netflix']}
```

Grab JSON from the network

```
import json
from urllib.request import urlopen

def getjson(url):
  return json.loads(urlopen(url).read().decode("utf-8"))
```

Grab JSON from the network

Someone's first name is Blandine. Man or woman? What age approximately?

```
import json
from urllib.request import urlopen

def getjson(url):
  return json.loads(urlopen(url).read().decode("utf-8"))
```

Find someone's age and gender

Someone's first name is Blandine. Man or woman? What age approximately?

Guess Blandine's age

```
data = getjson("https://api.agify.io/?name=Blandine")
print(data)
print(data["age"])
```

```
{'name': 'Blandine', 'age': 40, 'count': 5522}
40
```

Guess Blandine's gender

```
data = getjson("https://api.genderize.io/?name=Blandine")
print(data)
print(data["gender"])
```

```
{'name': 'Blandine', 'gender': 'female', 'probability': 0.98, 'count': 5768}
female
```

https://replit.com/@lemire/QueasyAustereCalculators#main.py

Homework

Find out what JSON data you can find online.