Programming Languages

Closures-ish in Python and C++

Higher-order programming

- Higher-order programming, e.g., with **`map`** and **`filter`**, is great
- Language support for closures makes it very pleasant
- Without closures, we can still do it more manually / clumsily
  - In OOP (e.g., C++) with classes with private members
  - Python has closures, though they are slightly clunky
- Working through this:
  - Shows connections between languages and features
  - Can help you understand closures and objects

C++

- In the beginning, there was the function pointer.
  - Not a substitute for full-fledged closures, because a function pointer points to code only, no environment.

C++

- In the beginning, there was the function pointer.
  - Not a substitute for full-fledged closures, because a function pointer points to code only, no environment.
- Then came objects, capable of having "private data" by using private members.
  - People started using "function objects" by creating classes that overload the parentheses operator (yes, you can do that).

C++

- In the beginning, there was the function pointer.
  - Not a substitute for full-fledged closures, because a function pointer points to code only, no environment.
- Then came objects, capable of having "private data" by using private members.
  - People started using "function objects" by creating classes that overload the parentheses operator (yes, you can do that).
- In 2011, the C++ standard was updated to include "function objects" as well as lambdas.
  - This is super-awesome and really cool, but not all compilers support everything yet.

Lambda expressions in C++

[capture option] (args) { block of code }

C++ will try to automatically figure out the return type for you, but if it can't, you can also do:

[capture option] (args) -> returnType { block of code }

Lambda expressions in C++

[capture option] (args) { block of code }

The "capture option" tells C++ how to handle non-local variables in the block of code.

[ ] means capture nothing (if you have no non-locals)

[=] means capture non-locals by value (make a copy)

[&] means capture non-locals by reference (don't make a copy)

Map and filter

- Part of the C++ algorithms library
  - map is called "transform"
  - many of these higher-order functions don't use vectors or arrays directly, but operate on "iterators."
  - An iterator is a data type that expresses the abstraction of a "sequence," whether that sequence is implemented as a vector, array, or looping through some other data structure
  - other things that have iterators
    - strings (sequence of characters)
    - keys in a hashtable

Python

- Python has fewer hoops to jump through because the language was designed with closures in mind (e.g., nested function defs are OK).
  - C++'s "closures" (which aren't really closures) had to be hacked in later.
- Syntax is cleaner in Python as well because no capture by value/capture by reference, plus no data type declarations needed because it's a dynamically typed language.
  - Summary: syntax closer to Racket/Scheme's syntax.