First-Class Functions

What is functional programming?

"Functional programming" can mean a few different things:
1. Avoiding mutation in most/all cases (done and ongoing)
2. Using functions as values (this unit)
...
- Style encouraging recursion and recursive data structures
- Style closer to mathematical definitions
- Programming idioms using laziness (later topic, briefly)
- Anything not OOP or C? (not a good definition)

Not sure a definition of "functional language" exists beyond "makes functional programming easy / the default / required"
- No clear yes/no for a particular language

First-class functions

Functions are first-class values, can be used wherever we use values:
- Arguments to function calls
- Results of function bodies
- Parts of cons cells
- Bound to variables
- ...

Common!
- Other function is called a higher-order function
- Powerful way to factor out common functionality

Function Closures

Sneak peak:
- Function bodies can use bindings from outside the function definition (in scope where function is defined)
- Distinct concept from first-class functions.
- Back to this powerful idea soon!
Onward

- How to use first-class functions and closures
- The precise semantics
- Multiple powerful idioms

Functions as arguments

We can pass one function as an argument to another function
- Not a new feature!

\[
\text{define (do-to-both f pair)}
\]
\[
\text{(cons (f (car pair)) (f (cdr pair)))}
\]

Elegant strategy for factoring out common code.
Combines well with anonymous functions.
[See separate code examples]

A style point

Compare:

\[
\text{(if x #t #f)}
\]

With:

\[
\text{(lambda (x) (f x))}
\]

So don't do this:

\[
\text{(n-times (lambda (x) (cdr x)) 2 (list 1 2 3 4))}
\]

When you can do this:

\[
\text{(n-times cdr 2 (list 1 2 3 4))}
\]

Map

\[
\text{(define (map f xs)}
\]
\[
\text{(if (null? xs) null}
\]
\[
\text{(cons (f (car xs)) (map f (cdr xs))))}
\]

Map is, without a doubt, in the “higher-order function hall of fame”
- Standard name (for any data structure)
- You use it all the time once you know it:
  • Saves space.
  • Clearly communicates intent.
- Predefined, but just a function.
Filter

\[
\text{(define \(\text{filter} f\ xs\))}
\text{(if \(\text{null? xs}\))}
\text{null}
\text{(if \(f\ (\text{car xs})\))}
\text{(cons \(\text{car xs}\) \(\text{filter} f\ (\text{cdr xs})\))}
\text{(filter f (cdr xs)))}}
\]

Filter is also in the hall of fame
– So use it whenever your computation is a filter.
– Predefined, but just a function.

Generalizing

Our examples of first-class functions so far all:
– Take one function as an argument to another function
– Process a number or a list

But first-class functions are useful anywhere for any kind of data
– Can pass several functions as arguments
– Can put functions in data structures (tuples, lists, etc.)
– Can return functions as results
– Can write higher-order functions that traverse your own data structures

Useful whenever you want to abstract over “what to compute with”
– No new language features

Returning functions

Remember: Functions are first-class values
– For example, can return them from functions

Silly example:

\[
\text{(define (double-or-triple-if-f-of-seven f))}
\text{(if \(f\ 7\))}
\text{(lambda (x) (* 2 x))}
\text{(lambda (x) (* 3 x)))}}
\]