Discussion 05

Higher-Order Functions

Kenneth Fang (kwf37), Newton Ni (cn279)

Feb. 11, 2019

Agenda

1. Review Options (optional) ((pun intended))
2. Higher-Order Functions
3. Recitation 6 (Please pull it up as you enter)

(Review) Options

Options are a common variant type built into the language:

- `type 'a option = Some of 'a | None`

(Review) Options

Options are a common variant type built into the language:

- `type 'a option = Some of 'a | None`
- This is an example of "parametric polymorphism." The `option` type is paramaterized (hence parametric) on type `'a`, which can be any type (hence polymorphic).

(Review) Options

Options are a common variant type built into the language:

- `type 'a option = Some of 'a | None`
- This is an example of "parametric polymorphism." The `option` type is paramaterized (hence parametric) on type `'a`, which can be any type (hence polymorphic).
- Example:

```
let safe_division (a:int) (b:int) : int option =
  if b = 0 then None
  else Some (a / b)
```

(Review) Option Examples

- ```
  let safe_division (a:int) (b:int) : int option =
    if b = 0 then None
    else Some (a / b)
  ```
- ```
  let text_ui (a:int) (b:int) : unit =
    match safe_division a b with
    | Some x -> print_endline (string_of_int x)
    | None -> print_endline "Error: Division by zero"
  ```

Why Options?

- My take: conceptually, options exist everywhere!
- In Java, any pointer could be pointing to a real value or NULL.
- However, using options makes this **explicit**.
- If a value is wrapped in an Option, you are forced to pattern match on the None case, which means you *don't unexpected exceptions*.

What is a Higher-Order Function?

What is a Higher-Order Function?

- Functions take in values as input arguments

What is a Higher-Order Function?

- Functions take in values as input arguments
- Functions are values

What is a Higher-Order Function?

- Functions take in values as input arguments
- Functions are values
- If a function takes another function in as an input argument, it is called a "higher-order function"

The Simplest Example

```
/**
 * [apply] takes the function [f] and applies it to [a].
 */

let apply (a: 'a) (f: 'a -> 'b) : 'b =
```

The Simplest Example

```
/**
 * [apply] takes the function [f] and applies it to [a].
 */
let apply (a: 'a) (f: 'a -> 'b) : 'b = f a
```

The Simplest Example

```
/**
 * [apply] takes the function [f] and applies it to [a].
 */
let apply (a: 'a) (f: 'a -> 'b) : 'b = f a
```

`f` is a function, so [apply] is higher order!

But Why Tho

- Higher-order functions are a functional programming *idiom*, or commonly used pattern in code.

But Why Tho

- Higher-order functions are a functional programming *idiom*, or commonly used pattern in code.
- What are some Object-Oriented programming idioms? What do they do?

But Why Tho

- Higher-order functions are a functional programming *idiom*, or commonly used pattern in code.
- What are some Object-Oriented programming idioms? What do they do?
  - Encapsulation- hides unnecessary information to make code easier to understand
  - Inheritance- Used to share code between related classes

Examples using Higher-Order Functions

Factoring Out Code

These are some things I think about when trying to factor code with higher-order programming

- Each function does some different kinds of *computation*
- For example, to sum a list, your computation includes *iterating* through the list and *summing* the list elements.
- Higher-order programming can be used by abstracting the *iterating* part of the computation, and passing in a function that does the *summing* part.
- The abstracted *iterating* function is the **fold** function