Introduction to Functional Progamming

OOSD

School of Information Technology
Otago Polytechnic
Dunedin, New Zealand

What is object-oriented programming?

Typically prgramming can be thought of as managing *state*.

- We have some variables that hold values.
- Those values change over time as the program is executed.
- Our programs are sets of instructions on how to change state.
- The value of those variables at any point in time characterises the program's state.

What is object-oriented programming?

The difficulty with this type of programming in that it is often difficult to reason about a program's state.

- A bug in a program occurs when a variable doesn't hold the corect value. The program's state is incorrect.
- In OO programming, we use *objects* to encapsulate state.
- Objects protect their states and provide methods to control changes to their states.

What is functional programming?

The aim of functional programming is to allow us to program without state. Instead of supplying instructions about how to change state, our programs describe to desired solution in terms of the application of functions.

Functional programming principles

1. Immutable data: Once we assign a value to a variable we avoid changing that value, since that gets us back to managing state.

Functional programming principles

2. Pure functions: Pure just functions take arguments and return a values. They are *deterministic*, meaning they always return the same values when given the same arguments. Pure functions avoid *side effects*, since side effects generally involve state.

Functional programming principles

3. First class functions: Functions are just values that can be passed around just like we pass around an integer value in a program.

Functional programming principles

4. Higher order functions: We write fuctions that operate on other functions.

Example

```
def doubler(f):
    def tmp(x):
        return 2 * f(x)
    return tmp
```

Why bother with FP

- Functional programming facilitates very rapid development since fuctional code is typiclaly very expressive.
- Functional programs have fewer bugs and are easier to debug.

Functional programming languages

If our goal is to do functional programming, then there are languages that implement these principles and facilitate programming functionally.

- Lisp
- Haskell
- Clojure
- Scheme
- Erlang
- and many others

Functional programming in Python

We can also write functional programs in Python. The language provides features that facilitate this. We can write purely functional programs, but more commonly we use a mix of OO and functional constructs as appropriate.

If we simple realise that managing state is difficult and we learn to do it in a careful and deliberate way, we get some of the benefits of FP without having to discard OO.

Functional tools: Lambda

In FP we often write short, and frequently anonymous functions.
Lambda expressions are useful for this.
The following are equivalent.

```
addone = lambda x: x + 1

def addone(x):
    return x + 1
```

Functional tools: List operations

When we program without state, we can't have loops anymore. Instead, we operate (recursively) on lists, often with higher-order functions. Three important one are

- map
- filter
- reduce

Functional tools: map

`map` takes a function and a list and returns a new list produced by applying the function to each element in the original list.

```
map(f, [1, 2, 3]) -> [f(1), f(2), f(3)]

map(lambda x: 2 * x, [1, 2, 3]) # returns [2, 4, 6]
```

Functional tools: filter

`filter` takes a boolean function and a sequence and returns a new list that contains the elements of the original sequence whose members satisfy the boolean condition.

```
filter(lambda x: x % 2 == 0, range(10)) #returns [2, 4, 6, 8]
```

Functional tools: reduce

`reduce` is used to reduce a list of values down to a single value. It takes a function of two values and a sequence and returns the result obtained by successively applying the function to items in the sequence and the previously computed value.

```
reduce(lambda x, y : x + y, range(10)) #returns 45
```

Functional tools: reduce

`reduce` has an optional 3rd argument that is an initial value to use in the reduction function.

```
reduce(lambda x, y : x + y, range(10)) #returns 45
reduce(lambda x, y : x + y, range(10), 5) #returns 50
```

Exercises

1. Write a lambda expression that computes factorials. (Use `reduce`.)
2. Write a function that takes a string and a character and returns a new string that is the original string with all occurences of the character removed.
3. Write a function that is similar to the one above, but that returns the number of occurences of the character.
4. Write a function that takes a string and a character and returns the number of words that start with the character.
5. Write a function that takes a string and a character and returns a new string that is the original string with all occurences of the character in uppercase.
6. Write your own version of `map` (call it `mymap`) using a loop.
7. Rewrite `mymap` using recursion.