204: Swift Functional Programming

Part 1: Overview

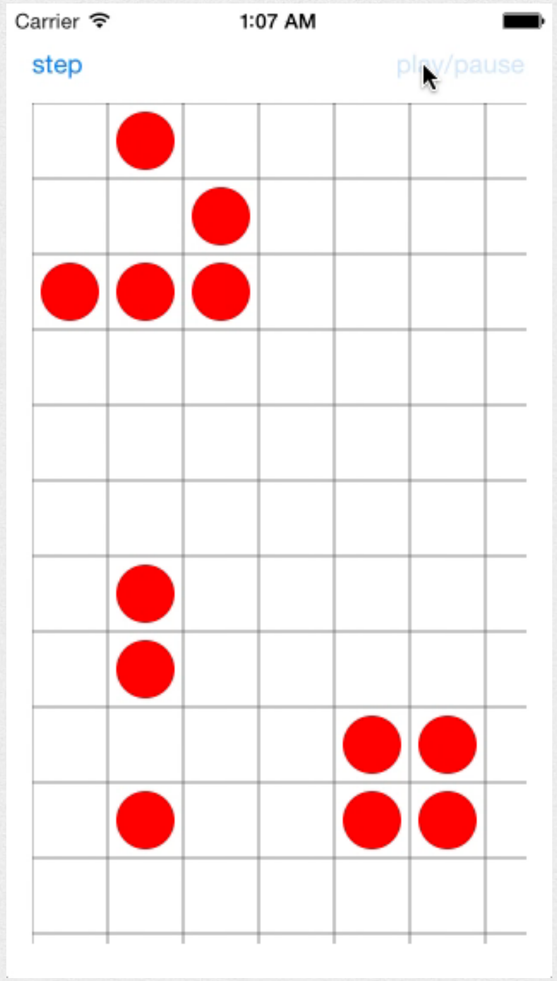
Functional Programming

- A *style* of programming ...
- ... treating the *mathematical function* as the primary unit of abstraction

Agenda: Elements of FP style

- the classic *higher-order functions*:
  - map, filter, & reduce
- functional *problem modeling*
  - pure functions (vs side-effecting functions)
  - values (vs mutable objects)

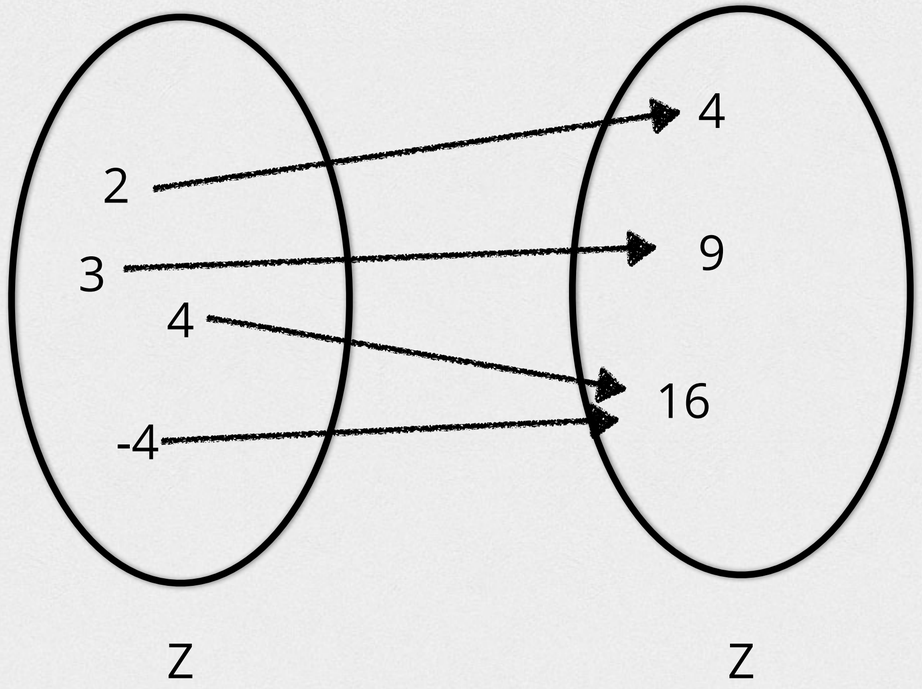
Conway's Game of Life



- Grid of live/dead cells
- Current liveness of cell and its neighbors determines future liveness of cell

Pure Function (vs Impure)

- maps values to values
- mathematical value
  - immutable
  - no identity,
    only equality
- *"does"* nothing



f: Z -> Z

f(3) = 9

f(x) = x * x

Values (vs Objects)

$$t_0 = 5$$

```
NSNumber * t0 = @5;
```

$$t_0 = 5$$

```
t0 = @6;
```

$$t_0 = 5$$

```
[t0 setIntegerValue:@7];
```

$$f(t_0) = 50$$

```
f(t0); // => 70
```

$$f(t_0) = 50$$

```
f(t0); // => 80
```

How To Prefer Values in Swift

	BEST	BETTER	WORST
what?	immutable value	mutable value	object
how?	constant struct[1]	variable struct	class
why?	It is immutable, so *no one can change it. (*Sharing moot)	Changeable but not shared, so *only you can change it*	Changeable and shared, so *anyone can change it*

[1]. Or, create a *value object*, by defining a class with a total initializer, only read-only properties, a member-wise `isEqual`. Like NSString, or NSArray holding value objects.