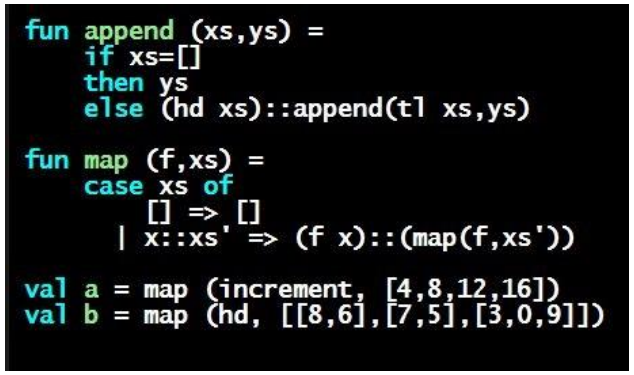


# Programming Languages

## Dan Grossman

Dynamic Dispatch Versus Closures

# *A simple example, part 1*

In ML (and other languages), closures are closed

```
fun even x = if x=0 then true  else odd  (x-1)
and odd  x = if x=0 then false else even (x-1)
```

So we can shadow **odd**, but any call to the closure bound to **odd** above will "do what we expect"

- Does not matter if we shadow **even** or not

```
(* does not change odd - too bad; this would
   improve it *)
fun even x = (x mod 2)=0
```

```
(* does not change odd - good thing; this would
   break it *)
fun even x = false
```

# *A simple example, part 2*

In Ruby (and other OOP languages), subclasses can change the behavior of methods they do not override

```
class A
  def even x
    if x==0 then true  else odd  (x-1) end
  end
  def odd x
    if x==0 then false else even (x-1) end
  end
end
class B < A  # improves odd in B objects
  def even x ; x % 2 == 0 end
end
class C < A  # breaks odd in C objects
  def even x ; false end
end
```

# *The OOP trade-off*

Any method that makes calls to overridable methods can have its behavior changed in subclasses even if it is not overridden

- Maybe on purpose, maybe by mistake
- Observable behavior includes calls-to-overridable methods

- So *harder* to reason about “the code you're looking at”
  - Can avoid by disallowing overriding
    - “private” or “final” methods

- So *easier* for subclasses to affect behavior without copying code
  - Provided method in superclass is not modified later