"Functional" Language

* Anonymous functions
* Higher-order-functions
* Partial application

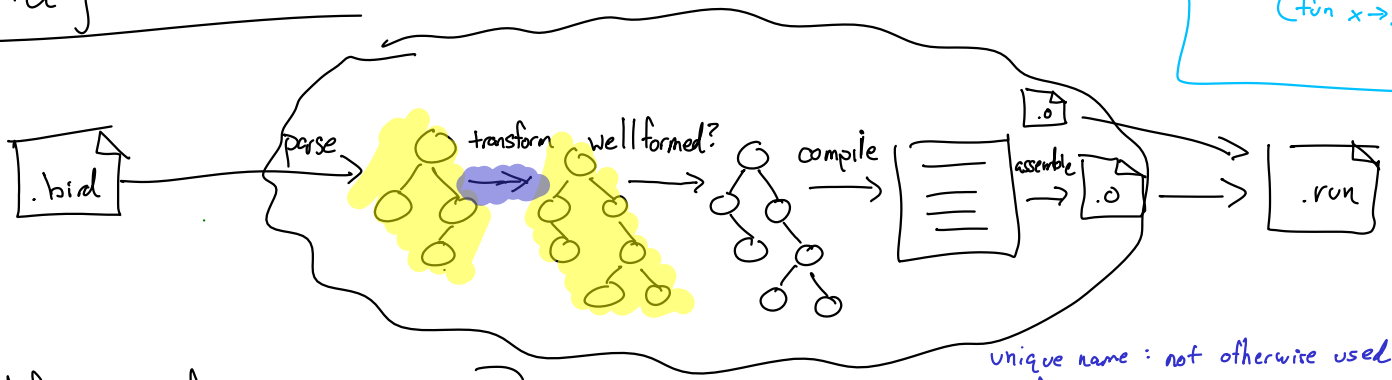
} Falcon } Finch

Finch:

<exp> ::= ... | fun <param_list> → <expr>

Magic Cloud!

(fun x → x x)
(fun x → x x)




```
def twice f x =
  f (f x)
end
twice (fun x → x * 3) 4
```

Falcon

unique name : not otherwise used

```
def anon$0 x =
  x * 3
end
def twice f x =
  f (f x)
end
twice anon$0 4
```

```
def twice f x =
  f (f x)
end
let y = 2 in
twice (fun x → x + y * 3) 4
```

⇌

```
def anon$0 y x =
  x + y * 3
end
def twice f x =
  f (f x)
end
let y = 2 in
twice (anon$0 y) 4
```

```
convert (p : program) : program =
  let Program (decls, main) = p in
  let main', decls' = convert_expression main in
    convert_expression
    ⋮
```

For every anonymous function in any expression, add a declaration with a unique name to the decl list with same body & params of anonymous function. Replace the anonymous function with its new name.

convert_expression (e : expression) : expression * declaration list

```
def foo x =
  fun y -> x + y
end

foo 5 8
```

Finch

```
def anon$0 x y =
  x + y
end
def foo x =
  anon$0 x
end
foo 5 8
```

```
def bar x y z =
  fun q -> q + y
end

bar 3 4 5 6
```

Falcon

```
def anon$0 y q
  q + y
end
def bar x y z =
  anon$0 y
end
bar 3 4 5 6
```

Question: given an expr like "fun q -> q + y", what set of non-local vars does it use?

needs "a+b" => {a, b}

needs "let a=4 in a+b" => {b}

"free variables"