Falcon

- ☆ Anonymous functions
- ☆ First-class functions
- ☆ Partial application

Finch

Falcon

```
def add x y =
  x+y
end

let inc = add 1 in
```

```
def inc y =
  1+y
end
```

Every function that can exist at runtime is either

1. a function that we compiled or
2. a specialization of such a function

Represent functions as closure values:

size in args	# params	ptr to code	bird arg	.....	bird arg

Last lecture: algorithm for handling expressions of form f x EAppl(..., ..)

<closure@0x00......5e9821(0/1)>

let g=f in

```
def f x =
  x
end
```

f ← compile in an environment where fn names map to initial closures

```
section .text
  ;
fun_f:
  push rbp
    ;
  pop rbp
  ret
bird_main:
  push rbp
    ;
  mov rax, closure_f+1
    ;
  pop rbp
section .data
  align 8
  heap_cursor: dq 0
  align 8
  closure_f: dq 0x8000000000000000, 1, fun_f
```

Falcon:

errors are different

Dove	Falcon
unbound vars	unbound vars
undefined fn	unbound vars
wrong # args	not a complie error
duplicate params	duplicate params
duplicate fns	duplicate fns

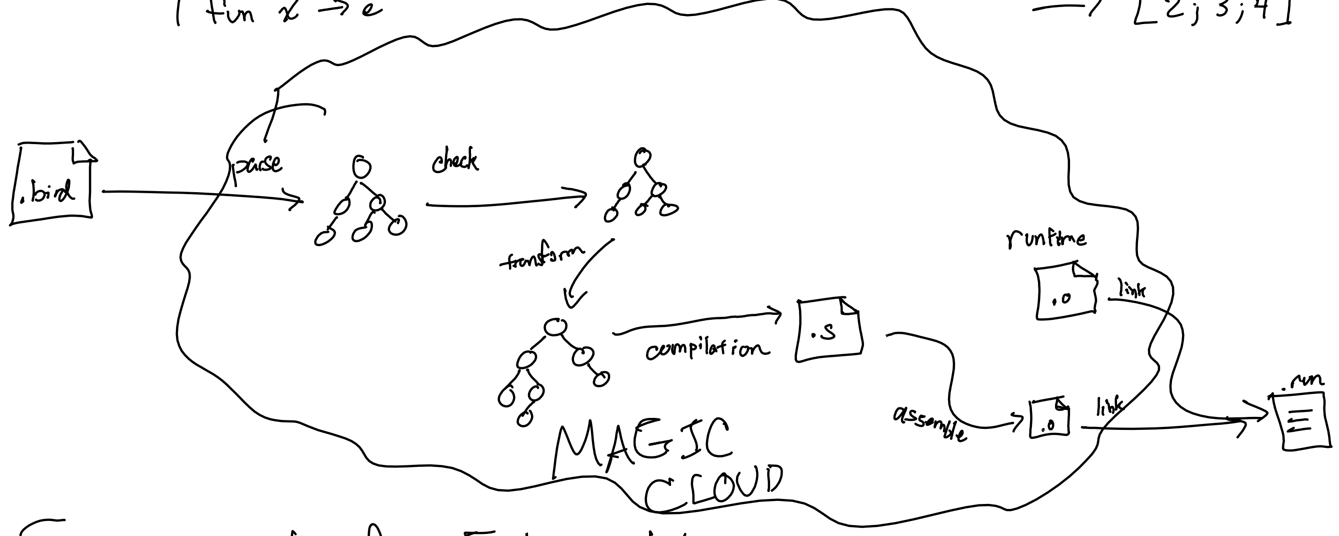
```
def f n =
  n
end
g

def add x y =
  x+y
end
add 1 2 3
```

Finch

(not required for lab)

```
List.map (fun a → a+1) [1;2;3]
  ⟹ [2;3;4]
```

```
<expr> ::= ....
         | fun x → e
```



Strategy: transform Finch into Falcon

Finch

```
def twice f x =
    f (f x)
end

twice (fun a → a+1) 4
```

Falcon

```
def twice f x =
    f (f x)
end
def $0 a =
    a+1
end
twice $0 4
```

```
let transform_expr
      (expr : expr)
        : expr * declaration list =
   ...
;;

let transform_decl
      (decl : declaration)
   : declaration * declaration list =
   ...
;;
```

```
def f x =
    let a=x in
    fun y → a+y
end
f 1 2
```

```
def f x =
     let a=x in
     $0 a
end
def $0 a y =
     a+y
end
f 1 2
```

1. Examine fn body to find all "free variables" — those vars not bound in that subtree

   ```
   let x=4 in
   x+y         } y is free
   ```

2. Subtract all params
3. Add those vars as params and args

closure conversion

```
let fn x =
   let loop a =
      ....
   in
   ...
```