Today

How do we add capability to the interpreter for functions we can't write, like +?

Why not

big eval loop

if (car(—))

- "if" ——

- "lambda" ——

- "define" ——

- "+" ——

- "-" ——

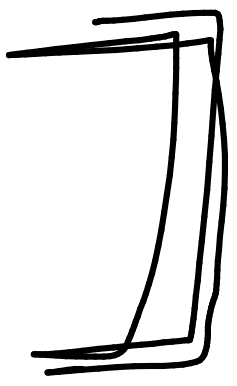
- "*" ——

- "/" ——

- "%" ——

} scales poorly

```
(define mygadd +)
(mygadd 3 5)
```



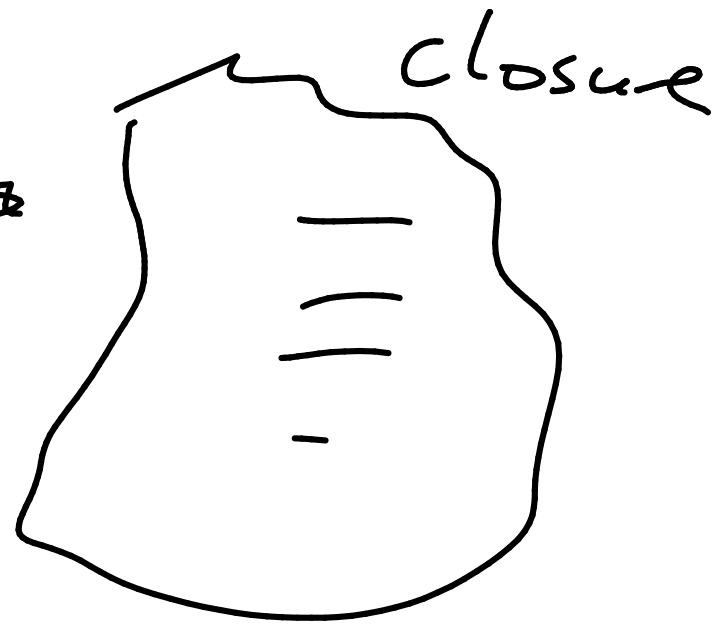
Code above would fail on this.

You need a way to treat functions as data so you can point at them

You are doing this right now w/ closures for functions defined in Scheme.

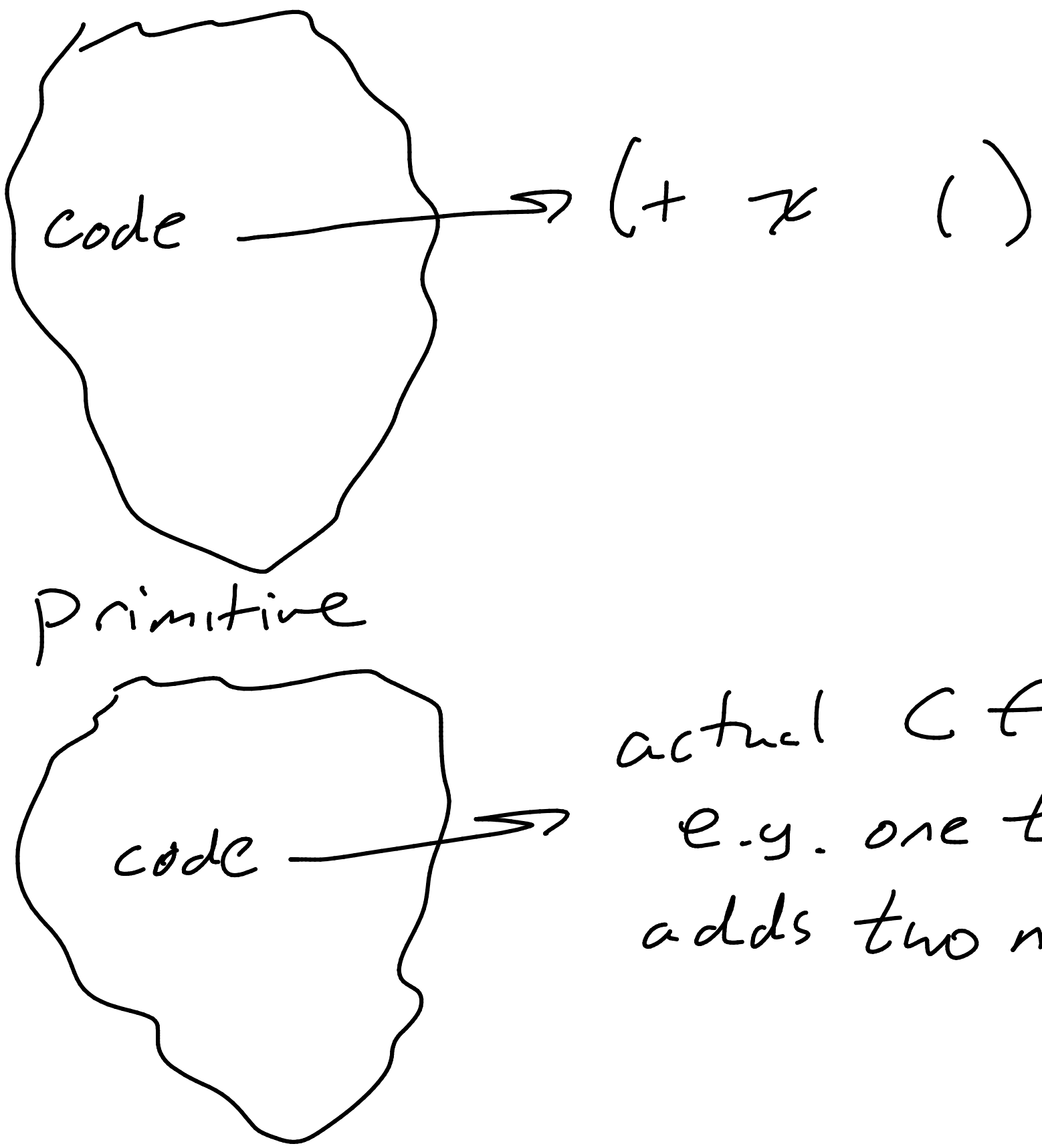
```
(lambda (x)
  (+ x 1))
```

→



Implemet something similar to closures, that we'll call primitives.

A primitive (PRIMITIVE_TYPE) is another SchemeVal that points to actual C code.



Look at function pointers in C.

```
typedef struct SchemeVal {
    objectType type;
    union {
        ...
        struct SchemeVal *(*pf)(struct SchemeVal *);
    };
} SchemeVal;
```

pf is a pointer

return type: it returns a ptr to a SchemeVal

name of the function pointer (pf)

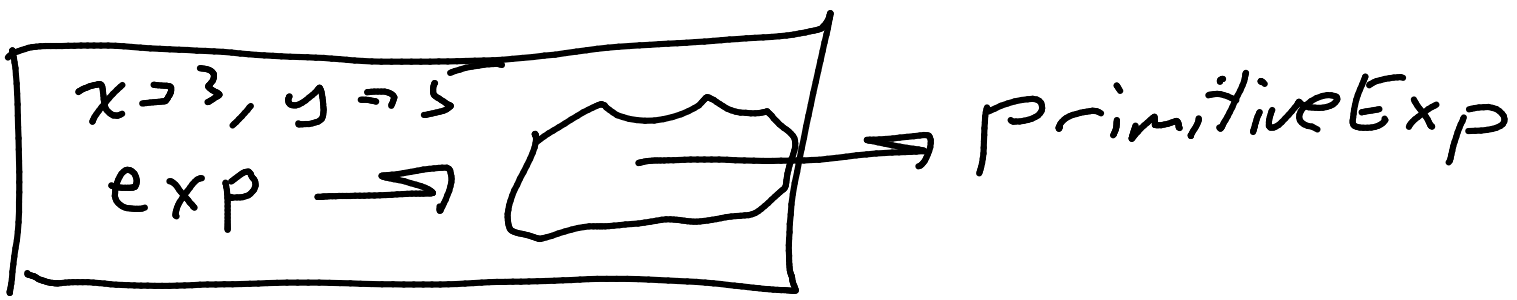
one param - ptr to a SchemeVal (i.e. a list of SchemeVals)

```
SchemeVal *primitiveExp(SchemeVal *args) {
    // check that args has length one and car(args) is numerical
    // assume that car(args) is of type double, should check that as we
    SchemeVal *result = talloc(sizeof(SchemeVal));

    result->type = DOUBLE_TYPE;

    // Uses built-in C function exp

    result->d = exp(car(args)->d);
    return result;
}

void bind(char *name, SchemeVal *(*function)(SchemeVal *), Frame *fram
    // Add primitive functions to top-level bindings list
    SchemeVal *value = talloc(sizeof(SchemeVal));

    value->type = PRIMITIVE_TYPE;

    value->pf = function;

    // Your code will differ from each other some on the following;
    // it adds a binding to a frame; can leave that out here
    frame->bindings = ....;
}

void interpret(SchemeVal *tree) {

    // Make top-level bindings list
    Frame *frame = talloc(sizeof(Frame));
    frame->bindings = makeEmpty();
    frame->parent = NULL;

    bind("exp", primitiveExp, frame);

    // ...
}
```

* Error checking missing

frame

You've done this



Preview of what's next:

types in programming languages

Languages generally fall into 2 categories regarding where the type ~~of~~ goes

C

```
int x = 3;
```

↑ type

x (int)

3

```
x = 9.6;
//error
```

Python

```
x = 3
```

x ⟶ 3 (int)

```
x = 9.6
```

x ⟶ 9.6 (double)