Types and Type Checking

optional annotation

Strong

Weak

int

inference

gradual

lazy

parsing

static

dynamic

OBJECT)

runtime

dependent

compile-time

void(*)

subtyping

inheritance

recursive

- Catch errors before program runs.

- Say something about infinite loops

- Additional restrictions (why?)
  beyond runtime error prevention

static / dynamic

```
var
void myprint(int i) {
    while(true) {
        System.out.println(i);
    }
}

psv main(blargs) {
    System.out.println("dynamic");
    myprint("a");
}
```

<exp> := (<op> <exp> <exp>)
| (if <exp> <exp> <exp>)
| NUM
| BOOL

<op> := + | <

```
data Expr:
  | e-op(op :: Op
         e1 :: Expr,
         e2 :: Expr)
  | e-if(c, t, e)
  | e-num(n)
  | e-bool(b)
```

type-check :: Expr -> Bool

(if 5 2 3) false

(if false (+ true 10) 11) true? (false?)

(if (user-input) (+ true 10) 11) false

(user-input)

(+ true 10)

DESIGN DECISION

true = definitely no errors

false = maybe error

```
fun type-of (e :: Expr) -> Type:
  cases (Expr) e:
    | e-num (n) => t-num | e-bool (b) => t-bool
    | e-op (op, e1, e2) =>
      te1 = type-of (e1)
      te2 = type-of (e2)
      cases (Op) op:
        | o-plus =>
          if (te1 == t-num) and (te2 == t-num):
            t-num
          else:
            raise (t-not-a-num)
          end
        | o-less =>
          (like o-plus, but t-bool)
    | e-if (c, t, e) =>
      ct = type-of (c)
      when not (ct == t-bool) : raise (t-not-a-bool) end
```

```
data Type:
  | t-num
  | t-bool
end
```

t-if
t-good
t-bad?

```
data TError:
  | t-not-a-num
  | t-not-a-bool
  | t-bad-if
end
```

Error

Design ☆

```
    tt = type-of(t)
    te = type-of(e)

    when not(tt == te): raise(t-bad-if) end
    tt
end
```

```
(if true 3 false)
false
```

```
(if (if true 3 false)
    4
    5)
```

if's type is a

DESIGN

DECISION