

Types

Sets of computational objects with uniform behavior

- Set of values
- Set of operations

Type Errors

Applying operation to data for which those operations don't apply.

Does the Language Check Types?

What happens if you don't check types?

When Does the Language Check Types?

- Static type checking Check at "compile time"
- Dynamic type checking Check at run time

Strongly Typed Systems

A type system is *strong* if it admits only correctly typed expressions.

Doesn't guarantee there are no errors, just that there are no *type* errors.

Strong typing is *not* synonymous with

- Declaring types
- Static type checking
- Type inference

Strong typing significantly helps program development.

Typed Functions

You're probably used to thinking of variables as having types, but functions have types too:

Pascal:

```
function f (x, y : integer) : integer
```

C:

```
int f(int x, int y)
```

are both

```
f:(integer  $\times$  integer)  $\rightarrow$  integer
```

This is true even in languages in which functions aren't first class.

Strategies

Two general approaches for strong typing: *type checking* and *type inference*

Type checking (declarations)

- Programmer identifies the type of every variable
- Use knowledge of semantics to determine the type of every expression from declarations
- Check to make sure expressions and declarations match

Assignment
Parameter passing
Primitive functions
...

Type inference

- Programmer doesn't (generally) have to identify types
- System figures out what types things are
- Again, check to make sure types match