The need for types

Our current SPL interpreter has no problem with

write `4 - true;`

which produces 3.

Or, more troubling,

new `a := 10;`
write `a(8);`

which gives a `Segmentation fault`.

What is a type?

A *type* is a tag on some data in a program that indicates what it means or how it can be used.

Types can be *built-in* (e.g. `int`, `char`, ...)
or *user-defined* (e.g. with `class`, `enum`, `typedef`, ...)

Types can be *declared* (C, C++, Java, Ada, ...)
or *implicit* (*inferred*) (Scheme, Ruby, Perl, Haskell, ...)

Type Safety

Besides providing information to the compiler and the programmer, types help ensure data gets used correctly.

Type safety is a mechanism enforced by the compiler or interpreter to ensure that types are not used in an incorrect or meaningless way.

A language without type safety is highly prone to errors and exploits.
Nearly every modern language supports type safety to some extent.
Some languages allow explicit overwriting of type safety checks.

Dynamic vs Static Typing

Where is type information stored?

- **Dynamic Typing**:
  Types are stored with data objects, at run-time.
  Makes sense for interpreted languages.
- **Static Typing**:
  Types stored with symbols, and *inferred* for expressions, at compile-time.
  Very useful in compiled languages.

Type inference

This refers to the automatic determination of an expression's type.

- **Simple example**: `5 + 3`
  has type `int` because 5 and 3 are both `int`s.
- **More difficult**: `5 + 3.2`
  Is this a `double` or `int`?
  Depends on rules for *type promotion*/*coercion*.
- **Totally crazy**: Some languages like ML infer the types of all variables, arguments, and functions based on how they are used.
  Type consistency is ensured at compile-time!

What gets a type?

Constants or *literals* such as `-8`, `'q'`, `"some string"`, and `5.3`
will all have a type.
Expressions will generally have the type of whatever value they compute.

- **Names**: Only have a fixed type in *statically-typed* languages.
- **Functions**: Type is determined by number and types of parameters and type of return value.
  Can be thought of as pre- and post-conditions.
  May be left unspecified in dynamically-typed languages.
- **Types**: Do *types* have type? Only when they are first-class!

Type Checking

Type checks ensure type safety.
They are performed at compile-time (*static*) or run-time (*dynamic*).

- **Dynamic Type Checking**: Easy! Types of arguments, functions, etc. are checked *as they are applied*, at run-time. Every time an object is accessed, its type is checked for compatibility in the current context.

- **Static Type Checking**: Type safety is ensured at compile-time.
  The type of every node in the AST is determined statically.
  Some level of *type inference* is always necessary.
  Often, *type declarations* are used to avoid the need for extensive inference.

Class outcomes

You should know:

- What types are, and why we want them.
- The benefits of type safety in programming languages.
- The differences between static and dynamic typing.
- The meaning of type inference.

You should be able to:

- Demonstrate dynamic and static type checks for an example program.