The role of a type system

- As a means for representing **concepts** more explicitly in the application domain.
- As a means for **detecting** certain kinds of programming errors - type errors
- As a means for language developers to **optimize** for increased performance.

Static or ***Dynamic*** - type checking mechanism

- **Static typing**: the "type" of an attribute (e.g. and integer, a float, or an instance of a class) is known at *compile time*.
  - The type of the attribute is made explicit, and
  - The type checking is done at compile time.
  - Catch type mismatch errors at compile time.
- **Dynamic typing**: no explicit type is assigned to an identifier.
  - In some such languages, it is possible to interrogate the object at run time to determine the current type associated with the identifier.
- In the early phases of software development, especially when the language is used for prototyping, heavy type systems tend to interfere with flexibility, and may impede productivity for rapid prototyping.

Type Systems and Program Development

- *Type* means many things to many people:
  - strict *"types have **verifiable behavior** independent of any implementation"*
  - to the less formal *"type equals class*" definition.
- Type system - *strong* or *weak:*
  - **weak typing:** enforcement of type rules but with well-defined exceptions or an explicit type-violation mechanism.
  - Weak typing is "friendlier" to the programmer than strong typing, but catches fewer errors at compile time.
  - C and C++ are often said to be weakly typed, as they automatically coerce many types e.g. ints and floats.
    E.g. `int a = 5; float b = a;`
    Some people ignore this and call C, C++ strongly typed tho.

Strong Typing

- **Strongly typed** languages have *type signatures* and behavior that can be represented in an abstract way
- Strict enforcement of type rules with no exceptions. All types are known at compile time, i.e. are statically bound. With variables that can store values of more than one type, incorrect type usage can be detected at run-time.
- Strong typing catches more errors at compile time than weak typing, resulting in fewer run-time exceptions.
- **Type Signature**: the sequence of primitive datatypes (and possibly, any other formal constraints)
  E.g.: {double, int, char}.
- **Type** refers to the specification, independent of the implementation (the actual 'class').

Types

- In a perfect strongly typed language, the type **carries all the information** about the denoted object and **can be reasoned about without execution**.
- On average, a function can be written in a **dynamically typed** language much more *quickly* and can be less than *half the size* syntactically than a statically typed function in Java or a flexible template implementation in C++.

Type Inference

- An algorithm for ascribing types to expressions in some language, based on the types of the constants of the language and a set of *type inference **rules*** such as:

```
f :: A -> B,  x :: A
---------------------  (App)
      f x :: B
```

- This type inference rule, called "App" for application, says:
  - if expression **f** has type **A - > B** and
  - expression **x** has type **A** , then we can deduce that
  - expression **(f x)** has type **B**.
  - The expressions above the line are the premises and below, the conclusion.
- May be used to: • Generate warnings, • Improve efficiency