Type Systems

Ken Q Pu, Faculty of Science, Ontario Tech University

Why types?

1. Static analysis of code to prevent type related runtime errors.
2. Static annotation of source code to improve code readability.

Let's first understand what a type system is.

Types

Types are descriptions of values.

Values can be:

- Atomic scalars (numbers, strings, ...)
- Composite data (records, tuples)
- Containers (list, hashmap, arrays...)
- Functions
- Objects

Each value can be described by one or more types.

```
3
```

```
Integer
```

```
"Hello"
```

```
String
```

```
{:name "Einstein"
 :IQ 160}
```

```
Record
  :name String
  :IQ Int
```

```
(defn add [x y]
  (+ x y))
```

```
Function
 (Num, Num) -> Num
```

Type Signature

Expressions to be evaluated during runtime.

```
"Hello"
```

```
String
```

Type signature, or simply type of the expression, to be determined at compile time.

```
{:name "Einstein"
 :IQ 160}
```

```
Record
  :name String
  :IQ Int
```

```
(defn add [x y]
  (+ x y))
```

```
Function
 (Num, Num) -> Num
```

Type System

A type system is a formal sublanguage that can express the type signatures for **all possible** values supported by the programming language.

Statically typed language

A programming language is **statically typed** if ***all expressions*** in the source code have a valid type signature. Furthermore, the expression types must be determined **before** the program executes.

```
(let [pi 3.14]
  (+ pi
     (inc 4)))
```

Statically typed language

A programming language is **statically typed** if ***all expressions*** in the source code have a valid type signature. Furthermore, the expression types must be determined **before** the program executes.

Float

```
(let [pi 3.14]
  (+ pi
     (inc 4)))
```

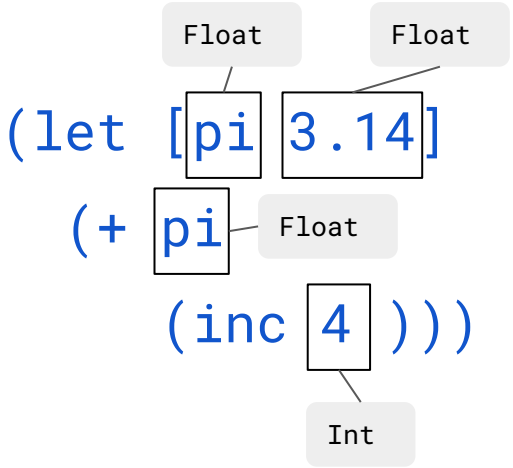
Statically typed language

A programming language is **statically typed** if ***all expressions*** in the source code have a valid type signature. Furthermore, the expression types must be determined **before** the program executes.

```
Float        Float
(let [pi 3.14]
  (+ pi      Float
     (inc 4)))
```

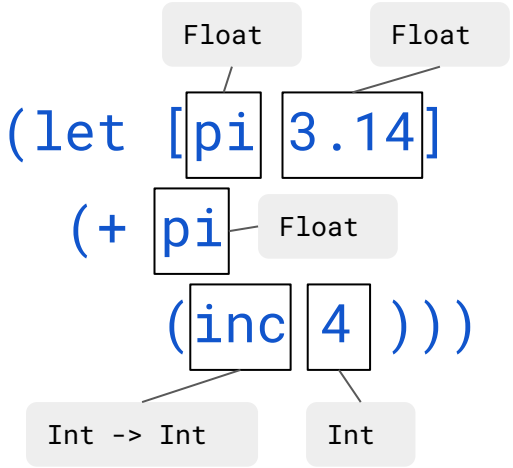
Statically typed language

A programming language is **statically typed** if ***all expressions*** in the source code have a valid type signature. Furthermore, the expression types must be determined **before** the program executes.



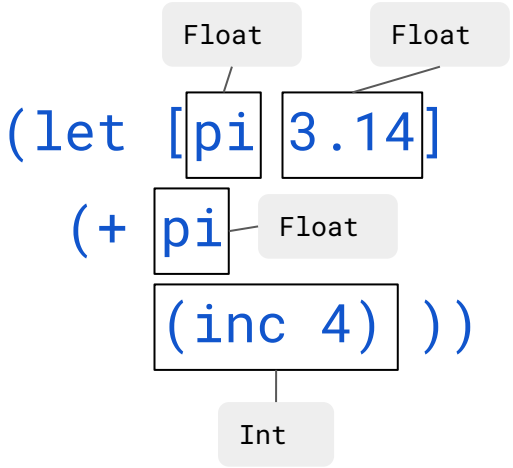
Statically typed language

A programming language is **statically typed** if ***all expressions*** in the source code have a valid type signature. Furthermore, the expression types must be determined **before** the program executes.



Statically typed language

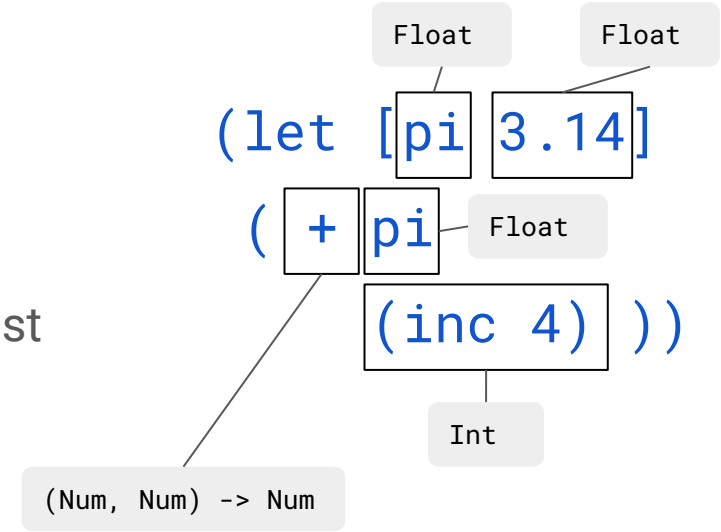
A programming language is **statically typed** if ***all expressions*** in the source code have a valid type signature. Furthermore, the expression types must be determined **before** the program executes.



Ken Q Pu, Faculty of Science, Ontario Tech University

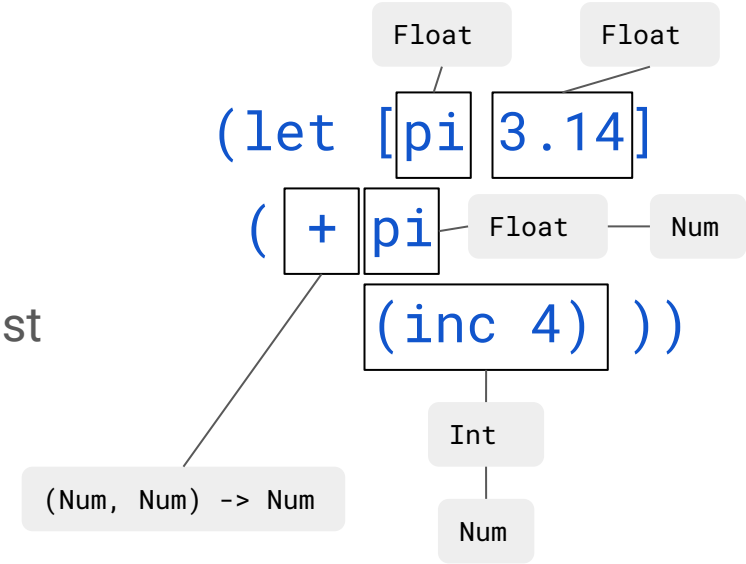
Statically typed language

A programming language is **statically typed** if ***all expressions*** in the source code have a valid type signature. Furthermore, the expression types must be determined **before** the program executes.



Statically typed language

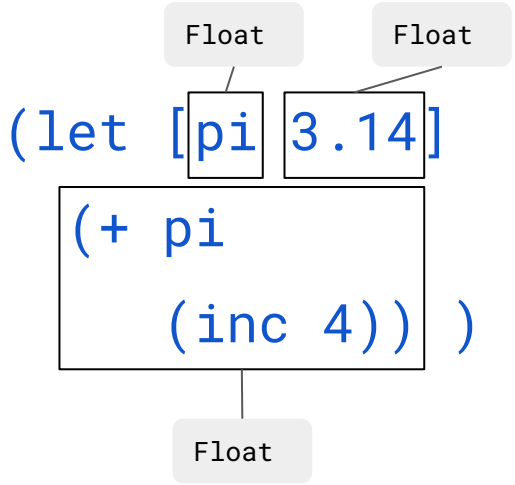
A programming language is **statically typed** if ***all expressions*** in the source code have a valid type signature. Furthermore, the expression types must be determined **before** the program executes.



Ken Q Pu, Faculty of Science, Ontario Tech University

Statically typed language

A programming language is **statically typed** if ***all expressions*** in the source code have a valid type signature. Furthermore, the expression types must be determined **before** the program executes.



Statically typed language

A programming language is **statically typed** if ***all expressions*** in the source code have a valid type signature. Furthermore, the expression types must be determined **before** the program executes.

```
(let [pi 3.14]
  (+ pi
     (inc 4)))
```

Float

Computing the type signature of all expressions is called **type inference**.

Type inference is only done for statically typed languages. Dynamic typed languages computes the type signatures based on data values at runtime.

Clojure is not statically type.

Clojure compiler does not determine the type information of expressions during compile time.

All type information are determined at runtime.

```
(let [x (get-value ...)]
  x)
```

Clojure only knows the type of x **after** get-value is executed.

Why types?

1. Static analysis of code to prevent type related runtime errors.
2. Static annotation of source code to improve code readability.

Runtime errors

Certain computation cannot be carried out at runtime, and the operating system has no choice but to stop the program execution immediately.

```
(let [x [:a :b :c]]
  (println (nth x 5)))
```

Illegal data access. (nth x 5) cannot be evaluated because x only has three elements.

```
(+ 3 "four")
```

Illegal function invocation. Parameters of + cannot be mixed between numbers and strings.

Runtime errors are dangerous

Runtime errors can cause the system to be stuck in undesirable states.

The program can terminate prematurely beyond the expectation of the programmer.

```
(defn move-file [source target]
  (let [content (read-file source)]
    (remove-file source)
    (println (+ "Bytes to move: " (count content)))
    (write-file target content)))
```

What is the runtime behaviour?

```
(move-file "important.txt"
           "passwords.txt")
```

Static Analysis: type checking

Type checking is the verification that all expressions have **valid** types.

More details will be discussed on the type checking algorithm.

```
( + "Bytes to move: " ( count content ) )
```

Static Analysis: type checking

```
( + "Bytes to move: " ( count content ) )
```

Array ✔

Iterable ✔

Static Analysis: type checking

```
( + "Bytes to move: " ( count content ) )
```

```
Iterable -> Int  ✔
```

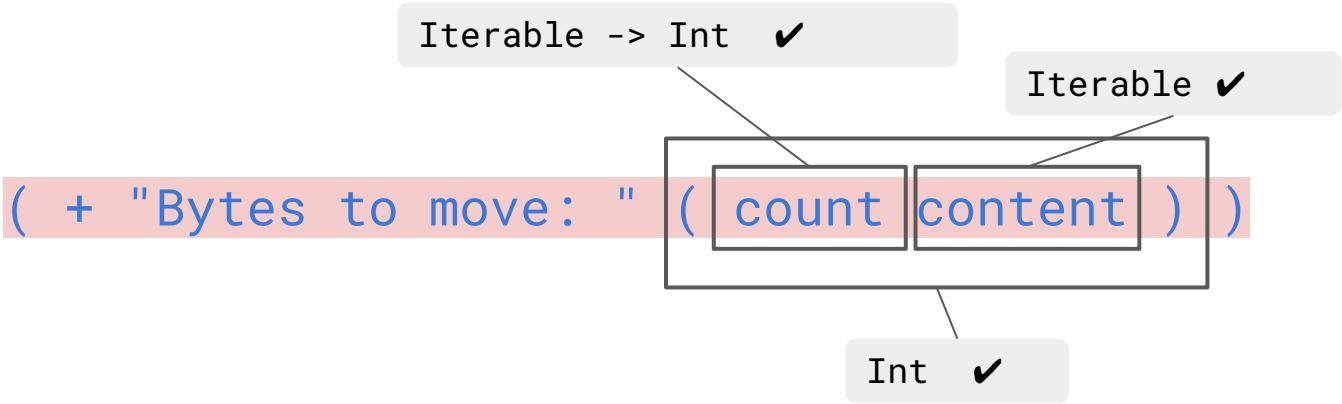
Static Analysis: type checking

```
Iterable -> Int  ✔
```

```
Iterable ✔
```

```
( + "Bytes to move: " ( count content ) )
```

Static Analysis: type checking



Static Analysis: type checking

```
Int, Int -> Int
            or
String, String -> String
```

```
( + "Bytes to move: " ( count content ) )
```

String ✔

Int ✔

Static Analysis: type checking

```
Int, Int -> Int
            or
String, String -> String
```

```
( + "Bytes to move: " ( count content ) )
```

String ✔

Int ✔

ERROR ✖