# Lisp types and checking

- Variables (and parameters) don’t have a fixed type – the data type is based on whatever actual data value they’re currently holding
- We can pass any kind of value to a function, not necessarily of the type it expects/needs
- Functions can crash if run on the wrong data type
- We must have some way of figuring out what kind of data a variable or parameter actually holds right now
- As with all things lispy, this is done through function calls

## Type checking

- For each built in data type, there is a function that can check if something is of that type
- These functions return true (t) if it is that type, false (nil) otherwise
- By convention, the names of most of these functions end in p, e.g. `(integerp x)` returns t iff x is an integer
- Available functions include `integerp`, `rationalp`, `floatp`, `complexp`, `listp`, `stringp`, `functionp`, `symbolp`, `vectorp`, `arrayp`, and many others

## Type checking habits

- Type checking becomes part of life for a lisp programmer
- When you read a value from the user (or a file), the first thing you’ll typically do is check which type it is
- When you write a script that takes command line arguments, or a function that takes parameters, the first thing you’ll typically do is check that the type meets your expectations

## Other typechecking functions

- We can look up the type of something using `(type-of x)`, which returns the name of the type as a symbol (e.g. ‘float)
- `typecase`, a type-based case statement (like a C++ switch, but for data types) let’s us check something against a series of types
- We can also see if something’s type matches a specific type using `(typep x sometype)`

## User-defined types

- Some lisp functions also allow us to generate user defined types (e.g. structures), and will at the same time create functions for us that allow us to check if an item is of our defined type