Types in Asteroid

- Asteroid is dynamically type checked
  - Infers types of variables at runtime
  - No type hierarchy, explicit type conversions!
  - Be careful, incorrect type conversions can lead to information loss!

```plaintext
ast> let i = 1 + 2.5.
error: found 'integer + real' expected 'real + real'
ast> let i = toreal(1) + 2.5.
ast> gettype(i)
real
ast> i
3.5
ast> let i = 1 + tointeger(2.5).
ast> gettype(i)
integer
ast> i
3
ast>  
```
Lists in Asteroid are polymorphic in the sense that they do not enforce any kind of type restrictions on their elements.

This is similar to Python and very different from languages like C++ where this kind of polymorphism can only be achieved via class inheritance.

The following is legal in Asteroid,

```
ast> let l = [1,2.5,"three"].
ast> gettype(l)
list
ast> l
[1,2.5,three]
```
Types in Asteroid

- **Tuples:**
  - One way to think about tuples is as “fixed length lists” that are immutable, i.e.
    - Once you have decided on the number of components of a tuple you cannot change it.
    - Tuples with different number of components are incompatible.
    - You cannot change the contents of a tuple.

```plaintext
ast> let (x,y) = (1,2,3).
error: pattern match failed: term and pattern lists/tuples are not the same length

Asteroid 2.0.1
(c) University of Rhode Island
Type "help" for additional information
ast> let t = (1,2).
ast> let t@0 = 2.
error: term '(1,2)' is not a mutable structure
```
Types in Asteroid

- **Structures**
  - Asteroid uses name equivalence when computing the compatibility of two constructed types

```plaintext
ast> structure Type1 with
  ....  data a.
  ....  data b.
  ....  end
ast> structure Type2 with
  ....  data a.
  ....  data b.
  ....  end
ast> let q:%Type2 = Type1(1,2).
error: pattern match failed: conditional pattern match failed
ast> let q:%Type1 = Type1(1,2).
```