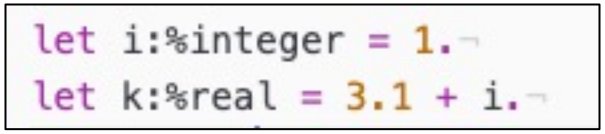
Types in Asteroid

- Primitive types
  - Asteroid's type hierarchy,
    boolean < integer < real < string
    allows for convenient type coercions.
  - The following is a valid program,

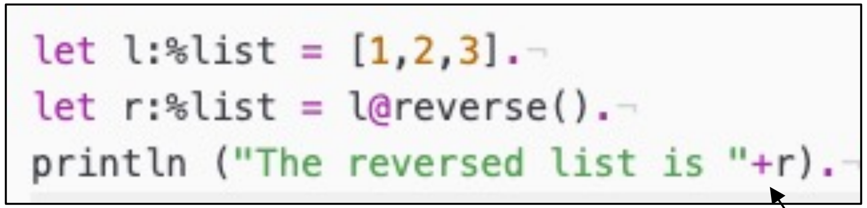


```
let i:%integer = 1.
let k:%real = 3.1 + i.
```

Type coercion from integer to real.

Types in Asteroid

- Consider part of Asteroid’s type hierarchy,
  - boolean < integer < real < string
  - list < string
  - tuple < string
- Printing values is convenient since everything is a subtype of string,

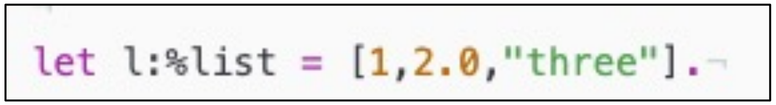


```
let l:%list = [1,2,3].
let r:%list = l@reverse().
println ("The reversed list is "+r).
```

Type coercion to string

Types in Asteroid

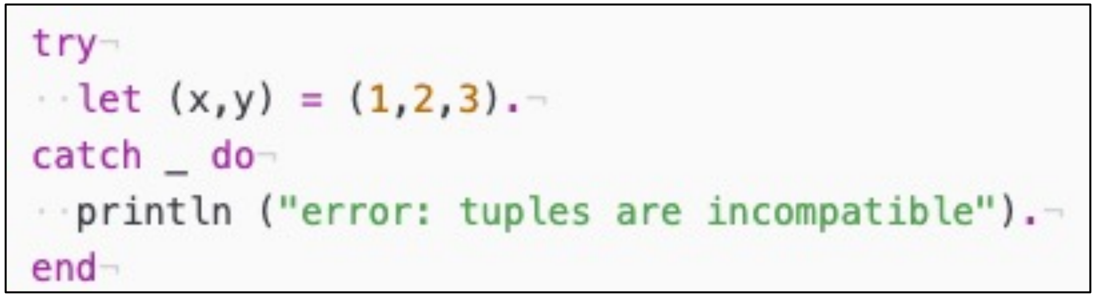
- Lists in Asteroid are polymorphic in the sense that they do not enforce any kind of type restrictions on their elements.
- This is similar to Python and very different from languages like C++ where this kind of polymorphism can only be achieved via class inheritance.
- The following is legal in Asteroid,



```
let l:%list = [1,2.0,"three"].
```

Types in Asteroid

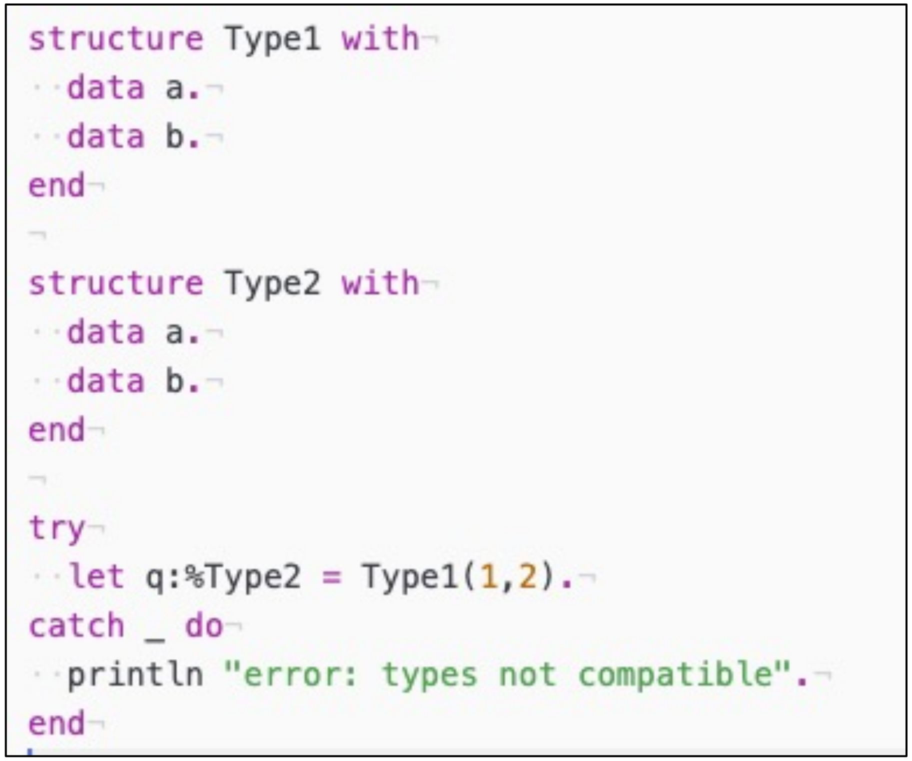
- One way to think about tuples is as “fixed length lists”.
  - Once you have decided on the number of components of a tuple you cannot change it.
  - Tuples with different number of components are incompatible.
- The following program will not succeed,



```
try
  let (x,y) = (1,2,3).
catch _ do
  println ("error: tuples are incompatible").
end
```

Types in Asteroid

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



```
structure Type1 with
  data a.
  data b.
end

structure Type2 with
  data a.
  data b.
end

try
  let q:%Type2 = Type1(1,2).
catch _ do
  println "error: types not compatible".
end
```