Imperative Programming in Asteroid – the Basics

- **Imperative programming** –
  - explicit statements that change the program state
- All three of our programming languages are at their core imperative programming languages.
- Here we look at basics of Asteroid programming

Names

- In Asteroid names are alpha-numeric symbols starting with an alpha character (as in most languages)
  - x
  - my_function
  - pi

Constants

- Constants are available for all the primitive data types,
  - integer, e.g. 1024
  - real, e.g. 1.75
  - string, e.g. "Hello, World!"
  - boolean, e.g. true

Primitive Data Types

- Asteroid arranges primitive data types in a type hierarchy,
  - boolean < integer < real < string
- Type hierarchies facilitate automatic type promotion, e.g.

```
let x:%string = "value: " + 1.
```

Type promotion: plus as string concatenate op

Structured Data Types

- Asteroid also supports the built-in data types:
  - list
  - tuple
- These are structured data types in that they can contain entities that belong to other data types.
- Lists are mutable objects whereas tuples are immutable.
- Some examples,

Note: (1,) ≠ (1)

```
let l = [1,2,3]. -- this is a list
let t = (1,2,3). -- this is a tuple
let one_tuple = (1,). -- this is a 1-tuple
```

Structured Data Types

- Lists and tuples themselves are also embedded in type hierarchies, although very simple ones:
  - list < string
  - tuple < string
- That is, any list or tuple can be viewed as a string. This is very convenient for printing lists and tuples,

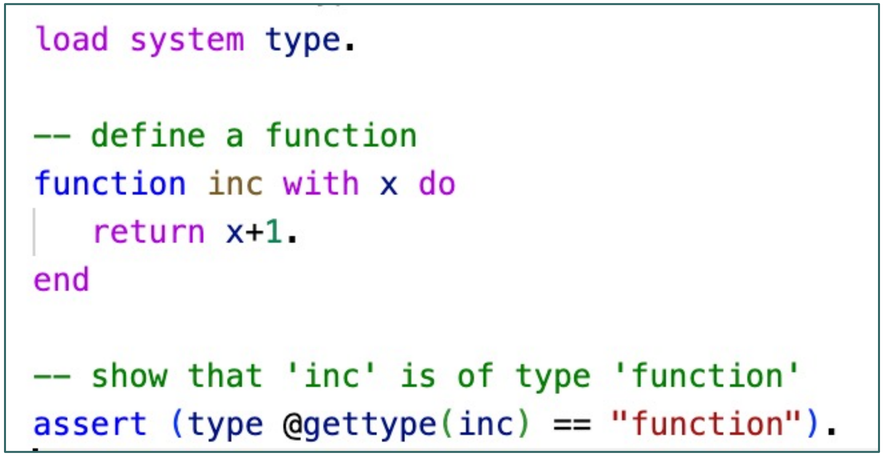
```
load system io.
io @println ("this is my list: " + [1,2,3]).
```

The None Type

- Asteroid supports the none type.
- The none type has only one member
  - A constant named none.
  - The empty set of parentheses () can be used as a shorthand for the none constant.
  - That is: none = ()

Other Data Types

- In Asteroid we also have additional data types:
  - function
  - pattern
  - user defined data types via structures



```
load system type.

-- define a function
function inc with x do
   return x+1.
end

-- show that 'inc' is of type 'function'
assert (type @gettype(inc) == "function").
```

In002/ftype.ast