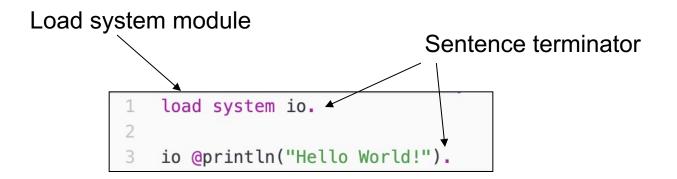
Asteroid Basics

https://asteroid-lang.readthedocs.io/

Imperative Asteroid

The "Hello World" program...



In002/hello.ast

Iteration

'while', 'for', 'loop' constructs are all supported

```
-- compute the factorial
                               Function argument
load system io.
function fact with n do
  let val = 1.
                         Iteration
 while n > 1 do
   let val = val*n.
                          Assignment
   let n = n-1.
 end
                                                          Function Call
  return val.
                    Type conversion
end
let x = tointeger(io @input("Enter a positive integer: ")).
io @println ("The factorial of " + tostring(x) + " is " + tostring(fact x)).
```

Function Calls

 In Asteroid function calls are constructed by juxta positioning a function with a value, e.g.

fact 3.

no parentheses necessary! But the traditional

fact(3).

also works.

Data Structures

- Built-in lists
 - [1,2,3]
- Built-in tuples
 - (x,y)
- Element access
 - a@i

```
-- the bubble sort
load system io.
function bubblesort with 1 do
  loop
                                 Element access
    let swapped = false.
    for i in 0 to len(l)-2 do
      if l@(i+1) <= l@i do
        let (l@i, l@(i+1)) = (l@(i+1), l@i).
        let swapped = true.
      end
    end
    if not swapped do
      break.
    end
  end
  return 1.
end
let k = [6,5,3,1,8,7,2,4].
io @println("unsorted array: "+tostring(k)).
io @println("sorted array: "+tostring(bubblesort k)).
```

Structures & Objects

- Asteroid is object-based
- Bundle operations with data
- No object-inheritance
 - Construct new objects from other objects via object composition
- New languages with a full object-oriented type system are waning
 - Of the three "big" new languages (Rust, Go, Swift) only Swift supports OO with objectinheritance, the others are object-based.

Structures

```
-- rectangle structure
load system io.

structure Rectangle with
data xdim.
data ydim.
end
Default Constructor

let r = Rectangle(4,2).
io @println ("Rectangle with x="+tostring(r@xdim)+" and y="+tostring(r@ydim)).
```

- Structures consist of 'data' fields and are associated with a default constructor
- Member access is via the '@' operator

Structures

```
-- rectangle structure
load system io.
structure Rectangle with
  data xdim.
  data ydim.
 -- member function
  function area with () do
    return this@xdim * this@ydim.
  end
end
                         Object member access
let r = Rectangle(4,2).
                                 Member function call
let x = tostring(r@xdim).
let y = tostring(r@ydim).
let area = tostring(r@area()).
io @println ("The area of rectangle <" + x + "," + y + "> is " + area).
```

In002/rect-OO.ast

- Member functions
- Object identity is given with the 'this' keyword
- Member functions are called on objects with the '@' operator
 - E.g., r@area()

Structures: Rust & Go

```
struct Rectangle {
    width: u32,
    height: u32,
}

impl Rectangle {
    fn area(&self) -> u32 {
        self.width * self.height
    }
}
```

Rust

```
type rect struct {
    width int
    height int
}

func (r *rect) area() int {
    return r.width * r.height
}
```

Asteroid Exercises

- Ex1: Write an Asteroid program that prints out the integers 10 through 1.
- Ex2: Write an Asteroid program that has a structure for the type 'Circle' that holds the coordinates of the center of a circle and its radius.
 - Your program should instantiate a number of different circle objects and print them out using 'io @println'.
 - Add a member function to your Circle structure that computes the circumference of the given circle using 2*pi*r. Your program should instantiate a number of circles and print out their circumference.

- Asteroid has a set of primitive data types:
 - integer
 - real
 - string
 - boolean
- Asteroid does not order these data types into a type hierarchy like Java, Python, or C. In that it closely aligns itself with languages like Rust and ML.

(more on type hierarchies later)

- Asteroid has two more built-in data types:
 - list
 - tuple
- These are structured data types in that they can contain entities of other data types.

```
lutz$ asteroid
Asteroid 2.0.1
(c) University of Rhode Island
Type "help" for additional information
ast> let a = [1,2,3].
ast> let s = ["hello", "world"].
ast>
                        Asteroid 2.0.1
                        (c) University of Rhode Island
                        Type "help" for additional information
                        ast> structure Person with
                               data first name.
                               data last name.
                        ... end
                        ast> let people = [Person("Joe", "Smith"), Person("Helen", "Jackson")].
                        ast>
```

- Using the 'structure' keyword Asteroid also supports user defined types.
 - The name of the structure becomes a new type available in the program.

- Finally, Asteroid supports one more type, namely the none type.
 - The none type has a constant named conveniently 'none'.
 - The empty pair of parentheses () can be used as a short-hand for the constant none.

Running Asteroid

- Install the interpreter on your machine
 - See https://asteroid-lang.org

Assignments

- Reading: MPL Chap 6
- Do Assignment #1 see BrightSpace