

Asteroid Basics

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

Imperative Asteroid

- The “Hello World” program...

Load system module

Sentence terminator

```
1 load system io.  
2  
3 io @println("Hello World!").
```

In002/hello.ast

Iteration

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

```
-- compute the factorial

load system io.

function fact with n do
  let val = 1.
  while n > 1 do
    let val = val*n.
    let n = n-1.
  end
  return val.
end

let x = tointeger(io @input("Enter a positive integer: ")).
io @println ("The factorial of " + tostring(x) + " is " + tostring(fact x)).
```

Function argument

Iteration

Assignment

Type conversion

Function Call

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 l do
  loop
    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 l.
end

let k = [6,5,3,1,8,7,2,4].
io @println("unsorted array: "+toString(k)).
io @println("sorted array: "+toString(bubblesort k)).
```

Element access



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 object-inheritance, the others are object-based.

Structures

```
-- rectangle structure
load system io.

structure Rectangle with
  data xdim.
  data ydim.
end

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

Diagram illustrating member access in the code:

- An arrow points from the text "Default Constructor" to the function call `Rectangle(4,2)`.
- An arrow points from the text "Member access" to the expressions `r@xdim` and `r@ydim` in the `println` statement.

- 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

let r = Rectangle(4,2).
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).
```

Object member access

Member function call

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  
}
```

Go

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.
 1. Your program should instantiate a number of different circle objects and print them out using 'io @println'.
 2. Add a member function to your Circle structure that computes the circumference of the given circle using $2 \cdot \pi \cdot r$. Your program should instantiate a number of circles and print out their circumference.

Types in Asteroid

- 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)

Types in Asteroid

- 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>
```

Types in Asteroid

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

```
1  -- user defined types~  
2  structure Person with~  
3  ..data name.~  
4  ..data profession.~  
5  end~  
6  ~  
7  let p:%Person = Person("Fred","Carpenter").~
```

↑
Type Pattern

Types in Asteroid

- 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>
- Note: Windows users will have to make sure that the pyreadline3 module is installed on their machine
 - <https://pypi.org/project/pyreadline3/>

Assignments

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