

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!").
```

Iteration

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

```
1  -- compute the factorial
2
3  load system io.
4  load system type.
5
6  function fact with n do
7    let val = 1.
8    while n > 1 do
9      let val = val*n.
10     let n = n-1.
11   end
12   return val.
13 end
14
15 let x = type @tointeger(io @input("Enter a positive integer: ")).
16 io @println ("The factorial of "+x+" is "+(fact x)).
```

Function argument

Iteration

Assignment

Type conversion

Function Call

Function Calls

- In Asteroid function calls are constructed by juxtapositioning 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

```
1  -- the bubble sort
2  load system io.
3
4  function bubblesort with l do
5
6      loop
7          let swapped = false.
8          for i in 0 to len(l)-2 do
9              if l@(i+1) <= l@i do
10                 let (l@i,l@(i+1)) = (l@(i+1),l@i).
11                 let swapped = true.
12             end
13         end
14         if not swapped do
15             break.
16         end
17     end
18
19     return l.
20 end
21
22 let k = [6,5,3,1,8,7,2,4].
23 io @println("unsorted array: "+k).
24 io @println("sorted array: "+(bubblesort k)).
```



Structures & Objects

- Asteroid is object-based
- Bundle operations with data
- No 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 inheritance, the others are object-based.


Structures

```
1  -- rectangle structure
2  load system io.
3
4  structure Rectangle with
5      data xdim.
6      data ydim.
7  end
8
9  let r = Rectangle(4,2).
10 io @println ("Rectangle with x="+r@xdim+" and y="+r@ydim).
```

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

Structures

```
1  -- rectangle structure
2  load system io.
3  load system type.
4
5  structure Rectangle with
6    data xdim.
7    data ydim.
8
9    -- member function
10   function area with () do
11     return this@xdim * this@ydim.
12   end
13 end
14
15 let r = Rectangle(4,2).
16 io @println ("The area of Rectangle (" + r@xdim + ", " + r@ydim + ") is " + r@area()).
```



- 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 the 'io @println' statement.
 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.

<https://replit.com/@lutzhamel/asteroid-csc301>

Types in Asteroid

- Asteroid has a set of **primitive data types**:
 - integer
 - real
 - string
 - boolean
- Asteroid arranges these data types in a **type hierarchy** in order to facilitate automatic type promotion:

$\text{boolean} < \text{integer} < \text{real} < \text{string}$

(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.
- 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.

```
let l = [6,5,3,1,8,7,2,4].  
println("unsorted array: "+l).
```



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 Constraint
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.
 - The none data type does not belong to any type hierarchy.

Running Asteroid

- Install the interpreter on your machine
 - See <https://asteroid-lang.org>
- Run Asteroid in the cloud
 - <https://replit.com/@lutzhamel/asteroid-csc301>

Assignments

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