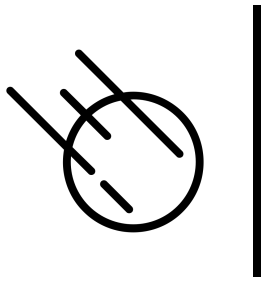


Data Structures

- We saw that Asteroid has built-in data structures such as lists and tuples
- Let's look at these a bit closer



Lists

```
load system io.  
  
let a = [1,2,3].      -- construct list a  
let b = [a@2, a@1, a@0]. -- reverse list a  
io @println b.
```

In003/reverse1.ast

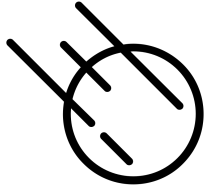
```
load system io.  
  
let a = [1,2,3].      -- construct list a  
let b = a@[2,1,0].    -- reverse list a using slice [2,1,0]  
io @println b.
```

In003/reverse2.ast

```
load system io.  
  
let a = [1,2,3].  
let b = a @reverse (). -- reverse list using member function 'reverse'  
io @println b.
```

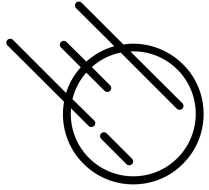
In003/reverse3.ast

In Asteroid lists are considered objects with member functions.



The General Access Operator

- The @ operator is Asteroid's general access operator:
 - individual elements, slices, or member functions of lists.
 - members and functions of tuples and objects.
- The println function:
 - the io module is an object and println is a member function, therefore
io @println <string>
 - In Asteroid all system modules are objects

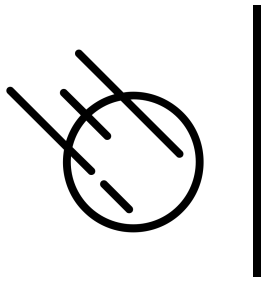


List Comprehensions

- In Asteroid a list comprehension consist of a range specifier together with an optional step specifier allowing you to generate a list of integer values within that range,
[<start> to <end>]
or
[<start> to <end> step <value>]
- If a comprehension is invalid Asteroid returns an empty list, e.g.
[0 to 4 step -1]

```
load system io.  
  
-- build a list of odd values  
let a = [1 to 10 step 2]. -- list comprehension  
io @println ("list: " + a).  
  
-- reverse the list using a slice computed as comprehension  
let slice = [4 to 0 step -1]. -- list comprehension  
let b = a@slice.  
io @println ("reversed list: " + b).
```

In003/comprehension.ast



Tuples

```
-- build a list of tuples
let b = [("a","b","c"),
         ("d","e","f"),
         ("g","h","i")].
-- Access an element in the nested structure.
assert (b@0@1 == "b").
```

In003/tuples1.ast

```
load system io.

let b = ("a","b","c"). -- build a tuple

try
|   | let b@1 = "z". -- attempt to modify an element in the tuple
catch Exception (kind,message) do
|   | io @println (kind+": "+message).
end.
```

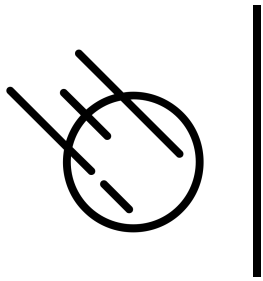
In003/tuples2.ast

Tuples are immutable objects!



Data Structures

- Asteroid also support custom data structures via the 'structure' keyword
- Structures allow us to instantiate objects with a particular internal structure



Structures & Objects

```
load system type.

-- define a structure of type A
structure A with
  data a.
  data b.
end

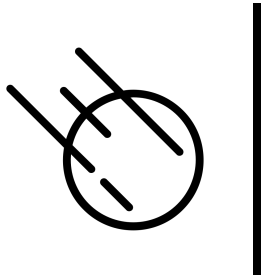
let obj = A(1,2).    -- default constructor, a<-1, b<-2

-- show that 'obj' is of type 'A'
assert (type @gettype obj == "A").

-- access the components of the new data type
assert (obj@a == 1). -- access first data member
assert (obj@b == 2). -- access second data member
```

In003/struct1.ast

- Structures in Asteroid are similar to classes in Python and almost identical to structures in Rust.
- A structure introduces a data structure as a new type
- For each structure Asteroid creates a **default constructor**



Reading

- Data Structures
 - asteroid-lang.readthedocs.io/en/latest/User%20Guide.html#data-structures