Object-Oriented Programming with Asteroid

- Structures with behavior
  - No inheritance
  - No member protection, everything is public
- Member function specification
  - Uses standard function syntax within structures
  - Internal object identity is given via the ‘this’ keyword.
  - Special member functions:
    - __init__
    - __str__
Object Identity

Internal and external object identities are the same

```
structure A with
    function identity with none do
        return this.
    end
end

let o = A().
-- getid maps an object to a unique identifier
assert (getid(o) == getid(o @identity ())).
```

ln009/objid.ast
Basic Objects with Behavior

- Data and function members
- Member functions are functions defined in the context of a structure.
- Notice the use of ‘this’
- We are using the default constructor that fills out the data members according to the order they appear.
- Taking advantage of default behavior when mapping object to a string.

```
structure Rectangle with
    data xdim.
    data ydim.
    function area with () do
        return this@xdim * this@ydim.
    end
end

load system type.
let r = Rectangle(3,2).
assert (r @area () == 6).
assert (type @tostring r == "Rectangle(3,2)").
```
Custom Constructors and String Mapping

- Taking advantage of the special functions `__init__` and `__str__`
- We use the constructor `__init__` to enforce that we only want real values for dimensions
- The `__str__` functions allows us to create a custom string representation for Rectangle objects

```
structure Rectangle with
  data xdim.
  data ydim.
  function __init__ with (xdim:%real,ydim:%real) do
    let this@xdim = xdim.
    let this@ydim = ydim.
  end
  function area with () do
    return this@xdim * this@ydim.
  end
  function __str__ with () do
    return "Rectangle with dimension "+this@xdim+"x"+this@ydim.
  end
end

load system type.
let r = Rectangle(3.0,2.0).
assert (r @area () == 6.0).
assert (type tostring r == "Rectangle with dimension 3.0x2.0").
```
Pattern Matching on Objects

- During pattern matching on objects member functions are ignored
  - It doesn’t matter where the functions appear.
  - You cannot pattern-match on functions!
  - You can only pattern-match on data members.

```plaintext
load system io.

structure Person with
  data name.
  data age.
  function hello with none do
    io @println ("Hello, my name is "+this@name).
  end
end

-- functions are ignored during pattern matching
let Person(name,age) = Person("Scarlett",28).
assert(name == "Scarlett").
assert(age == 28).
```
Pattern Matching on Objects

- It is not a surprise that object patterns can be used as constraints.
- All patterns we have looked at so far also apply to objects.

```plaintext
load system io.

structure Person with
  data name.
  data age.
  function hello with none do
    io @println ("Hello, my name is "+this@name).
  end
end

-- pattern match only successful for objects with
-- names that contain two lower case t's
let scarlett:Person(".*t.*t.*",_)= Person("Scarlett",28). scarlett @hello ().
```
Object Composition

- We already looked at object composition as a way of modeling compound objects.
- Note: in Asteroid we can have nested objects but not nested structures.

```plaintext
structure Address with
  data street.
  data city.
  data state.
  data zip.
end

structure Person with
  data name.
  data age.
  data address.
end

let address = Address("123 Main St", "Anytown", "CA", "12345").
let person = Person("John Doe", 30, address).

-- complete destructuring of the person object
-- => pattern matching on nested objects
let Person(name, age, Address(stree, city, state, zip)) = person.
```
Duck Typing

As long as objects have common interfaces they act as polymorphic structures – duck typing.

The do not need be related via a common supertype.

```
load system io.

structure Circle with
data name.
  -- draw interface
  function draw with () do
    io @println ("Drawing a circle "+this@name).
  end
end

structure Square with
  data name.
  -- draw interface
  function draw with () do
    io @println ("Drawing a square "+this@name).
  end
end

let v = [].
v @append (Circle("Circle1"))
v @append (Square("Square1"))
v @append (Circle("Circle2"))
for i in range (len v) do
  v@i @draw ().
end
```
String & List Objects

- In Asteroid, similar to Python, strings and lists are considered objects and have member functions.

```plaintext
-- calling member function 'flip' on a string
let s = "abc" @flip ()
assert (s == "cba").

-- check where 2 is on the list using the
-- member function 'index'
assert ([1,2,3,4] @index 2 == 1).
```