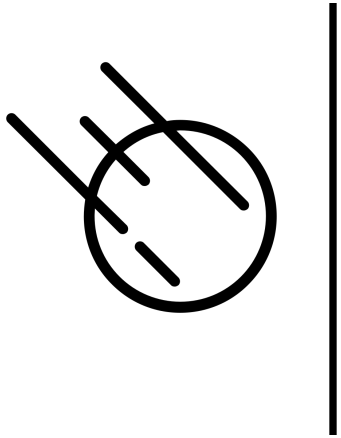


Asteroid

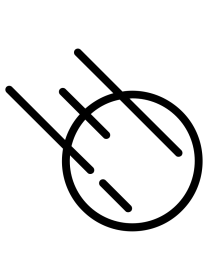
The Programming Language

Dr Lutz Hamel

Dept. of Computer Science & Statistics

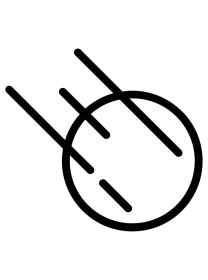
University of Rhode Island

asteroid-lang.org



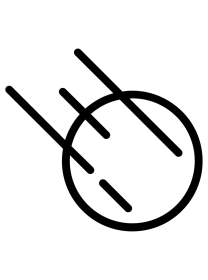
Asteroid: The Programming Language

- The Asteroid programming language is,
  - modern
  - application-oriented
  - open-source
  - dynamically typed
  - multi-paradigm
  - heavily influenced by Python, Rust, ML, and Prolog
  - currently under development at the University of Rhode Island
- Project page: https://asteroid-lang.org
- A cloud-based version is available for this talk: https://replit.com/@lutzhamel/asteroid-talk-f22
- Documentation: https://asteroid-lang.readthedocs.io



Design Objectives

- Seamless integration of imperative, functional, and object-oriented programming.
- Full support of first-class patterns.
- Expressive, conversational syntax geared towards use in a classroom setting.



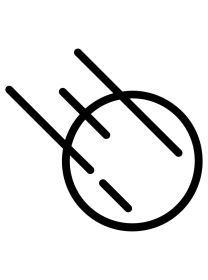
“Hello, World!”

- As is tradition when looking at a new programming language…hello world…

```
1  -- the customary hello world program
2
3  load system io.
4
5  io @println "Hello World!".
```

hello.ast

```
> asteroid hello.ast
Hello World!
>
```



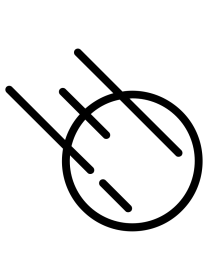
Imperative Programming

- Should look familiar.
- Here is an imperative version of computing a factorial…

```
1  -- compute the factorial iteratively
2  -- this program makes use of the fact that multiplication
3  -- is commutative, i.e. 1*3*2 = 3*2*1
4
5  -- load modules
6  load system io.
7  load system type.
8
9  -- our factorial function
10 function fact with n do
11   let val = 1.
12   while n > 1 do
13     let val = val*n.
14     let n = n-1.
15   end
16   return val.
17 end
18
19 -- talk to the user
20 let x = type @tointeger (io @input "Enter a positive integer: ").
21 io @println ("The factorial of "+x+" is "+(fact x)).
```

fact-iter.ast

```
> asteroid fact-iter.ast
Enter a positive integer: 3
The factorial of 3 is 6
>
```



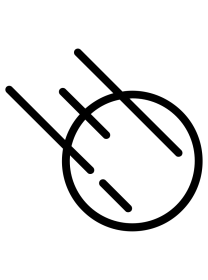
Imperative Programming

- Something a bit more interesting – the bubble sort.
- Note the access operator '@' for list element access.
- '@' is a universal access operator:
  - Member functions
  - Tuple components
  - List elements

bubble.ast

```
asteroid bubble.ast
unsorted list: [6,5,3,1,8,7,2,4]
sorted list: [1,2,3,4,5,6,7,8]
```

```
1  -- the bubble sort
2  load system io.
3
4  -- sort list l in ascending order
5  function bubblesort with l do
6    loop -- forever
7      let swapped = false.
8      for i in 0 to len(l)-2 do
9        if l@(i+1) < l@i do -- out of order
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. -- done!
16     end
17   end -- loop
18   return l.
19 end
20
21 -- sort a list
22 let k = [6,5,3,1,8,7,2,4].
23 io @println ("unsorted list: "+k).
24 io @println ("sorted list: "+(bubblesort k)).
```

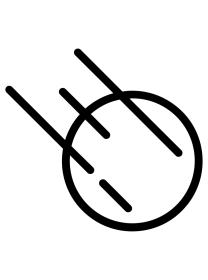


Strongly Typed

- Asteroid supports several type hierarchies,

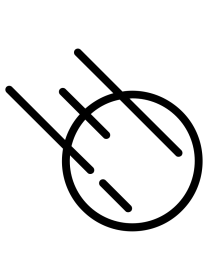
  boolean < integer < real < string
  list < string
  tuple < string
  none (or ‘()’)

- These are all built-in types.
- User defined types are introduced with the ‘structure’ keyword (more on that later).
  - User defined types do not belong to any hierarchy
- No generics,
  - Dynamic typing together with duck typing cover most of the use cases of generics in Asteroid.



Functional Programming

- Asteroid has a complete functional sublanguage.
  - ‘asteroid –F’ turns the Asteroid interpreter into a functional language interpreter.
  - Lisp/Scheme style functional programming – no monads or algebraic data types here.
  - But Asteroid offers pattern-matchable objects similar to Rust.



Functional Programming

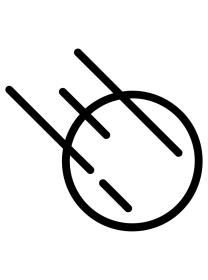
- The functional version of the factorial computation…

fact-rec.ast

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

Multi-dispatch with pattern matching on function arguments.

```
> asteroid -F fact-rec.ast
Enter a positive integer: 3
The factorial of 3 is 6
>
```



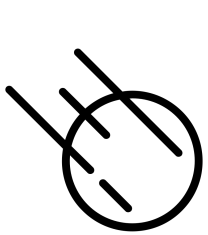
Functional Programming

- Something a bit more interesting – the quick sort
- Note:
  - [1|[2,3]] = [1,2,3]

qsort-fun.ast

```
> asteroid -F qsort-fun.ast
[0,1,2,3]
>
```

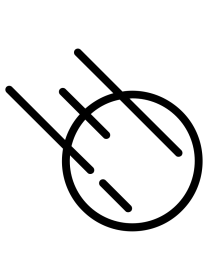
```
1  -- functional implementation of quicksort
2
3  load system io.
4
5  function qsort
6    with [] do -- empty list
7      []
8    with [a] do -- single element list
9      [a]
10   with [pivot|rest] do -- head-tail operator
11     function filter -- local function
12       with (e,[],fcmp) do
13         []
14       with (e,[a|rest],fcmp) do
15         [a]+filter(e,rest,fcmp)
16           if fcmp(a,e)
17           else filter(e,rest,fcmp)
18     end
19     let less=filter(pivot,rest,lambda with (x,y) do x < y).
20     let more=filter(pivot,rest,lambda with (x,y) do x >= y).
21     qsort less + [pivot] + qsort more.
22 end
23
24 io @println (qsort [3,2,1,0]).
```



Multi-Paradigm Programming

- Asteroid allows you to “mix ‘n match” paradigms.
- E.g. in the QuickSort we keep the functional multi-dispatch with structural pattern matching but replace the ‘filter’ functions with a ‘for’ loop from the imperative paradigm.
- Our experience is that the various paradigms complement each other in a very natural way.

```
1  -- Quicksort
2
3  load system io.
4
5  function qsort
6    with [] do -- empty list
7      [].
8    with [a] do -- single element list
9      [a].
10   with [pivot|rest] do -- head-tail operator
11     let less=[].
12     let more=[].
13     for e in rest do -- iteration instead of recursion
14       if e < pivot do
15         let less = less + [e].
16       else do
17         let more = more + [e].
18       end
19     end
20     qsort less + [pivot] + qsort more.
21 end
22
23 io @println (qsort [3,2,1,0]).
```



Function Calls

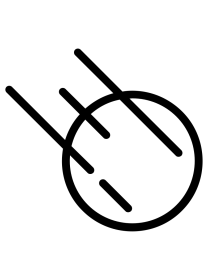
- In the functional programming tradition, Asteroid's function calls are constructed by juxtaposing a function with a value, e.g.

  ```
  fact 3.
  ```

- The implication is that all functions have only a single argument. If you want to pass more than one value to a function you have to construct a *tuple of values*, e.g.

  ```
  foo (1,2).
  ```

- Syntactically this looks the same as a function call to foo in Python but semantically it is very different – call foo with the *value* (1,2) in Asteroid as apposed to call foo with the *list* of values (1,2) in Python.
- This slight change of perspective enables effective pattern matching in the multi-dispatch within function definitions in Asteroid.



Function Calls

- The interpretation of function arguments as a list of values has unexpected implications in Python
  - foo (1,2) ≠ foo ((1,2)), but
  - (1,2) = ((1,2))
- Inconsistent handling of parenthesized tuples!

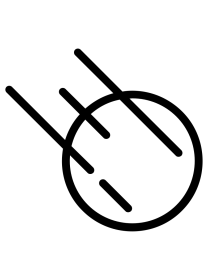
```
Python 3.8.11 (default, Jun 28 2021, 10:57:31)
[GCC 10.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> def foo(a,b):
...     pass
...
>>> foo (1,2)
>>> foo ((1,2))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: foo() missing 1 required positional argument: 'b'
>>>
```

but…

```
>>> (1,2) == ((1,2))
True
>>>
```

In Asteroid it works as expected…

```
Asteroid Version 1.1.2
Run "asteroid -h" for help
Press CTRL+D to exit
> function foo with (a,b) do . end
> (1,2) == ((1,2))
true
> foo (1,2)
> foo ((1,2))
>
```



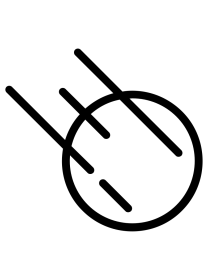
Higher-Order Programming

- Asteroid implements a very clean and intuitive framework for higher-order programming, e.g. the ‘map’ function
  - A program that creates a list of alternating positive and negative ones.
  - The list constructor [1 to 10] constructs a list of values [1, 2,...,10].
  - The first map turns this list into the list [1,0,1,...0].
  - The second map turns that list into the list [1,-1,1,-1,...,-1].

map.ast

```
1  -- higher-order programming with the 'map' function
2  -- this program that creates a list of alternating 1 and -1
3
4  load system io.
5  load system math.
6
7  let a = [1 to 10] @map (lambda with x do math @mod (x,2))
8                    @map (lambda with x do 1 if x else -1).
9
10 io @println a.
```

```
> asteroid -F map.ast
[1,-1,1,-1,1,-1,1,-1,1,-1]
>
```



Higher-Order Programming

- The improvements in the conceptual framework for higher-order programming in Asteroid are non-trivial.
- Let's try the same program in Python…

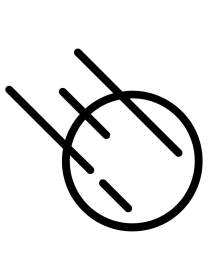
```
> python map.py
[1, -1, 1, -1, 1, -1, 1, -1, 1, -1]
>
```

Python:

```
1  l = [x for x in range(1,10+1)]
2  iter = map(lambda x : x%2, l)
3  out = list(map(lambda x : 1 if x else -1, iter))
4  print(out)
```

Compared to Asteroid:

```
3
4  load system io.
5  load system math.
6
7  let a = [1 to 10] @map (lambda with x do math @mod (x,2))
8                    @map (lambda with x do 1 if x else -1).
9
10 io @println a.
```



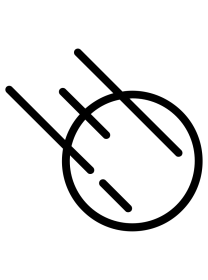
Structures

- Like in many modern programming languages such as Rust and Go, classes have given way to structures with member functions in Asteroid,
  - No member protection
  - No inheritance
  - But object identity (‘this’)

```
> asteroid rect.ast
The area of Rectangle(4,2) is 8
>
```

rect.ast

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



Structures

Asteroid

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

Go

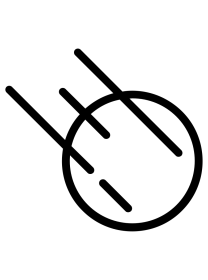
```
type rect struct {
    width int
    height int
}

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

Rust

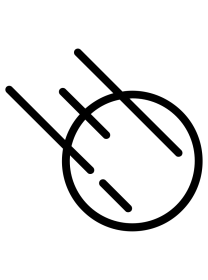
```
struct Rectangle {
    width: u32,
    height: u32,
}

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



First-Class Patterns

- The support of first-class patterns implies that patterns can be
  - stored in variables
  - passed to/from functions
- Asteroid implements the idiom
  - *Patterns as values and values as patterns*

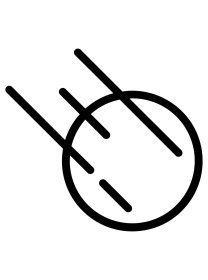


First-Class Patterns

- In classical pattern matching patterns are syntactically static – consider the quick sort

qsort-fun.ast

```
1  -- functional implementation of quicksort
2
3  load system io.
4
5  function qsort
6    with [] do -- empty list
7      []
8    with [a] do -- single element list
9      [a]
10   with [pivot|rest] do -- head-tail operator
11     function filter -- local function
12       with (e,[],fcmp) do
13         []
14       with (e,[a|rest],fcmp) do
15         [a]+filter(e,rest,fcmp)
16           if fcmp(a,e)
17           else filter(e,rest,fcmp)
18     end
19     let less=filter(pivot,rest,lambda with (x,y) do x < y).
20     let more=filter(pivot,rest,lambda with (x,y) do x >= y).
21     qsort less + [pivot] + qsort more.
22 end
23
24 io @println (qsort [3,2,1,0]).
```



Pattern Reuse

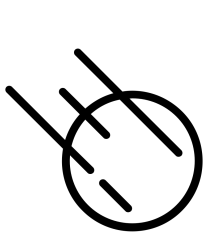
- First-class patterns are values and therefore dynamic in their nature
- First-class patterns enable pattern reuse

n:*pos_int ≡ n if n is *pos_int

fact-pat.ast

```
1  -- first-class patterns: pattern reuse
2  load system io.
3
4  let pos_int = pattern x if (x is %integer) and (x > 0).
5  let neg_int = pattern x if (x is %integer) and (x < 0).
6
7  function fact
8    with 0 do
9      1
10   with n:*pos_int do
11     n*fact(n-1)
12   with *neg_int do
13     throw Error "negative values not supported".
14 end
15
16 function sign
17   with 0 do
18     1
19   with *pos_int do
20     1
21   with *neg_int do
22     -1
23 end
24
25 io @println (fact 3).
26 io @println (sign 3).
```

```
> asteroid fact-pat.ast
6
1
>
```

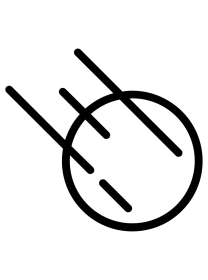


Pattern Factoring

- Patterns can become quite complex, first-class patterns allow us to break patterns into smaller chunks.
- In the process we can also give sub-patterns meaningful names making pattern expressions much more comprehensible.

```
1  -- first-class patterns: factoring
2
3  load system io.
4
5  -- without first-class patterns
6  function foo1 with (x if (x is %integer) or (x is %real), y) do
7    io @println (x,y).
8  end
9
10 -- with first-class patterns
11 let scalar = pattern v if (v is %integer) or (v is %real).
12
13 function foo2 with (x:*scalar, y) do
14   io @println (x,y).
15 end
16
17 foo1 (1,2).
18 foo2 (1,2).
```

factoring.ast



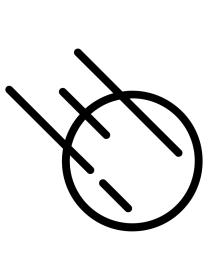
Enhancing Type Systems

- First-class patterns can act like types
- Here we use the first-class pattern ‘Shape’ to define a subtype polymorphic function

basetype.ast

```
asteroid basetype.ast
circle: 10
rectangle: 5, 20
```

```
1  -- first-class patterns: subtype polymorphism
2  load system io.
3
4  structure Circle with
5     data radius.
6     function print_shape with () do
7       io @println ("circle: "+this@radius).
8     end
9  end
10
11 structure Rectangle with
12    data a.
13    data b
14    function print_shape with () do
15      io @println ("rectangle: "+this@a+", "+this@b).
16    end
17 end
18
19 -- define abstract base type Shape with Circle and Rectangle as subtypes
20 let Shape = pattern x if (x is %Circle) or (x is %Rectangle).
21
22 -- define function in terms of the abstract base type Shape
23 -- the argument of the function is restricted to values described
24 -- by the pattern Shape
25 function print_any with obj:*Shape do
26    obj @print_shape ().
27 end
28
29 -- the function print_any is subtype polymorphic
30 print_any (Circle(10)).
31 print_any (Rectangle(5,20)).
```

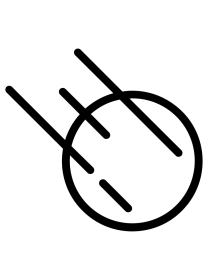


Structures and Objects

- In Asteroid the ‘let’ statement is a pattern-match statement of the form,
  let <pattern>=<value>
- We pattern-match objects for data decomposition!

```
1  -- pattern matching on objects
2
3  structure A with
4    data a.
5    data b.
6  end
7
8  let o = A(1,2).  -- call constructor
9  let A(x,y) = o.  -- pattern match object o
10 assert(x==1 and y==2).
```

struct-pat.ast

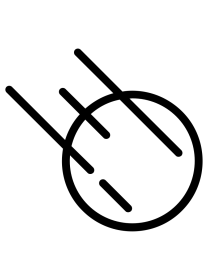


Structures and Objects

- Here are some fun pattern-match identities on objects using first-class and static patterns.

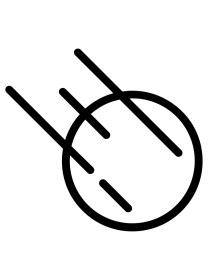
```
1   -- some pattern-match identities on objects
2
3   structure A with
4      data a.
5      data b.
6   end
7
8   let A(1,2) = A(1,2). -- pattern A(1,2) matching new object A(1,2)
9   let o = A(1,2).      -- variable o as pattern for new object A(1,2)
10  let *o = A(1,2).     -- object o as pattern matching new object A(1,2)
11  let *o = o.          -- object o as pattern matching object o
```

ident.ast



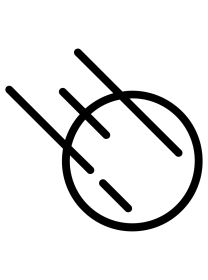
Asteroid in the Classroom

- In CSC301 (Foundations of PLs) I use Asteroid mostly to teach functional programming concepts,
  - “Everything is a Value”
  - Higher-order programming
  - Pattern matching
- In CSC493 (Multi-Paradigm Programming) we look deeply into the different programming paradigms and study how they interact
  - In particular, we look at the effect first-class patterns have on programming in general
- The fact that Asteroid is dynamically typed and basically looks familiar to most students let’s us get to the interesting bits very in functional programming quickly…
  - …in contrast to using something like Haskell or ML where we would have to wrangle the type system in non-trivial ways before we get to the interesting bits.
  - …or Lisp/Scheme where we would have to wrestle the uncommon syntax before we get to the interesting bits.



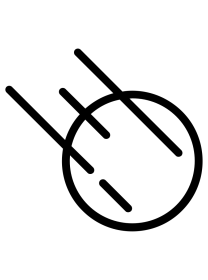
Future Work

- Near term,
  - We are developing a compiler for Asteroid that produces native code.
  - Key to this development is the Asteroid Virtual Machine (AVM) framework.
- Long term,
  - Asteroid has a niche as a development platform for performant programs within the WebAssembly (https://webassembly.org) framework geared toward frontend developers that are not used to working in C or Rust.



Thank You!

- I wanted to take this opportunity to thank the folks who have contributed to this project over the years, in particular,
  - Ariel Finkle
    Calvin Higgins
    Christian Tropeano
    Oliver McLaughlin
    Theodore Henson
    Timothy Colaneri
- If you are interested in programming language design and implementation, we are always looking for contributors!



Questions?

- lutzhamel@uri.edu
- or stop by my office for a chat.

- Homepage
  - https://asteroid-lang.org
- Example code at
  - https://replit.com/@lutzhamel/asteroid-talk-f22