Asteroid Functions

- 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.
- As we will see, this slight change of perspective enables effective pattern matching within function definitions in Asteroid.

Lambda Calculus

- The mathematical idea of function application to values was used by the logician Alonzo Church to create the lambda calculus as a computational foundation of mathematics in the 1930’s.
- It can be considered as an alternative to the Turing machine
- It is Turing-complete
  - Anything a TM can compute can also be computed with the lambda calculus
- It is considered the semantic foundation of our modern functional languages such as Haskell, Ocaml, Clojure, etc

https://en.wikipedia.org/wiki/Lambda_calculus

Lambda Calculus

- Here is an example of an increment function as a lambda expression applied to a value,

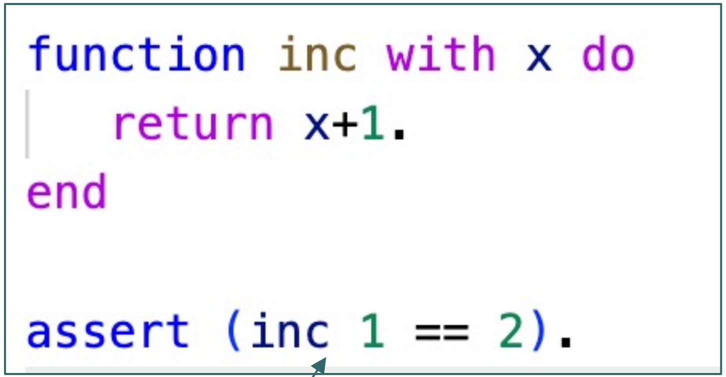
$$(\lambda x. x + 1)\; 1 \;\Longrightarrow\; 2$$

Lambda Calculus

- Another example that scales a point in 2D space (a pair of values),

$$(\lambda(x, y).(2x, 3y))\ (1,2)\ \Longrightarrow (2,6)$$

Asteroid Functions



```
function inc with x do
   return x+1.
end

assert (inc 1 == 2).
```

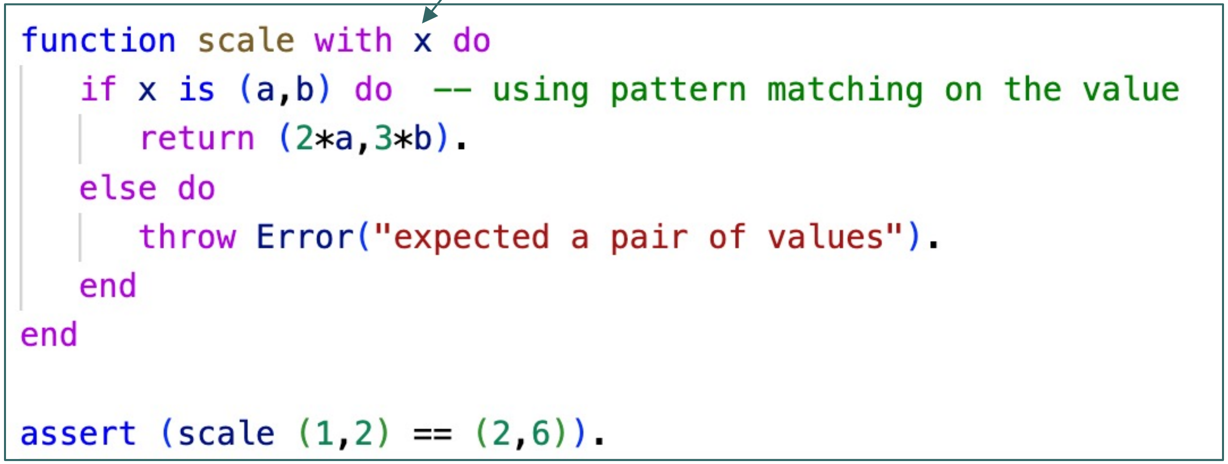
ln006/inc1.ast

Function call via juxtaposition

Asteroid Functions

- In Asteroid functions have only a single formal parameter,

Single, formal parameter



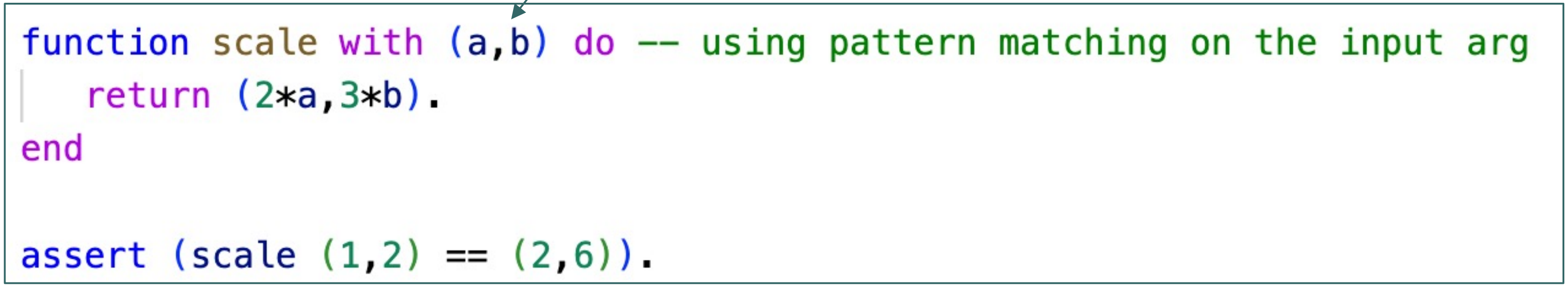
```
function scale with x do
   if x is (a,b) do  -- using pattern matching on the value
      return (2*a,3*b).
   else do
      throw Error("expected a pair of values").
   end
end

assert (scale (1,2) == (2,6)).
```

In006/scale1.ast

Asteroid Functions

- We can pattern match on the single formal parameter,

Single, formal parameter pattern matched

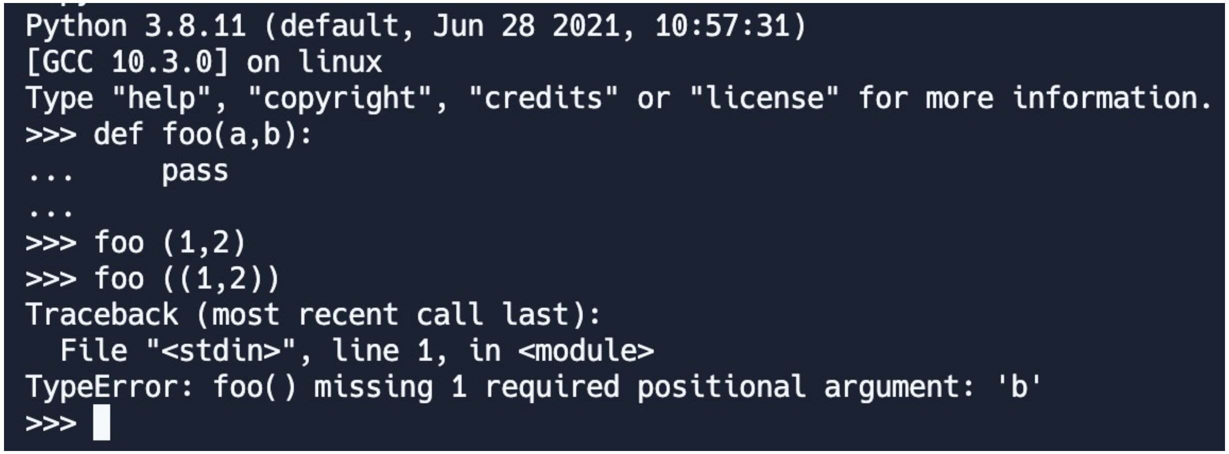


```
function scale with (a,b) do -- using pattern matching on the input arg
   return (2*a,3*b).
end

assert (scale (1,2) == (2,6)).
```

ln006/scale2.ast

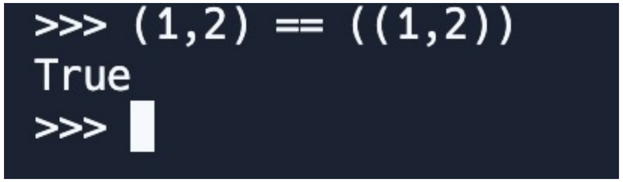
Function Calls

- 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'
>>> █
```

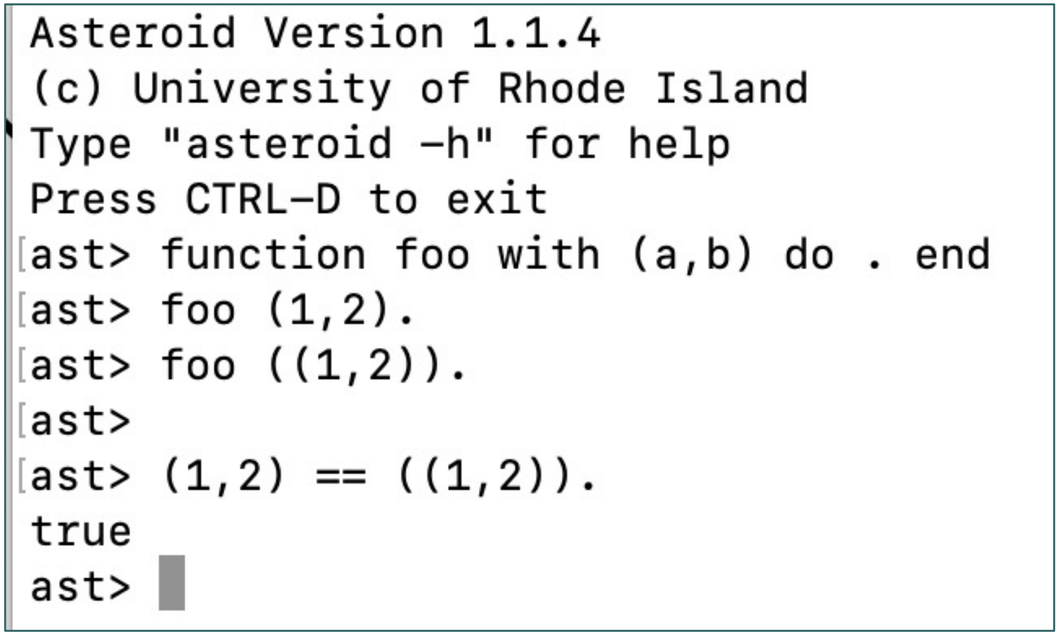
but…



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

Function Calls

- But it works fine in Asteroid,

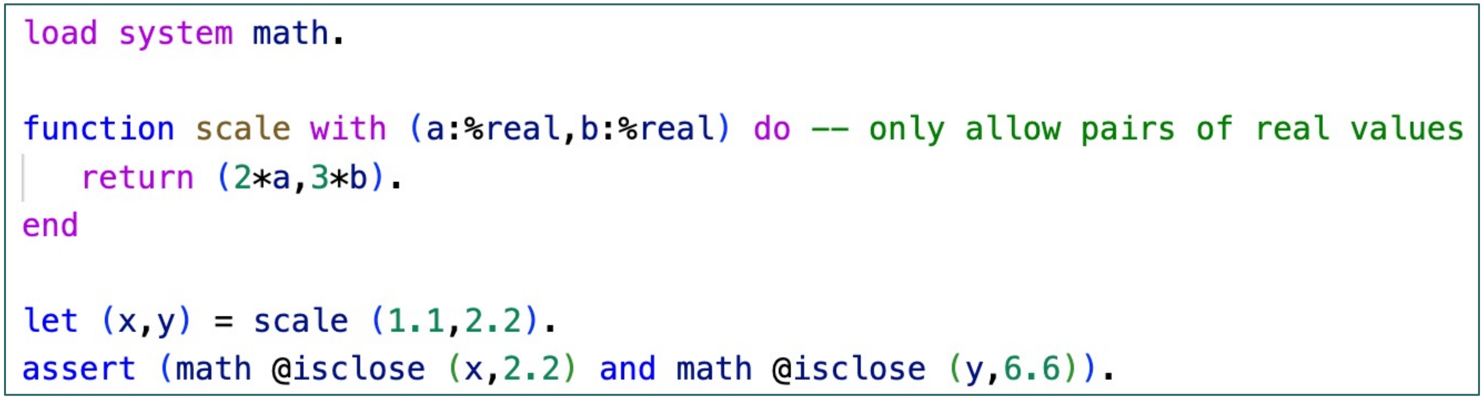


```
Asteroid Version 1.1.4
(c) University of Rhode Island
Type "asteroid -h" for help
Press CTRL-D to exit
[ast> function foo with (a,b) do . end
[ast> foo (1,2).
[ast> foo ((1,2)).
[ast>
[ast> (1,2) == ((1,2)).
true
ast> █
```

Pattern Matching in Functions

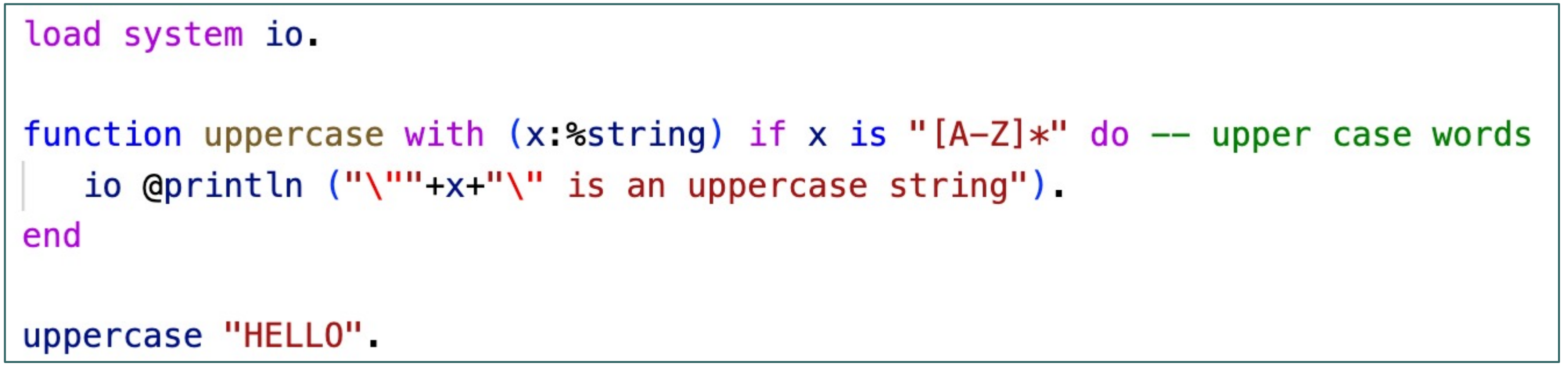
- As we have seen, we can pattern match on the function argument
- That means we can use all the patterns we have learned so far

ln006/scale3.ast



```
load system math.

function scale with (a:%real,b:%real) do -- only allow pairs of real values
    return (2*a,3*b).
end

let (x,y) = scale (1.1,2.2).
assert (math @isclose (x,2.2) and math @isclose (y,6.6)).
```

ln006/string1.ast



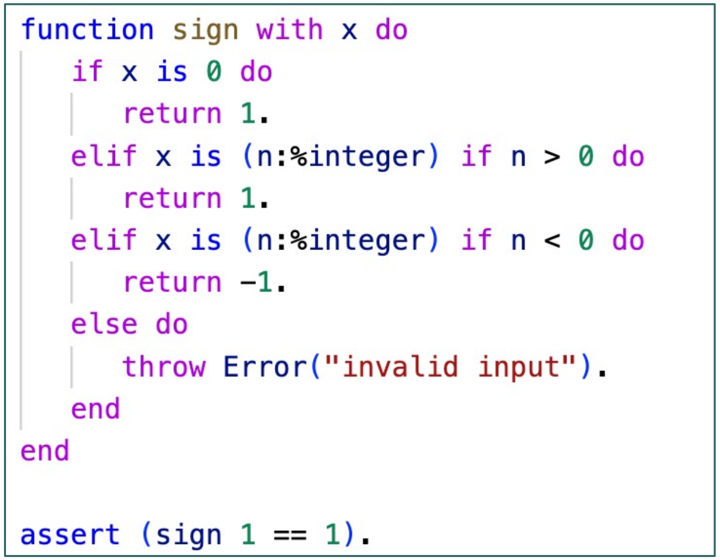
```
load system io.

function uppercase with (x:%string) if x is "[A-Z]*" do -- upper case words
    io @println ("\""+x+"\" is an uppercase string").
end

uppercase "HELLO".
```

Functions are Multi-Dispatch

- In Asteroid functions are multi-dispatch:
  - a single function can have multiple bodies each attached to a different pattern matching the actual argument.
- This is along the line of declarative programming
  - Highlight programmer’s intention instead of computational logic

Functions are Multi-Dispatch

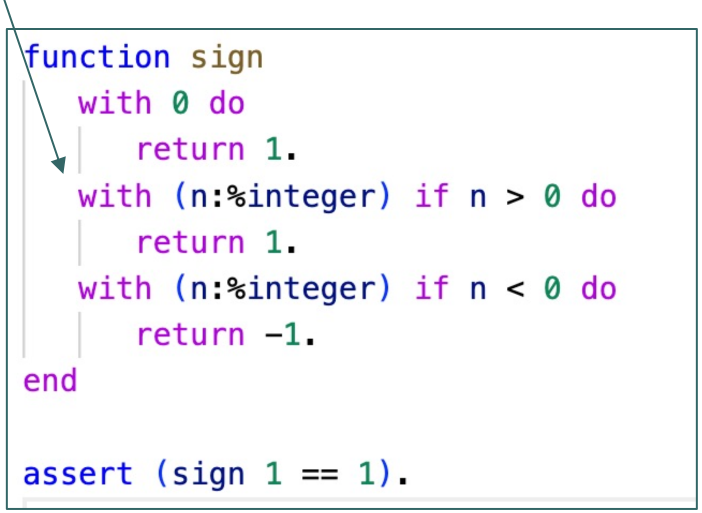
$$sign(x) = \begin{cases} 1\ if\ x = 0 \\ 1\ if\ x\ > 0 \\ -1\ if\ x < 0 \end{cases}\quad \text{only defined for } x \in Int$$



```
function sign with x do
    if x is 0 do
        return 1.
    elif x is (n:%integer) if n > 0 do
        return 1.
    elif x is (n:%integer) if n < 0 do
        return -1.
    else do
        throw Error("invalid input").
    end
end

assert (sign 1 == 1).
```

ln006/sign1a.ast

Multi-Dispatch



```
function sign
    with 0 do
        return 1.
    with (n:%integer) if n > 0 do
        return 1.
    with (n:%integer) if n < 0 do
        return -1.
end

assert (sign 1 == 1).
```

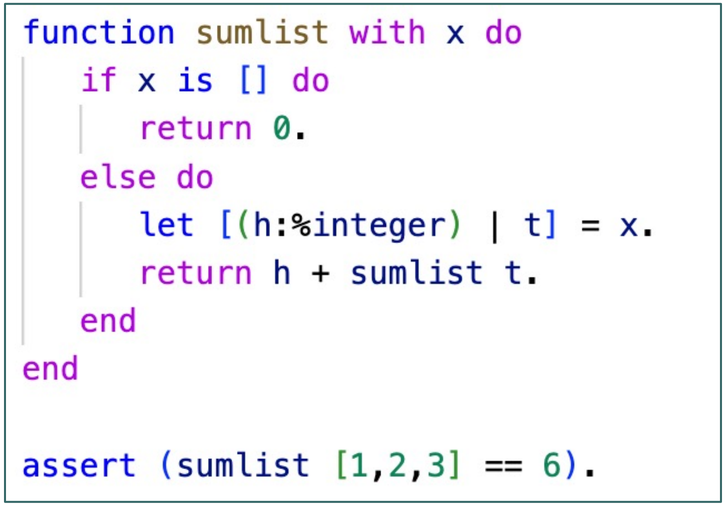
ln006/sign1b.ast

Multi-Dispatch and Recursion

- Multi-dispatch works exceptionally well with recursive functions
  - Separate ‘with’ clauses for base- and recursive cases

Multi-Dispatch and Recursion

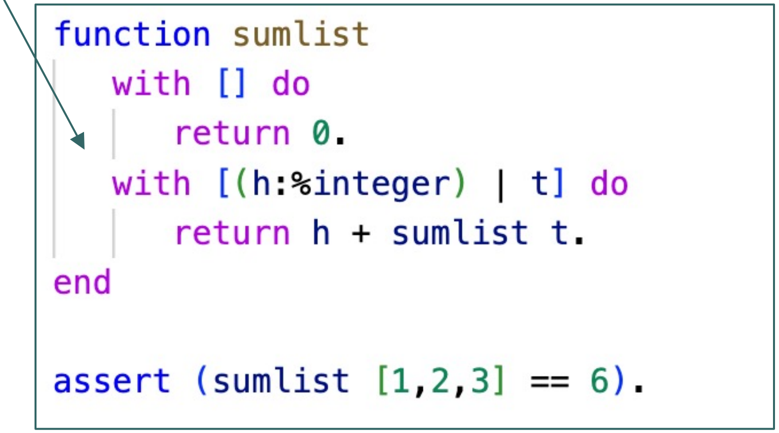
- Example: Recursive function that sums the elements of an integer list.
  - Observation: multi-dispatch preserves the declarative nature of pattern matching



```
function sumlist with x do
    if x is [] do
        return 0.
    else do
        let [(h:%integer) | t] = x.
        return h + sumlist t.
    end
end

assert (sumlist [1,2,3] == 6).
```

ln006/sumlist1a.ast

Multi-dispatch



```
function sumlist
    with [] do
        return 0.
    with [(h:%integer) | t] do
        return h + sumlist t.
end

assert (sumlist [1,2,3] == 6).
```

ln006/sumlist1b.ast

Reading

- asteroid-lang.readthedocs.io/en/latest/User%20Guide.html#functions
- asteroid-lang.readthedocs.io/en/latest/User%20Guide.html#pattern-matching