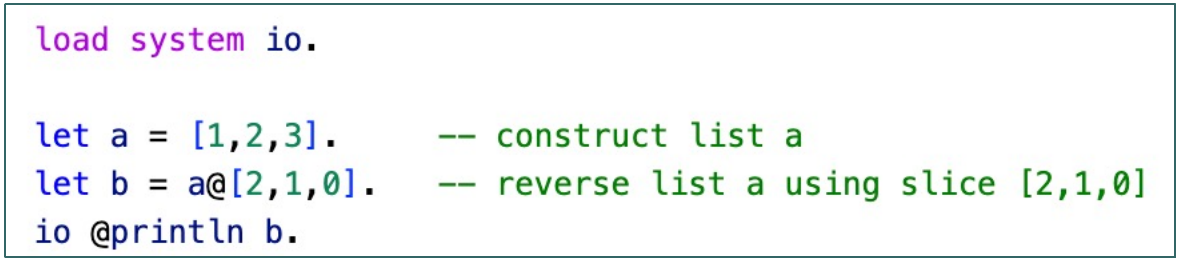
The Let Statement & Basic Pattern Matching

- Up till now we have used the let statement basically as an assignment statement into a single variable in the imperative fashion

  let <var> = <value>.



```
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.
```

The Let Statement & Basic Pattern Matching

- However, the let statement is a pattern-match statement in Asteroid,

```
let <pattern> = <value>.
```

- where the pattern on the left side of the equal sign is matched against the value of the right side of the equal sign.
- **Simple patterns are expressions that consist purely of constructors and variables**

Pattern Matching – Foundations

- In programs values are represented by **constructors**,
  - 1
  - “Hello, World!”
  - [1,2,3]
  - (“Harry”, 32)
- Any structure that cannot be reduced any further consists purely of constructors and is the **minimal/canonical representation** of a value.
- The following are all representations of the value two:
  - 1+1; 3-1; 2*1; 2+0; 2
  - Only the last one is the canonical representation of the value two.
  - We say that 2 is a constructor for the value two.
  - In this case the constructor happens to be a constant.

Pattern Matching – Foundations

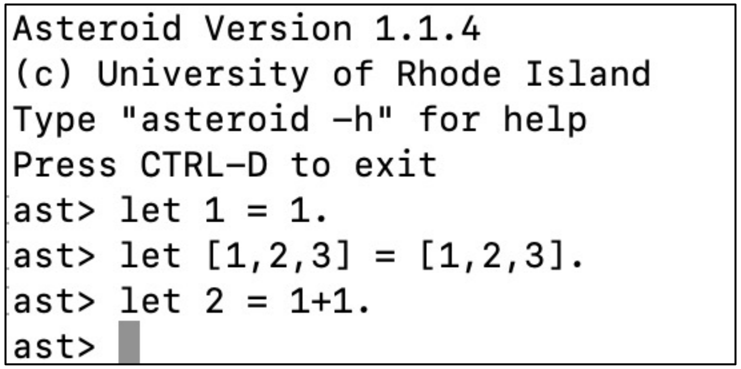
- Here is another example using lists
- The following are all representations of a list with the values one, two, and three
  - [1]+[2]+[3]; [1,2]+[3]; [1,2,3]+[]; [1,2,3]
- Again, only the last one is the canonical representation of the list
  - It represents the value of a list with integer values one, two, and three.

Pattern Matching – Foundations

- Constructors are interesting,
  - When they are part of an expression being evaluated, they represent values
  - Otherwise, they represent structure.
- We see this with the let statement,
  let <pattern> = <value>.
  - On the right of the = sign constructors represent values
  - On the left of the = sign constructors represent structure
- In a let statement, when the structure of the value on the right matches the structure of the pattern on the left, we say that we have a **successful pattern match**.

Pattern Matching – Foundations

- For example,

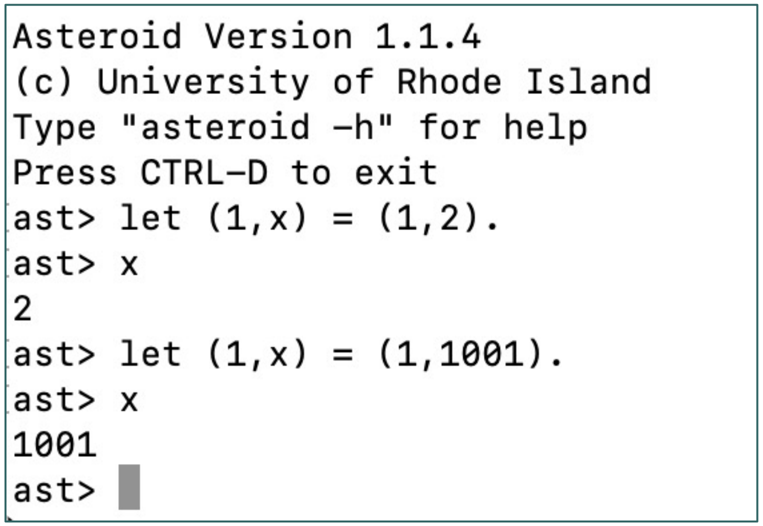


```
Asteroid Version 1.1.4
(c) University of Rhode Island
Type "asteroid -h" for help
Press CTRL-D to exit
ast> let 1 = 1.
ast> let [1,2,3] = [1,2,3].
ast> let 2 = 1+1.
ast> 
```

The last example is interesting, the right is not in the canonical representation for the value 2, so it is first reduced (evaluated) to its canonical form and then successfully pattern matched.

Pattern Matching – Foundations

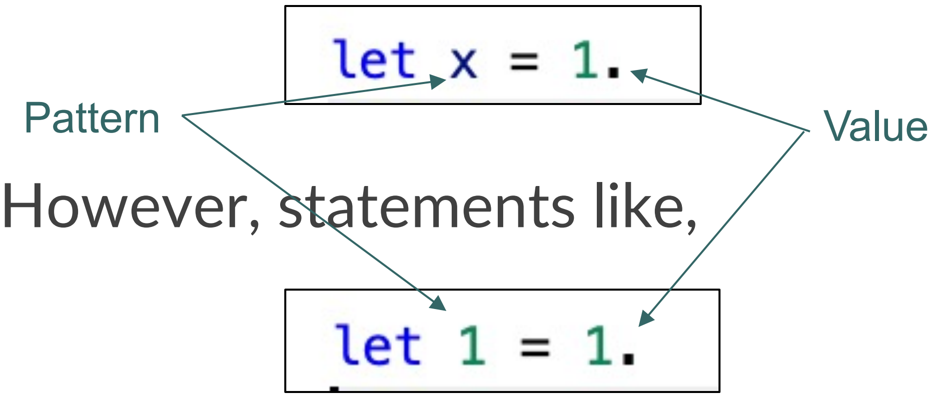
- You can think of variables in a pattern as a “I don’t care” structure
- During a pattern match the variable will receive the structure that was actually matched during the pattern match



```
Asteroid Version 1.1.4
(c) University of Rhode Island
Type "asteroid -h" for help
Press CTRL-D to exit
ast> let (1,x) = (1,2).
ast> x
2
ast> let (1,x) = (1,1001).
ast> x
1001
ast>
```

Pattern Matching – Foundations

- When the pattern is just a single variable then the let statement looks like an assignment statement,



```
let x = 1.
```

- However, statements like,

```
let 1 = 1.
```

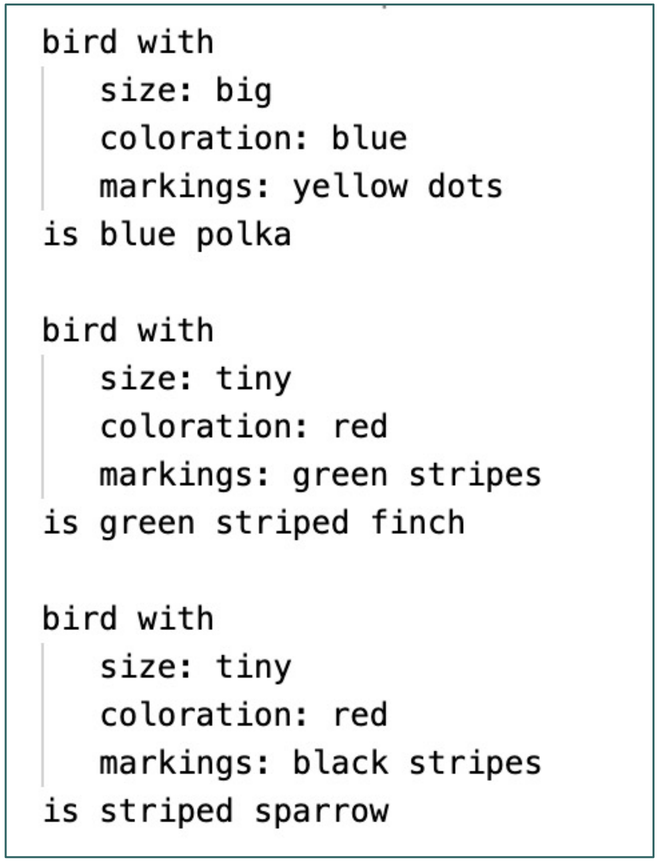
- are completely legal,
  - the 1 on the left is a constructor viewed as pattern, the 1 on the right is a constructor viewed as a value.
  - highlighting the fact that the let statement is not equivalent to an assignment statement.

Pattern Matching – Foundations

- Patterns are all about structure
- For example,
  - a wildlife biologist might use pattern matching to identify a specific species of bird based on its size, coloration, and distinctive markings on its feathers – **structure**.
  - They would compare these characteristics to a known set of **patterns** for different bird species from a field guide and use this information to make an accurate identification.
- Observe, the structure of a value (unknown bird) is pattern-matched against a set of known patterns. If one of the patterns matches the value (bird) then we have a match (identification).

Pattern Matching – Foundations

- We can code that biologist example using pattern matching
- Assume we have a field guide with the following patterns

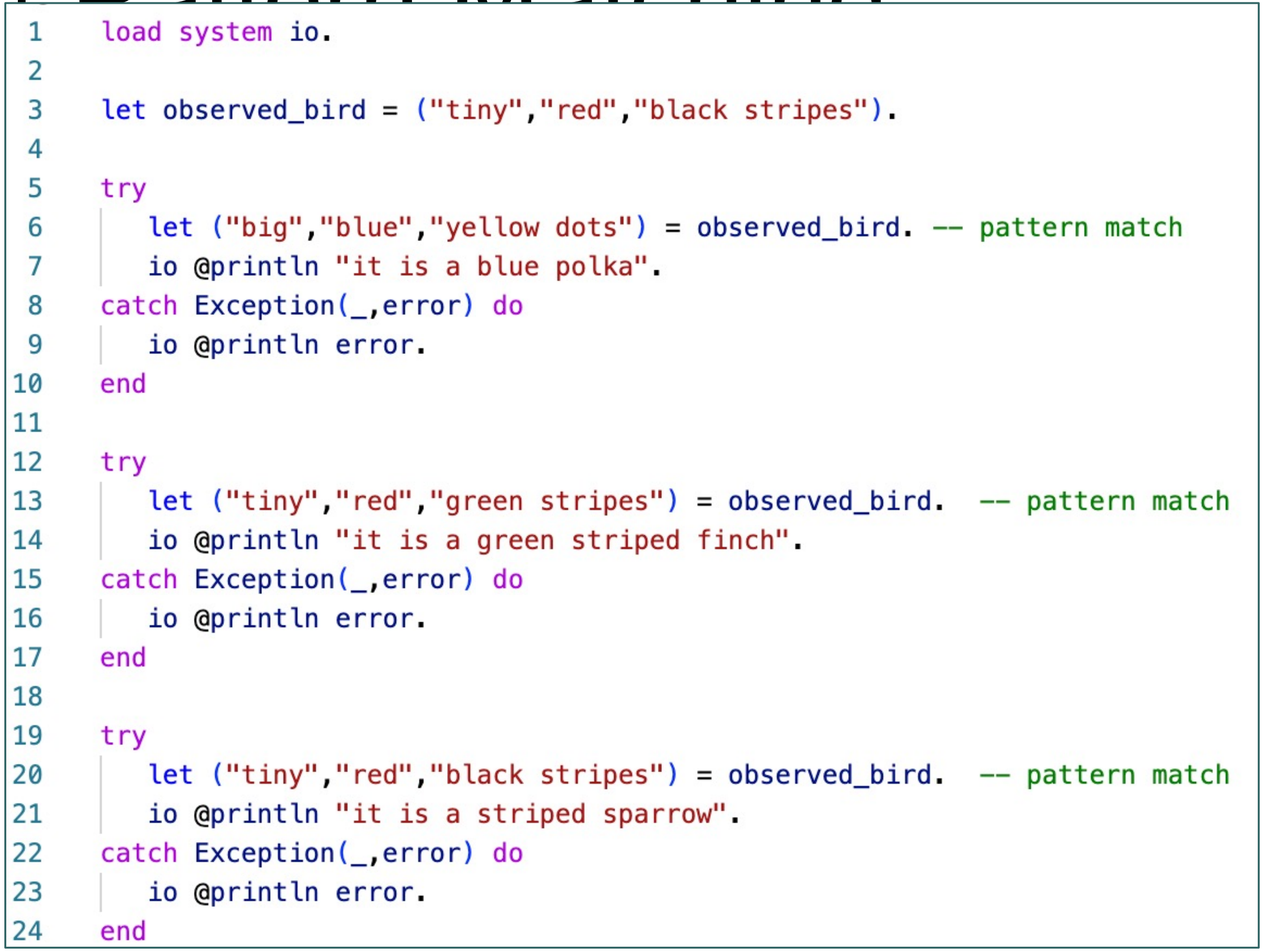


```
bird with
    size: big
    coloration: blue
    markings: yellow dots
is blue polka

bird with
    size: tiny
    coloration: red
    markings: green stripes
is green striped finch

bird with
    size: tiny
    coloration: red
    markings: black stripes
is striped sparrow
```

Pattern Matching – Foundations

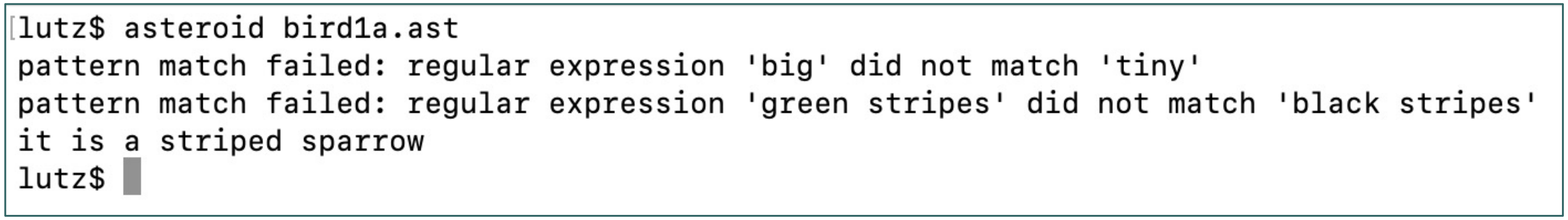
- We can solve this problem nicely with pattern matching in Asteroid,
  - We will encode the patterns as 3-tuples
  - We write a let statement for each pattern
  - When let statements fail they throw an exception, we will embed the let statements in a try-catch block so we can detect the pattern match failure

Pattern Matching



```
load system io.

let observed_bird = ("tiny","red","black stripes").

try
    let ("big","blue","yellow dots") = observed_bird. -- pattern match
    io @println "it is a blue polka".
catch Exception(_,error) do
    io @println error.
end

try
    let ("tiny","red","green stripes") = observed_bird.  -- pattern match
    io @println "it is a green striped finch".
catch Exception(_,error) do
    io @println error.
end

try
    let ("tiny","red","black stripes") = observed_bird.  -- pattern match
    io @println "it is a striped sparrow".
catch Exception(_,error) do
    io @println error.
end
```

In004/bird1a.ast



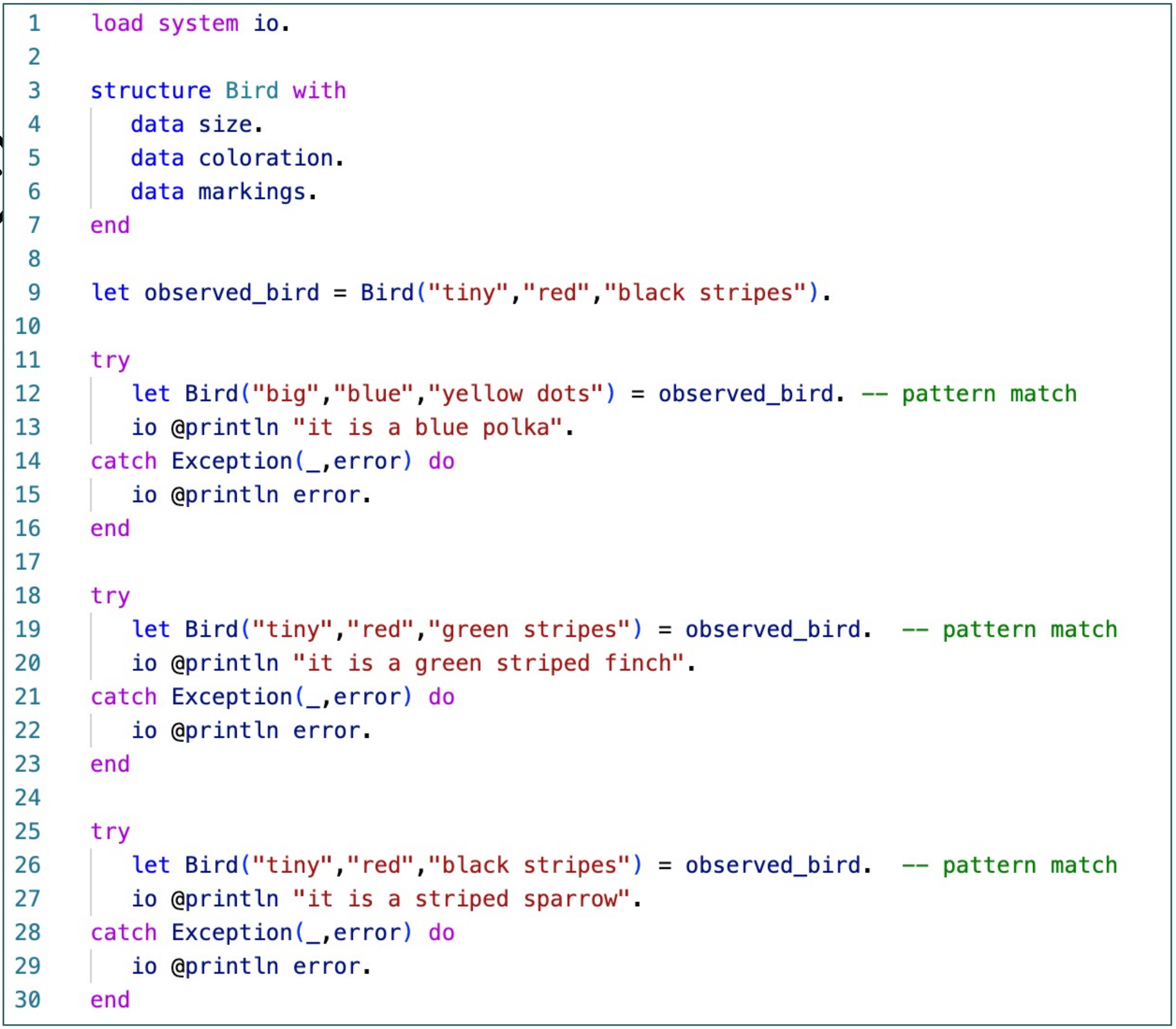
```
lutz$ asteroid bird1a.ast
pattern match failed: regular expression 'big' did not match 'tiny'
pattern match failed: regular expression 'green stripes' did not match 'black stripes'
it is a striped sparrow
lutz$
```

Pattern Matching – Foundations

- It is nicer to represent the patterns as bird objects
- This way we stay closer to the original problem setting. E.g.,

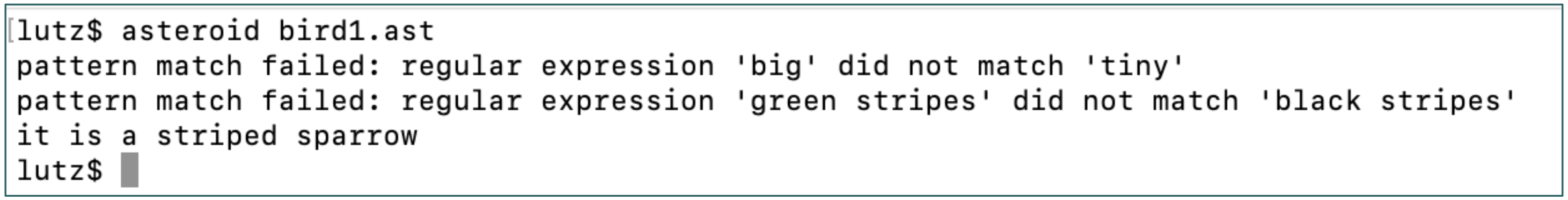
```
bird with
   size: big
   coloration: blue
   markings: yellow dots
is blue polka
```

```
structure Bird with
   data size.
   data coloration.
   data markings.
end

let observed_bird = Bird("tiny","red","black stripes").

try
   let Bird("big","blue","yellow dots") = observed_bird. -- pattern match
   io @println "it is a blue polka".
catch Exception(_,error) do
   io @println error.
end
```



```
load system io.

structure Bird with
    data size.
    data coloration.
    data markings.
end

let observed_bird = Bird("tiny","red","black stripes").

try
    let Bird("big","blue","yellow dots") = observed_bird. -- pattern match
    io @println "it is a blue polka".
catch Exception(_,error) do
    io @println error.
end

try
    let Bird("tiny","red","green stripes") = observed_bird.  -- pattern match
    io @println "it is a green striped finch".
catch Exception(_,error) do
    io @println error.
end

try
    let Bird("tiny","red","black stripes") = observed_bird.  -- pattern match
    io @println "it is a striped sparrow".
catch Exception(_,error) do
    io @println error.
end
```

ln004/bird1b.ast



```
lutz$ asteroid bird1.ast
pattern match failed: regular expression 'big' did not match 'tiny'
pattern match failed: regular expression 'green stripes' did not match 'black stripes'
it is a striped sparrow
lutz$ 
```

Pattern Matching – Foundations

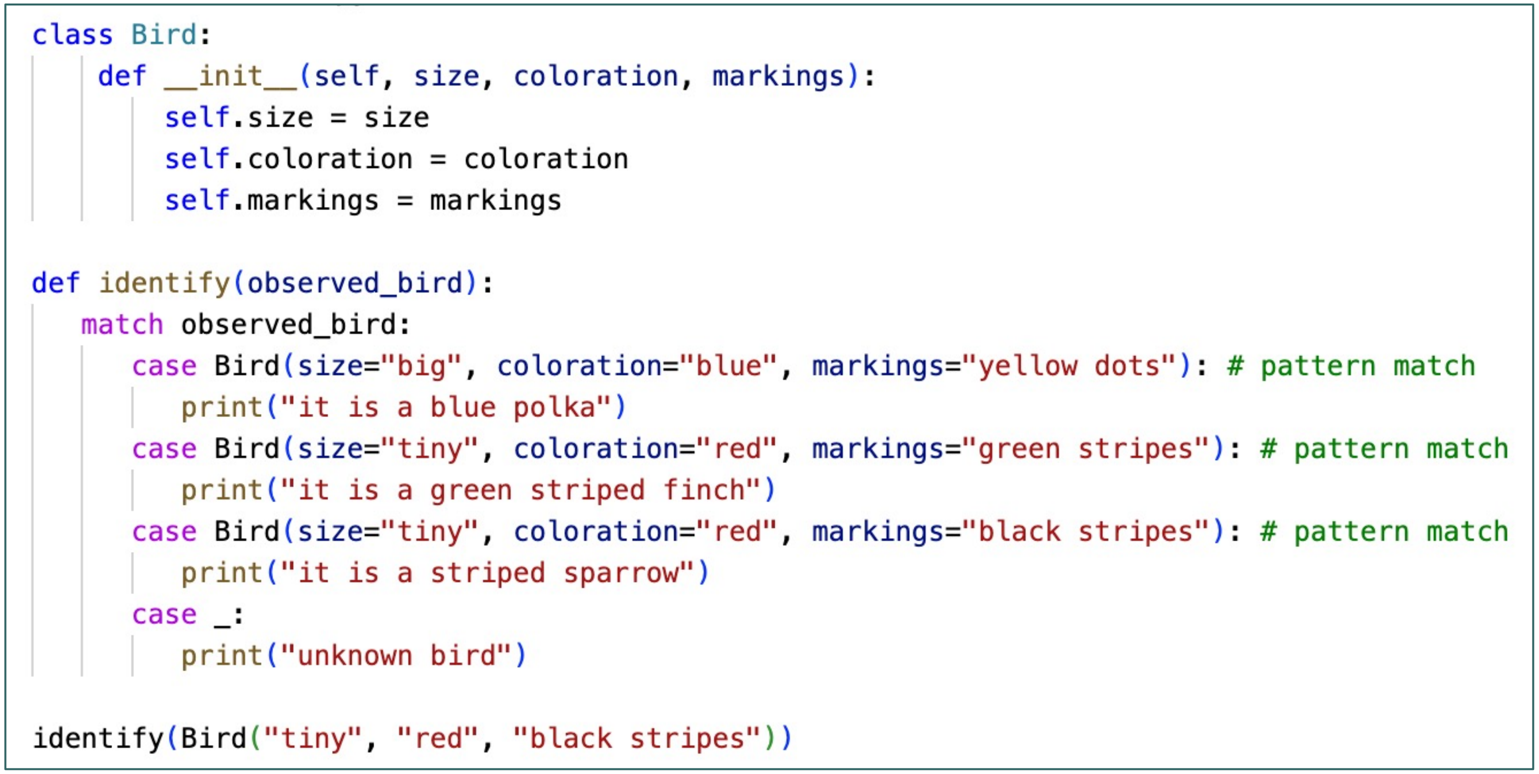
- Here is a much more elegant solution using pattern matching in functions

```
load system io.

structure Bird with
   data size.
   data coloration.
   data markings.
end

function identify
   with Bird("big","blue","yellow dots") do -- pattern match
      io @println "it is a blue polka".
   with Bird("tiny","red","green stripes") do  -- pattern match
      io @println "it is a green striped finch".
   with Bird("tiny","red","black stripes") do  -- pattern match
      io @println "it is a striped sparrow".
   with _ do
      io @println "unkown bird".
end

identify (Bird("tiny","red","black stripes")).
```

ln004/bird2.ast

Pattern Matching – Foundations

- Here is a solution using pattern matching in Python

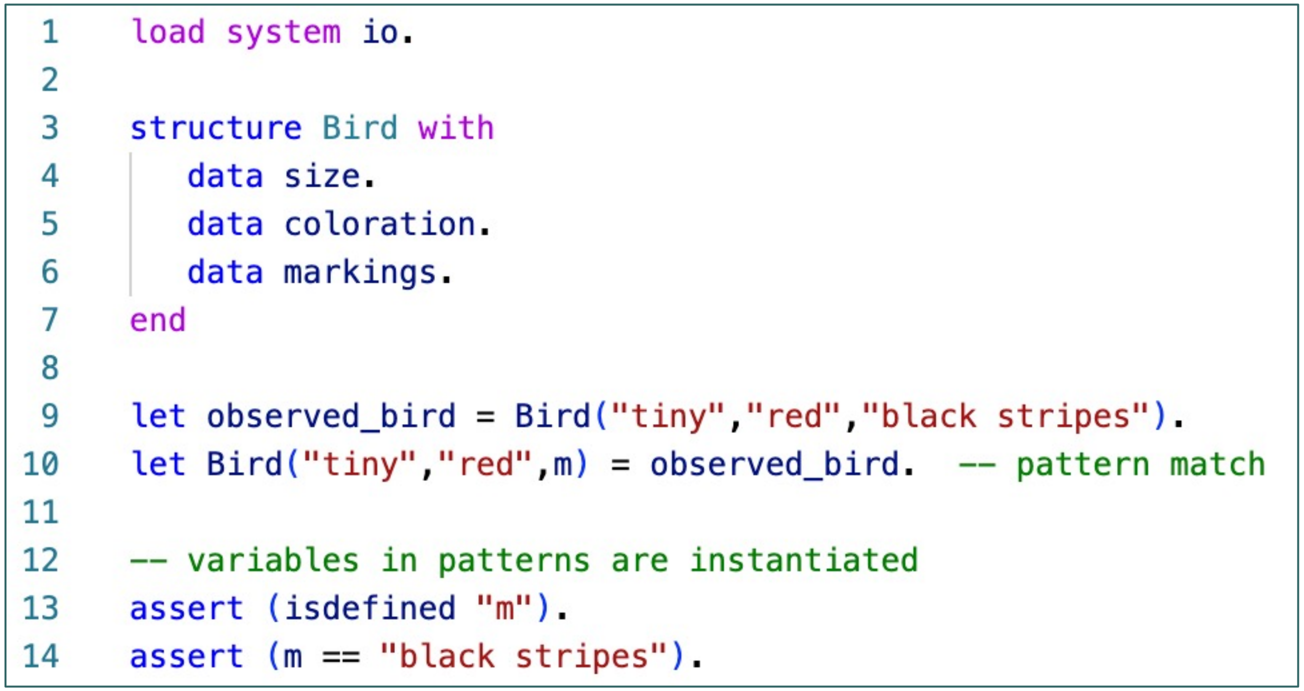


```python
class Bird:
    def __init__(self, size, coloration, markings):
        self.size = size
        self.coloration = coloration
        self.markings = markings

def identify(observed_bird):
    match observed_bird:
        case Bird(size="big", coloration="blue", markings="yellow dots"): # pattern match
            print("it is a blue polka")
        case Bird(size="tiny", coloration="red", markings="green stripes"): # pattern match
            print("it is a green striped finch")
        case Bird(size="tiny", coloration="red", markings="black stripes"): # pattern match
            print("it is a striped sparrow")
        case _:
            print("unknown bird")

identify(Bird("tiny", "red", "black stripes"))
```

ln004/bird2.py

Pattern Matching – Foundations

- Variables allow for partial matches
- Variables in patterns are instantiated in the current environment

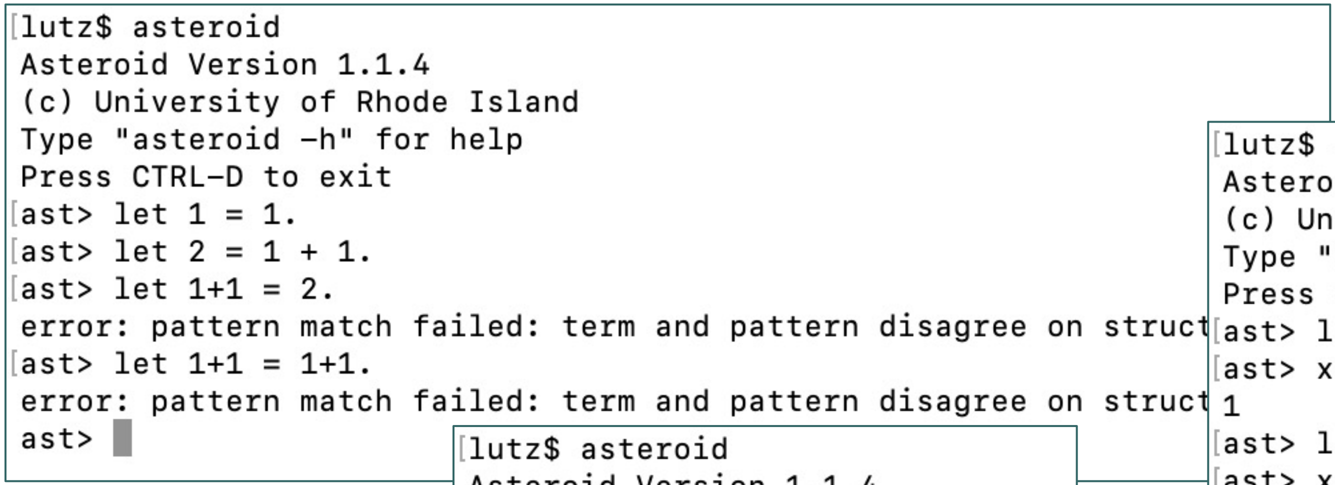


```
load system io.

structure Bird with
    data size.
    data coloration.
    data markings.
end

let observed_bird = Bird("tiny","red","black stripes").
let Bird("tiny","red",m) = observed_bird.  -- pattern match

-- variables in patterns are instantiated
assert (isdefined "m").
assert (m == "black stripes").
```

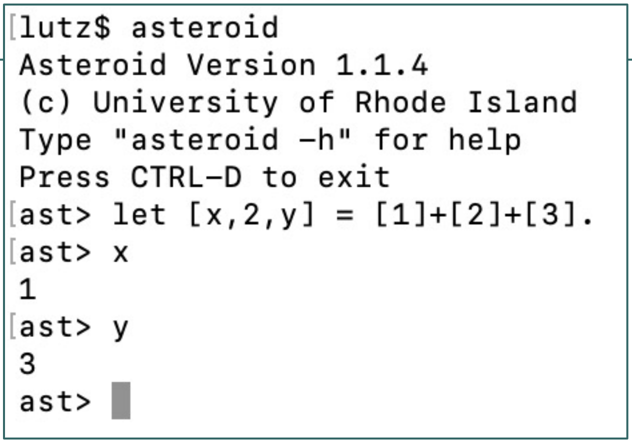
ln004/bird3.ast

Basic Patterns

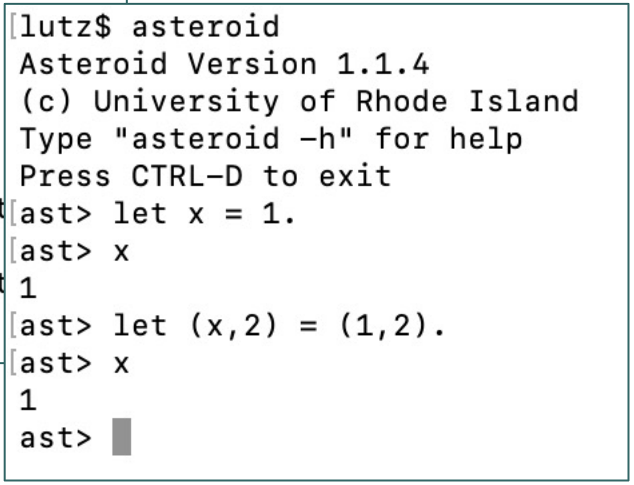
- Something a bit more CS related



```
[lutz$ asteroid
Asteroid Version 1.1.4
(c) University of Rhode Island
Type "asteroid -h" for help
Press CTRL-D to exit
[ast> let 1 = 1.
[ast> let 2 = 1 + 1.
[ast> let 1+1 = 2.
error: pattern match failed: term and pattern disagree on struct
[ast> let 1+1 = 1+1.
error: pattern match failed: term and pattern disagree on struct
ast> 
```

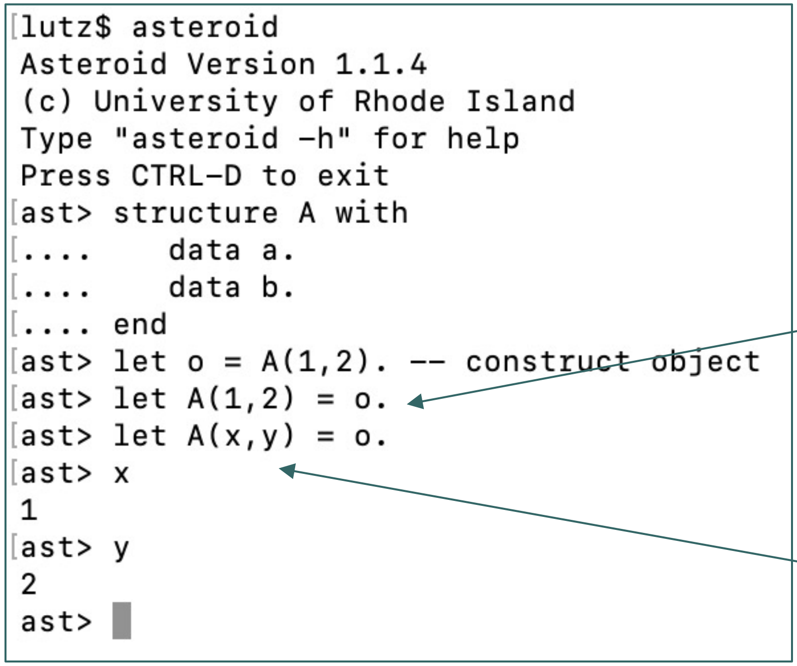


```
[lutz$ asteroid
Asteroid Version 1.1.4
(c) University of Rhode Island
Type "asteroid -h" for help
Press CTRL-D to exit
[ast> let [x,2,y] = [1]+[2]+[3].
[ast> x
1
[ast> y
3
ast> 
```



```
[lutz$ asteroid
Asteroid Version 1.1.4
(c) University of Rhode Island
Type "asteroid -h" for help
Press CTRL-D to exit
[ast> let x = 1.
[ast> x
1
[ast> let (x,2) = (1,2).
[ast> x
1
ast> 
```

Basic Patterns



```
[lutz$ asteroid
 Asteroid Version 1.1.4
 (c) University of Rhode Island
 Type "asteroid -h" for help
 Press CTRL-D to exit
[ast> structure A with
[....     data a.
[....     data b.
[.... end
[ast> let o = A(1,2). -- construct object
[ast> let A(1,2) = o.
[ast> let A(x,y) = o.
[ast> x
 1
[ast> y
 2
 ast>
```

- The idea of constructors on the right representing values and, on the left, representing structure/patterns also works for objects!
- The expression A(1,2) on the left side is a constructor for the object considered as a pattern
- We can insert variables into the constructor, A(x,y), for easy access to the components of the object o
  - **destructuring**

Destructuring

- The idea of destructuring is fundamental to pattern matching
- It makes access to substructures much more readable (and efficient).

Without structural pattern matching

```
let p = (1,2).    -- create a structure
let x = p@0.      -- access first component
let y = p@1.      -- access second component
assert (x==1 and y==2).
```

In004/destruct1.ast

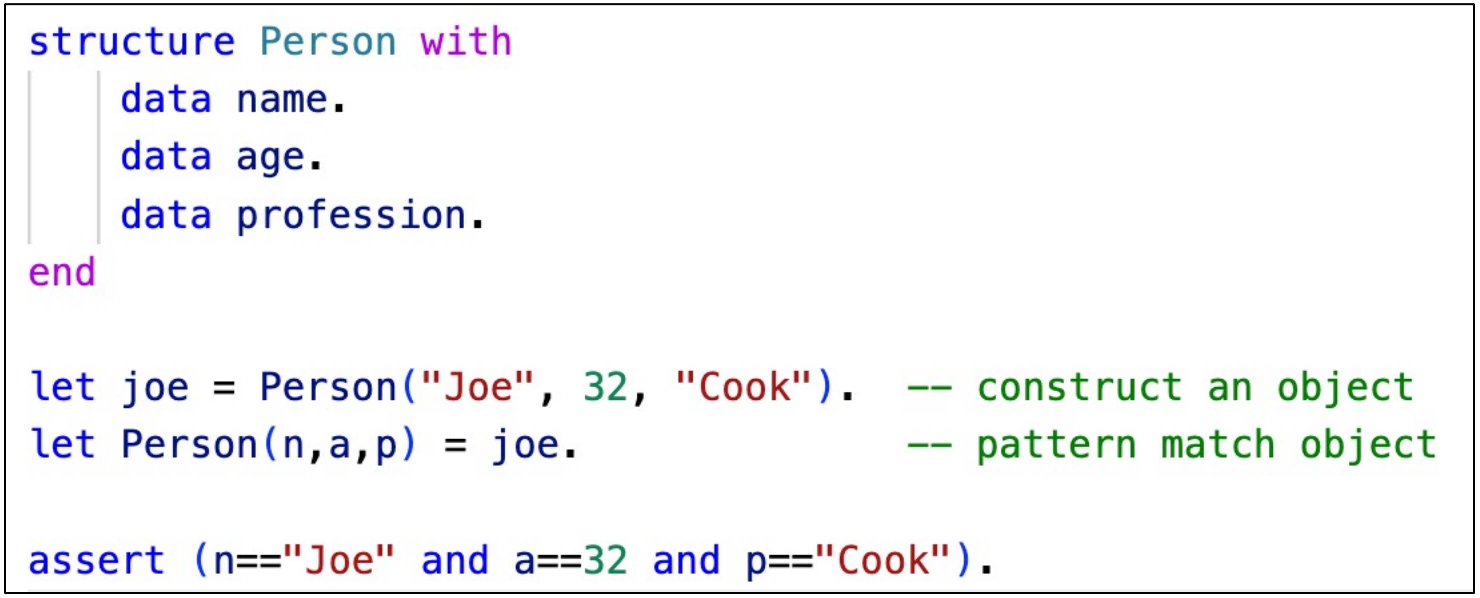
With structural pattern matching

In004/destruct2.ast

```
let p = (1,2).    -- create a structure
let (x,y) = p.    -- structural pattern matching, access to components
assert (x==1 and y==2).
```

Destructuring

- Here is another example using structures and objects



```
structure Person with
    data name.
    data age.
    data profession.
end

let joe = Person("Joe", 32, "Cook").   -- construct an object
let Person(n,a,p) = joe.               -- pattern match object

assert (n=="Joe" and a==32 and p=="Cook").
```

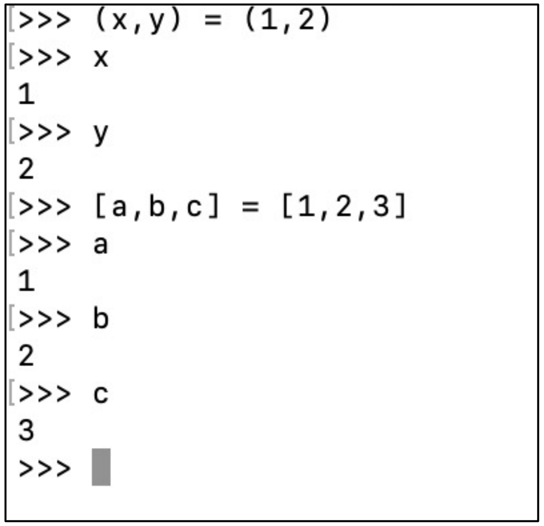
ln004/destruct3.ast

Basic Pattern Matching Summary

- The let statement
  let <pattern> = value .
- On the right side of equal sign constructors represent values
  - Operators/functions are allowed
- On the left side constructors represent structure
  - Operators/functions are **not** allowed
  - Constructors must minimally represent structure
- Variables are allowed in patterns for partial matches/destructuring
- Pattern matching is part of a programming paradigm called **declarative programming**
  - We will look at this more carefully when we examine control structures in Asteroid.

Pattern Matching in Python

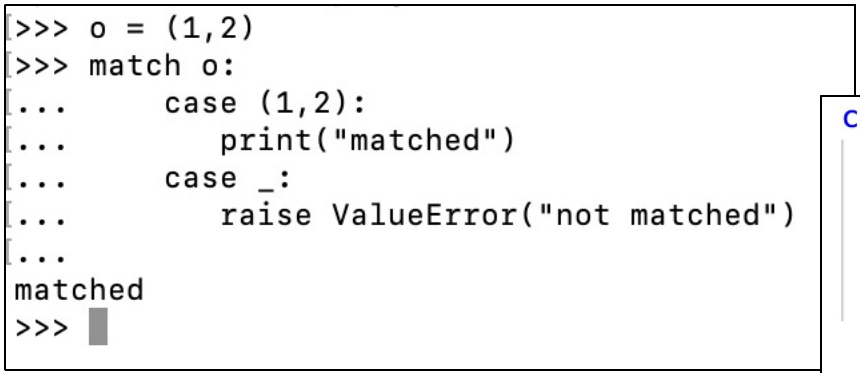
- Limited pattern matching available with the assignment statement
  - Called **destructuring** assignment



```
>>> (x,y) = (1,2)
>>> x
1
>>> y
2
>>> [a,b,c] = [1,2,3]
>>> a
1
>>> b
2
>>> c
3
>>> 
```

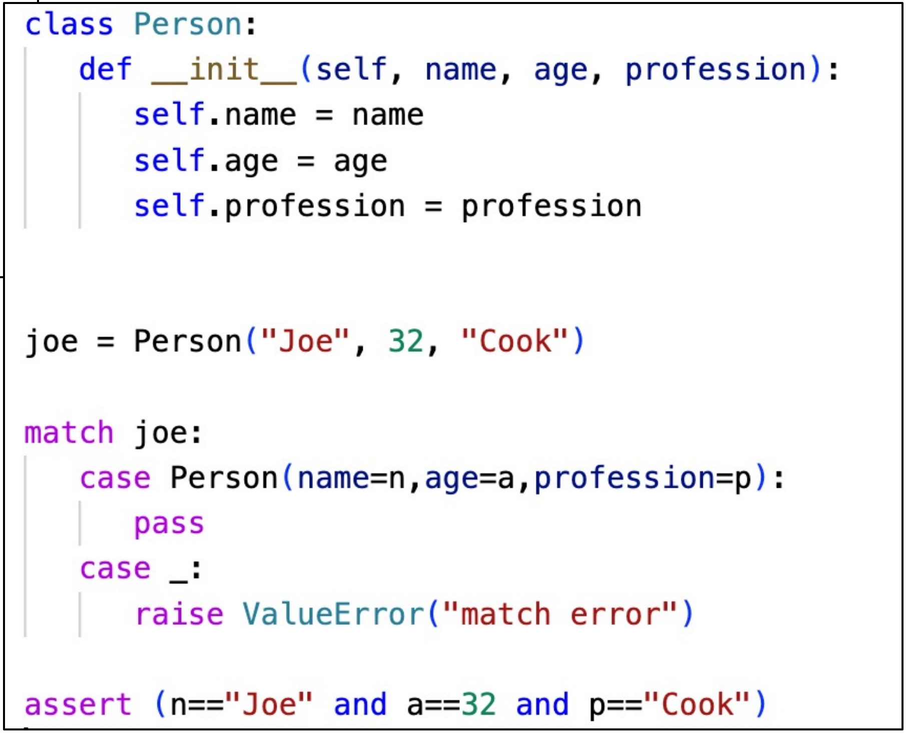
Pattern Matching in Python

- The match statement as of 3.10 provides a bit more functionality



```
>>> o = (1,2)
>>> match o:
...     case (1,2):
...         print("matched")
...     case _:
...         raise ValueError("not matched")
...
matched
>>>
```

ln004/destruct3.py



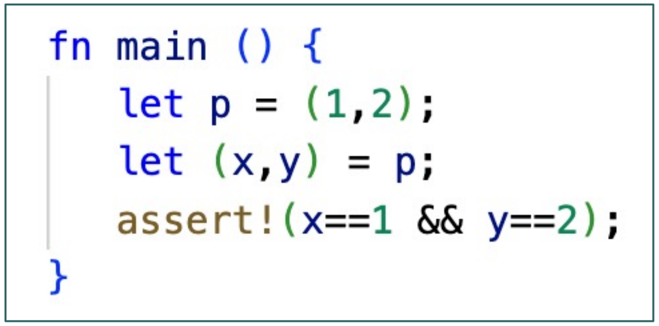
```python
class Person:
    def __init__(self, name, age, profession):
        self.name = name
        self.age = age
        self.profession = profession


joe = Person("Joe", 32, "Cook")

match joe:
    case Person(name=n,age=a,profession=p):
        pass
    case _:
        raise ValueError("match error")

assert (n=="Joe" and a==32 and p=="Cook")
```

https://peps.python.org/pep-0636/

Pattern Matching in Rust

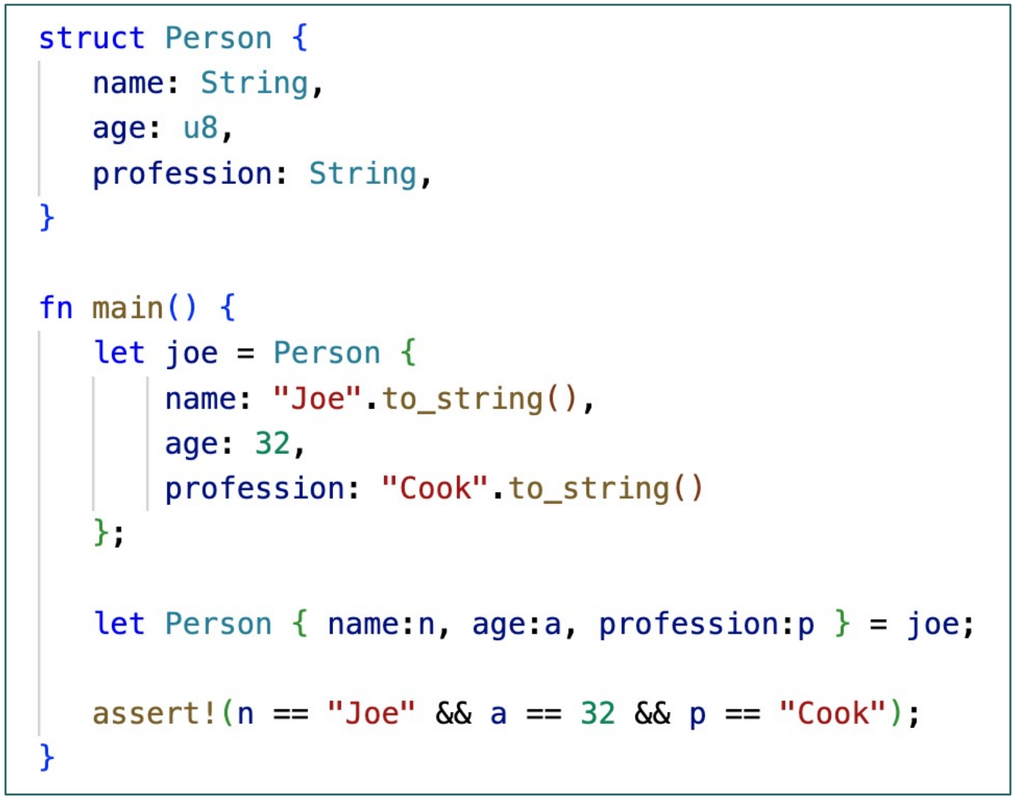
- Rust also supports pattern matching

ln004/destruct2.prs



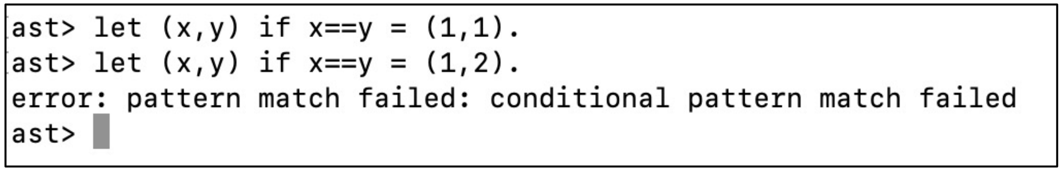
```rust
fn main () {
    let p = (1,2);
    let (x,y) = p;
    assert!(x==1 && y==2);
}
```

ln004/destruct3.rs

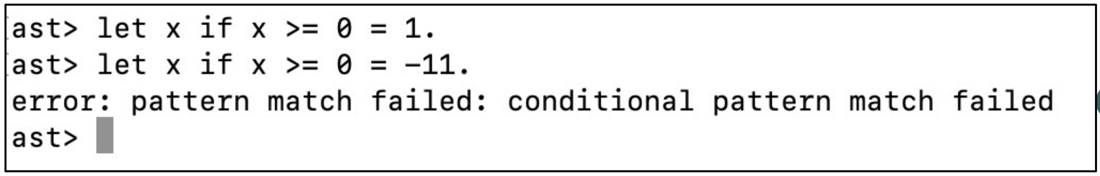


```rust
struct Person {
    name: String,
    age: u8,
    profession: String,
}

fn main() {
    let joe = Person {
        name: "Joe".to_string(),
        age: 32,
        profession: "Cook".to_string()
    };

    let Person { name:n, age:a, profession:p } = joe;

    assert!(n == "Joe" && a == 32 && p == "Cook");
}
```

Conditional Pattern Matching



```
ast> let (x,y) if x==y = (1,1).
ast> let (x,y) if x==y = (1,2).
error: pattern match failed: conditional pattern match failed
ast>
```



```
ast> let x if x >= 0 = 1.
ast> let x if x >= 0 = -11.
error: pattern match failed: conditional pattern match failed
ast>
```

- Only assign a pair if the two component values are the same
- Only assign positive values to x

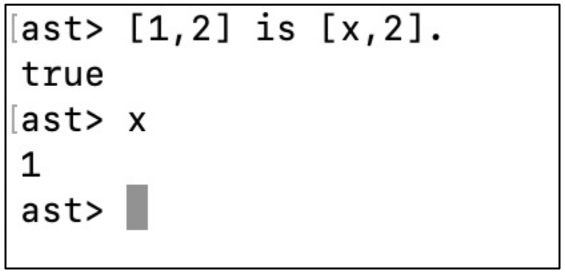
The is Predicate

Note: a predicate is a function/operator that always returns true or false. No other return value is permitted.

- The is predicate is of the form

  <value> is <pattern>

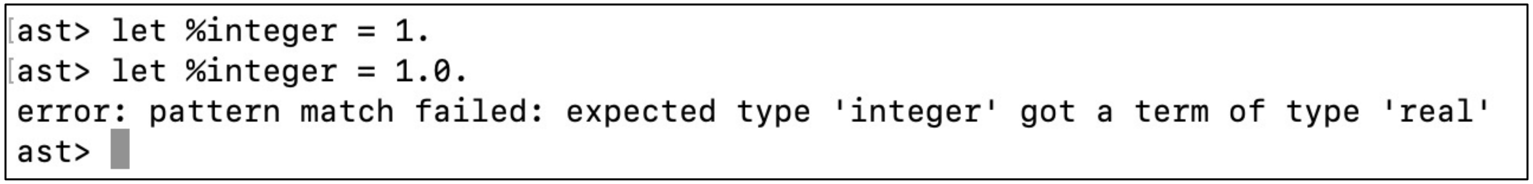
  and returns true if the value matches the pattern otherwise it will return false
- The is predicate allows us to do pattern matching is expressions



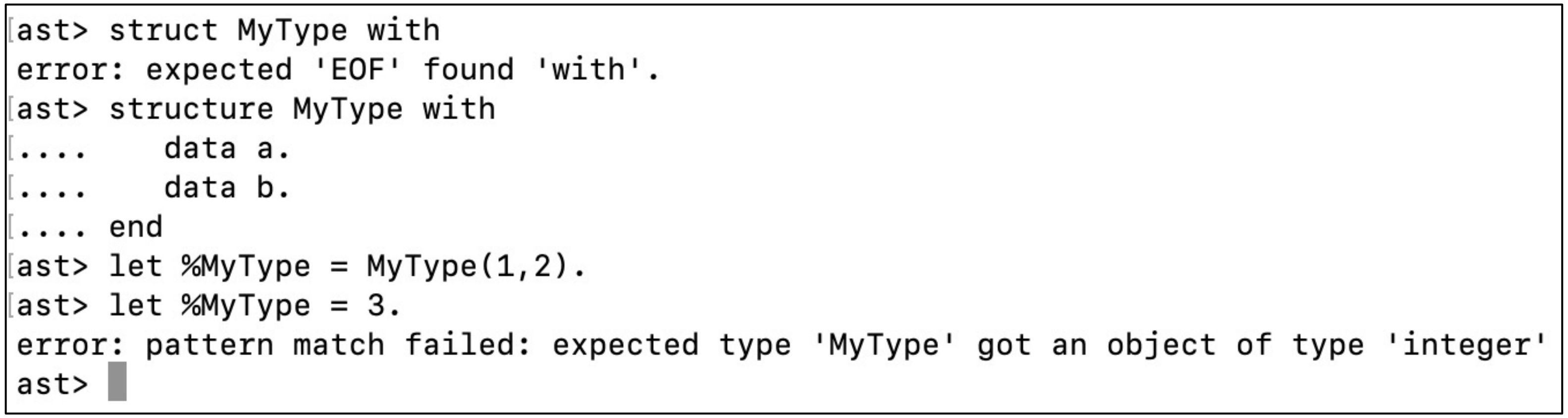
```
ast> [1,2] is [x,2].
true
ast> x
1
ast>
```

Type Patterns

- Type patterns are patterns of the form
  %<type name>
  and match all instances of the <type name>
- All built-in types have associated type patterns such as %integer, %real, %string etc.
- User defined types are also supported,
  %<user defined type name>



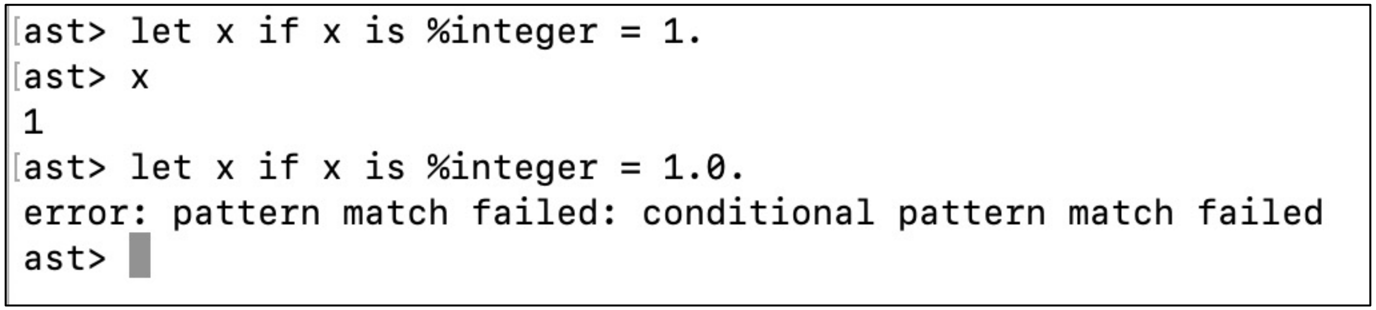
```
ast> let %integer = 1.
ast> let %integer = 1.0.
error: pattern match failed: expected type 'integer' got a term of type 'real'
ast>
```



```
ast> struct MyType with
error: expected 'EOF' found 'with'.
ast> structure MyType with
....    data a.
....    data b.
.... end
ast> let %MyType = MyType(1,2).
ast> let %MyType = 3.
error: pattern match failed: expected type 'MyType' got an object of type 'integer'
ast>
```

Advanced Pattern Match Expressions

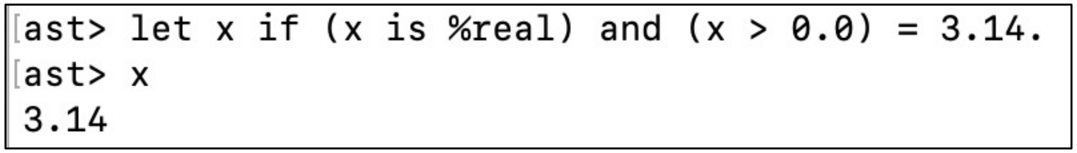
- We can combine conditional pattern matching with type patterns and the is predicate to express sophisticated patterns
- E.g., only assign a value to x if it is an integer value



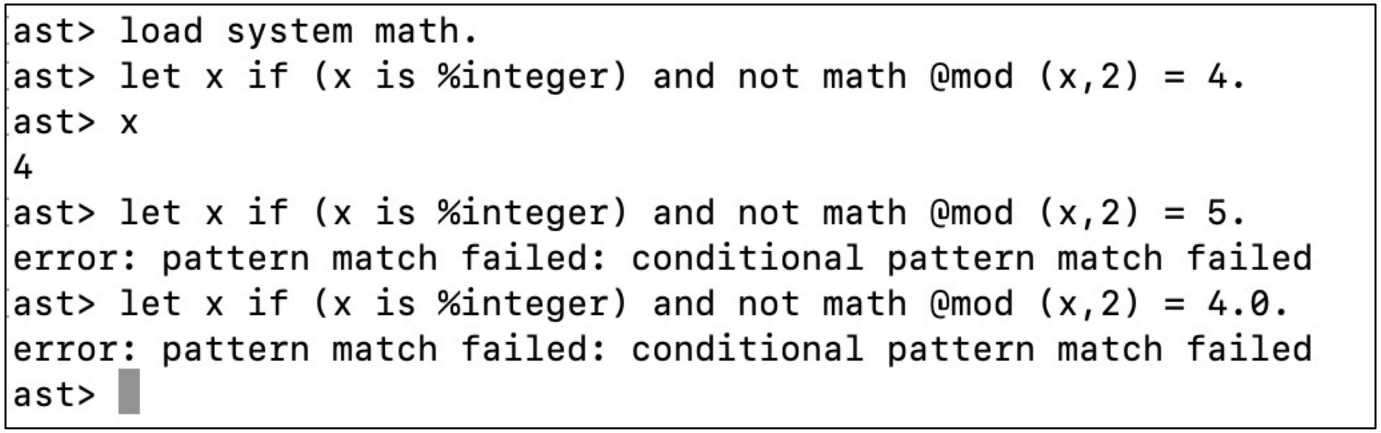
```
ast> let x if x is %integer = 1.
ast> x
1
ast> let x if x is %integer = 1.0.
error: pattern match failed: conditional pattern match failed
ast> 
```

Advanced Pattern Match Expressions

- Here are some additional examples,



```
[ast> let x if (x is %real) and (x > 0.0) = 3.14.
[ast> x
3.14
```

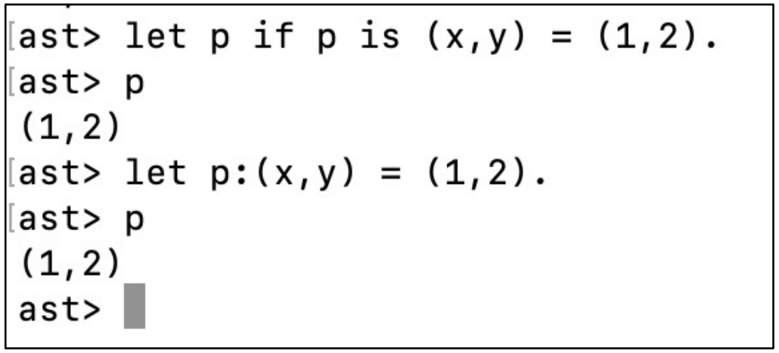


```
ast> load system math.
ast> let x if (x is %integer) and not math @mod (x,2) = 4.
ast> x
4
ast> let x if (x is %integer) and not math @mod (x,2) = 5.
error: pattern match failed: conditional pattern match failed
ast> let x if (x is %integer) and not math @mod (x,2) = 4.0.
error: pattern match failed: conditional pattern match failed
ast>
```

Note: ‘mod’ is the modulus function

Named Patterns

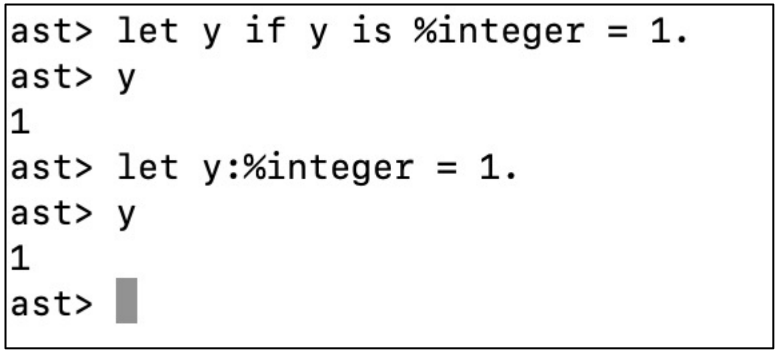
- The simple conditional pattern
  x if x is <pattern>
  appears a lot in Asteroid programs
- Named patterns of the form
  x:<pattern>
  represent a shorthand for the simple conditional pattern above
- E.g.



```
ast> let p if p is (x,y) = (1,2).
ast> p
(1,2)
ast> let p:(x,y) = (1,2).
ast> p
(1,2)
ast>
```

Named Patterns

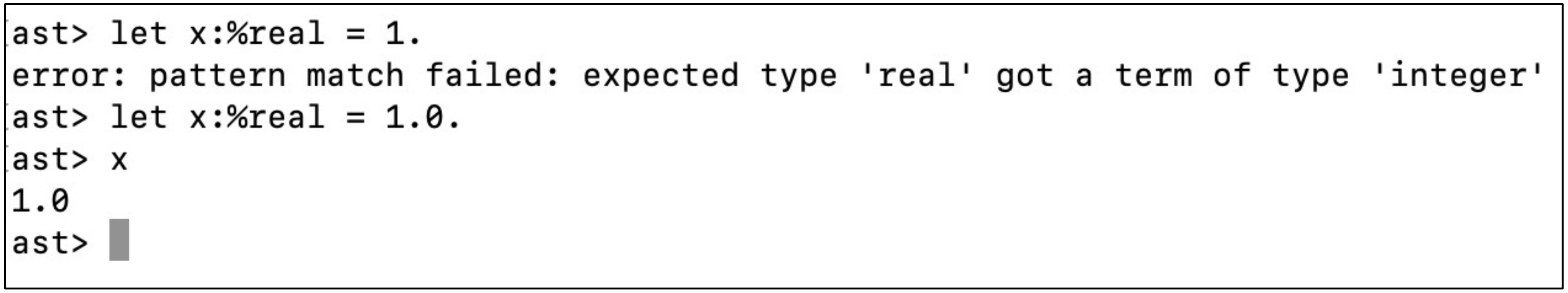
- This shorthand notation is especially useful when combined with type patterns,



```
ast> let y if y is %integer = 1.
ast> y
1
ast> let y:%integer = 1.
ast> y
1
ast> 
```

Named Patterns

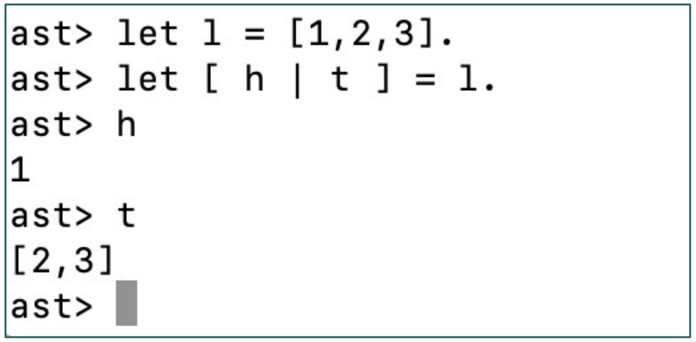
- Beware: even though named patterns with type patterns look like a declarations they are not!
- They are pattern match statements; consequently, implicit type conversions we are used to from other programming languages do not work!



```
ast> let x:%real = 1.
error: pattern match failed: expected type 'real' got a term of type 'integer'
ast> let x:%real = 1.0.
ast> x
1.0
ast>
```

Head-Tail Pattern

- The head-tail pattern
  [<head var> | <tail var>]
  is a useful pattern that allows us to destructure a list into into its first element and the rest of the list; the list with its first element removed.
- As we will see later, this pattern will prove extremely useful when dealing with recursion or iteration over lists.



```
ast> let l = [1,2,3].
ast> let [ h | t ] = l.
ast> h
1
ast> t
[2,3]
ast> 
```

Pattern Matching with Regular Expressions

- Regular expressions are patterns that can be applied to strings
- e.g., the regex

  “a(b)*”

  matches any string that starts with an a followed by zero or more b’s.
- In Asteroid regular expressions are considered patterns and therefore we can write expressions like

  “abbbb” is “a(b)*”
- Asteroid’s regex syntax follows Python’s regex syntax
  - https://docs.python.org/3/library/re.html

Pattern Matching with Regular Expressions

- Regular expressions is a formal language that defines lexical patterns of character strings
- As shown before, the regular expression
  “a(b)*”
  describes a pattern that matches any string that starts with an ‘a’ character followed by zero or more ‘b’ characters.
- Possible matches are
  “a”, “ab”, “abb”, “abbb”, etc

Pattern Matching with Regular Expressions

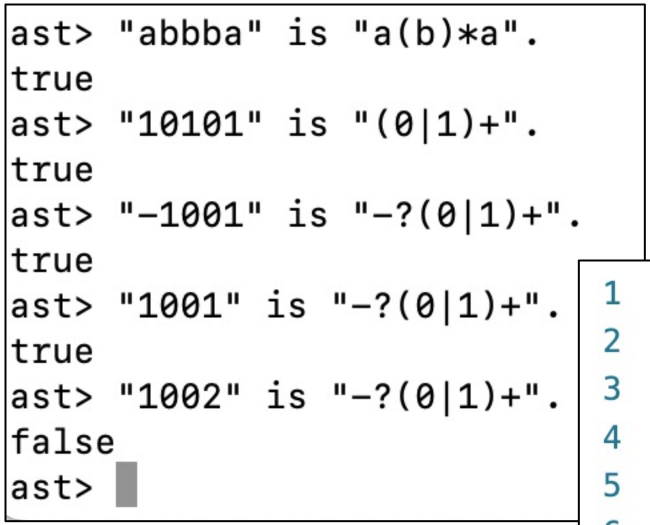
- Any single, printable character is a RE, e.g., “A” or “1”
- The concatenation “<RE1><RE2>” is also an RE, e.g. “ab”
- The “<RE>*” operator means match the RE zero or more times, e.g. “a*” and “(ab)*”
- The “<RE>+” operator means match the RE one or more times, e.g. “a+” and “(ab)+”

“a+” = “a(a)*”

- The “<RE>?” operator means match the RE if it exists, e.g. “a(b)?c”
- The “<RE1>|<RE2>” operator means match either RE1 or RE2.
- The ”.” operator matches any character

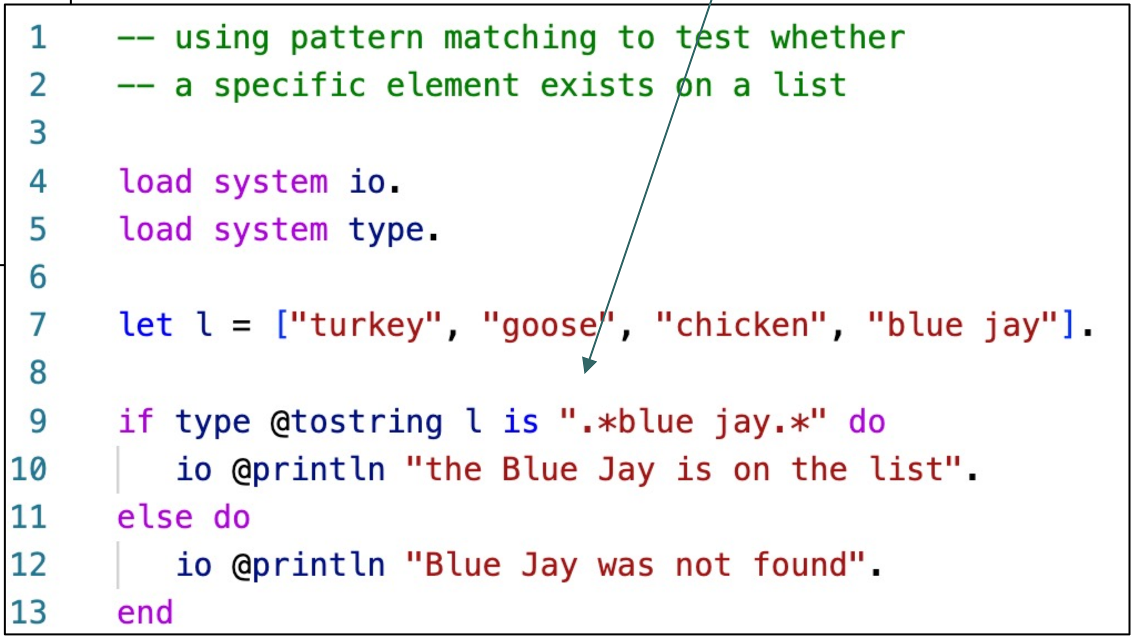
Note: REs are a very rich language, see more at https://docs.python.org/3/library/re.html

Pattern Matching with Regular Expressions



```
ast> "abbba" is "a(b)*a".
true
ast> "10101" is "(0|1)+".
true
ast> "-1001" is "-?(0|1)+".
true
ast> "1001" is "-?(0|1)+".
true
ast> "1002" is "-?(0|1)+".
false
ast> 
```

Pattern matching with regex



```
-- using pattern matching to test whether
-- a specific element exists on a list

load system io.
load system type.

let l = ["turkey", "goose", "chicken", "blue jay"].

if type @tostring l is ".*blue jay.*" do
   io @println "the Blue Jay is on the list".
else do
   io @println "Blue Jay was not found".
end
```

ln004/list1.ast

Reading

- The Let Statement
  - asteroid-lang.readthedocs.io/en/latest/User%20Guide.html#the-let-statement