CMPU 101 § 53 · Computer Science I

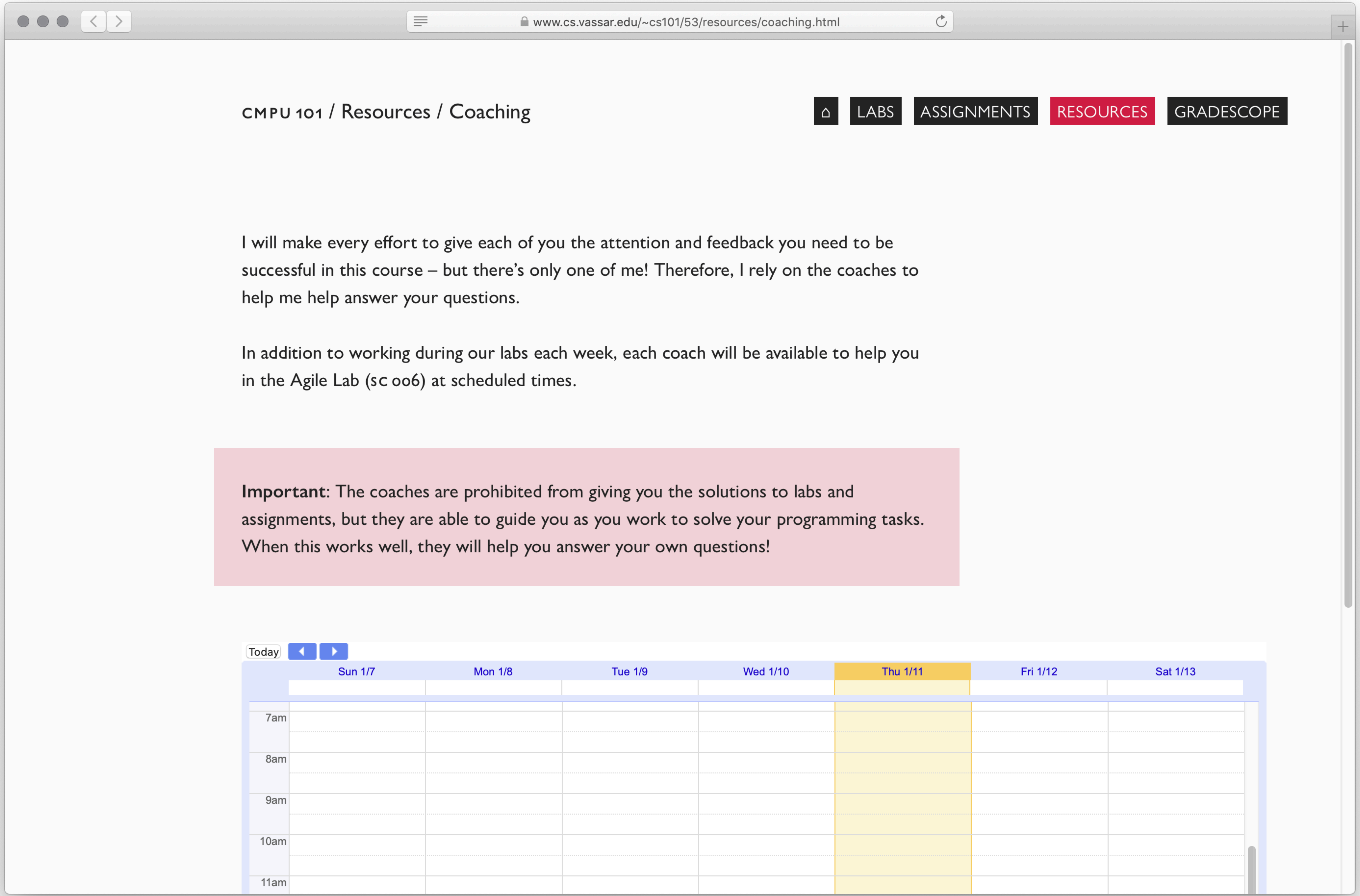
Evaluating Functions and Conditionals

25 January 2024

Assignment 1

Out today, at 5 p.m.

Due on Wednesday by 11:59 p.m.



www.cs.vassar.edu/~cs101/53/resources/coaching.html

CMPU 101 / Resources / Coaching

⌂ LABS ASSIGNMENTS RESOURCES GRADESCOPE

I will make every effort to give each of you the attention and feedback you need to be successful in this course – but there's only one of me! Therefore, I rely on the coaches to help me help answer your questions.

In addition to working during our labs each week, each coach will be available to help you in the Agile Lab (SC 006) at scheduled times.

Important: The coaches are prohibited from giving you the solutions to labs and assignments, but they are able to guide you as you work to solve your programming tasks. When this works well, they will help you answer your own questions!

Today

	Sun 1/7	Mon 1/8	Tue 1/9	Wed 1/10	Thu 1/11	Fri 1/12	Sat 1/13
7am							
8am							
9am							
10am							
11am							

Where are we?

We've been using Pyret to write expressions using

data, including

numbers like `0`, `-10`, and `0.4`;

strings like `""`, `"hi"`, and `"111"`; and

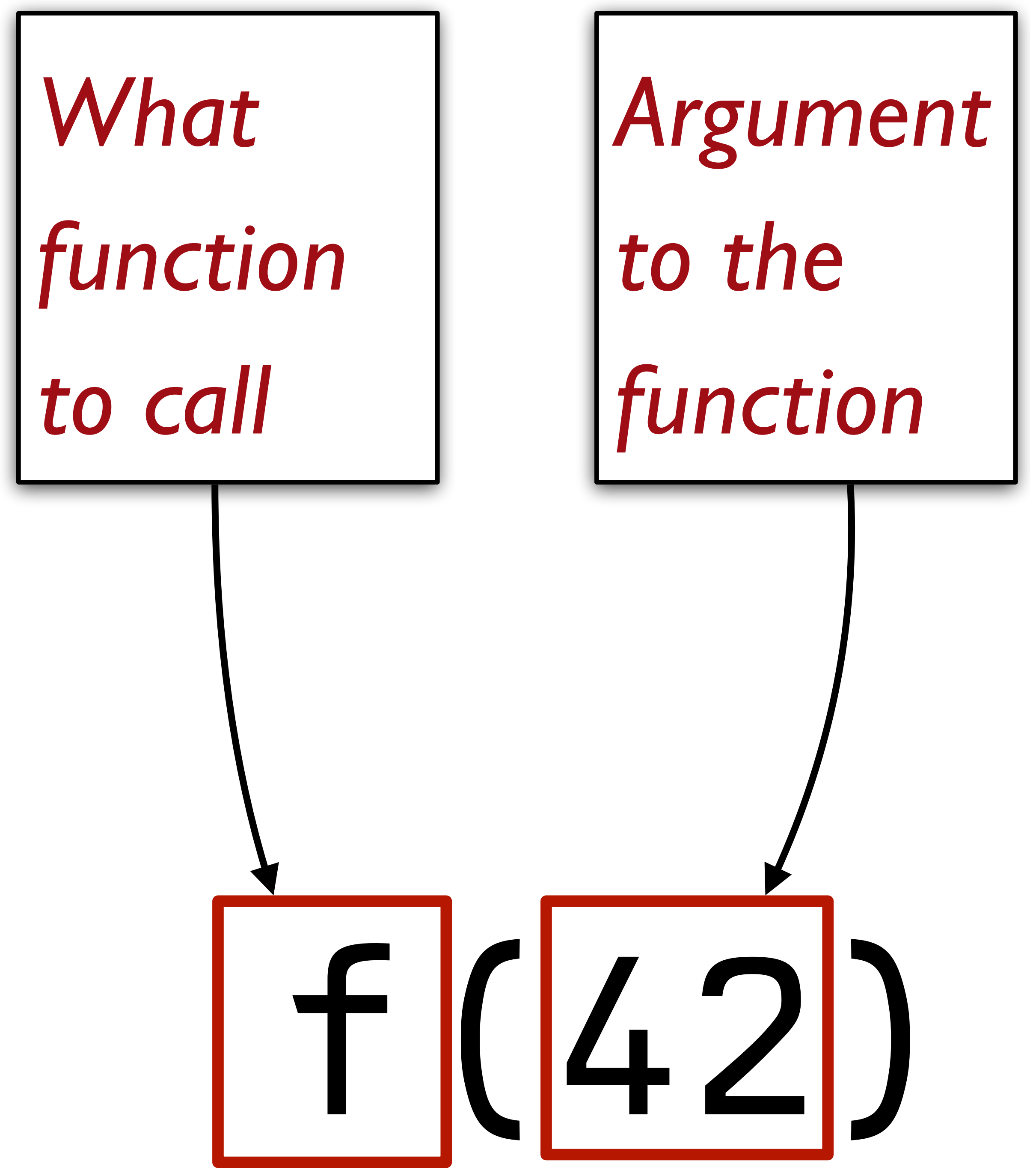
images like ●, a.k.a., `circle(2, "solid", "red")`,

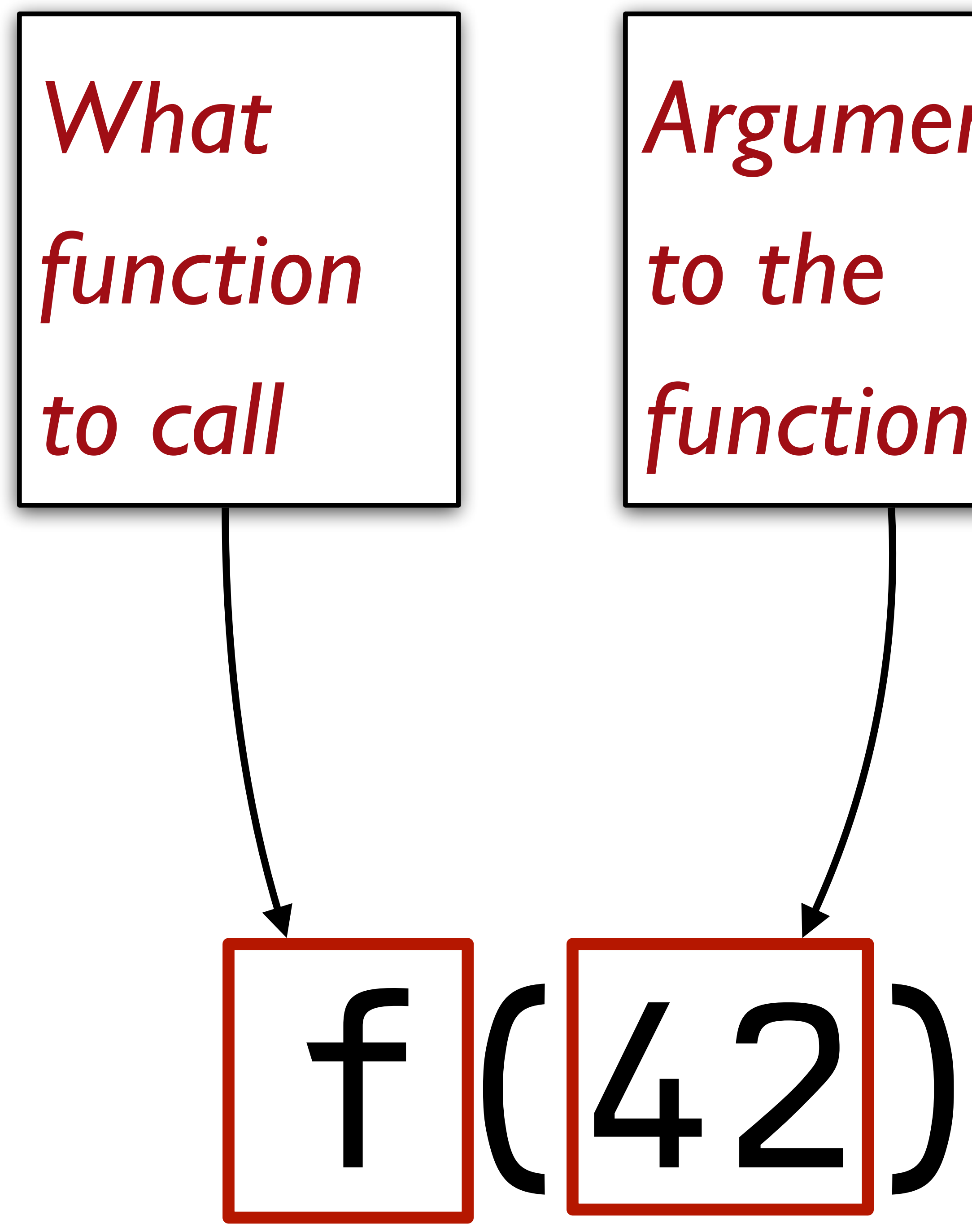
which we modify or combine using operators like **`+`** and **`*`** and functions like **`string-append`** and **`above`**.

```
f(42)
```

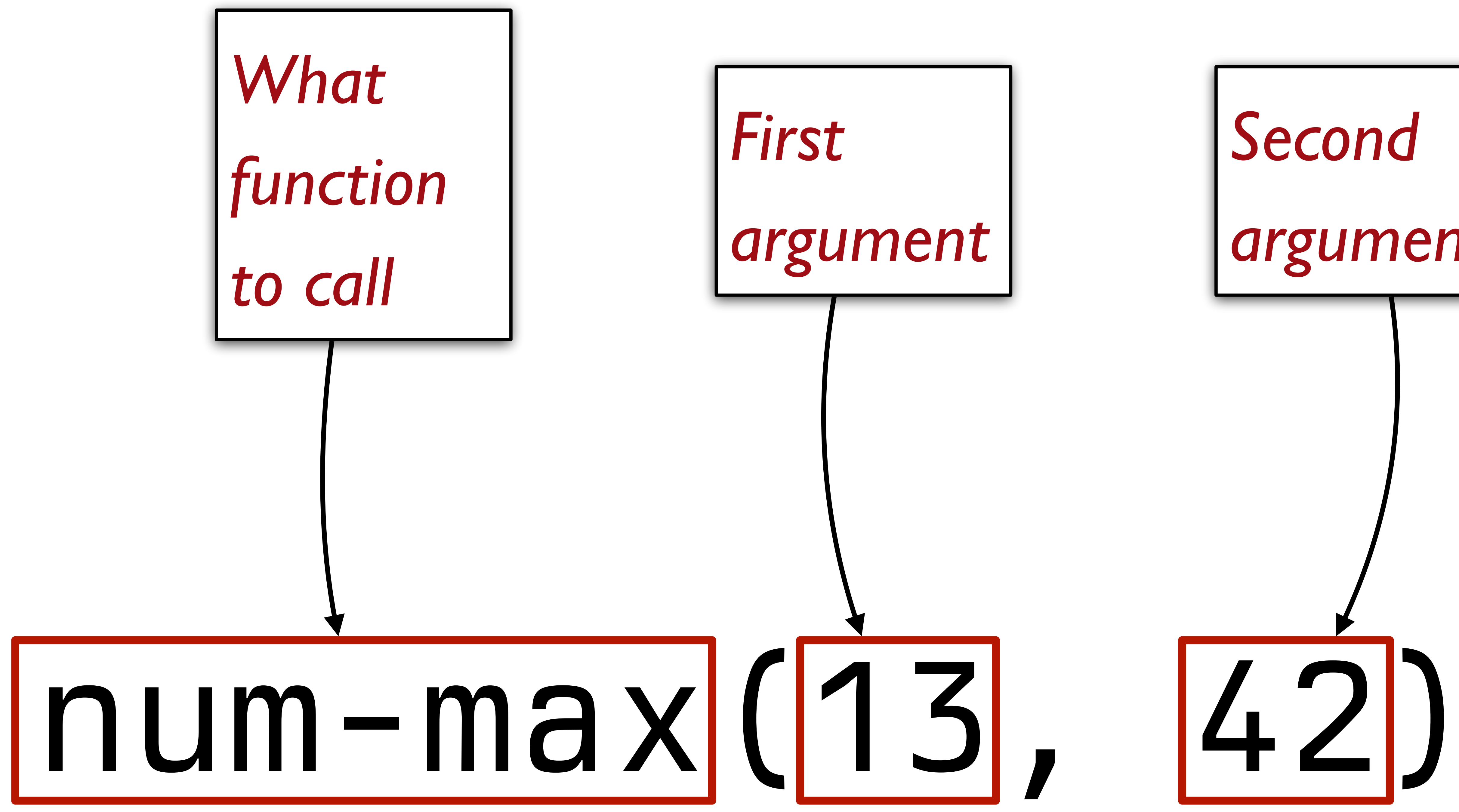
What function to call

```
f(42)
```


What function to call
Argument to the function
f(42)



"Call f on 42."


What function to call
First argument
Second argument
num-max(13, 42)

Distinguishing types of data helps to catch mistakes.

If you try to give

  a string to `/` or

  a number to **`overlay`**,

we want Pyret to catch the problem right away,
giving a helpful error message.

en.wikipedia.org/wiki/Fail-fast_system

WIKIPEDIA
The Free Encyclopedia

Search Wikipedia | Search

Create account Log in

Fail-fast system

2 languages

Article Talk | Read Edit View history Tools

Contents hide

From Wikipedia, the free encyclopedia

This article includes a list of general references, but **it lacks sufficient corresponding inline citations**. Please help to improve this article by introducing more precise citations. *(June 2016)* *(Learn how and when to remove this template message)*

In systems design, a **fail-fast system** is one which immediately reports at its interface any condition that is likely to indicate a failure. Fail-fast systems are usually designed to stop normal operation rather than attempt to continue a possibly flawed process. Such designs often check the system's state at several points in an operation, so any failures can be detected early. The responsibility of a fail-fast module is detecting errors, then letting the next-highest level of the system handle them.

Hardware and software [edit]

Fail-fast systems or modules are desirable in several circumstances:

- Fail-fast architectures are based on an error handling policy where any detected error or non-contemplated state makes the system fail (fast). In some sense the error handling policy is the opposite of that used in a fault-tolerant system. In a fault-tolerant system an error handling policy is established to have redundant components and move computation requests to alive components when some component fails. Paradoxically fail-fast systems make fault-tolerant systems more resilient. We can have 10 redundant servers for a given database, but if the shared configuration for the 10 servers is updated with wrong authentication data for clients, all of them will "redundantly fail". In that sense, a fail-fast system will get sure that all the 10 redundant servers fail as soon as possible to make the DevOps react fast.
- Fail-fast components are often used in situations where failure in one component might not be visible until it leads to failure in another component as a consequence of lazy initialization. e.g. "The system that is "doomed" to fail because a file-system path is wrongly setup, does it not fail at startup because the file-system path is not checked at startup. Only when a client-request arrives the system fails, at random, later on.
- Finding the cause of a failure is easier in a fail-fast system, because the system reports the failure with as much information as possible as close to the time of failure as possible. In a fault-tolerant system, the failure might go undetected, whereas in a system that is neither fault-tolerant nor fail-fast the failure might be temporarily hidden until it causes some seemingly unrelated problem later.
- A fail-fast system that is designed to halt as well as report the error on failure is less likely to erroneously perform an irreversible or costly operation.

Developers also refer to code as fail-fast if it tries to fail as soon as possible at variable or object initialization. In object-oriented

We've seen that we can create more complicated programs by composing function calls, e.g.,

```
    1 + (2 / 3)
```

or

```
    string-append("hello ",
      string-append("Pyret ", "world!"))
```

And we can give a name to the result of an expression, e.g.,

```
        total = 2 + 3
```

And we can give a name to the result of an expression, e.g.,

```
total = 2 + 3
```

Directory

Name	***Value***
total	

And we can give a name to the result of an expression, e.g.,

```
  total = 2 + 3
→ total = 5
```

Directory

Name	***Value***
total	

And we can give a name to the result of an expression, e.g.,

```
  total = 2 + 3
→ total = 5
```

Directory

Name	***Value***
total	5

And we can give a name to the result of an expression, e.g.,

```
  total = 2 + 3
→ total = 5

  new-total = total + 1
```

Directory

Name	***Value***
total	5

And we can give a name to the result of an expression, e.g.,

```
  total = 2 + 3
→ total = 5

  new-total = total + 1
```

Directory

Name	***Value***
total	5
new-total	

And we can give a name to the result of an expression, e.g.,

```
   total = 2 + 3
→  total = 5

   new-total = total + 1
```

Directory

Name	***Value***
total	5
new-total	

And we can give a name to the result of an expression, e.g.,

```
  total = 2 + 3
→ total = 5

  new-total = total + 1
→ new-total = 5 + 1
```

Directory

Name	***Value***
total	5
new-total	

And we can give a name to the result of an expression, e.g.,

```
  total = 2 + 3
→ total = 5

  new-total = total + 1
→ new-total = 5 + 1
→ new-total = 6
```

Directory

Name	***Value***
total	5
new-total	

And we can give a name to the result of an expression, e.g.,

```
  total = 2 + 3
→ total = 5

  new-total = total + 1
→ new-total = 5 + 1
→ new-total = 6
```

Directory

Name	***Value***
`total`	5
`new-total`	6

Defining and evaluating functions

Remember functions from middle-school math:

Given $f(x) = \cos(x) + 2$

$f(0) = 1 + 2 = 3$

The parameter x stands for varying values

Pyret functions work much the same way:

```
    fun f(x): num-cos(x) + 2 end

    f(0)
 →  num-cos(x) + 2
 →  num-cos(0) + 2
 →  1 + 2
 →  3
```

Directory

Name	***Value***
x	`0`

Note that the parameter names are only defined inside the function body:

```
››› fun f(x): num-cos(x) + 2 end
››› f(0)
3
››› x
Error!
```

Once the function is finished, the names are removed from the directory.

We say a parameter name has only *local scope*, while names defined outside a function have *global scope*.

Example

Mary Berry needs to know how many cakes to bake for her cake shop.

To avoid running out or having too many, she wants to bake two cakes more than the number she sold the previous day.

E.g., if Mary sells eight cakes on Monday, she makes ten cakes on Tuesday.

Let's write some code to help Mary.

```
fun cakes-to-make(num-sold):
  num-sold + 2
end
```

Keyword to define a ***fun****ction*

```
fun cakes-to-make(num-sold):
  num-sold + 2
end
```

```
fun cakes-to-make(num-sold):
  num-sold + 2
end
```

Name of the function

```
fun cakes-to-make(num-sold):
  num-sold + 2
end
```

```
fun cakes-to-make(num-sold):
  num-sold + 2
end
```

Parameter

```
fun cakes-to-make(num-sold):
  num-sold + 2
end
```

```
fun cakes-to-make(num-sold):
  num-sold + 2
end
```

```
fun cakes-to-make(num-sold):
  num-sold + 2
end
```

How to transform the data

```
fun cakes-to-make(num-sold):
  num-sold + 2
end
```

```
fun cakes-to-make(num-sold):
  num-sold + 2
end
```

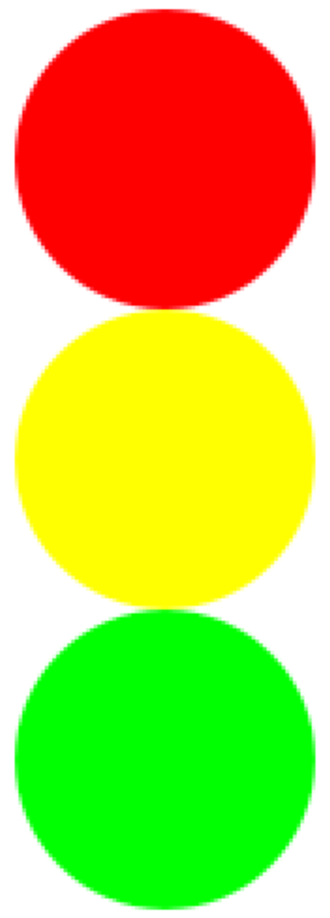
Keyword to signal the ***end*** *of the function definition*

```
fun cakes-to-make(num-sold):
  num-sold + 2
end
```

Functions are abstractions over specific computations

```
# Draw a traffic light
above(  circle(40, "solid", "red"),
  above(circle(40, "solid", "yellow"),
        circle(40, "solid", "green"))
```

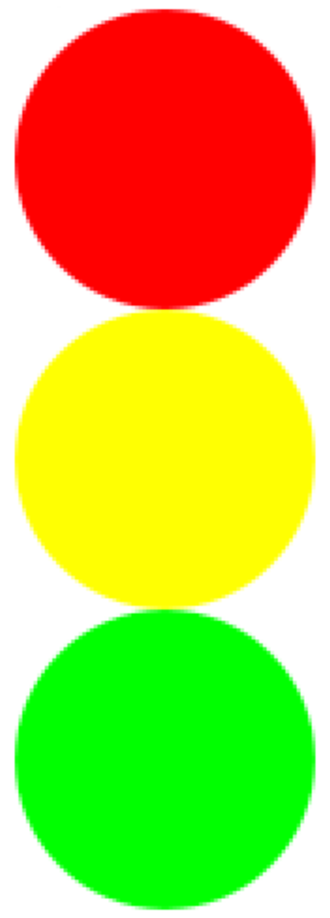
```
# Draw a traffic light
above(  circle(40, "solid", "red"),
  above(circle(40, "solid", "yellow"),
        circle(40, "solid", "green"))
```



```
# Draw a traffic light
above(   circle(40, "solid", "red"),
   above(circle(40, "solid", "yellow"),
         circle(40, "solid", "green")))
```

Unchanging

Varying



```
# Draw a traffic light
above(circle(40, "solid", "red"),
  above(circle(40, "solid", "yellow"),
    circle(40, "solid", "green")))
```

```
# Draw a traffic light
above(circle(40, "solid", "red"),
  above(circle(40, "solid", "yellow"),
    circle(40, "solid", "green")))

# Can be changed to
fun bulb(color):
  circle(40, "solid", color)
end

above(bulb("red"),
  above(bulb("yellow"),
    bulb("green")))
```

```
fun bulb(color):
  circle(40, "solid", color)
end

above(bulb("red"),
  above(bulb("yellow"),
    bulb("green")))
```

```
fun bulb(color):
  circle(40, "solid", color)
end

fun traffic-light():
  above(bulb("red"),
    above(bulb("yellow"),
      bulb("green")))
end
```

Remember: Each function has *one* job!

Example

For Mary’s cake shop, we want to determine the price of each cake based on the cost of the ingredients and the time to prepare it.

As the price, she uses twice the cost of the ingredients plus ¼ of the preparation time in minutes.

For Mary’s cake shop, we want to determine the price of each cake based on the cost of the ingredients and the time to prepare it.

As the price, she uses twice the cost of the ingredients plus ¼ of the preparation time in minutes.

Chocolate cake
Ingredients: $10
Prep. time: 20 min.

For Mary's cake shop, we want to determine the price of each cake based on the cost of the ingredients and the time to prepare it.

As the price, she uses twice the cost of the ingredients plus ¼ of the preparation time in minutes.

Chocolate cake
Ingredients: $10
Prep. time: 20 min.

```
choc-cake-price = (2 * 10) + (1/4 * 20)
```

For Mary's cake shop, we want to determine the price of each cake based on the cost of the ingredients and the time to prepare it.

As the price, she uses twice the cost of the ingredients plus ¼ of the preparation time in minutes.

Chocolate cake
Ingredients: $10
Prep. time: 20 min.

```
choc-cake-price = (2 * 10) + (1/4 * 20)
```

Cheesecake
Ingredients: $15
Prep. time: 36 min.

For Mary's cake shop, we want to determine the price of each cake based on the cost of the ingredients and the time to prepare it.

As the price, she uses twice the cost of the ingredients plus ¼ of the preparation time in minutes.

Chocolate cake
Ingredients: $10
Prep. time: 20 min.

```
choc-cake-price = (2 * 10) + (1/4 * 20)
```

Cheesecake
Ingredients: $15
Prep. time: 36 min.

```
cheesecake-price = (2 * 15) + (1/4 * 36)
```

We use functions to avoid repetitive code when we need to perform the same operations on different values.

```
choc-cake-price = (2 * 10) + (1/4 * 20)

cheesecake-price = (2 * 15) + (1/4 * 36)
```

We use functions to avoid repetitive code when we need to perform the same operations on different values.

```
choc-cake-price = (2 * 10) + (1/4 * 20)

cheesecake-price = (2 * 15) + (1/4 * 36)
```

```
(2 * ingredients-cost) + (1/4 * prep-time)
```

We use functions to avoid repetitive code when we need to perform the same operations on different values.

```
choc-cake-price = (2 * 10) + (1/4 * 20)

cheesecake-price = (2 * 15) + (1/4 * 36)
```

```
fun cake-price(ingredients-cost, prep-time):
  (2 * ingredients-cost) + (1/4 * prep-time)
end
```

Parameters

```
fun cake-price(ingredients-cost, prep-time):
  (2 * ingredients-cost) + (1/4 * prep-time)
end
```

The parameters are the values passed into the function that it needs to know for each operation.

```
fun cake-price(ingredients-cost, prep-time):
  (2 * ingredients-cost) + (1/4 * prep-time)
end
```

Expression repeated each time the function is called

```
fun cake-price(ingredients-cost, prep-time):
  (2 * ingredients-cost) + (1/4 * prep-time)
end
```

```
# Price of chocolate cake
cake-price(10, 20)

# Price of cheesecake
cake-price(15, 36)
```

To calculate the price of chocolate cake or cheesecake, you just call your function and pass in the relevant values!

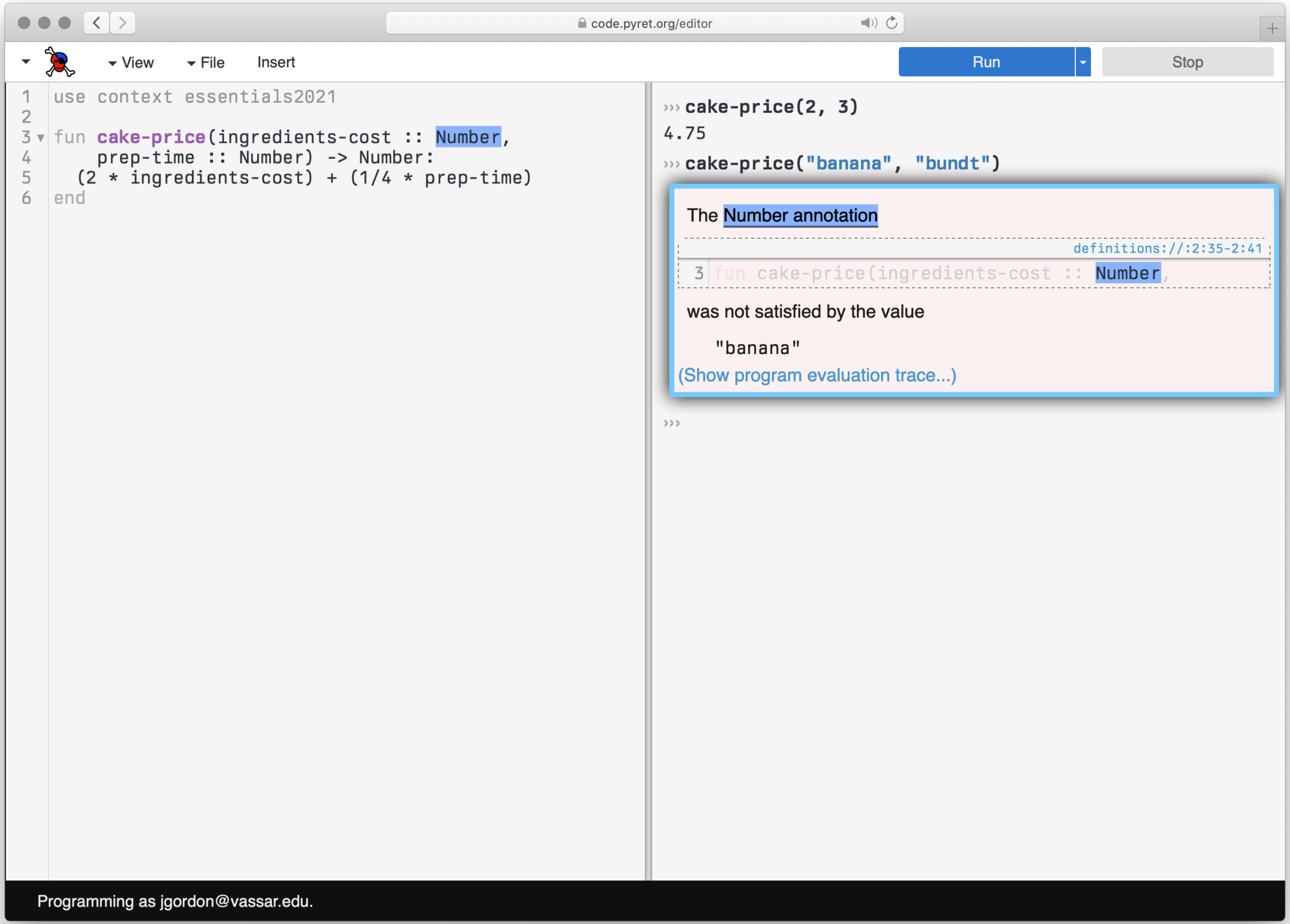
Improving our function definitions

```
fun cake-price(ingredients-cost :: Number,
    prep-time :: Number):
  (2 * ingredients-cost) + (1/4 * prep-time)
end
```

*We specify the **type** of each parameter so that Pyret will check that we pass in the right kind of values, just like for built-in operations like* `+` *and* `above`*.*

```
fun cake-price(ingredients-cost :: Number,
    prep-time :: Number) -> Number:
  (2 * ingredients-cost) + (1/4 * prep-time)
end
```

*And we can specify the type of value the function **returns**.*



```
use context essentials2021

fun cake-price(ingredients-cost :: Number,
    prep-time :: Number) -> Number:
  (2 * ingredients-cost) + (1/4 * prep-time)
end
```

```
››› cake-price(2, 3)
4.75
››› cake-price("banana", "bundt")
```

The Number annotation

```
definitions://:2:35-2:41
3  fun cake-price(ingredients-cost :: Number,
```

was not satisfied by the value

```
"banana"
```

(Show program evaluation trace...)

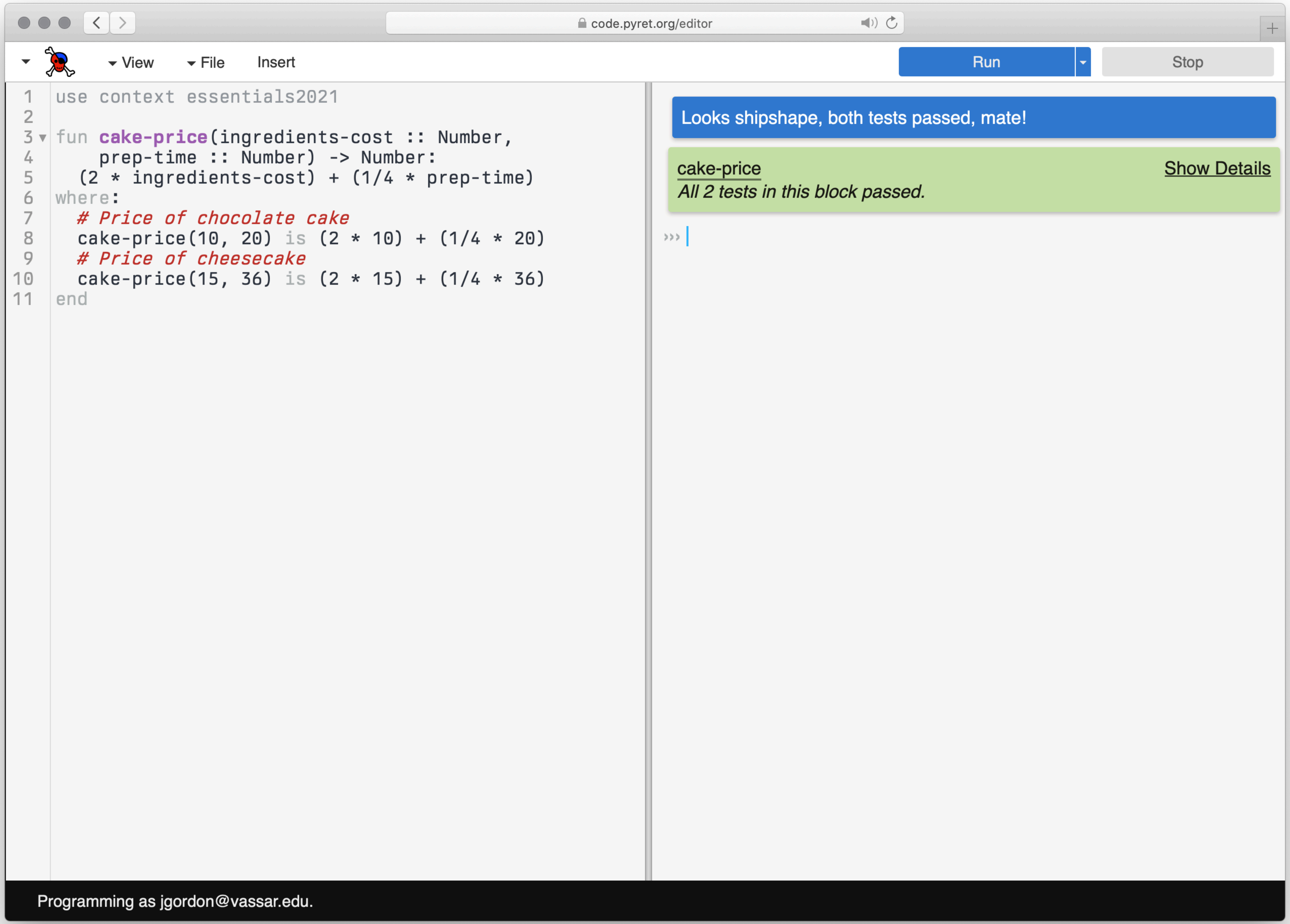
```
›››
```

Programming as jgordon@vassar.edu.

```
fun cake-price(ingredients-cost :: Number,
    prep-time :: Number) -> Number:
  doc: "Calculate price of cake based on
ingredient cost and preparation time"
  (2 * ingredients-cost) + (1/4 * prep-time)
end
```

*Additionally, a **docstring** explains what the function does.*

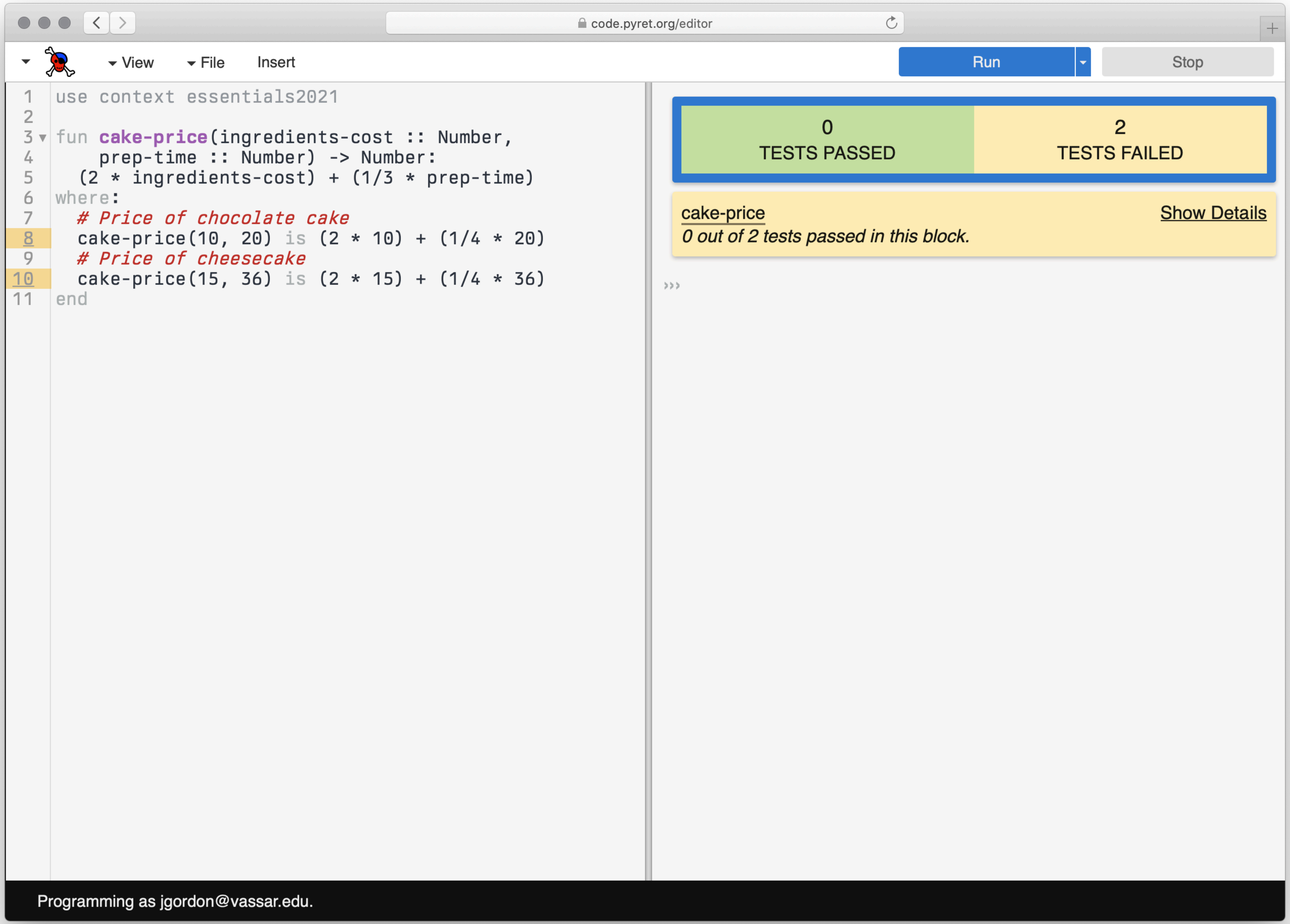
```
fun cake-price(ingredients-cost :: Number,
    prep-time :: Number) -> Number:
  doc: "Calculate price of cake based on
ingredient cost and preparation time"
  (2 * ingredients-cost) + (1/4 * prep-time)
where:
  # Price of chocolate cake
  cake-price(10, 20) is (2 * 10) + (1/4 * 20)
  # Price of cheesecake
  cake-price(15, 36) is (2 * 15) + (1/4 * 36)
end
```



```
use context essentials2021

fun cake-price(ingredients-cost :: Number,
    prep-time :: Number) -> Number:
  (2 * ingredients-cost) + (1/4 * prep-time)
where:
  # Price of chocolate cake
  cake-price(10, 20) is (2 * 10) + (1/4 * 20)
  # Price of cheesecake
  cake-price(15, 36) is (2 * 15) + (1/4 * 36)
end
```

```
Looks shipshape, both tests passed, mate!

cake-price                                   Show Details
All 2 tests in this block passed.

›››
```

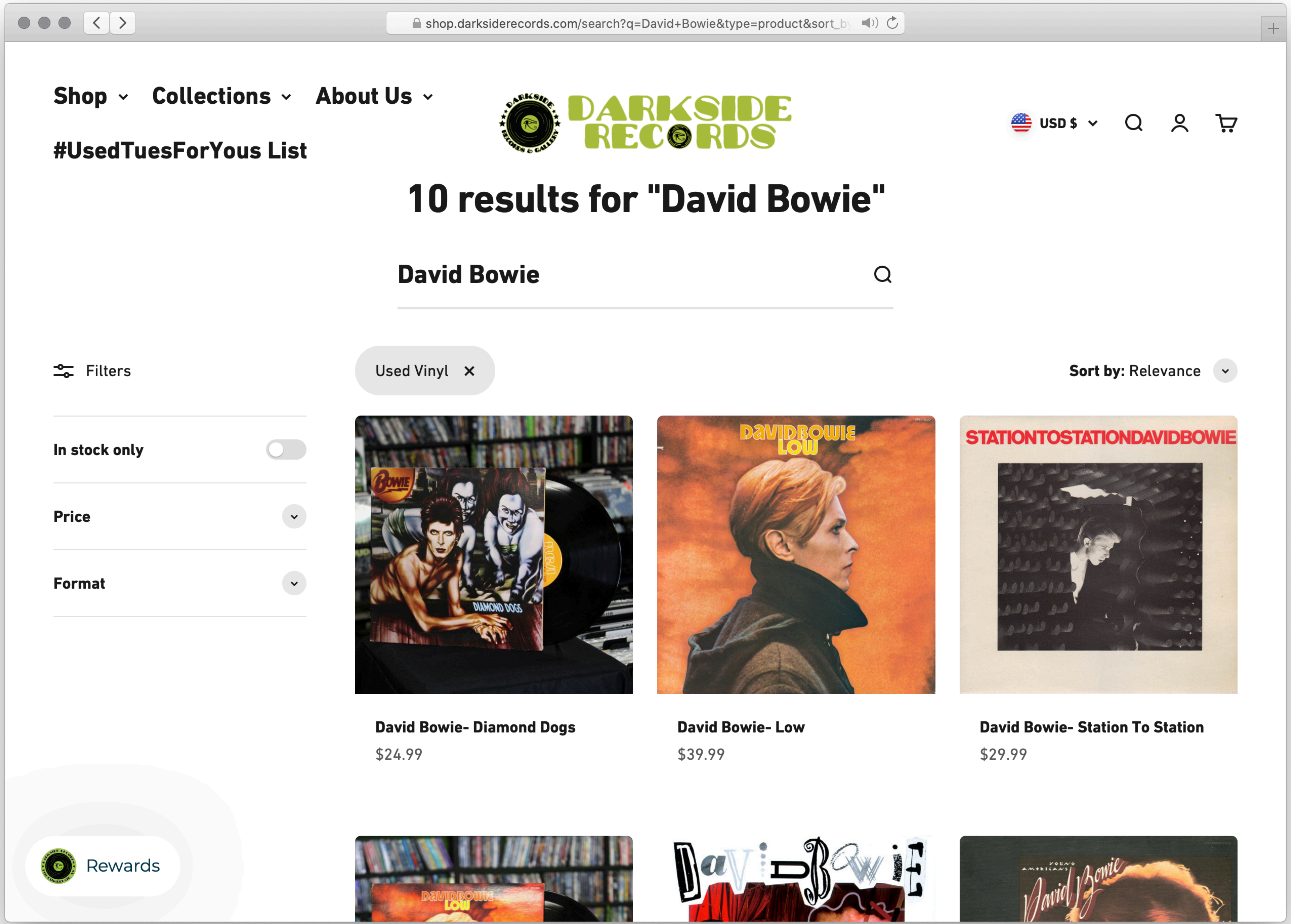
Programming as jgordon@vassar.edu.



```
use context essentials2021

fun cake-price(ingredients-cost :: Number,
    prep-time :: Number) -> Number:
  (2 * ingredients-cost) + (1/3 * prep-time)
where:
  # Price of chocolate cake
  cake-price(10, 20) is (2 * 10) + (1/4 * 20)
  # Price of cheesecake
  cake-price(15, 36) is (2 * 15) + (1/4 * 36)
end
```

```
0
TESTS PASSED

2
TESTS FAILED

cake-price                                   Show Details
0 out of 2 tests passed in this block.

›››
```

Programming as jgordon@vassar.edu.



shop.darksiderecords.com/search?q=David+Bowie&type=product&sort_by

Shop
Collections
About Us
#UsedTuesForYous List

USD $

10 results for "David Bowie"

David Bowie

Filters

Used Vinyl ✕

Sort by: Relevance

In stock only

Price

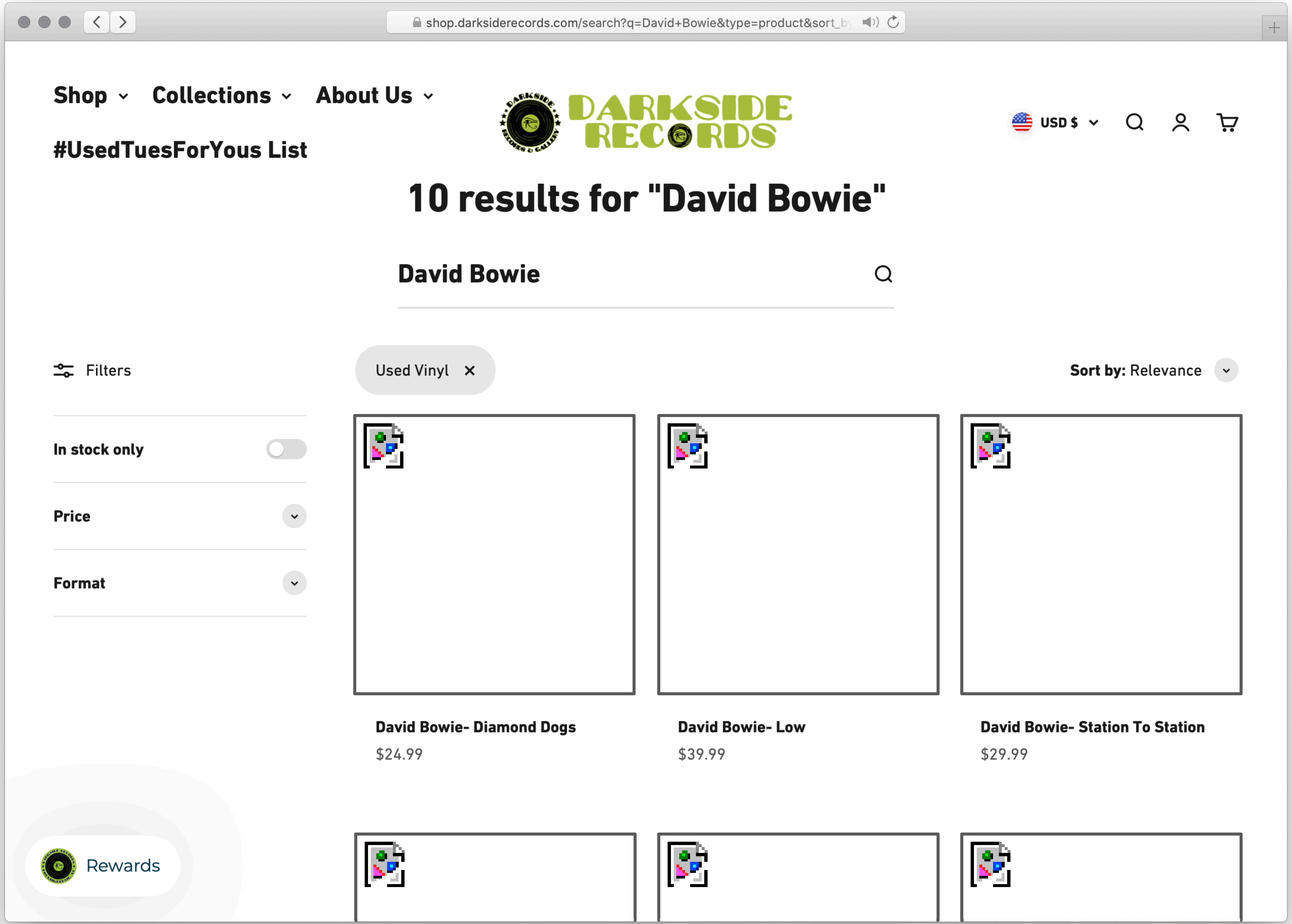
Format

David Bowie- Diamond Dogs
$24.99

David Bowie- Low
$39.99

David Bowie- Station To Station
$29.99

Rewards



shop.darksiderecords.com/search?q=David+Bowie&type=product&sort_by

Shop
Collections
About Us
#UsedTuesForYous List

USD $

10 results for "David Bowie"

David Bowie

Filters

In stock only

Price

Format

Used Vinyl ✕

Sort by: Relevance

David Bowie- Diamond Dogs	David Bowie- Low	David Bowie- Station To Station
$24.99	$39.99	$29.99

Rewards

```
fun rectangle-area(r):
  image-height(r) * image-width(r)
end
```

```
fun rectangle-area(r :: Image) -> Number:
  doc: "Return the rectangular area of the image"
  image-height(r) * image-width(r)
where:
  rectangle-area(rectangle(0, 0, "solid", "black"))
    is 0
  rectangle-area(rectangle(2, 3, "outline", "blue"))
    is 6
end
```

```
fun rectangle-area(r :: Image) -> Number:
  doc: "Return the rectangular area of the image"
  image-height(r) * image-width(r)
where:
  tiny = rectangle(0, 0, "solid", "black")
  rectangle-area(tiny) is 0

  blue = rectangle(2, 3, "outline", "blue")
  rectangle-area(blue) is 6
end
```

Booleans and `if` expressions

```
true

false
```

We can compare values using these operators

`<`	less than
`<=`	less than or equal to
`>`	greater than
`>=`	greater than or equal to
`==`	equal to
`<>`	not equal to

which produce `true` or `false` as a result.

Be careful:

```
x = 2
```

is assigning the name **x** to have the value **2** in the directory.

```
x == 2
```

is asking the question “is **x** equal to **2**?”

Boolean expressions can also be combined using the operators

and

`true` if both inputs are `true`;

`false` otherwise

or

`false` if both inputs are `false`;

`true` otherwise

```
››› true and false
false
››› true or false
true
››› (1 < 2) and (2 > 3)
false
››› (1 <= 0) or (1 == 1)
true
```

To change an expression that evaluates to `true` to be `false` – or vice versa – use the **`not`** function:

```
››› not(true)
false
››› not(1 == 0)
true
```

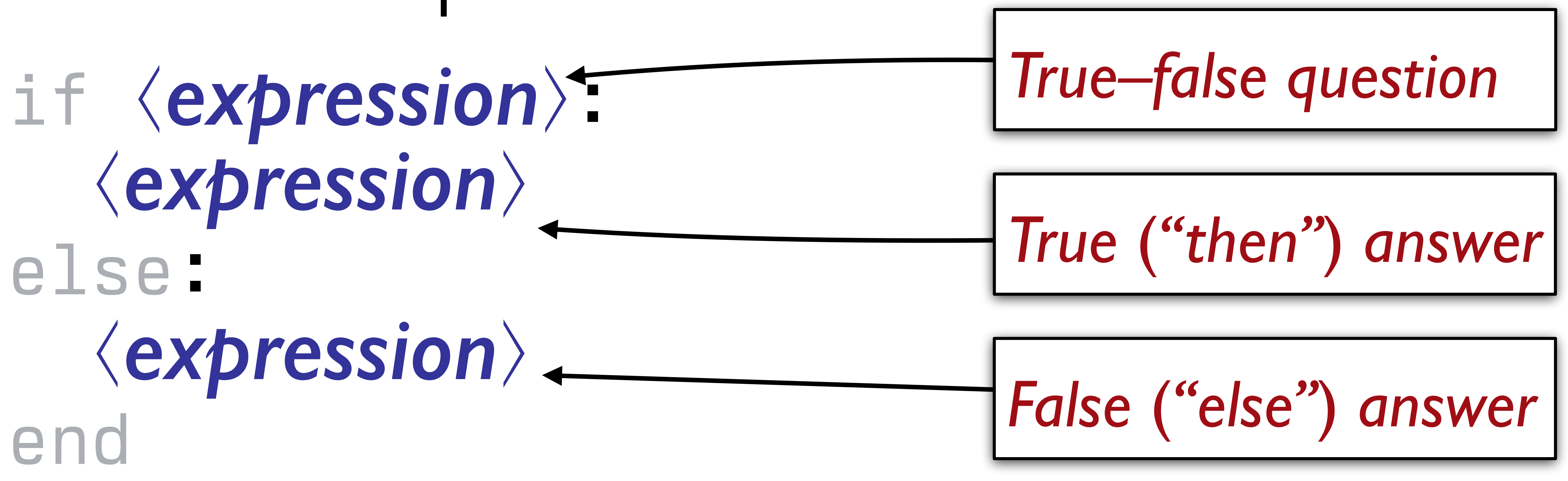
```
i1 = rectangle(10, 20, "solid", "red")
i2 = rectangle(20, 10, "solid", "blue")

image-width(i1) < image-width(i2)
```

```
rect = rectangle(10, 20, "solid", "red")

if image-width(rect) < image-height(rect):
  "portrait"
else:
  "landscape"
end
```

`if ... else ... end` is a *conditional expression*.

Conditionals allow us to *branch* – maybe we evaluate this expression, or maybe this other expression instead!

To form an **if** expression:



```
if ⟨expression⟩:
   ⟨expression⟩
else:
   ⟨expression⟩
end
```

How an `if` expression is evaluated

```
if 1 < 2:
  "All is right in the world"
else:
  "Watch out for flying pigs"
end
```

1. If the question expression is not a value, evaluate it, and replace with the resulting value.

How an `if` expression is evaluated

```
if true:
  "All is right in the world"
else:
  "Watch out for flying pigs"
end
```

1. If the question expression is not a value, evaluate it, and replace with the resulting value.

How an **`if`** expression is evaluated

```
if true:
  "All is right in the world"
else:
  "Watch out for flying pigs"
end
```

2 If the question is `true`, replace the entire **`if`** expression with the true (“then”) answer expression.

How an **`if`** expression is evaluated

```
"All is right in the world"
```

2 If the question is `true`, replace the entire **`if`** expression with the true (“then”) answer expression.

How an **`if`** expression is evaluated

```
if false:
  "All is right in the world"
else:
  "Watch out for flying pigs"
end
```

3 If the question is `false`, replace the entire **`if`** expression with the false (“else”) answer expression.

How an **`if`** expression is evaluated

```
"Watch out for flying pigs"
```

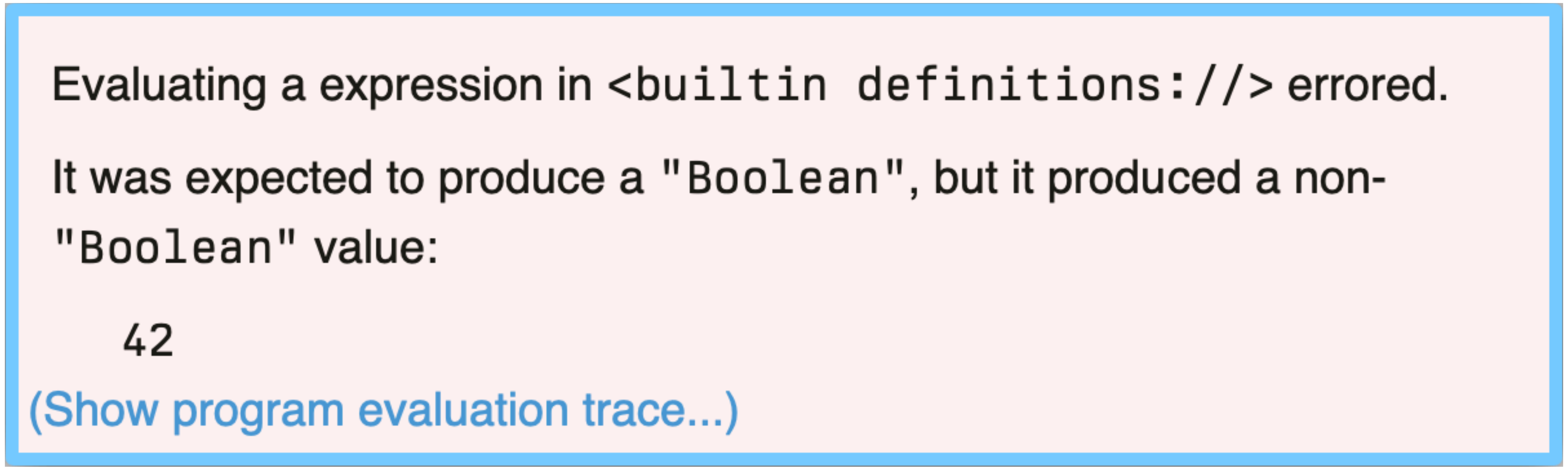
3 If the question is `false`, replace the entire **`if`** expression with the false (“else”) answer expression.

How an `if` expression is evaluated

```
if 42:
  "All is right in the world"
else:
  "Watch out for flying pigs"
end
```

4 Otherwise, the question must be a value other than `true` or `false`, so produce an error.

How an `if` expression is evaluated


Evaluating a expression in `<builtin definitions://>` errored.

It was expected to produce a `"Boolean"`, but it produced a non-`"Boolean"` value:

```
    42
```

(Show program evaluation trace...)

4 Otherwise, the question must be a value other than `true` or `false`, so produce an error.

```
rect = rectangle(10, 20, "solid", "red")

if image-width(rect) < image-height(rect):
  "portrait"
else:
  "landscape"
end
```

What’s wrong with this code?

```
rect = rectangle(10, 20, "solid", "red")

if image-width(rect) < image-height(rect):
  "portrait"
else if image-width(rect) == image-height(rect):
  "square"
else:
  "landscape"
end
```

```
rect = rectangle(10, 20, "solid", "red")

fun image-type(img :: Image) -> String:
  doc: "Classify an image as portrait, square, or landscape"
  if image-width(img) < image-height(img):
    "portrait"
  else if image-width(img) == image-height(img):
    "square"
  else:
    "landscape"
  end
where:
  image-type(rect) is "portrait"
end
```

```
rect = rectangle(10, 20, "solid", "red")

fun image-type(img :: Image) -> String:
  doc: "Classify an image as portrait, square, or landscape"
  if image-width(img) < image-height(img):
    "portrait"
  else if image-width(img) == image-height(img):
    "square"
  else:
    "landscape"
  end
where:
  image-type(rect) is "portrait"
  image-type(rectangle(10, 10, "solid", "blue")) is "square"
  image-type(rectangle(20, 10, "solid", "blue")) is "landscape"
end
```

Acknowledgments

This class incorporates material from:

Kathi Fisler, Brown University

Gregor Kiczales, University of British Columbia

Peter Lemieszewski, Vassar College