

CMPU 101 § 04 and 05 · Problem-Solving and Abstraction

# Names, Types, and Organization

1 September 2021



Where are we?

A *program* (or *script*) instructs a computer to do something.

These instructions must be very specific for the computer to carry them out.

But programs also need to be understood by people, so they must be readable!



**Armenia**



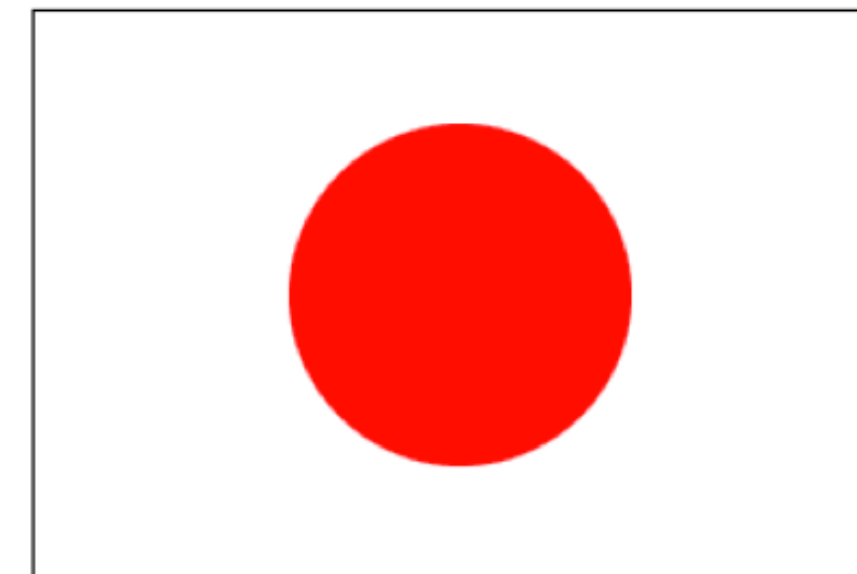
**Austria**



**Colombia**



**Zambia**



**Japan**

We saw we need at least three types of data:

*Numbers* to describe flag dimensions

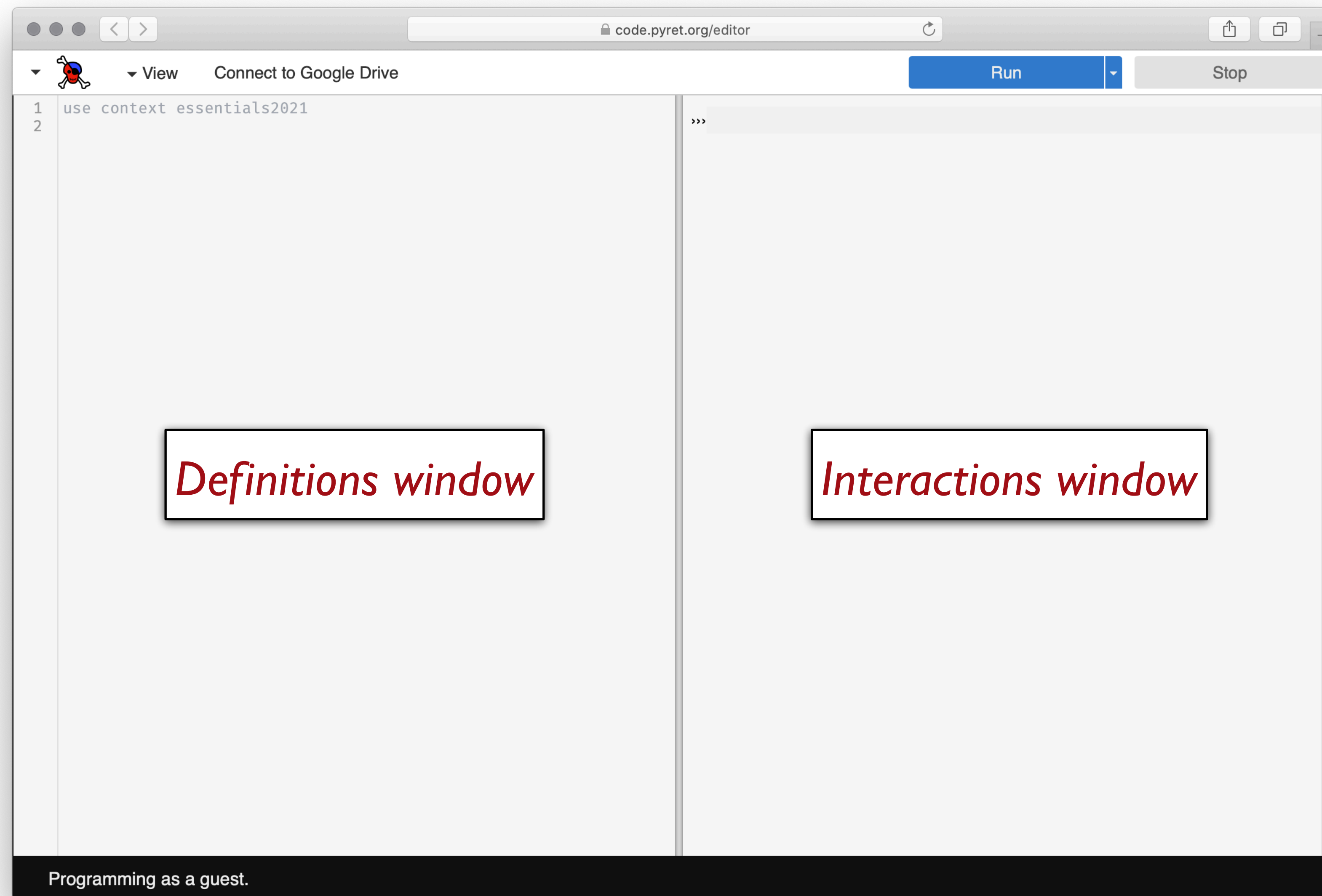
*Strings* to describe colors

*Images* for the flags and the shapes in them

To work with these data types, we need to use a *programming language* and *programming environment*.

We write our computation in the language.

We run the program in the environment.



code.pyret.org

Use the *interactions window* for:

- Trying out expressions

- Checking syntax

Use the *definitions window* for:

- Building complex expressions

- Naming expressions

- Using previously defined expressions

- Saving your code as files!



code.pyret.org/editor



Run



Stop

>>>

*Prompt*

```
>>> 3 + 4 * 5
```

Reading this expression errored:

[interactions://1:0:0-0:9](#)

```
1 3 + 4 * 5
```

The **+** and **\*** operations are at the same grouping level. Add parentheses to group the operations, and make the order of operations clear.

```
>>> 3 + (4 * 5)
```

```
23
```

```
>>> |
```

$(3 + 4) * (5 + 1)$  is an *expression* – a computation that produces an answer.

A program consists of one or more computations you want to run.

An individual number like **5** or string like `"red"` is a *value* – it can't be computed any further.

In other words, a value is its own answer.

We can call functions that will draw an image of a particular shape, e.g.,

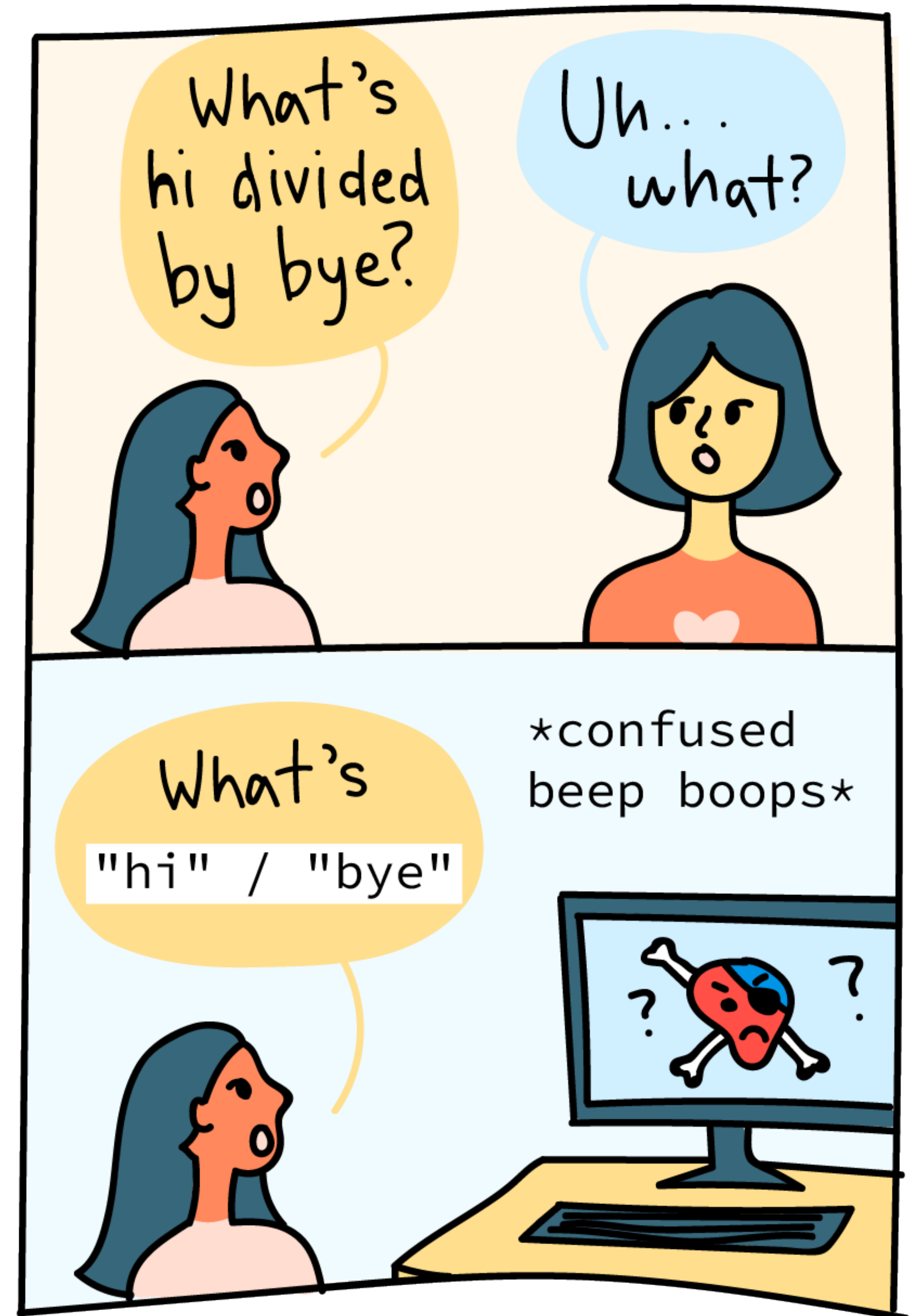
```
circle(30, "solid", "red")
```

We can manipulate images much like we can manipulate numbers.

Numbers can be added, subtracted, etc.

Images can overlaid, rotated, flipped, etc.

Operations may only work on certain types of data!



# Evaluation



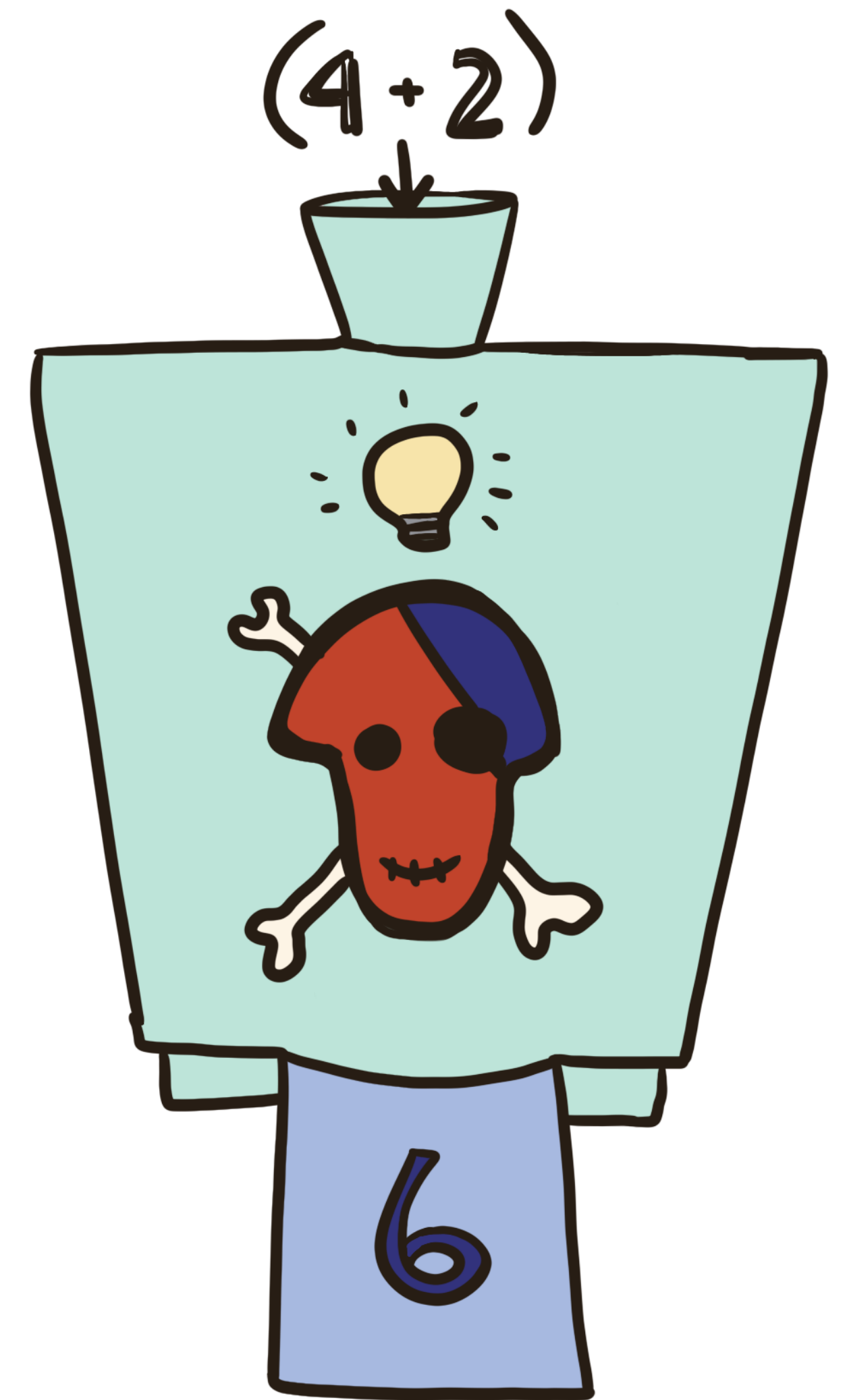
How does something like  $(4 + 2) / 3$  work?

What is the operator  $/$  dividing?

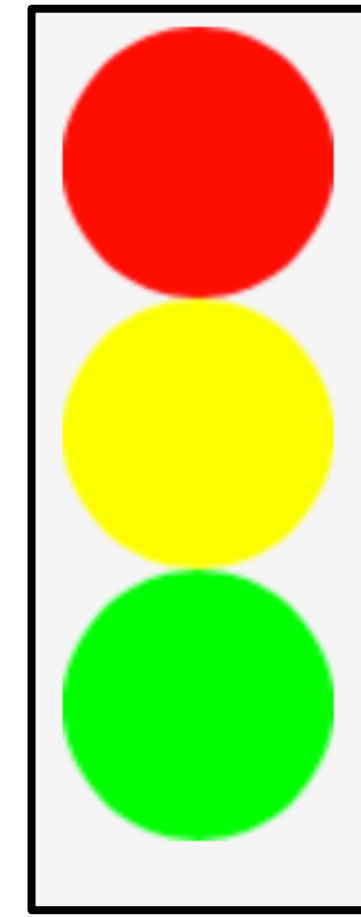
Shouldn't  $/$  expect two numbers?

Even though  $(4 + 2)$  isn't a number, it's an expression that *evaluates* to a number.

This works for all data types, not just numbers!



Let's say we want to make this image:



How could we do this using **above**?

**above** only takes two arguments, but we can make one of the arguments be the image that results from another call to **above**!

```
>>> above(above(circle(30, "solid", "red"), circle(30, "solid", "yellow")), circle(30, "solid", "green"))
```



What's in a name?

This is a lot to type:

```
above(  
  above(  
    circle(30, "solid", "red"),  
    circle(30, "solid", "yellow")),  
  circle(30, "solid", "green"))
```

But we can give it a name:

```
traffic-light = above(  
  above(  
    circle(30, "solid", "red"),  
    circle(30, "solid", "yellow")),  
  circle(30, "solid", "green"))
```


<

>

code.pyret.org/editor

+

▼



View

File

Insert

Run

Stop

1 use context essentials2021

2

3 **include** image

4

5 traffic-light = above(


6   above(

7     circle(30, "solid", "red"),

8     circle(30, "solid", "yellow")),

9   circle(30, "solid", "green"))

>>> traffic-light



>>> |

Programming as jgordon@vassar.edu.

An expression of the form

$\langle \textit{name} \rangle = \langle \textit{expression} \rangle$

tells Pyret to associate the value of  $\langle \textit{expression} \rangle$  with  $\langle \textit{name} \rangle$ .

Every time you type  $\langle \textit{name} \rangle$ , Pyret will substitute the value for you, e.g.,

$x = 5$   
 $x + 4$

will evaluate to 9.



code.pyret.org/editor

View

File


Insert

Run

Stop

```
1 use context essentials2021
2
3 r = circle(30, "solid", "red")
4 y = circle(30, "solid", "yellow")
5 g = circle(30, "solid", "green")
6
7 traffic-light = above(r, above(y, g))
```

```
>>> traffic-light
```



```
>>>
```

Programming as jgordon@vassar.edu.

*We can also give names to pieces so our traffic-light definition isn't so complex.*

As you build up more complex images from simpler ones, you're following a core idea called *composition*.

Programs are always built of smaller programs that do parts of the larger task you want to perform.

We'll often use composition in this course.

Note there's no output from entering a definition.

It only has a side effect of telling Pyret to associate the name with the value.

```
>>> star(40, "solid", "gold")
```



```
>>> my-star = star(40, "solid", "gold")
```

```
>>> my-star
```



Names must be given a value before being used.

In Pyret, names are *immutable*, which means they can only be defined once.

```
>>> x
```

The identifier x is unbound:

interactions://1:0:0-0:1

1	x
---	---

It is used but not previously defined.

```
>>> x = 3
```

```
>>> x
```

```
3
```

```
>>> x = 4
```

The declaration of x shadows a previous declaration of x

```
>>> x
```

```
3
```

```
>>> |
```

Windows and files

Using the definitions window also lets us save files of our code to load again later, as you'll do for labs and homework assignments.

CPO will save these to your Google Drive if you're logged in.

Which window would I use if...

I want to see if I can make a blue circle?

I want to define **my-shape** as a blue circle and use it later in my code?

I want to see if Pyret will accept this: **print "5"**?

I want to start my assignment now and finish it later?

# Summary

Programs are formed of expressions and definitions.

Expressions compose (or nest) to create larger programs.

The structure of data is reflected in the structure of the expressions that create the data.



