Defining Values

(Also available in Pyret)

Students learn to improve readability, performance and maintanability of code by defining values that repeat in code, just as we would define variables in math.

Lesson Goals	Students will be able to: • Define ***values*** of various types • Simplify a complex expression by replacing repeated parts with defined names, • Explain why ***variable***s are useful in math and programming
Student-facing Goals	• Let's learn how to clean up complex code by defining ***values*** for repeated expressions • Let's learn why ***variables*** are useful in math and programming
Prerequisites	• Simple Data Types • Contracts
Materials	• PDF of all Handouts and Page • Defining Values Starter File • Chinese Flag Starter File • Lesson Slides • Printable Lesson Plan (a PDF of this web page)
Supplemental Materials	• Matching Code to Images using overlay & put-image (Desmos)

Key Points For The Facilitator	• Learning how to define values is a big milestone! It will be used consistently throughout other lessons, so be sure to give students plenty of time to practice this new skill. • Check frequently for understanding of ***data types*** and ***Contract***s during this lesson and throughout subsequent lessons.

Glossary

contract :: a statement of the name, domain, and range of a function

data type :: a way of classifying values, such as: Number, String, Image, Boolean, or any user-defined data structure

value :: a specific piece of data, like 5 or "hello"

variable :: a name or symbol that stands for some value or expression, often a value or expression that changes

What's in Common? *30 minutes*

Overview

This activity introduces the problem with duplicate code, leveraging **Mathematical Practice 7 - Identify and Make Use of Structure**. Students identify a common structure in a series of expressions, and discover how to bind that expression to a name that can be re-used.

Launch

Invite students to take a look at the expressions below:

```
(star 50 "solid" "green")
(scale 3 (star 50 "solid" "green"))
(scale .5 (star 50 "solid" "green"))
(rotate 45 (star 50 "solid" "green"))
(rotate 45 (scale 3 (star 50 "solid" "green")))
```

- What code do they all have in common?
  - `(star 50 "solid" "green")`
- What would happen if you were asked to change the color of all the stars to gold?
  - *We'd have to change the color everywhere it appeared.*

There are lots of potential problems with duplicate code:

- **Readability:** The more code there is, the harder it can be to read.
- **Performance:** Why re-evaluate the same code a dozen times, when we can evaluate it *once* and use the result as many times as we need?
- **Maintainability:** Suppose we needed to change the size of the stars in the examples above. We would have to make sure every line is changed, which leaves a lot of room for error.

Duplicate code is almost always bad!

Since we're using that star over and over again, wouldn't it be nice if we could define a "nickname" for that code, and then use the nickname over and over in place of the expression? And then, if we wanted to change something about all of the stars, we would only have to make the change once, in the definition.

You already know how to do this in math:

$x = 4$ *defines* the nickname x to be the value 4.

WeScheme uses the word "define" to make this even clearer!

We can type `(define x 4)` to define `x` to be the value 4.

Once we've defined `x` to be the value 4 and clicked "Run", anytime we use `x`, the computer will remember that it is *defined* as 4. We can *use* that definition to get an answer. For example, $x + 2$ will evaluate to 6.

Of course, the whole point of defining a value is so that it sticks around and can be used later! That's why programmers put their definitions on the *left-hand side*, known as the **Definitions Area**.

Investigate

- Open the Defining Values Starter File and complete Defining Values - Explore with your partner.
- Add some definitions of your own in the Definitions Area. Be sure to click "Run" again before you try testing them out.
- Complete Which Value(s) Would it Make Sense to Define?.

Synthesize

- What data types can we define values for?
  - *All of them - Number, String, Image*
- What values did you decide to define? When might they be useful?

Support for English Language Learners

MLR 8 - Discussion Supports: As students discuss, rephrase responses as questions and encourage precision in the words being used to reinforce the meanings behind some of the programming-specific language, such as "define" and "value".

Using Defined Values

Overview

Now that we know *how* to define values, we've got two more things to consider:

- When it would be *useful* to define them?
- How do we *use* them once we've defined them?

Launch

Complete Defining Values - Chinese Flag.

Note: The above worksheet will direct students to open the Chinese Flag Starter File once they complete the first half of the questions.

Investigate

- Open a new file in WeScheme and name it `sunny`, then turn to Why Define Values?
- The first row of the table has been completed for you. What is happening in that first row?
  - *The original Circle of Evaluation has been simplified by using a defined value* `sunny`.
- What code is being replaced by `sunny`?
  - `(radial-star 30 20 50 "solid" "yellow")`
- Write the code on the line provided in question 2. Then type it into the Interactions Area and click "Run".
- Define the value `sunny` in the Definitions Area.
- Complete the page and test your code in the editor.
- When you're done, turn to Writing Code using Defined Values and follow the directions to work with a new function you'll define called `PRIZE-STAR`.

Synthesize

- Why is defining values useful?
  - *Defining values allows the programmer to reuse code and make changes easily. It allows us to more easily use elements inside other functions, and it saves time!*

Additional Exercises

- Matching Code to Images using overlay & put-image (Desmos)