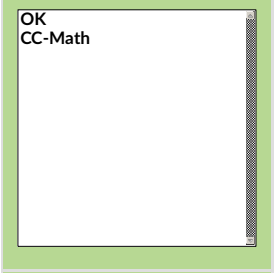
Piecewise Functions

(Also available for Pyret)

Students will learn how one function can have different behaviors based on the input.

Prerequisites	Simple Inequalities
Relevant Standards 	*Select one or more standards from the menu on the left (⌘-click on Mac, Ctrl-click elsewhere).*
Lesson Goals	Students will be able to: • Explain what a piecewise function is. • Give examples of inputs and outputs of a given ***piecewise function***.
Student-Facing Lesson Goals	• I can describe how piecewise functions work.
Materials	• Lesson slides template (Google Slides) • Alice's Restaurant starter file (WeScheme) • Restaurant - Intro (Original (Page 48), Solution (Page 48)) • Restaurant - Explore (Original (Page 49), Solution (Page 49))
Preparation	• Make sure all materials have been gathered • Decide how students will be grouped in pairs
Supplemental Resources	• Domain & Range of Piecewise Functions (Desmos Activity)
Key Points for the Facilitator	• The Design Recipe looks a bit different for piecewise, or ***conditional***, ***function***s. Check that students are taking time to write ***example***s and circle what is changing.

For a textbook-like version of materials similar to these, you may wish to see the prior unit-based version

Glossary

conditional :: a code expression made of questions and answers

example :: shows the use of a function on specific inputs and the computation the function should perform on those inputs

function :: a mathematical object that consumes inputs and produces an output

piecewise function :: a function that computes different expressions based on its input

String :: a data type for any sequence of characters between quotation marks (examples: "hello", "42", "this is a string!")

Warmup

Students should have their computer, workbook, contracts page, and pencil and be logged in to WeScheme and have their workbooks with a pen or pencil.

Not Every Function is Smooth

15 minutes

Overview

Students are challenged via counterexamples to see just how far the Vertical Line Test will go: into behaviors that *feel* like functions but don't act like a straight line or smooth curve!

Launch

Have students stand up and put some space between themselves, as if on a number line (each student essentially represents an "x-coordinate"). Give directions to distinct groups of students. For example:

- If you have brown eyes, wave your arms in the air.
- If you have blue eyes, walk in place.
- If you have green or hazel eyes, flap your arms like a chicken.
- If you like sushi, go back to your seat.

Every student should have an activity to perform. Ask a student walking in place why they aren't waving their arms in the air, or how they knew what to do. Their behavior is essentially the y-coordinate, though for a more direct connection you can specify that different groups sit, kneel, or stand so that their literal *height* represents the y-axis.

The Vertical Line Test says that to be a function, every input has to be matched with exactly one output.

Ask students: Is this activity representing a function? What is the input? What is the output? *Since each student ("input") has only one action ("output"), it *is still a function** .

Up until now, almost all the functions students have seen are continuous and smooth. Make a big deal about this, so they recognize how big of a shift this is!

Explain that students have just acted out what is called a *piecewise function* . Even though their behavior didn't follow a smooth pattern (or even a continuous one!), it clearly followed a set of rules and each input had exactly one output. Math has functions like this as well!

Example: Suppose I sell boxes of candy for $2 each. We could imagine that a graph of sales-v-revenue looks like a straight line with a slope of 2: a linear function! But then I want to offer a "bulk discount", where the price drops to $1 for the 21st box of candy and every box after that. Suddenly our line has a kink in it at 20 boxes, where the slope suddenly changes from 2 to 1.

Can students come up with their own examples?

Investigate

Students open the Alice's Restaurant starter file, and turn to Welcome to Alice's Restaurant! (Page 48).

Students investigate the file using their workbook page as a guide.

Notice and Wonder

Have students take time to think and discuss what they Notice and Wonder about this file, which contains some new elements they haven't seen before!

Synthesize

- **What are some familiar things you notice in this file?**

  Answers vary: `define`, `string=?`, *a contract and purpose statement, etc.*

- **What new things did you notice in this file?**

  Answers vary: the `cond` *keyword, the square brackets,* `else`, *the general look of the* `order` *function, etc.*

- **What function is being defined here? What is its contract?**

  `order`, takes in a String and produces a Number.

- **How do you think this function works?**

  Answers vary - let students drive discussion!

The `order` function is *also* piecewise function! Each input has a single output, but the behavior isn't smooth (there's no relationship between one item's price and another!) or continuous (there are plenty of items not on the menu!).

Partial Functions

For Algebra 2 or pre-calculus teachers, this is a useful time to address *partial functions*. The students who liked sushi had *no rule at all*, meaning that the function was *undefined* at those points. The candy-sales analogy can be extended to say that no one can order more than 100 boxes at a time, making the function undefined for values of x greater than 99.

Defining Piecewise Functions

30 minutes

Overview

Having acted out a piecewise function and examined the code for one on their computers, students take the first step towards writing one, by modifying a function that's already been written for them.

Launch

Students turn to Alice's Restaurant - Explore (Page 49) and complete the exercises with their partner. Students should have added as least one extra option to the menu before moving on.

- **Why do you get an error when you try to use the `sales-tax` function for an item not on the menu?**

  Let students discuss - move towards the realization that the contract for `order` *is* `order : String -> Number`, *and the "catch-all" branch at the bottom returns a* ***String*** *instead of a Number.*

- **What should we do about this?**

  Since we want the program to stop if we give it an invalid input, we should just delete the last branch altogether. Think about other functions that don't work when we give them an invalid input, like dividing by zero!

Investigate

So how do we actually *write* a piecewise function? And more importantly, how does the Design Recipe help us get there? The Contract and Purpose Statements don't change: we still write down the name, Domain and Range of our function, and we still write down all the information we need in our Purpose Statement (of course, now we might need to write a lot more, since there's more information!).

The examples are also pretty similar: we write the name of the function, followed by some example inputs, and then we write what the function produces with those inputs.

How many examples are needed to fully test this function?

More than two! In fact, we need an example for at least every possible item on the menu!

```
(EXAMPLE (order "hamburger")   6.00)
(EXAMPLE (order "onion rings") 3.50)
(EXAMPLE (order "fried tofu")  5.25)
(EXAMPLE (order "pie")         2.25)
```

Now we circle and label everything that is *change* -able, just as we always have. So what changes?

- The input changes (the String, representing the food being ordered)
- The price changes (the Number, representing the price of the food)

> **Pedagogy Note**
>
> Up until now, there's been a pattern that students may not have noticed: the number of things in the Domain of a function is *always* equal to the number of labels in the example step, which is *always* equal to the number of variables in the definition. Make sure you explicitly draw students' attention to this here, and point out that this pattern **no longer holds** when it comes to piecewise functions.

If there are more unique labels in the examples than there are things in the Domain, we're probably looking at a piecewise function.

We have two things changing (the item and the price), but only one thing is in our Domain. That's how we know this function is piecewise function!
We start writing the definition as we normally would, using the function name and the input label from the examples step (`define (order item) ...)` . But since we know it's a piecewise function, now we add `(cond ...)` to the body of the function.
Then, for each different behavior we wrote in our examples, we add a condition to the body of our `cond` expression. Each condition has a test and a result, and we copy the results from our examples just as we always do.

```
(define (order item)
  (cond
    [      ...                    6.00]
    [      ...                    3.50]
    [      ...                    5.25]
    [      ...                    2.25]))
```

Finally, we fill in the tests with an expression that tells us *when* the function should behave that way. When should `order` return `6.00` ? *when the menu item is "hamburger"!* :

```
(define (order item)
  (cond
    [ (string=? item "hamburger")    6.00]
    [          ...                   3.50]
    [          ...                   5.25]
    [          ...                   2.25]))
```

Additional Exercises:

- Option 1: Students create another function in the code that displays an image of the food instead of the price. This integrates earlier-learned skills in creating images and defining values.
- Option 2: Students create a *visual representation* of how the computer moves through a conditional function.