# Piecewise Functions and Conditionals

(Also available in Pyret)

Students learn how to define a function so that it behaves differently depending on the input.

| | |
|---|---|
| **Lesson Goals** | Students will be able to:<br>• Explain what a piecewise function - or *conditional* - is.<br>• Give examples of inputs and outputs of a given ***piecewise function***. |
| **Student-Facing Lesson Goals** | • Let's learn how piecewise functions work in math.<br>• Let's learn how conditionals work in programming. |
| **Prerequisites** | • Simple Data Types<br>• Contracts<br>• Simple Inequalities |
| **Materials** | • PDF of all Handouts and Page<br>• Red Shape Starter File<br>• Lesson Slides<br>• Printable Lesson Plan (a PDF of this web page)<br>• Lesson Slides |
| **Key Points for the Facilitator** | • The Design Recipe looks a bit different for piecewise, or ***conditional***, ***functions***. Check that students are taking time to write examples and circle what is changing. |

***Glossary***

**conditional ::** a code expression made of questions and answers
**contract ::** a statement of the name, domain, and range of a function
**design recipe ::** a sequence of steps that helps people document, test, and write functions
**domain ::** the type or set of inputs a function expects, i.e., the independent variable(s) that govern the output of the function
**function ::** a relation from a set of inputs to a set of possible outputs, where each input is related to exactly one output

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

**purpose statement ::** a concise, detailed description of what a function does with its inputs

**range ::** the type or set of outputs that a function produces, i.e., the dependent variable(s)

# Intro to Piecewise Functions *15 minutes*

## *Overview*

Students are introduced to piecewise functions through a kinesthetic activity, and then brainstorm real world applications of piecewise functions.

Decide whether this activity would work better for your class if students stand up and spread out around the classroom or stay in their seats.

## *Launch*

How would you explain a *function* to someone else? What are some rules that all functions follow?

A function has exactly one output for each input.

Explain to students that today, we are going to act out a special kind of function. Give directions to distinct groups of students so that every student ends up with an activity to perform.

- If your birthday is in the summer, make an O with your arms.
- If your birthday is in the fall, make an X with your arms or body.
- If your birthday is in the winter, put your hand on your head.
- If your birthday is in the spring, flap your arms like a chicken.

Debrief how students decided what to do. You might do this while students are still in motion or it might work better with your class to stop the activity before discussing it. Example questions:

- Ask a student with their hands on their head why they aren't flapping their arms like a chicken.
- Ask a student making an O with their arms how they knew what to do.

- What is the input of the function we just acted out?
  - *Student.*
- What is the output of the function we just acted out?
  - *Action.*

- How do we know that you just acted out a function?
  - *Since each student ("input") has only one action ("output"), it is still a function.*

Up until now, all of the functions students have seen can be described by a single rule. In this activity their behavior followed a *set of rules* for which each input still had exactly one output. 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*** in math, and a ***conditional*** in programming. The world is full of piecewise functions and conditionals!

Let's try acting out another set of rules.

- Everyone wearing sneakers put your hands on your head.
- Everyone wearing a T-shirt make a T with your arms.
- Everyone wearing pants put your hands on your hips.
- Everyone wearing a T-shirt make an O with your arms.

Some students should have nothing to do. Others should look confused as to what to do. They may be wearing sneakers and pants and not be able to put their hands on their head and on their hips at the same time. They may be wearing a T-shirt and not be able to make a T and an O with their arms simultaneously.

Observe that this set of rules doesn't seem to have worked quite as well as the last. Debrief how students decided what to do. Example Questions:

- Ask a student wearing a T-shirt with their arms making an O… "I see that you're wearing a T-shirt - why aren't you making a T with your arms?"
- Ask a student wearing pants and sneakers… "Why don't you have your hands on your head and your hips?""
- Ask a student who isn't doing anything… "Why aren't you doing anything?"

---

A function has exactly one output for each input.

---

- Why doesn't this set of directions (above) represent a function?
  - *People wearing T-shirts - and wearing sneakers and pants - were told to do two conflicting things.*

## *Investigate*

Have students work with their partner to think of examples of piecewise functions. Here are a few examples to get you started:

- Places like movie theaters and museums often have different ticket prices for students, children, and senior citizens. That means the total price can't be determined simply by asking how many tickets there are — the price-per-person is conditioned on what kind of ticket is being purchased!
- The US Postal Service charges a different rate for differently-sized letters and packages. That means the total price can't be determined just by asking how many things are being mailed — the price is conditioned on what kind of things are shipped!
- Many phone plans include a certain price-per-gigabyte for data, but only up to a maximum cutoff amout. After that, the price-per-gigabyte gets a lot higher. That means we can't calculate the cost simply by knowing how many gigabytes there are — the cost is conditioned on what the cutoff is!

## *Synthesize*

Share your findings as a class. You may also want to discuss whether square root and absolute values are piecewise functions.

**Partial Functions**

Piecewise functions apply different rules over different "pieces" of their domains. But what happens if there's an "empty piece", for which there is no rule?

For Algebra 2 or pre-calculus teachers, this is a useful time to address *partial functions*. These are functions which are undefined over parts of their domain (like division, which is undefined when the denominator is zero). These definitions are independent from one another: a function can be piecewise *and* partial, just piecewise, or just partial. But partiality comes up much more frequently when defining piecewise functions, because students need to think through all the possible inputs.

In the USPS example, the cost to mail tiny cards is *undefined* because the postal service doesn't ship packages that are too small.

# Conditionals in Programming *20 minutes*

## *Overview*

Having acted out a piecewise function, students take the first step towards writing one, by exploring one or two programs that make use of piecewise functions, developing their own understanding, and modifying the programs.

## *Launch*

So far, all of the functions we've written had a *single rule*. The rule for `gt` was to take a number and make a solid, green triangle of that size. The rule for `bc` was to take a number and make a solid, blue circle of that size.

What if we want to write functions that apply different rules, based on certain conditions?

## *Investigate*

- Open the Red Shape Starter File.
- Complete Red Shape - Explore.
- *Optional:* Not all piecewise functions are one-to-one! If you're ready to think about *Onto Functions*, try Decide & Defend - Piecewise Onto Functions.

If you have more time to devote to piecewise functions, we have additional materials in Additional Resources.

## *Synthesize*

- What happened when you gave `red-shape` a shape that wasn't defined in the program?
  - *The program told us that the shape was unknown. Think about other functions that don't work when we give them an invalid input, like dividing by zero!*
- What is the syntax for writing piecewise functions?
  - *WeScheme allows us to write piecewise functions as follows:*
    1. *the keyword* `cond`*, followed by a list of conditions*
    2. *each condition is a Boolean expression, followed by a rule for what the function should do if the condition is* `true`.
    3. *ending with an* **`else`** *statement, being our fallback in case every other condition is* `false`.

# Extending the Design Recipe *20 minutes*

## *Overview*

Students think through how much of the Red Shape program we could have written using the Design Recipe.

## *Launch*

Let's see how the ***Design Recipe*** could help us to write a piecewise function.

## *Investigate*

- Turn to Word Problem: red-shape.
- How do the ***Contract*** and ***Purpose Statement*** compare to other Contracts we've seen?
  - *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 have more important information to write - like our condition(s)!).*

> **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 was *always* equal to the number of labels in the example step, which was *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. When it doesn't hold, that's how we *know* we need a piecewise function!

- How are the examples similar to other examples we've seen?
  - *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 are these examples different from other examples we've seen?
  - *Instead of every example being the same, each one is different.*

Circle and label everything that is *change*-able.

- What changes? What did you label?
  - *In this case, there are more things to circle-and-label in the examples than there are things in our Domain.*

---

If there are more unique labels in the examples than there are things in the Domain, we're probably looking at a piecewise function. And if the examples cannot be explained by a single pattern or rule, it's definitely a piecewise function!

---

Think back to our examples of piecewise functions (ticket sales, postage, cell-phone data plans, etc)… knowing the input isn't enough - we also need to know the conditions, and all the possible patterns!

Once we know that we're dealing with multiple patterns, we're ready to define them as a piecewise function!

**In this example, we have four patterns:**

- sometimes we produce `(circle 20 "solid" "red")`
- sometimes we produce `(triangle 20 "solid" "red")`
- sometimes we produce `(rectangle 20 20 "solid" "red")`
- sometimes we produce `(star 20 "solid" "red")`
- sometimes we produce `(text "Unknown shape name!" 20 "red")`

**To define a piecewise function, each condition has both a result ("what should we do") and a question ("when should we do it?").**

- When should we make circles?
  - *When* `shape == "circle"`
- When should we make triangles?
  - *When* `shape == "triangle"`
- When should we make rectangles?
  - *When* `shape == "rectangle"`
- When should we make stars?
  - *When* `shape == "star"`
- When should we draw the "Unknown shape name" text?
  - *When* `shape` *is….anything* **`else`**

**Adding the questions to our pattern gives us:**

- When `shape == "circle"` ...we produce `(circle 20 "solid" "red")`
- When `shape == "triangle"` ...we produce `(triangle 20 "solid" "red")`
- When `shape == "rectangle"` ...we produce `(rectangle 20 20 "solid" "red")`
- When `shape == "star"` ...we produce `(star 20 "solid" "red")`
- When `shape` is anything **`else`** ...we produce `(text "Unknown shape name!" 20 "red")`

**This practically gives away the body of our function definition!**

```
(define (red-shape shape)
  (cond
    [(string=? shape "triangle") (triangle 20 "solid" "red")]
    [(string=? shape "rectangle") (rectangle 20 20 "solid" "red")]
    [(string=? shape "star") (star 20 "solid" "red")]
    [else (text "Unknown shape name!" 20 "red")]))
```

If you have more time for working with Piecewise Functions, you may want to have students create a *visual representation* of how the computer moves through a conditional function. Students will enjoy getting more practice with piecewise functions while using emojis!

- For additional practice, check out the Mood Generator Starter File, which uses emojis.
- Although emojis look like images, they are actually characters in a string!
- *Optional:* On Word Problem: Mood Generator, try defining a function that translates moods into emojis.

## *Synthesize*

- How many examples are needed to fully test a piecewise function with four "pieces"?
  - *More than two! In fact, we need an example for every option - every "piece"! (And in some cases there is a "default"* **`else`** *or* `otherwise` *option, which we should write an example to test, too!)*
- What changes in a piecewise function?
  - *The input, and also the* ***rule the function applies to the input***

# Additional Resources:

We have one more program for your students to explore and scaffolded pages to support them through the process!

- Alice's Restaurant Starter File
- Alice's Restaurant - Explore
- Word Problem: Alice's Restaurant