PLTW COMPUTER SCIENCE

Activity 1.2.3

Procedural Abstraction: Price Per Slice

goals

- Create procedural abstractions in a program
- Learn how to use modulo to make choices in a program
- Identify what details are hidden or abstracted in block-based programming
- Develop an app as part of a pair programming collaboration

description of app

Create an app that allows the user to compare the price of pizza per slice when feeding a large group of people.

Essential Questions

1. How do my algorithms integrate arithmetic and logical concepts?
2. How is abstraction in the programming language I am using managing complexity in my program?
3. What details have been hidden or removed by an abstraction?

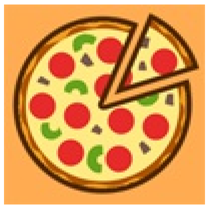
essential Concepts

- Problem Solving
- Procedural Abstraction
- Programming Language Abstraction: Methods and Functions
- Modulo

- User-centered Design, Iterative Design, and Debugging

Resources

Activity 1.2.3 Student Files



Design Overview: Price Per Slice

User Story

A school club has decided to serve pizza at the end of all of their meetings this year. After the first two meetings, they quickly realize they are having a hard time estimating how much pizza to order. Develop an app that they can use at the start of every club meeting to count how many slices of pizza people want, calculate the amount to charge per slice, and determine which pizza size is the best deal that day.

App Overview
Create a Price Per Slice app that allows a user to calculate the best deal on pizza based on the size and cost of each pizza. The app will allow a user who needs to order pizza for a large number of people to enter the total number of slices they want and receive back how many pizzas they need to order. The app will also return how much the user should charge each person per slice to cover the cost of the pizza.

This app will require you to use procedural abstraction to make calculations and return values for comparison and evaluation by a user.

- The first feature of the Price Per Slice app will ask a user to enter the size of the pizza (diameter in inches) and cost of a pizza. The app will need to return the total area of the pizza with how much each square inch of pizza costs (cost per square inch).
- The second feature of the app will add the ability for the user to input the total number of slices needed. The user will touch a button to increment the number of slices needed.
  Example: If there are 20 students and the user estimates one slice per student, they would tap the input button until they reach the number 20. With each increment, the app will calculate and update the total number of pizzas that a user would need for that number of slices.
- Once you have these procedures working, you and your partner will expand the app to call these procedures in multiple places. This way, a user can

compare different combinations of size and cost and decide what is the best value to feed everyone. By using procedural abstraction, it will be less time consuming to add these features.

Initial Backlog Breakdown

- ☐ Create a way for users to enter the cost of a whole pizza and the size of a pizza (diameter in inches).
- ☐ Create a math procedure that calculates the area of a pizza.
- ☐ Create a math procedure that calculates the cost per square inch for that pizza.
- ☐ Create a procedure that increments up to the number of slices needed.

  ☐ Create a procedure to determine how many pizzas to order and the price of each slice.

- ☐ Expand upon the app to so that users can compare multiple options all at the same time.
-

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

Create a User Interface

When you develop an interface design, be sure to consider usability—how easy it is for the user to interact with your app without help. Add in layout components and adjust component properties to make the design easy to navigate and use. Then, you and your partner will expand this app, so planning your program and user interface with that in mind will save you in the long run from having to do major redesign.

1. Navigate to MIT App Inventor, and using the naming conventions as directed by your teacher, set up a new app project.
2. To create a user interface, add the list of components below. You may choose to add additional components to make your design unique and increase the usability of the app.
    - To calculate the total pizza size based on the diameter and the cost per unit area ($/in^2), you will need the following UI components:
        - *Textbox* - to enter the price of a pizza, *Price1Input* (The number is included in the name, so you may expand the app to have multiple price inputs.)
        - *Textbox* - to enter the size of the pizza, *Size1Input*
        - *Label* - for the results of the calculations for area of pizza and price per square inch. Rename it *Result1Output.*

        - *Button* to *Calculate*.

    Important: Note that each criterion has a “1” in the name. Later, you will use the same procedures to make calculations for a second pizza, “2”, so the user can compare.

    - To increment up to the total number of slices needed and return the price per slice, you will need:
        - *Button* or *Canvas* - to tap to increment up to the desired *SliceCount*. You may choose to upload a picture, such as the provided Tap picture, so the user knows where to tap. Adjust the size properties of the canvas to make it easy to tap, but not too large for the screen.
        - *Button* - to click to *ResetPizzaCount*
        - *Label* - for how many pizza slices the user has tapped
        - *Label* - for the results of how many pizzas to order
        - *Label* - for the price per slice. Rename it “SlicePrice”*.*

    Important: You may choose to return some information within a single label instead of creating three.

Abstraction

Abstracted Programming Languages

Up to now, you have learned that block-based programming is an example of an abstracted programming environment. Abstraction allows you to focus on different levels of detail. Later in the course, you will program in a text-based environment representing a lower level of abstraction. This lower level of abstraction will give you more flexibility as a programmer, because you will not be dependent on what someone else decided should be pre-built into the coding environment. The trade-off is that you will need to learn some of the rules of that text-based programming language.

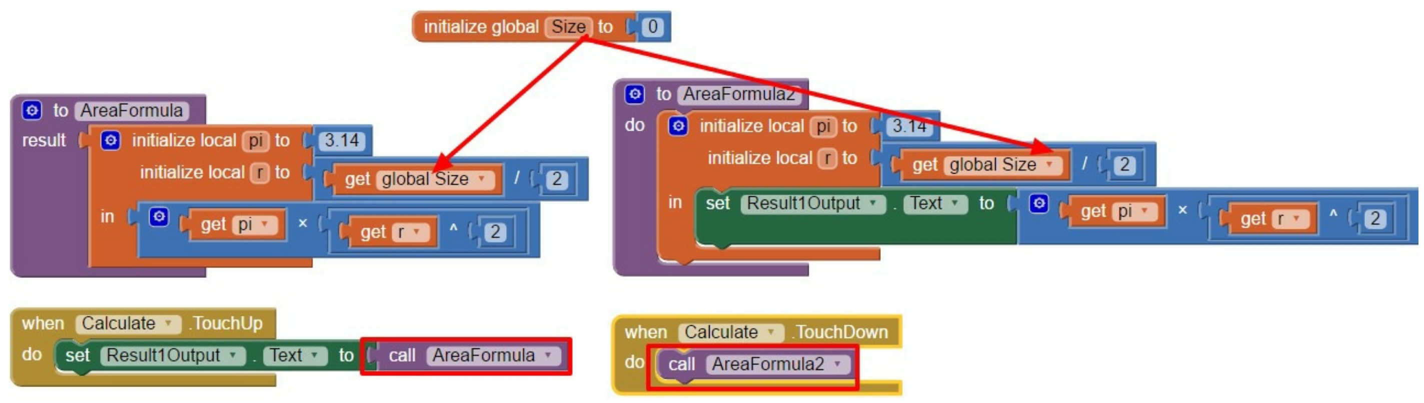
The term abstraction can be used to describe other concepts that hide details from us to make programs more manageable. At times, it can be confusing that abstraction means slightly different things based on the type of abstraction discussed.

In this activity, you focus on a slightly different idea related to abstraction, procedural abstraction. Procedural abstraction is creating a set of instructions under a specific name to call on that specific sequence of steps multiple times in the same program. In a sense, you are creating your own block of code made of smaller blocks that you can use over and over without needing to change it. Again, once the functionality is created, what is inside the procedure is no longer that important to us. It does what you want it to, and you can reuse it anytime you need that function. In this way, programmers spend less time expanding and debugging programs.

When you create procedures or operations that are used over and over, you are applying the idea of abstraction to manage complexity in a program. You can use the same coding construct over and over without concern about what’s inside it, just what it does.

You will use procedural abstraction in the Price Per Slice app when you allow a user to compare many criteria and results of calculations. The procedure will not change as you expand the app, but you will change how many times you call that specific procedure.

Look at the code samples below and decompose each sample. They perform similar tasks, use the same global variable, and accomplish the same result of displaying the area of a circle to the user.



```
initialize global Size to 0

to AreaFormula
result  initialize local pi to 3.14
        initialize local r to  get global Size / 2
        in  get pi × get r ^ 2

to AreaFormula2
do  initialize local pi to 3.14
    initialize local r to  get global Size / 2
    in  set Result1Output . Text to  get pi × get r ^ 2

when Calculate .TouchUp
do  set Result1Output . Text to  call AreaFormula

when Calculate .TouchDown
do  call AreaFormula2
```

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

In coding, multiple ways exist to accomplish the same goal. As you continue in the course, think about what is the most efficient and reusable way to code and build that into your program's design to save yourself time in the end.

Variables and Procedures

Create variables to use in the procedures. This way, each procedure can be reused by changing the variable at a local scope. The app will need variables for:

- Pizza size
- Pizza price
- The area formula, such as r = radius, D = diameter, π = 3.14
- Price per square inch (within your calculations)

3. Complete the following assessment to help you determine what types of variables you want to use to store the four pieces of information listed above.

   Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

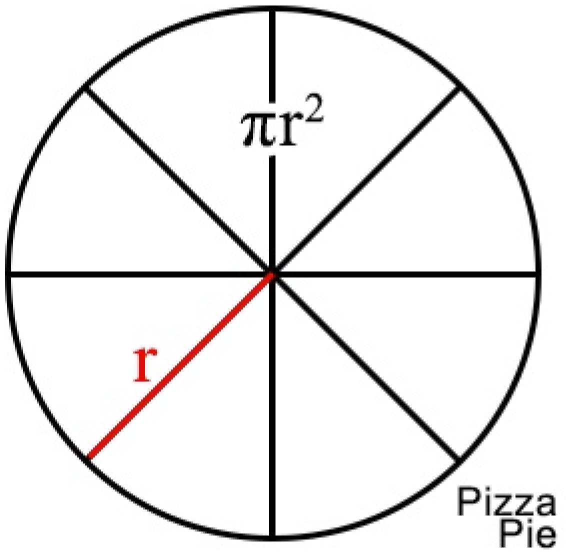
4. Set up and initialize the two global variables (*Size* and *Price*) to 0.

Procedures, Methods, and Functions

One of the main ways you have seen abstraction in previous activities is through built-in procedures. The terms procedure, function, and method are often used interchangeably depending on the language you are coding in. The following points might help you recognize the different uses of each term.

- A function returns a value, but a procedure does not have to.
- A function accepts input values and returns one or more values.
- A procedure is a small component of a computer program that may contain one or several specific algorithms to accomplish a certain task.
- Methods are similar to functions in text-based programming languages like *Python*®.

You are going to set up a procedure that will use local variables to calculate the area of the pizza using the formula to find the area of a circle, πr^2. The procedure will accept the integers stored in the variables π and r, pass them through the local variable formula stored in the procedure, and return the resulting area.



AreaFormula Procedure with Local Variables

Most pizza companies provide the diameter (not the radius) of a pizza, which is the size from one edge of the pizza to the other. Set up the math in the local variable procedure so that the user can enter the diameter and get the correct output.

Note: In the formula πr^2, π (pi) can be represented as 3.14, and r is the radius, the distance from the edge of a circle to the center of the circle.

5. Set up a procedure that will use local variables to calculate the area of the pizza.
   - Drag out a procedure that will return a result to where it is called from and name it "AreaFormula".
   - Name the local variable "pi" and initialize it to "3.14".
   - Use the blue *mutator* button to add a second local variable that will handle the *r* value of the formula.
   - Use arithmetic operators to initialize the r variable to the global *Size* variable divided by 2, because most pizza places provide the diameter of the pizza, but the formula for area requires the radius measurement.
   - In the local variable, use arithmetic operators to set up the formula "pi × r2".

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

The formula calculation will return a number to wherever the procedure call was made. Because the area is needed to calculate the price per area, you will call this procedure in the next procedure you make. The advantage of keeping them separate procedures is that you may call just the area procedure wherever you need it, or just the cost per area procedure.

Setting Up Variables and a Procedure for Price

Set up another procedure to get the result of calculating the cost of pizza per square inch.

6. Create a procedure named *PricePerArea*.
   - Create a local variable for area that calls the *AreaFormula* procedure and gets the resulting area from the procedure result. To call a procedure, from the *Built-in Procedures* drawer, grab the appropriate call block that will plug into the initialize local variable socket.
   - Add a local variable for cost that will get the global price of the pizza that the user entered.
   - Using arithmetic operators inside the local variable, get the local cost variable and divide it by the local area variable to find out how much the pizza costs per square inch.

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

7. Test, debug, and adjust. What does your app do?

   Important: The program has procedures to take multiple inputs and perform the same process with those inputs. However, the program does not yet include directions on when or where to access those inputs.

   The up side is that you’ve put the math into a lower level of abstraction. From now on, you can operate at a higher level of procedural abstraction by calling that procedure and not have to do the math again.

Calling a Procedure

To test the procedures, you will need to add an event handler, *Calculate.Click*, which will set the global variables to the user inputs for size and price.

8. Set up the *Calculate.Click* event handler.
   - Drag the *Calculate.Click* event handler to the *Blocks Viewer*.

  - Set *global Price* to *Price1Input.Text*.
  - Set *global Size* to *Size1Input.Text*.
  - Set *Result1Output.text* to concatenate the labels with the procedure calls for the *AreaFormula* and *AreaPerPrice* so the user can see the results of the procedural formulas.

> Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

9. Test, debug, and adjust.

   To test the algorithms in the procedures, you will store the values of size and price in variables before calling the procedures, *AreaFormula* and *PricePerArea*. Then output the procedures result in a label.

   Does the answer you get make sense?

   - Do you get a result that seems accurate for total pizza area?
     - Should the number of square inches of pizza be larger or smaller than the diameter input?
     - Does the number seem too large or too small?

   - Do you get a result that seems accurate for pizza area per price?
     - Should the price-per-square-inch be less than or greater than the total cost of the pizza?
     - Does the number seem too large or too small?

   - What happens if you do not provide any input?
     - What happens if you enter a diameter but no price?
     - What happens if a person enters “ten” instead of “10”?

Conditionals to Prevent a Broken Appearance

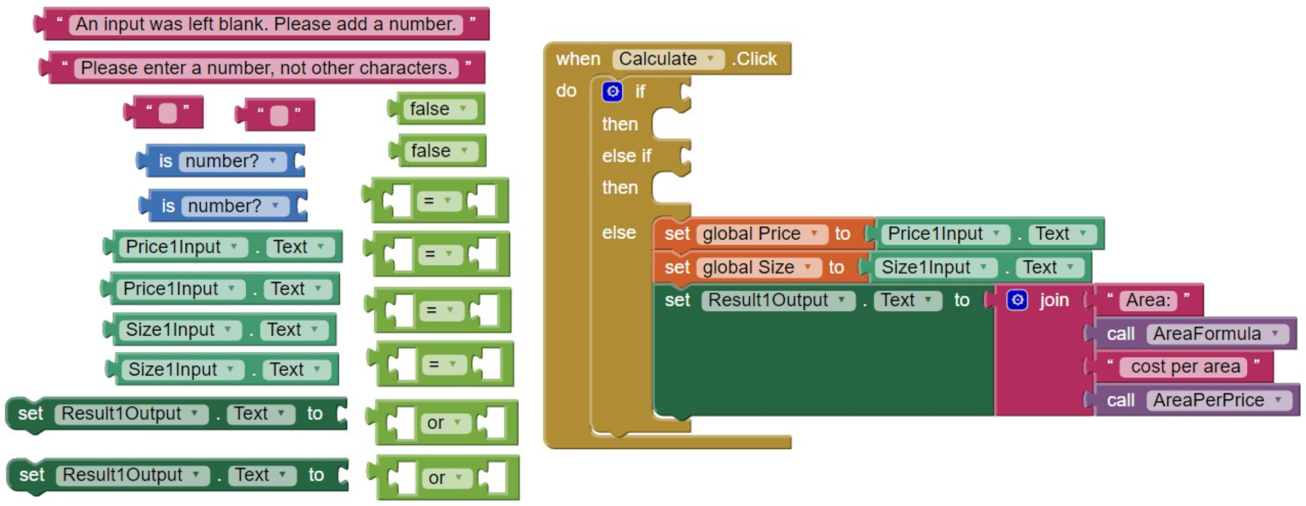
Certain inputs can make an app seem like it does not work even when it does. You may be wondering whether you programmed something incorrectly due to empty string errors.

For example, dividing by 0 (or no input) will cause the program to stop or give an error. (Maybe you heard a math teacher say something before about not dividing by 0?) If the user enters 0 or nothing, the program will try to divide by 0, unless you program the app to avoid dividing by 0.

There are multiple ways to protect against 0 inputs. Doing so is important in apps and other computer software to avoid errors that make users think the program is broken and to prevent programs from crashing. In the next steps, you will set up a conditional structure that can be used in different types of programming to avoid the divide-by-zero crash.

10. For when the user needs to change their inputs, use a chained conditional statement, Boolean expressions, and logical operators to provide input to the user through the *Result1Output*.

    The example below shows all the blocks for one way to set up the conditional. Pull out these blocks to create the conditional, or use what you know to create your own functional algorithm.



```
" An input was left blank. Please add a number. "
" Please enter a number, not other characters. "
" "    " "
false
false
is number?
is number?
=
=
=
=
Price1Input . Text
Price1Input . Text
Size1Input . Text
Size1Input . Text
or
or
set Result1Output . Text to
set Result1Output . Text to

when Calculate .Click
do  if
    then
    else if
    then
    else  set global Price to  Price1Input . Text
          set global Size to  Size1Input . Text
          set Result1Output . Text to  join  " Area: "
                                             call AreaFormula
                                             " cost per area "
                                             call AreaPerPrice
```

11. Test your algorithm. See what outputs you get when you leave a string blank or enter other characters, such as punctuation.
12. Enter a diameter and a pizza price.
    - Does the answer you get make sense?
    - Do you get what seems accurate for total pizza area?
    - Do you get what seems accurate for pizza area per price?
    - Compare with other pairs to determine whether your answers are reasonable.

13. Test your app as follows:
    - Provide no input. What happens?
    - Provide a 0. What happens?
    - Enter a size but no price. What happens?
    - Enter a price but no size. What happens?
    - Does the app always provide appropriate feedback?

14. If your app works by providing reasonable answers and not dividing by 0, then save the app as a new version before you start the next iteration.

Creating an Incrementing Counter

The user story included being able to tap the screen to quickly get a count of the number of pizza slices to order. To store the taps and increment this count at each additional tap, you need to create a *global Count* variable.

15. Using what you learned in previous activities as a guide:

- Set up a global variable for the *Count*.
- Drag out the *Canvas1.Touched* event handler.
- Set up an incremental counter inside *Canvas1.Touched* to increment the value stored in the global count variable by 1 for each time *Canvas1* is tapped.
- Provide an output to your screen that shows the total number of slices needed, based on the number of taps the person makes.
- Test.

So far, you have set up an output that expresses the total number of slices, but now you need to be sure that each tap adds one more slice.

16. Using previous activities as a guide, set up an *OutPut* procedure that will provide feedback to the user.
    - For now, use the *SlicePrice* label to display the count and a description of what that number represents.
    - You will need to use concatenate to display all this information in a single label.
17. After you make the *OutPut* procedure, add a *call OutPut* procedure to the *Canvas1.Touched* event handler.
18. Test, debug, and adjust.
    - If you tap the picture, does it add 1 each time?
    - What if the user needs to start the counter over again?

Adding a Reset Event

You may have noticed that there is no way to reset the counter if the user needs to start over.

19. Add the *ResetPizzaCount*.*Click* event handler to the viewer.
20. Use this event handler to reset the global variable and the *SlicePrice* label output back to 0.

How Many Slices Per Pizza?

At this point, the user can tap the button to get the total number of slices people want to eat, but how many whole pizzas need to be ordered? Typically there are 8 slices per pizza.

You need a variable to store and increment the *NumberOfPizzas* by 1 every 8 taps.

21. Set up a global *NumberOfPizzas* variable and initialize to 0.
22. To the *Canvas1*.*Touched* event handler, add some arithmetic operators so the app can automatically take the number of slices the user enters and divide it by the number of pieces per pizza. This example assumes 8 slices per round pizza.
23. Update your *OutPut* concatenation to include a get global *NumberOfPizzas*.

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

24. Test your app. As you tap for more pizza slices, does it give you a float or an integer type number? Pizza places would most likely not appreciate an order with a float type number.
25. Try adding a round block to the set *NumberOfPizzas* to round the number it gets from the division.
26. Test your app. Is the number more accurate for ordering pizzas now?

Important: Rounding does not work, because rounding will display 0 pizzas until you reach 5 slices, at which point it will say 1. To help with these situations, computer science has an arithmetic operator called modulo.

Modulo

You want the number of pizzas to be 1 as soon as the first slice is tapped, and then be increased by 1 after every 8th tap. You could program a conditional statement to add 1 to *NumOfPizzas* after every multiple of 8, but that would take a lot of time and work. This conditional might be the conditional for the first pizza, but would you want to do this each time (16 slices = 2 pizzas; 24 = 3 pizzas)?

You also have no way to know what multiple of 8 you could stop at for the app to always work. Ultimately, a user could order one more slice than you accounted for in your program.

Instead of programming for every multiple of 8, which as numbers, are infinite (you'd never be done), you are going to use a conditional statement with the **modulo** block.

Modulo does division but only returns the remainder, not the number of times the number is divided by 8.

For example, the result of 10 ÷ 8 is 1 with a remainder of 2. So 2 is the modulo that would be returned. More examples:

- 14 ÷ 8 would return a modulo of 6.
- 27 ÷ 8 would return a modulo of 3.
- What is the modulo result of 170 ÷ 8?

When do you need to order another whole pizza?

PLTW DEVELOPER'S JOURNAL

1. If each pizza has 8 slices, at how many slices would you need to order one more whole pizza?
2. What is the minimum number of slices ordered that would force you to buy 3 pizzas?
   - At 8 or 9 slices, how many pizzas would you need to order?
   - At 15, 16, or 17 slices, how many pizzas would you need to order?

- The following table divides the number of slices by 8. The table increments by 1 slice to show you the pattern developed when you look at the remainders. Complete the table to see what you get back with the different types of math blocks. Record your observations about using the different operators.

No. of Slices	Operation (divided by)		Output using Division operator (Quotient)	Output using Round operator	Modulo Answer (Remainder)
8	÷	8	1	1	0
9	÷	8	1.125	1	1
10	÷	8	1.25	1	2
11	÷	8	1.375	1	3
12	÷	8	1.5	2	4
13	÷	8			
14	÷	8			
15	÷	8			
16	÷	8			
17	÷	8			

Important

- The quotient (Division answer) is never the same twice.
- The rounded version is not helpful for knowing how many pizzas to order.
- However, the remainder (the Modulo) answer has a set pattern! After every 8th additional slice, the remainder is always the same. Once you identify patterns such as this, you can use them to program a procedure that will take advantage of these patterns.
- Ordering 1 more slice past a multiple of 8, means ordering another whole pizza!

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

You will use the idea of modulo to do the math for you and increment the number of whole pizzas each time the slice count passes a multiple of 8. (Look at the table and compare the Modulo answers for using a Division operator, a Round operator, and a Modulo operator with when you need to add another pizza.)

27. Next to the modulo answers, star the rows that represent each number of slices that indicate the user would need to buy another pizza.
28. Look at the patterns in your table.

> Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

Setting Up a Conditional with Modulo

Using modulo, you will set up a conditional statement that divides the *Global Count* by 8 and then checks whether the remainder result of the modulo equation is a specific number. The specific number should be whichever modulo answer in your table represents when you need to order another pizza (to have enough slices to meet the demand in the *NumOfSliceCount*).

29. Add the *OutPut* procedure back into the *Canvas1.touched* event handler.
30. To see the results of the math blocks, remove the testing blocks that you used with the *SlicePrice*.
31. Add an *if then* conditional to the *Canvas1.Touched* event handler.
    - The *if* part of the conditional should compare the result of a modulo equation (*Global Count* ÷ 8) with the modulo answer that represents needing to order another pizza.
    - In the *then* part of the conditional, increment by 1, the value stored in the global *NumberOfPizzas*.

> Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

32. Test, debug, and adjust.
    - ☐ All previous parts of the app still work.
    - ☐ After every 8th slice, the *NumberOfPizzas* count goes up by 1.

Price Per Slice

The last feature to add to the program will calculate the price per slice of pizza to cover the total price of the ordered pizza. To do this, take the price per pizza and multiply it by how many pizzas need to be ordered to get the total cost. Then divide the total by the number of pizza slices, so you know the price per slice of pizza to cover the overall pizza cost.

33. Make a procedure to get the *TotalPrice* the user would need to pay based on how many pizzas they need to order as stored in *NumOfPizzas*. The procedure should return the result of *NumOfPizzas* × *PizzaPrice.Text*.
34. Make a procedure to get the result of *PricePerSlice*, which is: taking the *TotalPrice* and

dividing it by the number of slices. The procedure should return the (*NumOfPizzas* × *PizzaPrice.Text*) / *global count* of pizza slices.

35. Update the *OutPut* procedure to call the new procedures in the concatenation of the message to the user. Be sure to update the strings so all the numbers make sense to a user.

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

36. Let your teacher know when you have a working app.
37. Your teacher may ask you to expand this program to be able to compare two or three pizzas at one time using the procedures you have already built.
    - What other features could you add to help make this an app that clubs at your school could use?
    - Does the total include tax and tip?
    - Would it be possible to expand or add a procedure that can add in the tax and tip amount to split across all the pizza slices to cover all expenses?

Conclusion

1. In your own words, describe what modulo is doing in your program.
2. How did you interpret and respond to the **essential questions**? Capture your thoughts for future conversations.