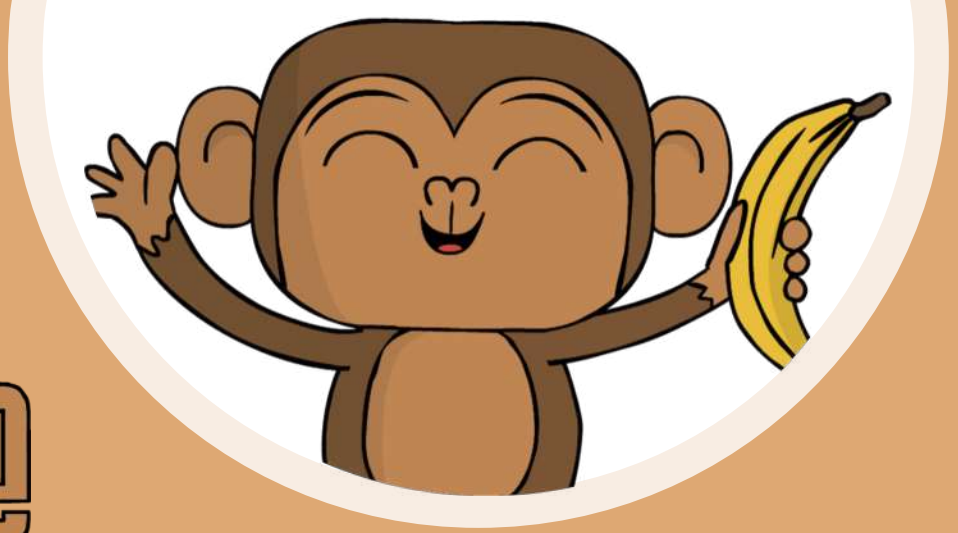
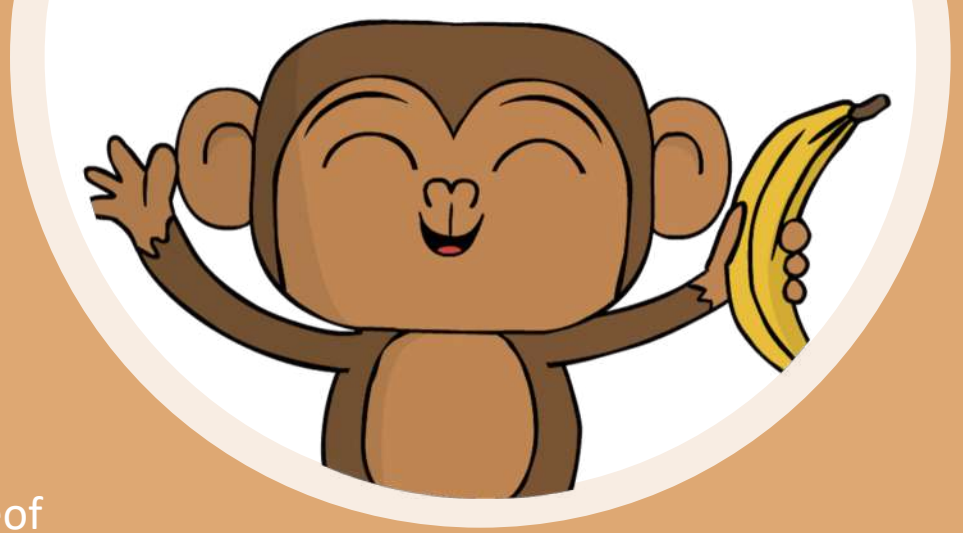
CODEMONKEY
LESSON PLANS



CODING ADVENTURE

FUNCTIONS & CONDITIONS

Lesson Plans
17-32



2345 Yale St., 1st floor
Palo Alto, CA 94306
info@codemonkey.com
www.codemonkey.com

Table of Contents

Table of Contents

Lesson 17 – Function Farm

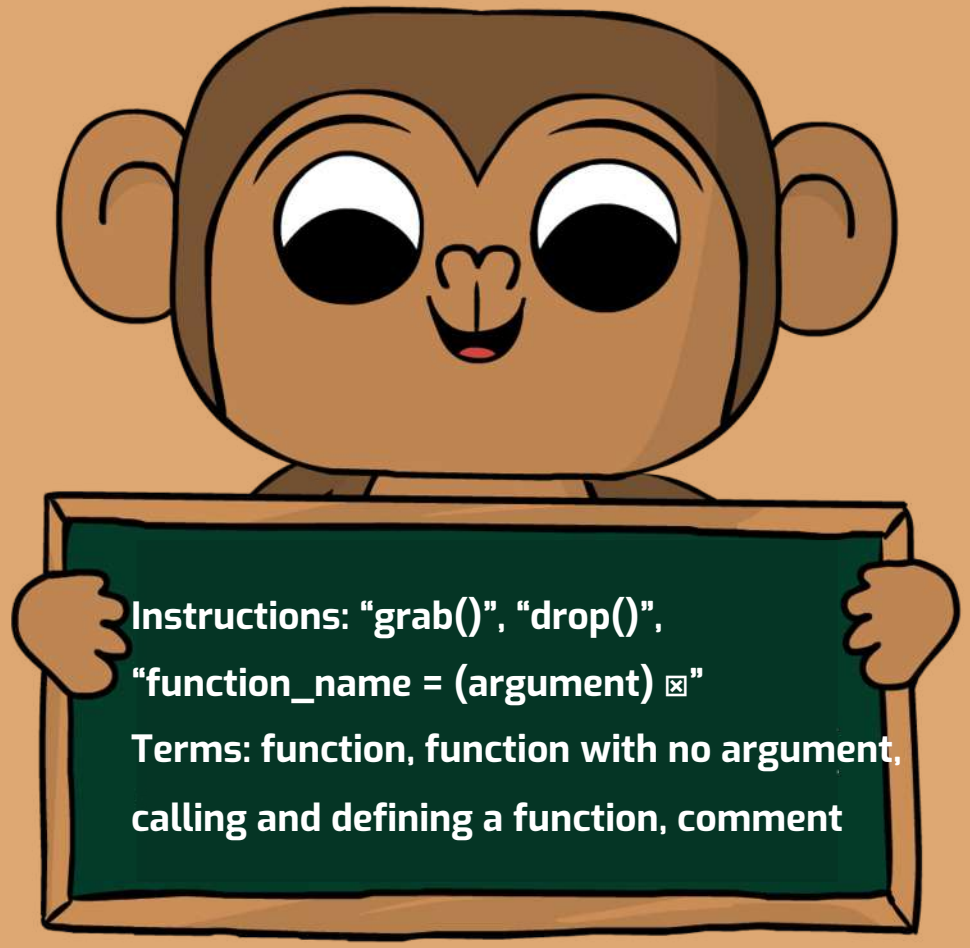
The chapter of Coding Adventure begins with challenge #76, where the gorilla destroys the bridge and Monkey can't get through. Luckily, the Rats come to help him rebuild the bridge and your students must help them! This chapter will introduce new characters and cool new functions and loops.

Objectives

In this lesson, students will:

- Revisit *function* and define *comments* as programming terms
- Define *main* code
- Work with functions
- Work with comments
- Complete challenges 76-80

Components



U.S. Standards Addressed

CSTA-K12 Computer Science Standards	
• 1B-AP-12 • 1B-AP-15	• 2-AP-10 • 2-AP-14 • 2-AP-16 • 2-AP-17

Part 1: 20 Minutes

Introduction

Explanation	**8 mins.**
In this lesson, we will get to know the character of the rat. We already met the rats, but now we can also control them and help them collect items. This lesson's key topic is learning how to read and write functions. The term function was introduced in Lesson 2 - Turn Around, but we will dive deeper in this lesson. So, why do we need functions?	

Part 1: 20 Minutes

Introduction Cont.

Explanation Cont.	**8 mins.**

Tell your students to imagine they have a block of code with the instructions for riding a bicycle (for example, sit on the seat, start to pedal, watch out for traffic, use your bell, etc.). Now, imagine you want to use this set of instructions in a few places in your code but with different bicycles in different places.

Like this:

```
go to mountain
sit on seat of a mountain bike, pedal, watch out for traffic, use bell
go to the city
sit on seat of city bike, pedal, watch out for traffic, use bell
```

Do we really need to write the entire set of instructions for riding over and over again? Wouldn't it be better to tell the computer once how to ride, and then to just use "ride" as an instruction? The code would look like this:

```
ride bike means:
    sit on seat of bike
    Pedal
    watch out for traffic
    use bell
go to mountain
ride mountain bike
go to the city
ride city bike
```

Part 1: 20 Minutes

Introduction Cont.

Explanation Cont.	**8 mins.**

Note: when we defined ride **bike**, we didn't say which bike it is. It could be any object, as long as it's a bike.
Explain: A function is a set of instructions that performs a specific task. The computer will only execute the function when we will call it, meaning use it in our code. Using a function is called 'calling the function', and creating a function is called 'defining it'. We have called (used) many functions before, but today we will learn how to define (create) functions.
Ask your students if they can think of reasons why it is good to write code with functions (like in the second example) as opposed to without functions (like in the first example).
Some correct answers:

- Make the code shorter
- Make the code more readable
- In case the instructions for riding change (e.g. if we add- use handlebars to turn), we'll only have to change the code once
- We can split the job between team members: one will write the function and one will write the code that uses it

Make sure to cover these answers whether or not your students thought of them.

Part 1: 20 Minutes

Introduction Cont.

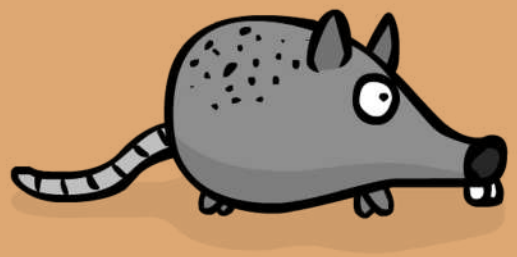
Walk-through	**8 mins.**

Functions that take an argument are written like this:

```
function_name = (argument) ->
 first_statement
 second_statement
 etc...
```

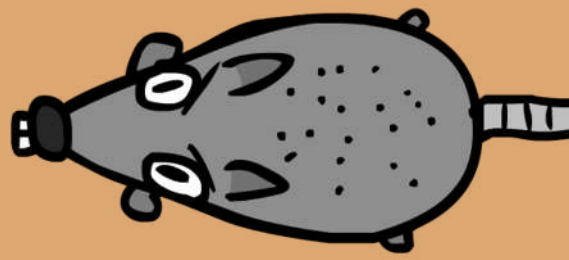
This is called the function definition.

Usually, underneath the function definition, we write the statements we want the computer to perform when we hit “run”. In these statements we may call the function that we defined.



Part 1: 20 Minutes

Introduction Cont.



Walk-through Cont.	**8 mins.**

Example:

```
goto = (a) ->
    turnTo a
    step distanceTo a
goto banana
```

Note that when we hit run, the computer will go straight to “goto banana”. That, in turn, will call the definition above.

Ask how a computer tells the difference between statements that are inside the function definition and statements that are not?
Answer: indentation, just like in loops (lesson 5). The name of the function is used for identification.
Programmers give the function a name that represents what it does, which helps make the code readable.
When we call a function, we need to join the name of the function and the name of the object we want the function to use. Look at the example on the right.

Part 1: 20 Minutes

Introduction Cont.

Walk-through Cont.	**8 mins.**

This might seem confusing at first, but the most important thing is to read this code slowly. Write this code on the board and go over how to read it with your students.

The function's name is **goto**, and the argument is **a**.

We start reading the code from line 7 where it says "**goto bridge**". This line and underneath it are called the *main* code. It is very important to understand that this is where the computer will start executing the code when we hit "run". The function's definition in the lines above is stored there and will only be executed by the computer if we will call the function in the main code.

In line 7, we use the function **goto** with the argument **bridge**. Therefore, **a=bridge**, so **a** inside of the function will be replaced with **bridge**. The computer goes back to line 3 and reads: "turnTo bridge, step distanceTo bridge".

After the computer went through all the statements in goto with bridge, it would continue with executing the main code (line 8). Next, we have another call to the same function. This time the argument is **match**, so the computer goes back to line 3, and the function will execute again **with a=match**.

```
1  # This is how you define a function:
2  goto = (a) ->
3      turnTo a
4      step distanceTo a
5
6  # Here we call the goto function:
7  goto bridge
8  goto match
9  grab()
10 # Add your code here:
11
12
13 drop()
```

Part 1: 20 Minutes

Introduction Cont.

Explanation	**4 mins.**

Another kind of function is a function without an argument. This type of function makes something happen, but it doesn't require us to *pass* any input. Calling such a function includes only a function's name and an empty parenthesis.
As we said, the rats love to collect items.
In these next few challenges, we will have to help them collect matches. In order to do so, we will use these simple functions: "grab()" and "drop()".
Ask your students, "Can you think of another statement from CodeMonkey that also includes a function?"

Answers: step, turn, and turnTo are all functions! The arguments in these functions are the number, direction, or object we add to the function.

When a function has no arguments (like grab and drop) we must use parenthesis otherwise the computer does not know we want to call a function. When a function has an argument (like step, turn, turnTo) we can call it with or without parenthesis.
step 10
step(10)
These two statements are equivalent. In Coding Adventure, to make it simple we call the functions without parenthesis.

Part 1: 20 Minutes

Introduction Cont.

Explanation Cont.	**4 mins.**

For example: step 20 (step is the function's name and 20 is the argument), turn left (left is the argument), turnTo banana (banana is the argument). In this case, the programmer who designed CodeMonkey saved us some work and already defined the actions that make these simple functions work. In programming, it is relatively common to use functions that someone else defined.

Lastly, another new concept we will meet in this lesson is the **comment**. A comment is a line in the code that is marked with the pound symbol (#) at the beginning. The computer does not treat this line as an instruction. Rather, it is used by the programmers who write and read the code in order to understand one another. Comments are mainly useful when we want to let someone else read our code and we want to write notes to help him or her. The comments will be very useful in part three of CodeMonkey. Let your students know that the comments in CodeMonkey will give them hints on what to fix in the code.

Part 2: 15 Minutes

Playtime

Log-in	**1 mins.**
Go to app.codemonkey.com. Instruct your class on how to log in to their CodeMonkey accounts. If your students use usernames and passwords to login, make sure they store their usernames and passwords where they can easily access them in the future. Optional: hand out user log-in cards. If a student forgets their password, you can reset it from the classroom dashboard, in the students tab.	
Playtime	**5 mins.**
All students should complete challenges 76-80 with at least two stars. (Students from the age of 12 and up should get three stars.) Use the classroom dashboard to keep track of students' achievements. Use this time to walk around the class and help students who are struggling. The topics in this lesson are complex so it is natural that some students will need more help. If you notice a few students who are struggling, bring them together in a small group and provide more explanation. Tip: when using grab() and drop(), don't forget the parentheses!	

Part 2: 15 Minutes

Playtime Cont.

Walk-through	**4 mins.**

Open challenge #80 and click reset to reset the code. This is the first challenge where a function definition is shown. Bring the comment to your students' attention and read the function definition with them. Run the code once, then fix it to achieve a solution to the challenge.

Ask your students "when I click run, where in the code does the computer start executing?"

The answer is at line 7 (goto bridge).

Part 2: 15 Minutes

Playtime

Playtime	**5 mins.**
Students continue working on challenges 76-80.	
Practice	
Encourage students who finish early to open skill mode on the map and complete unlocked challenges. After completing challenges 76-80, skill challenges 7-1 and 7-2 are unlocked.	

Part 3: 10 Minutes

Debriefing

Walk-through	**10 mins.**
Open challenge #80 and erase the code. Write the following solution that does not use the new function “goto”: turnTo bridge step distanceTo bridge turnTo match step distanceTo match grab() turnTo bridge step distanceTo bridge turnTo pile step distanceTo pile drop()	Your students will probably notice that this solution only gets one star. In order to get three stars, use the following code: goto = (a) -> turnTo a step distanceTo a goto bridge goto match grab() goto bridge goto pile drop()

Part 3: 10 Minutes

Debriefing Cont.

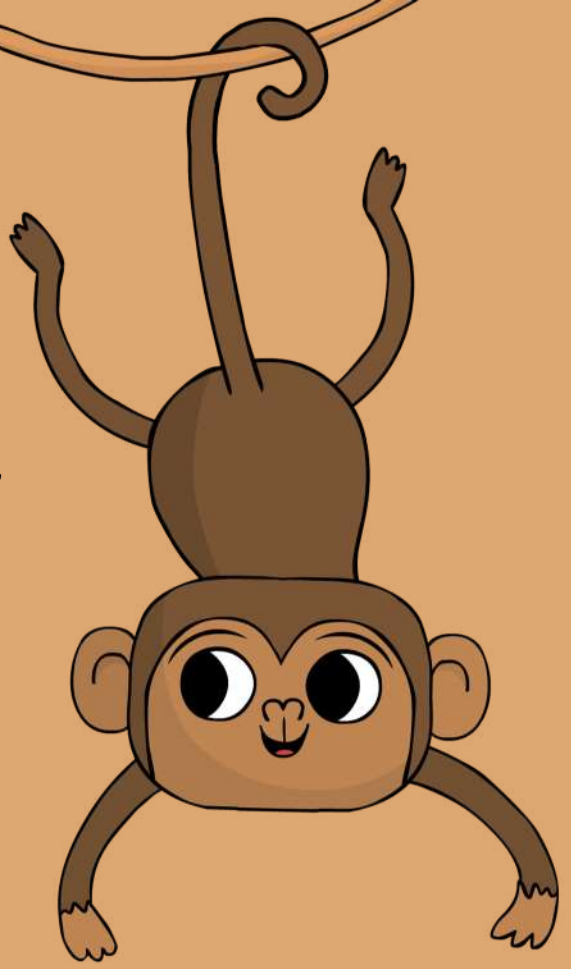
Walk-through Cont.	**10 mins.**

Write the code on the board. Go over the code with the students to make sure they understand how to read functions.
Tip: Draw arrows to show how the computer reads the code in the following order:

1. goto bridge
2. execute the inside of the goto function with a replaced by bridge
3. execute the inside of the goto function with a replaced by match
4. grab()
5. execute the inside of the goto function with a replaced by bridge
6. execute the inside of the goto function with a replaced by pile
7. drop()

Point your students' attention to the orange highlight that shows what line the computer is executing at every step.

Lesson 18 – Fun-ctions!



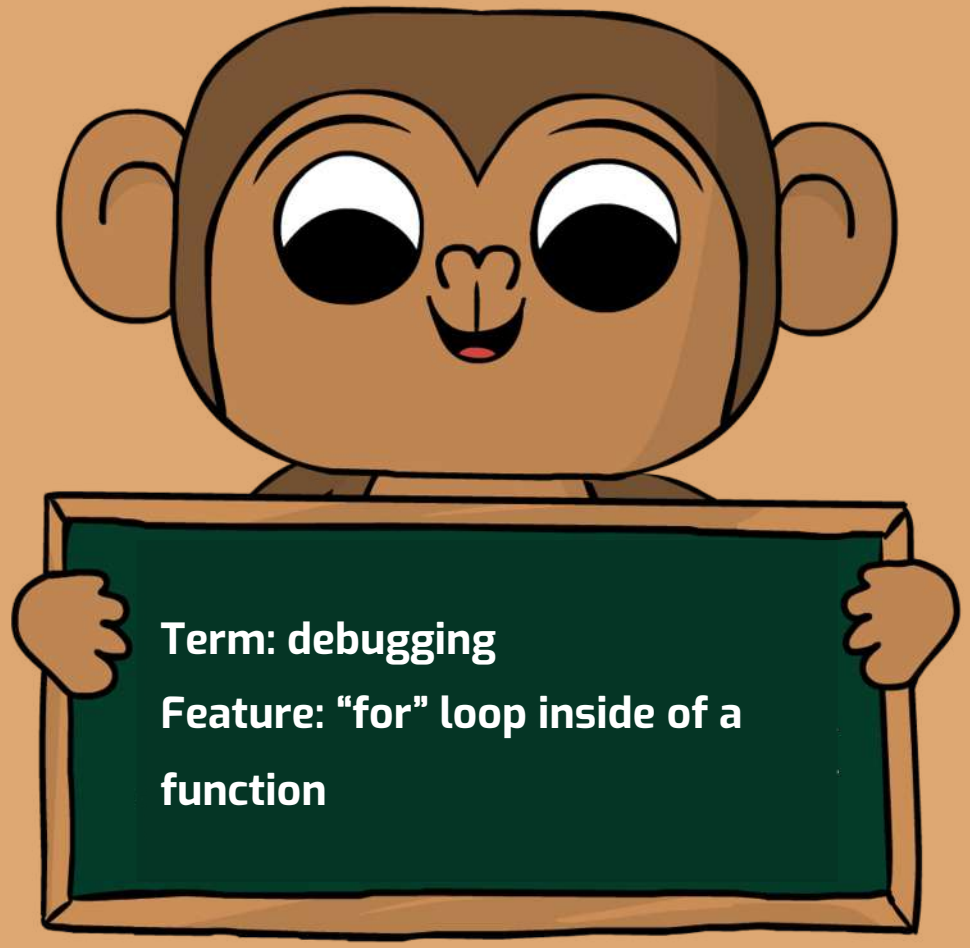
In this lesson, your students will continue practicing reading and writing functions, and will also learn about debugging.

Objectives

In this lesson, students will:

- Review functions
- Define *debugging*
- Read and write functions
- Complete challenges 81-85

Components



U.S. Standards Addressed

CSTA-K12 Computer Science Standards	
• 1B-AP-10 • 1B-AP-12 • 1B-AP-15	• 2-AP-10 • 2-AP-12 • 2-AP-14 • 2-AP-16 • 2-AP-17

Part 1: 15 Minutes

Introduction

Review	**10 mins.**

Start this lesson with a quick review of the previous lesson's topics, by asking the class a few questions:
Q: What is a function?
A: A function is a set of instructions that performs a specific task. The computer will only execute the function when we will *call* it, meaning use it in our code.
Q: What are some reasons for using functions?
A:

- Make the code shorter
- Make the code more readable
- In case the instructions change (e.g. if we add to the riding- use handlebars to turn), we'll only have to change the code once
- To split the job between team members: one will write the function and one will write the code that uses it

Part 1: 15 Minutes

Introduction Cont.

Review Cont.	**10 mins.**

Q: What are examples of functions we learned?
A: step, turn, turnTo, grab, drop, but not: for
Q: What is the meaning of *defining* a function and *calling* it?
A: Calling a function means using it in our code. Defining a function means creating it by writing the set of statements that tell the computer what to do when it is called.
Q: What are arguments? Give an example of how we call a function with and without arguments?
A: Argument is a number or an object that we *pass* to the function, that can be used by the code inside the function. Calling a function with an argument: step 10, turnTo banana, goto bridge. Without: grab(), drop(). Emphasize the parentheses.
Q: What is a function definition? Write it on the whiteboard
A: The answer has to be in the form

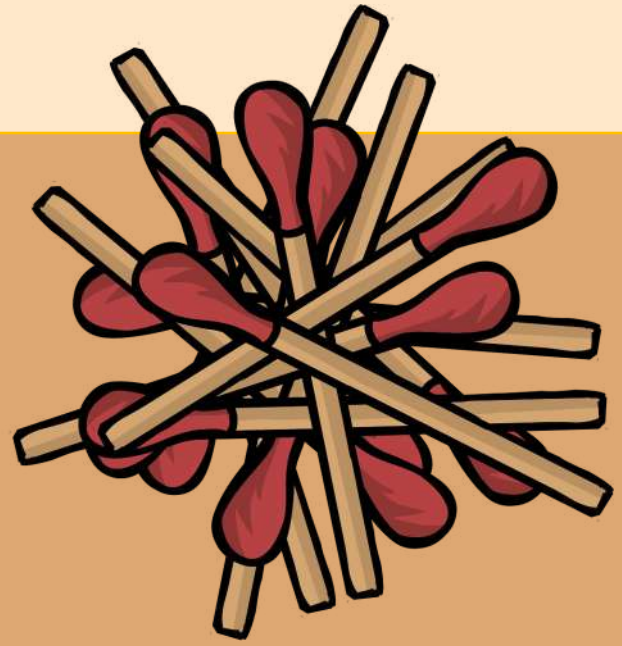
```
name = (argument) ->
 ...[indentation] statement1
 ...[indentation] statement1
etc.
```

Part 1: 15 Minutes

Introduction Cont.

Review Cont.	**10 mins.**

Q: What are comments in coding and what are they used for in Coding Adventure particularly?
A: Comments are lines inside the code that are not executed but rather are written and read by programmers. They help programmers understand each other when reading and writing code. In Coding Adventure, the code will sometimes contain comments that will provide students hints on what to do next.
Make sure to cover these answers whether or not your students recalled them.



Part 1: 15 Minutes

Introduction Cont.

Activity	**5 mins.**

Ask your students, “Have you ever heard of the term *debugging*?” Debugging is a methodical process of finding and reducing the number of bugs, or defects, in a computer program.
When we go over the code in Coding Adventure and scan it in order to see what needs to be fixed/changed, we are debugging the code.
Various sophisticated tools are available to help programmers debug their programs, but reading the code slowly and accurately is the best way to start.

Part 1: 15 Minutes

Introduction Cont.

Activity Cont.	**5 mins.**

Every programmer has his or her own favorite method to debug code. Ask for two student volunteers and instruct one of them to step out of the class until you call them back.
Open challenge #82 and ask the first student to read the code aloud so everyone can hear him. The class should focus on finding the bug.
After he is done, ask the second student to come back into the classroom and read the code aloud. The differences between the way each student reads the code demonstrate how everyone has their own habits when reading and debugging code, even though the program does the same thing.
Solve challenge #82 with your students.
Show your students that we can use “for” loops with functions. In the next lesson, we will even use “for” loops inside of the function’s definition.

Part 2: 25 Minutes

Playtime

Log-in	**1 min.**
To review the log-in instructions, please click here.	
Playtime	**24 mins.**
All students should complete challenges 81-85 with at least two stars. (Students from the age of 12 and up should get three stars.) Use the classroom dashboard to keep track of student achievements. Use this time to walk around the class and help students who are struggling. The topics in this lesson are complex so it is natural that some students will need more help. If you notice a few students who are struggling, bring them together in a small group and provide more explanation. **Tips**: When using grab() and drop(), don't forget the parentheses!	
Practice	
Encourage students who finish early to open skill mode on the map and complete unlocked challenges. After completing challenges 81-85, skill challenges 7-3 – 7-5 are unlocked.	

Part 3: 5 Minutes

Debriefing

Walk-through	**5 mins.**

Open challenge #82 and solve it. Then go over the “goto” function with your students. Draw their attention to the argument that is used (in this challenge it is ‘a’).
Open challenge #83 and solve it as well. Now go over the “goto” function in this challenge. Again, draw their attention to the argument that is used (in this challenge it is ‘c’).
Ask your students what is the difference in the “goto” function in these two challenges. You can write on the whiteboard the function’s code from the two challenges:

Challenge #82	Challenge #83

goto = (a) ->
  turnTo a
  step distanceTo a
	goto = (c) ->
	  turnTo c
	  step distanceTo c

Part 3: 5 Minutes

Debriefing Cont.

Walk-through Cont.	**5 mins.**

Both functions do the same action, they turn the monkey to an object and then step to it. The only difference is the argument that is used in each function. In one challenge it is ‘a’ and in the other challenge it is ‘c’.
When we write functions, we can use any argument that we want. Just like when we define variables.
Ask your students to change the argument to a different letter or word (make sure they change it in the body of the function as well!) and see that the function will still work just the same.
For example, you can change the code in challenge #82 to be:

```
goto = (cool) ->
    turnTo cool
    step distanceTo cool
for m in matches
    goto m
    grab()
    goto pile
    drop()
```

Lesson 19 – Fake it ‘til You Make it!

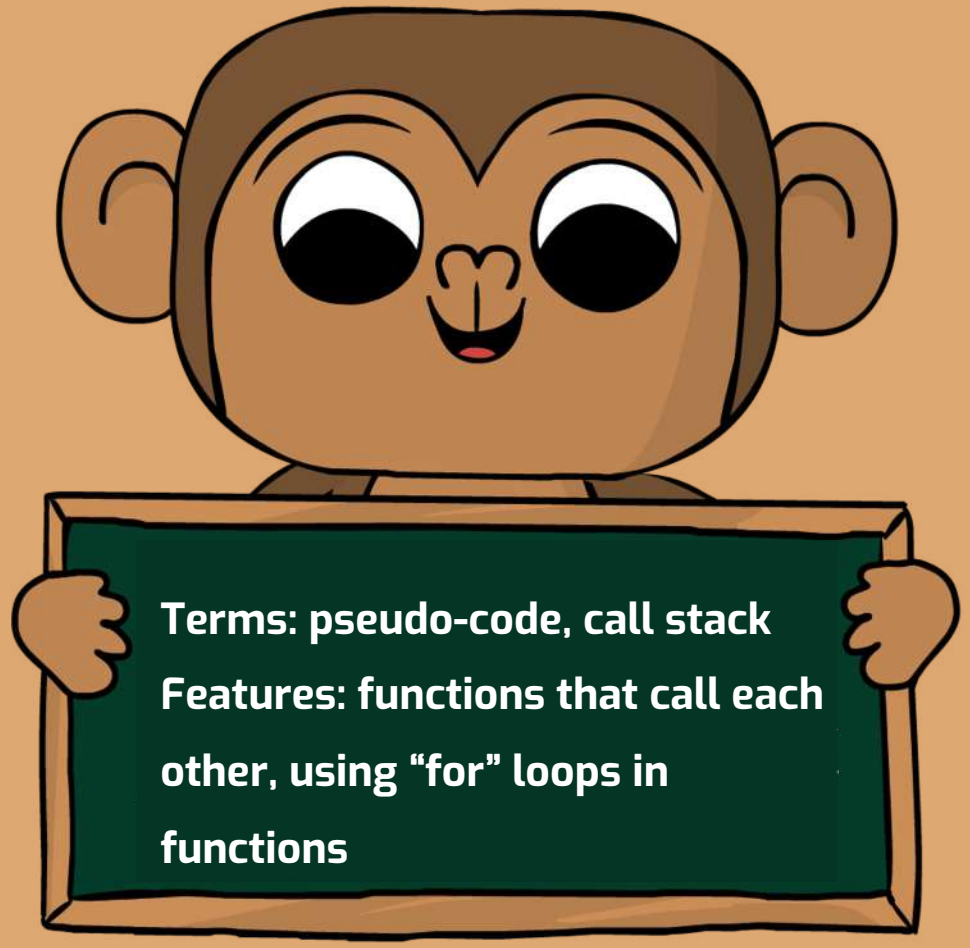
This is another lesson about functions. Your students will practice writing function definitions, and experiment with defining functions that call each other.

Objectives

In this lesson, students will:

- Review, practice and deepen their expertise in functions
- Complete challenges 86-90

Components



U.S. Standards Addressed

CSTA-K12 Computer Science Standards	
• 1B-AP-10 • 1B-AP-12 • 1B-AP-15	• 2-AP-10 • 2-AP-12 • 2-AP-14 • 2-AP-16 • 2-AP-17

Part 1: 10 Minutes

Introduction

Review	**5 mins.**

Return to challenge #73 by using the challenge map. Recall with your students that this pattern of stepping over a whole collection of objects appeared in the solution to a couple of other challenges as well.

```
for i in islands
    turnTo i
    step distanceTo i
```

Recall with your students that a loop is used when the same pattern repeats itself in the code several times in a row. That way, we do not have to write the same code over and over again.

Just like that, a function is used when the same pattern appears in different blocks of code, so that we do not have to repeat the same code in different places. Just like with loops, using functions will be all about identifying patterns.

Part 1: 10 Minutes

Introduction Cont.

Activity	**5 mins.**

Give your students the following assignment:
Imagine you are standing in the hall in front of the classroom door, which is closed. In your own words, write instructions for doing the following task: getting into the classroom and walking in the shape of a square, saying hi and shaking hands with each of the students, and finally leaving the classroom.
After they finish, ask one or two students to share the instructions they wrote with the class.
Show the following instructions:

open door
walk 3 steps
repeat 4 times
- walk 5 steps
- turn right

to every student in the class
- say hello
- shake hand

go to door
walk 3 steps

Part 1: 10 Minutes

Introduction Cont.

Activity Cont.	**5 mins.**
Point out that the instructions shown are not in a computer language, so they are not "real" code. However, they are still built in the same way as real code and represent the same ideas, such as repetition (simple loop - repeat 4 times), iteration (for loop - to every student in the class), and use a certain set of actions on an object (function - go to door) etc. This is a common way to express a set of actions (or an algorithm) without writing actual code. It is called *pseudo-code*, meaning bogus or fake code. The prefix pseudo has greek origins (pseudes = false) and is pronounced like soodoe.	

Part 2: 25 Minutes

Playtime

Log-in	**1 min.**
To review the log-in instructions, please click here.	
Playtime	**9 mins.**
All students should complete challenges 86-90 with at least two stars. (Students from the age of 12 and up should get three stars.) Use the classroom dashboard to keep track of students' achievements. Use this time to walk around the class and help students who are struggling. The topics in this lesson are complex so it is natural that some students will need more help. If you notice a few students who are struggling, bring them together in a small group and provide more explanation. **Tip:** 1. When you are asked to define goto or collect, it is a good idea to click the "function" block at the bottom of the editor. You can also go back to a previous challenge using the map to see the way they were defined earlier. 2. When asked to complete or fix a function definition, make sure it does what it's supposed to - you'll find that in the comment above the definition.	

Part 2: 25 Minutes

Playtime Cont.

Walk-through	**7 mins.**

Open challenge #88 and reset the code if necessary.
Click run to see what happens, and observe outload that something is not working right.
Ask your class if they can say, in their own words, what the rat should do to solve this. Then write that as pseudo-code on the whiteboard. The answer should be something like this:

step over all the islands
grab the match
step over all the rafts
drop the match

They can find something like this in the main code starting at line 12.
Ask your students if they can find a hint in the comments on what needs to be fixed. Of course, one of the comments says "fix this". So let's have a look at the main code first and compare it to the pseudo-code in the answer above. Show them that the pseudo-code matches the main code. You can try to run the code again and see that it still does not work. So the main code is correct and doesn't need any fixing.
Let's look at "gotoAll". What is this function supposed to do? It is given "stuff" as an argument, so we can tell by its name that it should go to all "stuff", whether stuff is islands, rafts, or something else.

Part 2: 25 Minutes

Playtime Cont.

Walk-through Cont.	**7 mins.**

It is probably correct to use a for loop to *iterate* over all stuff, but inside the for loop we are trying to go to all the stuff, so we shouldn't just step 1 each time, but step the right distance, just like we did in challenge 68. Fix the inside of the for loop to read:

```
turnTo s
step distanceTo s
```

Now run the code. It will only get one star, so ask the class if they can suggest a way to get 3 stars. Hint: use goto!

The answer is that inside gotoAll we can use goto, which does exactly the same as the two lines of code inside the for loop. This will save us 1 line of code.

Replace the code inside the for loop with:

```
goto s
```

Explain that the main code calls the function gotoAll, which in turn calls the function goto. This is a very common thing in coding: one function calls another function, which calls another function, and so on. This results in a whole stack of functions calling each other, which is named: *the call stack*.

Run the code again to achieve 3 stars.

Part 2: 25 Minutes

Playtime Cont.

Walk-through (2)	**3 mins.**

Open challenge #89 and reset the code if necessary.
Instruct your students to look for a hint in the comments, and tell you what to do to solve the challenge. Once they have concluded that they should fix the function getAndReturn, ask them to write pseudo-code that represents the code that will be inside the function. Start from the comment above the function:

```
get to r
grab
then return to the turtle
```

Now compare this pseudo code with the current contents of the function. Observe that the first line corresponds to “get to r”, so there are two things missing:
1. grab()
2. return to the turtle

Grab is simple. But how do we “return to the turtle”? Answer: by using the function “goto” with argument turtle.
The code inside the function should look like this:

```
goto r
grab()
goto turtle
```

Run this code to get a 3-star score.

Part 2: 25 Minutes

Playtime Cont.

Playtime	**5 mins.**
Students continue their work on challenges 86-90.	
Practice	
Encourage students who finish early to open skill mode on the map and complete unlocked challenges. After completing challenges 86-90, skill challenges 7-6 – 7-15 are unlocked.	

Part 3: 10 Minutes

Debriefing

Walk-through	10 mins.

Open challenge #90 and delete all the lines of code except for the last four lines.
Tell your students that this challenge can be solved with only these four lines of code.

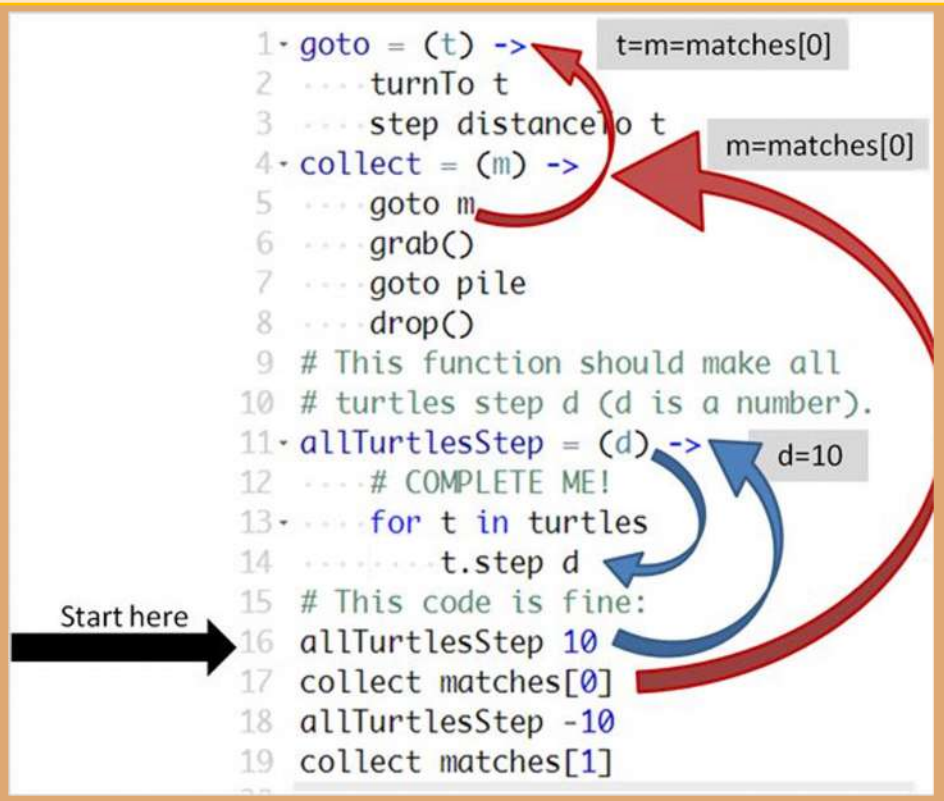
```
1 allTurtlesStep 10
2 collect matches[0]
3 allTurtlesStep -10
4 collect matches[1]
```

Click the **Run** button and let the error message pop up: "I don't know what **allTurtlesStep** is".
Explain that when we started playing Coding Adventure, most of our code looked like this, with short lines saying "step 20" or "turn left". The truth is that "behind the scenes" the computer had a definition of the functions we used, so we didn't get an error message like this one. Now, we have to define our own functions if we want the computer to do what we want. Click on the reset button to show the original code. Solve challenge #90 with your students. Explain to them using arrows how the different arguments are passed from one line to another (see graphic below).
The first line of the *main* code is "allTurtlesStep 10"; the number *10* is the argument. The argument is first passed to the function "allTurtlesStep= (d) ->", so *d=10*. Then it moves into the "for" loop: "for t in turtles" and the *t.step 10*.
The next line of the main code is "collect matches[0]", where the argument is matches[0]. Its first step is to pass into the function "collect = (m) ->" where now *m=matches[0]*. Then, we have a function called from within a function, so the next move for our argument is to pass into "goto = (t) ->", so *t=matches[0]*.

Part 3: 10 Minutes

Debriefing Cont.

Walk-through Cont.	**10 mins.**
Ultimately, your students should see how an argument can take on different names, and how useful and important functions are when programming.	



Lesson 20 – It Ain't Over Until It's Over!

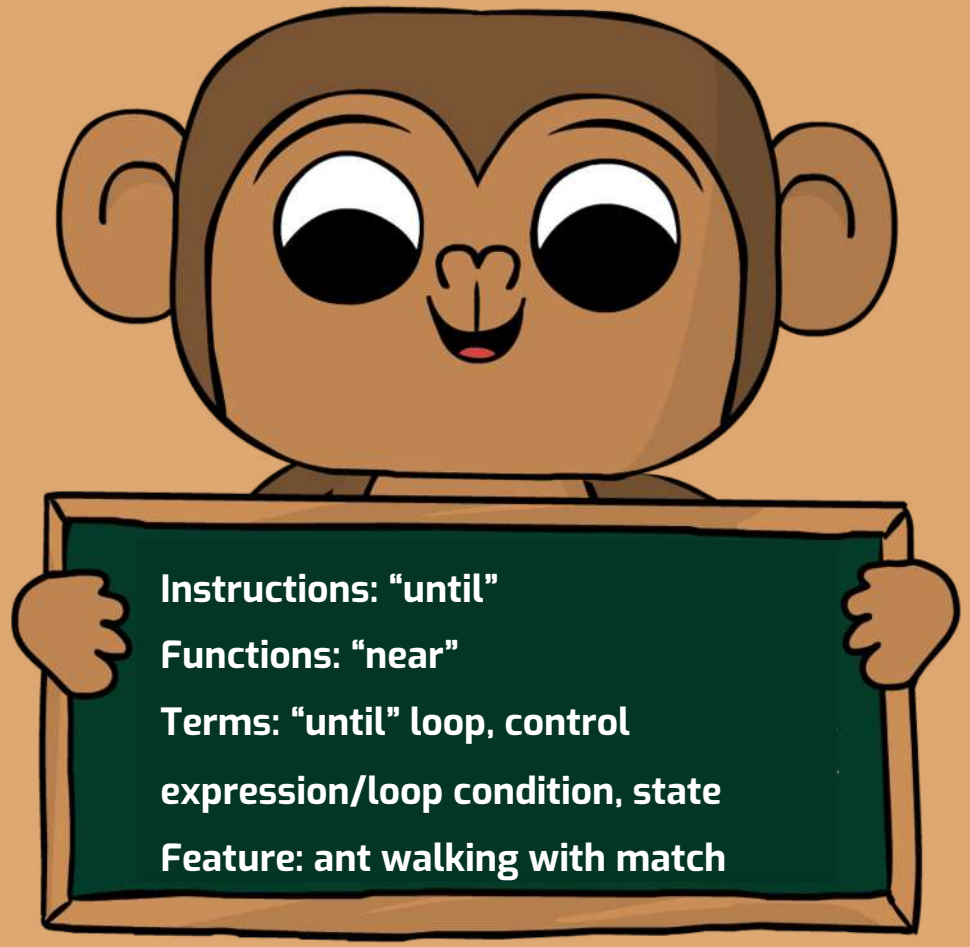
We're taking a break from functions to learn a new kind of loop. After covering simple loops and "for" loops, in this lesson, we will learn about the "until" loop.

Objectives

In this lesson, students will:

- Review simple loops and "for" loops
- Work with "until" loops
- Work with the function "near"
- Complete challenges 91-95

Components



U.S. Standards Addressed

CSTA-K12 Computer Science Standards	
• 1B-AP-10 • 1B-AP-12 • 1B-AP-15	• 2-AP-10 • 2-AP-12 • 2-AP-14 • 2-AP-16 • 2-AP-17

Part 1: 15 Minutes

Introduction

Activity	5 mins.

Some of our favorite childhood games are based on doing an action until the condition changes. For example, in freeze dance, you dance until the music stops and then you freeze. These activities reflect how the "until" loop works. Choose an activity to play according to your students' age group and the size of your classroom. Two possible games are musical chairs or freeze dance. You are welcome to choose a different game.

Have your students play the game you selected. After the activity, talk to them and try to simplify the actions they took. For example, if you picked freeze dance, then the students danced and when the music stopped, they froze. Write this on the board.

Ask your students to identify the types of loops they have learned. Have a brief discussion about simple loops (a sequence of instructions that repeats a specified number of times) and "for" loops (performs a set of actions on all objects in a collection).

Part 1: 15 Minutes

Introduction Cont.

Explanation	**5 mins.**

In this lesson, we will use a new kind of loop: the "until" loop. This loop is very useful but also dangerous when used carelessly. It is useful because it helps us when we have a repetitive task to be done, but it could be dangerous to use because if we are not careful, it might go on forever.

The "until" loop contains a block of code that will repeat *until* "something happens", or more accurately, until a specific condition is met. This condition is called a *control expression* or *loop condition*. The computer checks the condition at every repetition. If the answer is false, the loop will keep going. It will only stop once the answer is true. If we do not pay attention, we might give the "until" loop a condition that will not be met. This will cause the loop to keep going forever and **could even cause the program to crash**.

Part 1: 15 Minutes

Introduction Cont.

Activity	**5 mins.**

If we go back to our activity at the beginning of the lesson, we can now write it in pseudo-code as an "until" loop:

```
until music stopped
    dance
freeze
```

Observe that dance has an indentation because it is inside the loop, but freeze does not have an indentation because it is only carried out after after the loop ended. Ask your students, "What is the control expression for this 'until' loop?"

Answer: music stopped. As long as we heard music, we kept dancing. We can translate this to computer language: the computer kept asking, "Did the music stop?" and kept getting a false answer as long as there was music playing. Once the computer asked, "Did the music stop?" and the answer was true, we stopped dancing. The control expression we will use in this lesson is the function "near". The value returned by the function "near" will determine when the "until" loop will stop executing. Ask your students if they can think of other examples from everyday life that repeat until a certain condition is met (or until "something happens"), and write them in pseudo-code. Examples for possible answers:

```
until cream is solid
    whip
eat
```

Or

```
until time to sleep
    watch favorite tv show
go to sleep
```

Part 2: 25 Minutes

Playtime

Log-in	**1 min.**
To review the log-in instructions, please click here.	
Playtime	**4 mins**
All students should complete challenges 91-95 with at least two stars. (Students from the age of 12 and up should get three stars.) Use the classroom dashboard to keep track of students' achievements. Use this time to walk around the class and help students who are struggling. **Tip:** pay attention to what should happen before the loop, in the loop, and after the loop.	

Part 2: 25 Minutes

Playtime Cont.

Walk-through	**5 mins.**

Open challenge #92 and reset the code.
Ask your students: "what do you think the line say near match does?"
Click run to test the initial code and find out the answer. Note the speech bubble that appears as the rat is getting closer to the match. It says "no" twice and then "yes". Note by watching the orange highlight that after the "yes" appears, the execution continues to the code after the loop.
Explain how say near match works (similarly to step distanceTo banana):
First the computer asks the question "near match" and gets the answer "yes" or "no". Remember: asking the question is called *calling* the function and the answer is called the *return value*.
The return value then becomes the argument of the function say. In other words, near match is replaced by the answer, so the line becomes say "yes" or say "no".
Complete the solution by moving drop() from line 5 to line 8. Run the solution and observe that it achieves only 2 stars. Ask your students to read the hint and improve the solution. Which line should be removed?
Answer: The line that says "say" is great for learning about functions, but it is not necessary in order to complete the challenge. Remove the unnecessary line and run the improved solution to get 3 stars.

Part 2: 25 Minutes

Playtime Cont.

Walk-through (2)	**5 mins.**

Open challenge #93 and reset the code.
Ask your students: "what do you think will happen?"
Click run to test the initial code and find out the answer: the loop continues infinitely. Note 3 signs that tell us the loop is still running: the "run" button still shows up as "stop", line 6 is highlighted in orange, and of course we didn't get any completion/failure pop up screen.
Let's analyze why the loop didn't stop. What is the loop condition? Answer: near pile. But the rat keeps stepping a different direction so it never reaches the pile.
Fix the code by entering "turnTo pile" in line 4, and run the solution. Show that the loop no longer runs infinitely. We did not change the code of the loop itself in order to prevent it from running forever. The state at which we started the loop was the only thing that was different.
There is still a little bit of work left in order to solve this challenge completely (entering drag() and drop() in the right places). Solve it together with your students.

Part 2: 25 Minutes

Playtime Cont.

Playtime	**10 mins.**
Students continue working on challenges 91-95.	
Practice	
Encourage students who finish early to open skill mode on the map and complete unlocked challenges. After completing challenges 91-95, skill challenges 8-1 – 8-3 are unlocked.	

Part 3: 5 Minutes

Debriefing

Activity	**5 mins.**

Let's try to clarify how we can make a loop run forever. Ask for two student volunteers. Instruct the volunteers to stand on opposite sides of the classroom facing each other. Tell them to "step 1" until they are near each other. Once they are near each other (near means within arm's reach), they should stop walking. After every step, they should say "no" if they are far from each other or "yes" if they are near one another. You can also write this "until" loop on the board:

```
until near student
    step 1
    say near student
```

This is an example of a good "until" loop.

Now, ask them to stand back to back, just beyond arm's reach. Give them the same instructions from the previous activity. In this example, the students will never meet, and if they could, they would have kept going forever.

Emphasize that the code was the same in both examples, it is only the initial state that determined whether the loop will run forever or not. Therefore, we must always be careful when writing loops and think through when the loop condition will cause it to stop.

Add that a loop that goes on forever is called in infinite loop.

Lesson 21 – Cut to the Chase

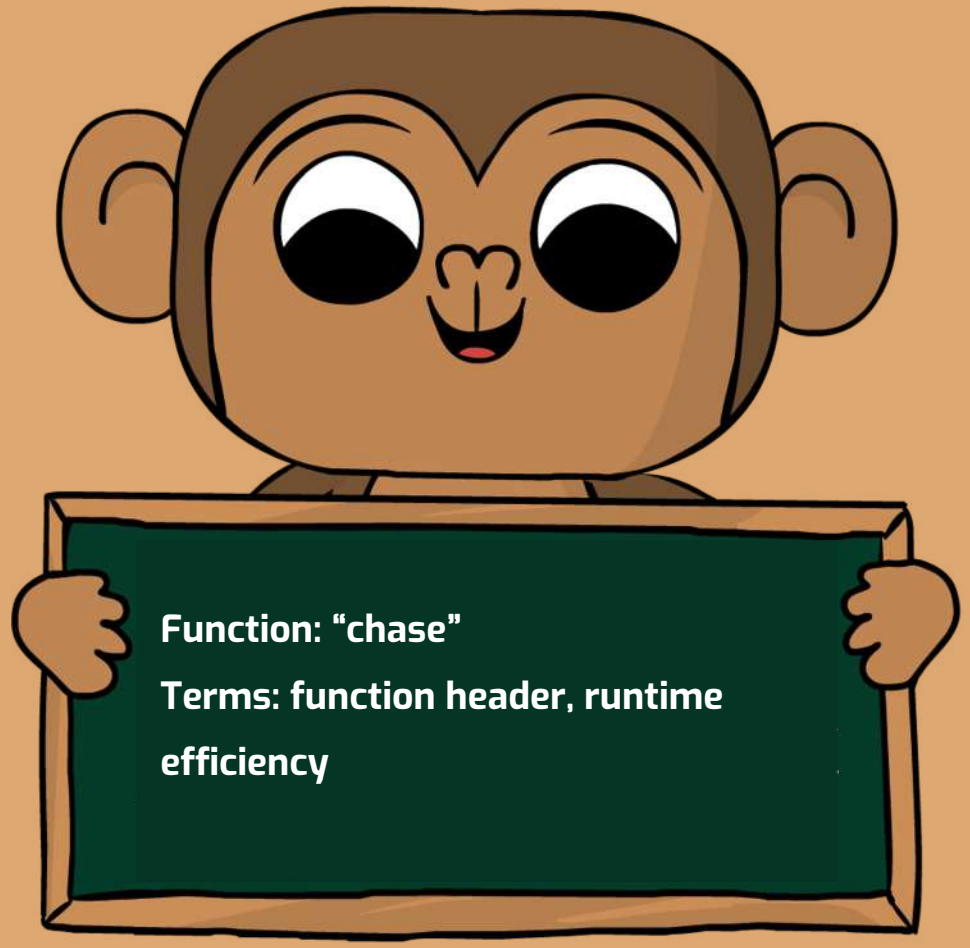
Your students will practice further the use of until loops, including writing loops with more than one statement inside them, and using an until loop inside a function definition.

Objectives

In this lesson, students will:

- Practice until loops and function definitions
- Write an until loop from scratch
- Complete challenges 96-100

Components



U.S. Standards Addressed

CSTA-K12 Computer Science Standards	
• 1B-AP-10 • 1B-AP-12 • 1B-AP-15	• 2-AP-10 • 2-AP-12 • 2-AP-14 • 2-AP-16 • 2-AP-17

Part 1: 10 Minutes

Introduction

Review	**10 mins.**

Practice using “until near” with your students: ask them to write 4 lines of code that replace the following code but do the exact same thing.

```
turnTo banana
step distance to banana
say “hooray”
```

The special instructions are that the new code has to use an until loop, and is not allowed to use the function distanceTo.

Answer:

```
turnTo banana
until near banana
    step 1
say “hooray”
```

Make sure your students have accurately used the syntax of the until loop, including the indentation in front of step 1, and no indentation in front of say “hooray”.

Part 1: 10 Minutes

Introduction Cont.

Review Cont.	**10 mins.**

Ask them what would be different if there was indentation in front of say, like this:

```
turnTo banana
until near banana
    step 1
    say “hooray”
```

Answer: the monkey would say hooray many times instead of once, because the “say” is inside the loop.

Review the definition of loop condition: the condition that the computer uses in order to decide whether it has to continue executing a loop or to move on to the next statement after the loop.

Function definitions: ask your students what a function definition is. Ask them to write the definition for a function that takes an argument, turns to face that object and says “done”.

Answer:

```
turnAndReport = (m) ->
    turnTo m
    say “done”
```

Make sure your students have used the correct syntax including the indentation and punctuation.

Part 2: 30 Minutes

Playtime

Log-in	**1 min.**
To review the log-in instructions, please click here.	
Playtime	**14 mins**
All students should complete challenges 96-100 with at least two stars. (Students from the age of 12 and up should get three stars.) Use your classroom dashboard to keep track of students' achievements. Use this time to walk around the class and help students who are struggling.	

Part 2: 30 Minutes

Playtime Cont.

Walk-through	**5 mins.**
Open challenge #98 and reset the code if necessary. Solve the challenge together with your students: try to run the initial code. Show that the rat follows the direction of the match but does not move, and point out that there could be more than one statement inside the until loop just like in for loops. Ask them to complete line 3. A possible correct answer would be step 1. What are other possibilities? Experiment with a few different numbers (0.5,2,4). Run the solution with two different options - 0.5 and 4, and observe that each represents a different algorithm. Ask your students to use a timer to time the execution of each algorithm. Explain that in this particular case, the second algorithm took less time to complete. In such a case we say that the second algorithm is more *runtime-efficient*.	
Playtime	**5 mins.**
Students continue working on challenges 97-100	

Part 2: 30 Minutes

Playtime Cont.

Walk-through	**5 mins.**

Open challenge #100 and reset the code if necessary.
Solve the challenge together with your students. Try using the following code for the loop in lines 2-4, and running the solution.

```
until near match
    turnTo match
    step 1
```

Show that the computer does not know what match is. Ask your students what the right object has to be instead of match. Explain that the code in the beginning of a function that contains the =, () and -> signs is called the *function header*. In this case it is the code in line 1, reading: chase = (m) ->. You can always check the function header to find out the name of the argument that is passed to the function.
So the right code for the loop in this case is:

```
until near m
    turnTo m
    step 1
```

Run this solution to get 3 stars.
Now experiment a little further by replacing line 4 by: step distanceTo match
Ask your students to describe the difference between the two algorithms. Answer: one of them takes 1 step at a time, then turns to face the target. The other steps the whole distance, then turns to face the target.
Ask your students to tell you which one of the algorithms is more runtime efficient.

Part 2: 30 Minutes

Playtime Cont.

Practice	
Encourage students who finish early to open skill mode on the map and complete unlocked challenges. After completing challenges 96-100, skill challenges 8-4 – 8-6 are unlocked.	

Part 3: 5 Minutes

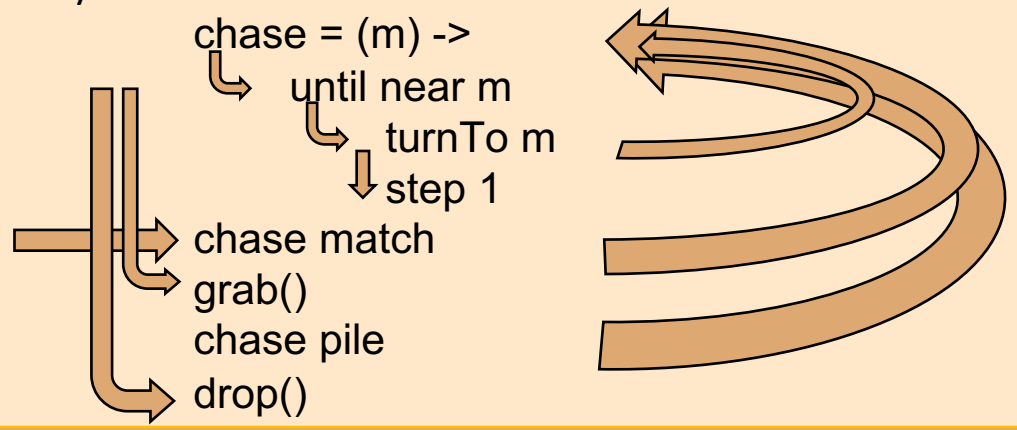
Debriefing

Debriefing	**5 mins.**

Copy the following code from a solution to challenge #99 to the whiteboard.

```
chase = (m) ->
    until near m
        turnTo m
        step 1
chase match
grab()
chase pile
drop()
```

Ask your students to trace the execution on the whiteboard with arrows. Make sure they start at “chase match”. For example:



Lesson 22 – wait() for It…

In this lesson, your students will finish chapter three of Coding Adventure. These last five challenges will cover everything we've learned in the last 22 lessons, and will also introduce two new functions: "wait()" and "sleeping()".

Objectives

In this lesson, students will:

- Review the different topics covered in Coding Adventure
- Work with the functions “cat.sleeping()” and “wait()”
- Use functions with “until” loops
- Complete challenges 101-105

Components

U.S. Standards Addressed

CSTA-K12 Computer Science Standards	
• 1B-AP-9 • 1B-AP-10 • 1B-AP-12 • 1B-AP-15	• 2-AP-10 • 2-AP-11 • 2-AP-12 • 2-AP-14 • 2-AP-16 • 2-AP-17

Part 1: 15 Minutes

Introduction

Review	**10 mins.**

The challenges in this lesson will cover all the topics that were covered since the start of Coding Adventure. Ask your students to name the different topics previously discussed. Write their answers on the board. Answers should include:

- Statements
- Function calls
- Arguments
- Loops
- Variables
- Array indexes
- Return values
- Output
- For loops
- Functions
- Until loops

Go over the topics one by one to make sure that your students remember the essences of each topic. You can use the Reference Card at the end of this guide for help.

Part 1: 15 Minutes

Introduction Cont.

Explanation	**5 mins.**

In this lesson, we will meet a new character and learn two new functions to go along with the “until” loop.
Cats love to do two things: sleep and catch mice. Well, our cat is no different. If he sees our rat, he will most definitely catch it. But once in a while, the cat falls asleep. This leaves our rat a window of opportunity to act and catch more matches.
Ask your students, “What do you think we need to put in the code in order for the rat not to be eaten by the cat?”
The two functions we are going to use are “cat.sleeping()” as the loop condition and “wait()” inside of the loop. The code will look like this:

```
until cat.sleeping()
    wait()
```

Part 1: 15 Minutes

Introduction Cont.

Explanation Cont.	**5 mins.**

As a result, when we click **Run**, the rat will wait until the cat is sleeping before he will move on to the next thing we tell him to do. Ask your students, “Should the other actions we want the rat to do be inside of the loop or outside of it?” The answer is of course outside of it! If we put other actions inside of the loop, the rat will do them while the cat is awake.
In this lesson, your students will have to use “until” loops, functions, and “for” loops all together. This is a good opportunity to go over planning. When we write code, we have to take into consideration that computers read the code from TOP to BOTTOM, and we have to think ahead about which instruction should come first. Now that we have to define our own functions, this matter is even more important.

Part 2: 25 Minutes

Playtime

Log-in	**1 min.**
To review the log-in instructions, please click here.	
Playtime	**19 mins**
All students should complete challenges 101-105 with at least two stars. (Students from the age of 12 and up should get three stars.) Use your classroom dashboard to keep track of students' achievements. Use this time to walk around the class and help students who are struggling.	

Part 2: 25 Minutes

Playtime Cont.

Practice	
Encourage students who finish early to open skill mode on the map and complete unlocked challenges. After completing challenges 101-105, skill challenges 8-7 – 8-15 are unlocked.	
Quiz	
After completing challenges 76 – 105, you can assign your class the first quiz – Part 2: Functions & Until Loops. The quiz includes 5 challenges. You can assign quizzes from the Quizzes tab on your teacher dashboard.	

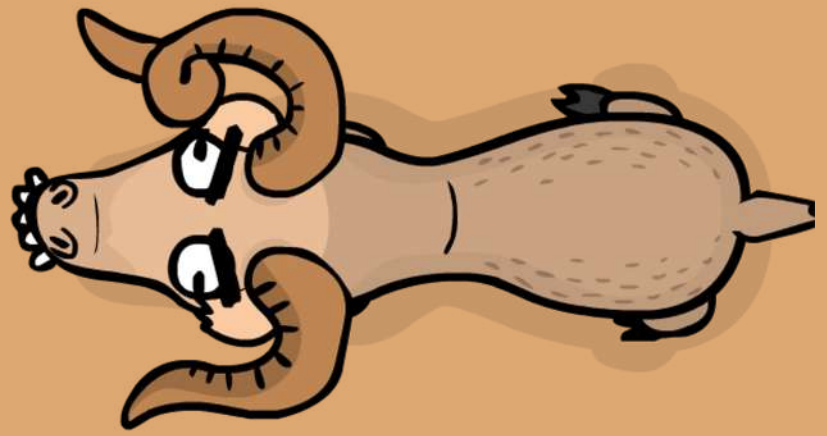
Part 3: 5 Minutes

Debriefing

Review	5 mins.
Open challenge #105 and ask your students how they solved this challenge. Ask them, “What part of the code did you start writing first, the code or the function’s definition? What do you think is the correct way?” Explain that there isn’t one correct way. As long as the end result is correct and the computer can read the code without giving an error message, it is good. Programmers usually develop their own habits and routines when writing programs, but as far as the code itself, they need to follow the conventions of the language.	

Lesson 23 – Act the Goat

This lesson marks the beginning of chapter 4 of Coding Adventure. In this lesson your students will meet the new character - the amusing goat. With the goat we will learn about the important concept of conditionals.

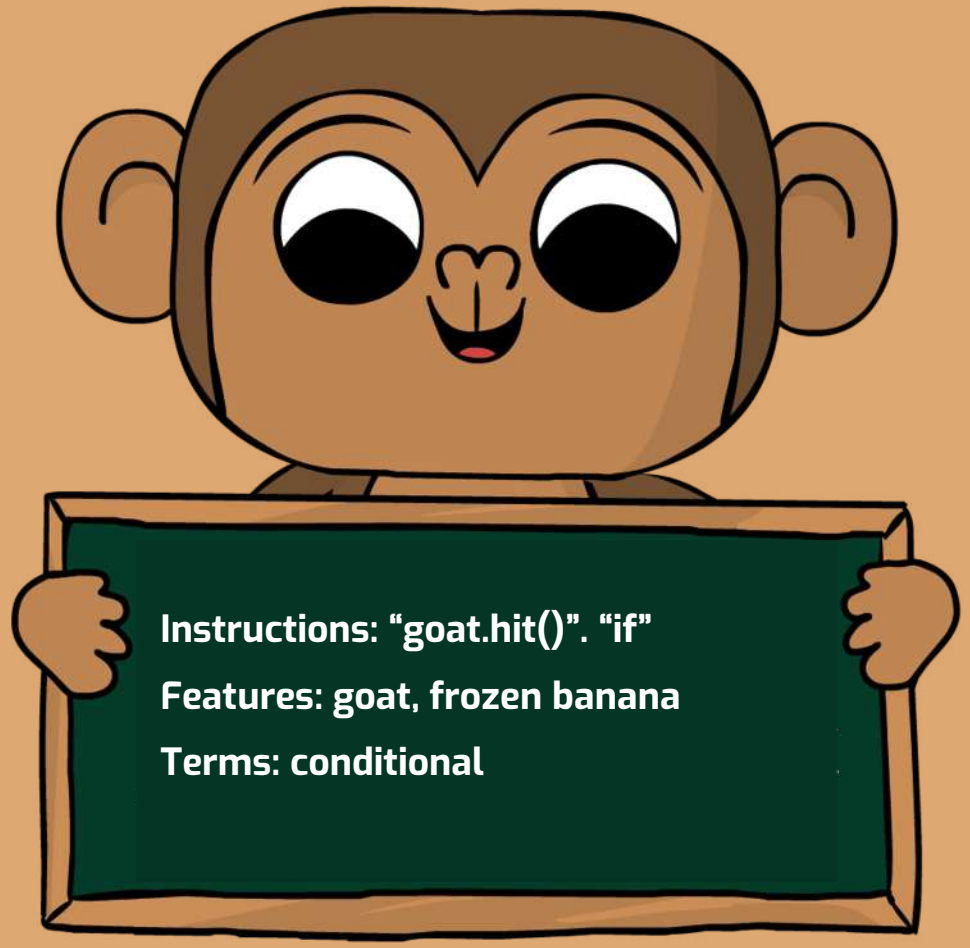


Objectives

In this lesson, students will:

- Learn about conditionals and use them in their code
- Complete challenges 106-115

Components



U.S. Standards Addressed

CSTA-K12 Computer Science Standards	
• 1B-AP-9 • 1B-AP-10 • 1B-AP-12 • 1B-AP-15	• 2-AP-10 • 2-AP-11 • 2-AP-12 • 2-AP-14 • 2-AP-16 • 2-AP-17

Part 1: 15 Minutes

Introduction

Activity	**7 mins.**
Ask for a student volunteer. Instruct the volunteer to walk around the classroom and ask every single student the same yes/no question. If the answer is “yes”, then the student who answered the question should join the volunteer and walk behind him as he completes his task, forming a long “snake”. If the answer is “no”, then the volunteer should move to the next student. Here are some examples for the question your volunteer can ask (choose one or think of your own): • Are you 9 years old? • Are you the youngest in your family? • Are you wearing something blue? • Do you like basketball?	
Questions to Ask	**3 mins.**
Ask the volunteer- • How was the activity? • How did it feel to go around the room and ask the same question over and over again? Ask the class- • What do you think was the point of this activity? • How does this relate to computer programming?	

Part 1: 15 Minutes

Introduction Cont.

Explanation	**5 mins.**

Sometimes we want our code to decide what to do based on a certain condition (e.g. decide whether to get up and walk based on the condition: do you like basketball?). The state might be unknown to us when we write our code, and the computer will have to decide what to do while the code is executed. Today we will learn how to instruct the computer to decide what to do, based on a condition.

We already know what to do when we have more than one banana or match, but what if we have two bananas, and one of them is frozen?

Today we will meet a new character that will help us deal with frozen bananas, and we will have to help the goat by identifying the frozen bananas from the regular ones.

Part 2: 25 Minutes

Playtime

Log-in	**1 min.**
To review the log-in instructions, please click here.	
Playtime	**4 mins**
All students should complete challenges 106-110 with at least two stars. (Students from the age of 12 and up should get three stars.) Use your classroom dashboard to keep track of students' achievements. Use this time to walk around the class and help students who are struggling.	

Part 2: 25 Minutes

Playtime Cont.

Walk-through	**4 mins.**

Open challenge #110 with its 3 star solution. Discuss the solution with your students. It should look like this:

```
goat.goto bananas[0]
goat.hit()
goat.goto bananas[2]
goat.hit()
for b in bananas
    goto b
```

Ask your students: “how did we come up with this solution?”. Explain that the process was that we identified by ourselves all the bananas that were frozen and made the goat step only to them. Then the monkey could collect all the bananas.

Ask: “What if there were 50 frozen bananas and 50 normal ones? Would it make sense for us to go over them one by one?”. Explain that we can make the computer ask a question and make a decision based on the answer. For example we can write a loop that goes over all the bananas and asks “is this banana frozen?”. Only if the answer will be yes, will we send the goat to hit that banana.

Part 2: 25 Minutes

Playtime Cont.

Walk-through Cont.	**4 mins.**

In programming this is called a *conditional*. Later this lesson we will learn how to use conditionals.
Open challenge 111 and reset the solution. Before solving the challenge, ask your students what the code will do. If needed, return to challenge #92 to show how yes/no return values work.

Run the initial solution and observe the speech bubbles to see if the answers were correct. Discuss if needed: explain that the function frozen() returns a value that is yes or no according to the state of the banana. That yes or no is in turn passed as an argument to the function say.

Part 2: 25 Minutes

Playtime Cont.

Playtime	**4 mins.**
All students should complete challenges 111-113 with at least two stars. (Students from the age of 12 and up should get three stars). Use your classroom dashboard to keep track of students' achievements. Use this time to walk around the class and help students who are struggling.	

Part 2: 25 Minutes

Playtime Cont.

Walk-through	**5 mins.**

Open challenge 114 and delete the code. Solve the challenge together with your students:
Start with checking which bananas are the frozen ones (click a banana to see which one it is), and then writing explicitly the code that gets the goat to hit that banana. This will give you the following “naive” code

```
goat.goto bananas[0]
goat.hit()
goat.goto bananas[2]
goat.hit()
for b in bananas
    goto b
```

Show that it only gets one star and notice the hint that says we should use if. Click reset, and replace line 3 by:

```
goat.goto b
goat.hit()
```

Show that it gets 3 stars, and the code is shorted. But most importantly, point out that we didn’t have to check the bananas manually to get the code to work, the computer did it for us!

Part 2: 25 Minutes

Playtime Cont.

Playtime	**7 mins.**
All students should complete challenges 114-115 with at least two stars. (Students from the age of 12 and up should get three stars). Use your classroom dashboard to keep track of students' achievements. Use this time to walk around the class and help students who are struggling.	
Practice	
Encourage students who finish early to open skill mode on the map and complete unlocked challenges. After completing challenges 106-115, skill challenges 9-1 – 9-8 are unlocked.	

Part 3: 5 Minutes

Debriefing

Explanation	**5 mins.**

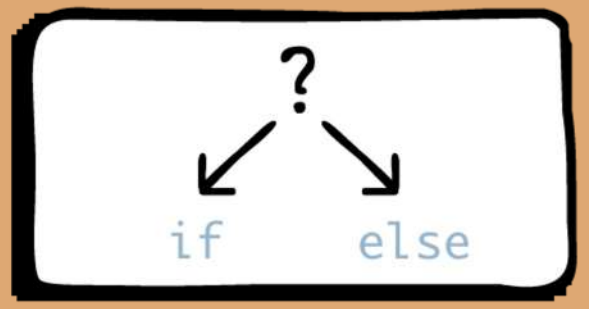
Use this debriefing session to review the new concept we learned today.
Ask: “What do conditionals do in our code?”
Answer: A conditional makes the computer execute statements according to the answer to a question (i.e a condition).
Ask: “What is the statement in the code that we use in order to write a conditional?”
Answer: if
Ask: “What is the syntax of an if statement?” (you can ask a student to solve this on the whiteboard)
Answer:

```
if [condition]
    statement
    statement
```

Ask: “When are conditionals useful?”
Answer: When we want to computer to do an action according to the answer to a certain questions.

Lesson 24 – Green Banana Sorbet

In this lesson your students will continue to use if statements, and learn a more sophisticated conditional called if-else.



Objectives

In this lesson, students will:

- Practice using if statements
- Use if-else statements
- Complete challenges 116-121

Components

U.S. Standards Addressed

CSTA-K12 Computer Science Standards	
• 1B-AP-9 • 1B-AP-10 • 1B-AP-12 • 1B-AP-15	• 2-AP-10 • 2-AP-11 • 2-AP-12 • 2-AP-14 • 2-AP-16 • 2-AP-17

Part 1: 20 Minutes

Introduction

Activity	**5 mins.**

Today we're going to play the same game we played in the previous lesson, but with a little twist.
Choose a volunteer, ask the volunteer to walk around the class and ask each student in your class the same yes/no question. If they answer "yes", they should go stand on the right side of the classroom. If they answer "no", they should go stand on the left side.
Here are some examples for the question your volunteer can ask (choose one or think of your own):

- Are you 9 years old?
- Are you the youngest in your family?
- Are you wearing something blue?
- Do you like basketball?

Explanation	**5 mins.**

Let's write the instructions of the game as an "if" statement in pseudo-code:

```
for s in students
    if s.likesBasketball
        stand right
    if not s.likesBasketball
        stand left
```

Part 1: 20 Minutes

Introduction Cont.

Ask the Following	**5 mins.**

- Can you think of a way to write the instructions using only one if statement?

The instructions say that IF you answer “yes”, then you go right, **otherwise** go left. Meaning any other answer except for “yes”, will lead to going left. For this purpose we can use an If-Else statement.

```
If s.likesBasketball
        go right
else
        go left
```

Part 1: 20 Minutes

Introduction Cont.

Explanation	**5 mins.**

An if statement helps the computer decide what to do based on a certain condition. When we use an if-else statement, we are telling the computer what to do in case the condition is true (under the “if”), and what to do in case the condition was not met, meaning it is false (under the “else”).

```
if [condition]
    code to be executed if the condition is true
else
    code to be executed if the condition is false
```

The state might be unknown to us when we write our code, but just to be sure we are telling the computer what to do in both cases, and it will decide what to do while the code is executed.

Today we will learn how to instruct the computer to decide what to do when the condition is true and when it is false.

Part 2: 20 Minutes

Playtime

Log-in	**1 min.**
To review the log-in instructions, please click here.	
Playtime	**9 mins.**
All students should complete challenges 116-121 with at least two stars. (Students from the age of 12 and up should get three stars). Use your classroom dashboard to keep track of students' achievements. Use this time to walk around the class and help students who are struggling.	

Part 2: 20 Minutes

Playtime Cont.

Walk-through (1)	**7 mins.**

Open challenge #120 without its solution. Discuss the starting code with your students. It should look like this:

```
if banana.green()
    say “It’s green!”
else
    say “It’s yellow!”
```

Run the initial code and observe the speech bubble to see if the condition is true or false and how it affects the action. Explain that the function green() returns a value that is yes or no according to the state of the banana.

The starting code takes into consideration two scenarios, the first is if the banana is green, and the second if it is not green.

Why is the second scenario “not green”? should it just be “yellow”?

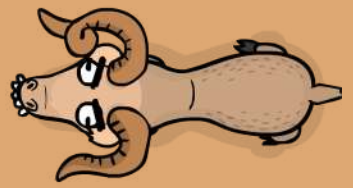
When using an If-Else statement, we are telling the computer what to do in case the condition is true or false, so if the condition is “banana.green()” the second scenario is “not green”.

Solve this challenge with your students:

- If the banana is green, what will happen?
- If the banana is not green, what will happen?

Part 2: 20 Minutes

Playtime Cont.



Walk-through (2)	**3 mins.**
Open challenge #121 without its solution. Ask your students: • What will the initial code do? • Why are the goat and monkey taking turns moving? ○ Because of the for loop and the order of the bananas in the array (if needed, go back to **lesson 11 - A for Array** to remind your students of array indexing). • What do we need to fix to solve this challenge? Solve the challenge with your students.	
Practice	
Encourage students who finish early to open skill mode on the map and complete unlocked challenges. After completing challenges 116-121, skill challenges 9-9 – 9-11 and 10-1 and 10-2 are unlocked.	

Part 3: 5 Minutes

Debriefing

Below you'll find two suggestions for debriefing activities.
You can choose whichever one you want to do, or you can do both if you have the option.

Activity 1: Explanation	**5 mins.**

Ask your students to think of different scenarios from their life where they have to choose between two things, depending on something other than themselves. for example:

- Should I wear a sweater today? If it's cold outside, yes. If not, then no.
- Should I use my bike to get to school? If I'm late, I should take my bike. If I'm not, then I can walk.
- Should I text my mom or should I call? If I have a lot to say, I should call. If not, I can text.

Ask each student to think of a similar scenario from their life, and write it in pseudo code. Pay attention that some scenarios can fit into an if statement, and some should be in an if-else statement.

Examples:

```
if cold.outside
    wear sweater
if LateToSchool
    ride.bike
else
    walk
```

Have your students share their scenarios with one another, or have some write their pseudo code on the board.

Part 3: 5 Minutes

Debriefing Cont.

Activity 2: Walk-through	**5 mins.**

Open skill challenge 9-10 (note that this challenge will unlock for students only after they complete challenge #116 in ‘Story mode’).
Ask your students if they can see something different in this challenge compared to other challenges that they solved?
Possible answers:

1. There are 3 goats instead of just one.
2. In this challenge we need to use a “for” loop and a “times” loop.

To solve this challenge we first have to break all the frozen bananas. Only after all the bananas will be regular yellow bananas, the monkey will be able to collect all of them using a for loop.
Tell your students they need to solve this challenge in two parts:

- First, using a “for” loop and a “times” loop, they need to hit the frozen bananas
- Then, using a “for” loop, they can get the bananas (actually, this part is already written in the code - lines 12-13)

Part 3: 5 Minutes

Debriefing Cont.

Activity 2: Walk-through Cont.	**5 mins.**

Let's start with the first part, the code that we have is:

```
x = 0
for g in goats
    3.times ->
        # Complete me!
        x = x + 1
```

Ask your students what is missing here. We need to check if the bananas are frozen and hit them if they are. The first loop we will use is a "for" loop on the goats. Then, we will use a "times" loop so each goat will check if 3 bananas are frozen and hit them if they are. Ask a volunteer to solve this part:

```
x = 0
for g in goats
    3.times ->
        if bananas[x].frozen()
            g.goto bananas[x]
            g.hit()
        x = x + 1
```

Before you run this code, add the # symbol at the beginning of lines 12-13 to make them Comment lines. Click on run. Ignore the hint message and show your class how at the end, all the bananas are unfrozen. Remove the # sign, press "run" to finish the challenge.

Lesson 25 – If All Else Fails

In this lesson we will focus on practicing all that was learned in lessons 23-24.

Objectives

In this lesson, students will:

- Practice using if-else
- Write code that uses if-else from scratch
- Use if-else statements within function definition
- Complete challenges 122-124

Components

U.S. Standards Addressed

CSTA-K12 Computer Science Standards	
• 1B-AP-9 • 1B-AP-10 • 1B-AP-12 • 1B-AP-15	• 2-AP-10 • 2-AP-11 • 2-AP-12 • 2-AP-14 • 2-AP-16 • 2-AP-17

Part 1: 10 Minutes

Introduction

Activity	**10 mins.**

By now you and your students played more than 120 challenges on Coding Adventure! You collected bananas, crossed bridges, made friends with a turtle, crocodiles, rats and a goat! Don't you think it's time you'll learn to share those bananas? Goats love green bananas, but unfortunately in this hard gorilla-takes-bananas-from-monkey world, some bananas are frozen. Even green bananas can be frozen.

Ask your students:

- If I have a yellow banana, what should I do? Answer: catch it
- If I have a frozen yellow banana, what should I do? Answer: send the goat to hit it, and then catch it.
- If I have a green banana, what should I do? Answer: send the goat to eat it
- If I have a frozen green banana, what should I do? Answer: send the goat to hit it and then eat it
- What should I do if I have three bananas, one yellow, one frozen yellow, and one frozen green? Solve this scenario with your students, ask them to describe how they would solve it and write their answer on the board.

Part 1: 10 Minutes

Introduction Cont.

Activity Cont.	**10 mins.**

```
for b in bananas
    if b.frozen()
        goat.goto b
        goat.hit()
    if b.green
        goat.goto b
    else
        goto b
```

Sometimes we have to use more than one condition. In this case, we first make sure the ice is broken using an if statement because we do not want the monkey to get to the frozen banana and not be able to eat it. After adding the if statement, we will add an if-else statement to determine which character should be sent to each banana. Ask your students they think what will be the order of the actions carried out on the stage? Remind your students that a for loop runs each time on a specific object, meaning for every banana the computer runs the test:

- Is the banana frozen?
- Is the banana green?

To determine what will happen. So in actual fact, each banana will have all the right actions performed on it at once (i.e. goat hitting it or not, then either monkey or goat catching it), and then the computer will move on to the next banana.

Part 2: 25 Minutes

Playtime

Log-in	**1 min.**
To review the log-in instructions, please click here.	
Playtime	**7 mins.**
All students should complete challenges 122-124 with at least two stars. (Students from the age of 12 and up should get three stars). Use your classroom dashboard to keep track of students' achievements. Use this time to walk around the class and help students who are struggling.	

Part 2: 25 Minutes

Playtime Cont.

Walk-through (1)	**5 mins.**

Open challenge #123 and reset the code.
Ask your students to describe in their own words the algorithm (sequence of actions) that solves this challenge.
The answer will be something similar to:
Go over all the bananas / for each banana:
    if the banana is green - tell the goat to go to it
    if not then tell the monkey to go to it.
Translate the above into code and run it:

```
for b in bananas
    if b.green()
        goat.goto b
    if not b.green()
        monkey.goto b
```

Note that it only gets one star. Ask your students what can be fixed in order to get 3 stars (look at Gordo's hint!).

Part 2: 25 Minutes

Playtime Cont.

Walk-through (1) Cont.	**5 mins.**

Change line 4 into this version of the code:

```
for b in bananas
    if b.green()
        goat.goto b
    else
        monkey.goto b
```

Run this code to get 3 stars.

Part 2: 25 Minutes

Playtime Cont.

Walk-through (2)	**10 mins.**

Open challenge #124 and reset the code. Read the initial code with your students and pay attention to the comments. Observe that the comments are telling us that the *main* code should be left intact, but we should fill in both of the function definitions so they would do what is expected of them.

Start with line 2 - the definition of breakBanana(f).

Ask: “How do we tell the computer to break the ice if a banana **b** frozen?”

Answer:

```
if b.frozen
    goat.goto b
    goat.hit()
```

Part 2: 25 Minutes

Playtime Cont.

Walk-through (2) Cont.	**10 mins.**

Observe that in the function breakBanana the banana is called f, not b.
So the code in the function definition should read:

```
if f.frozen
    goat.goto f
    goat.hit()
```

Now similarly conclude with your students what the code should be to get the right character to eat banana according to its color: (you can use the solution to challenge #123 as a reference)

```
if e.green()
    goat.goto e
else
    monkey.goto e
```

Part 2: 25 Minutes

Playtime Cont.

Practice	
Encourage students who finish early to open skill mode on the map and complete unlocked challenges. After completing challenges 122-124, skill challenges 10-3 – 10-8 are unlocked.	
Quiz	
After completing challenges 106 – 124, you can assign your class the second quiz – Part 2: If & If-Else. The quiz includes 5 challenges. You can assign quizzes from the Quizzes tab on your teacher dashboard.	

Part 3: 10 Minutes

Debriefing

Explanation	**10 mins.**

Ask your students what is the difference between using two “if” statements rather than an if-else statement?
Answer: We use an if statement when we want something to run only if “something happens”. We use if-else when we want to be in control also of what will happen in case that “something” does not “happen”.
For example:

If statement:

```
if raining
    wear a coat
```

If-else statement:

```
if raining
    wear coat
else
    take sunglasses
```

Part 3: 10 Minutes

Debriefing Cont.

Explanation Cont.	**10 mins.**

In the first scenario, we wouldn't take sunglasses at all, where in the second we will if it isn't raining. Ask your students to add the condition for taking sunglasses as another if statement.

Answer:

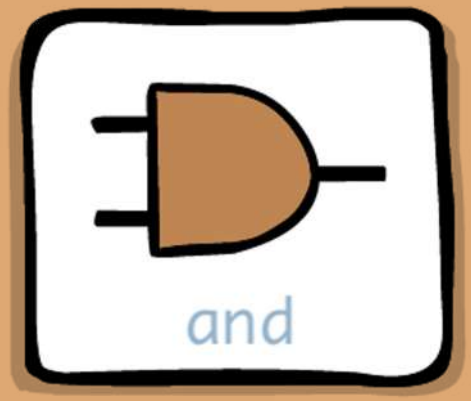
```
if raining
    wear a coat
if not raining
    take sunglasses
```

Ask your students what do they think is the better way to write our conditions?

Using an if-else statement is more convenient and also makes the code readable. However there will be times where using two if statements would be the better way to go, and as programmers we need to know when and why to use a certain method.

Lesson 26 – And In the End

This lesson introduces us to a tool that is very useful when using conditionals: the logical operator. Specifically they will learn about the “and” operator.

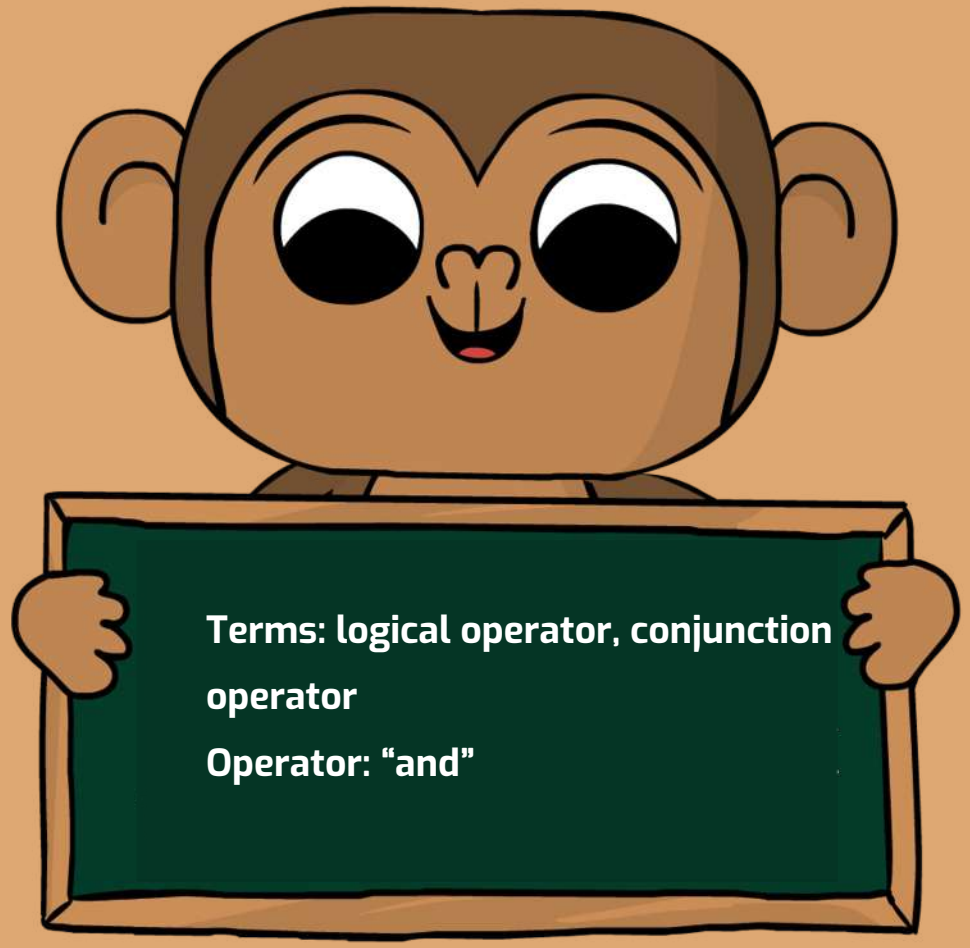


Objectives

In this lesson, students will:

- Use the “and” operator
- Complete challenges 125-130

Components



U.S. Standards Addressed

CSTA-K12 Computer Science Standards	
• 1B-AP-9 • 1B-AP-10 • 1B-AP-12 • 1B-AP-15	• 2-AP-10 • 2-AP-11 • 2-AP-12 • 2-AP-14 • 2-AP-16 • 2-AP-17

Part 1: 15 Minutes

Introduction

Review	**5 mins.**
Start the lesson with a short review of the “until” loop. You can use lesson 20 as a reference. The “until” loop contains a block of code that will repeat *until* “something happens”, or more accurately, until a specific condition is met. This condition is called a *control expression*or *loop condition*. The computer checks the condition at every repetition. If the answer is false, the loop will keep going. It will only stop once the answer is true.	
Activity	**10 mins.**
Today we are going to learn that an until loop can have a loop condition that is made out of two different conditions. That way we can make the computer wait until both are true before it executes the code below. Let’s imagine we are standing in a crosswalk. Cars are driving by and the traffic light for the pedestrians is red. Suddenly there are no more cars, but the light is still red. Would you cross the road? Similarly, if the light is green but a car passes, nobody would imagine crossing the road either. Let’s translate this scenario into pseudo-code: until road.clear **and** light.green wait() cross road	

Part 1: 15 Minutes

Introduction Cont.

Activity Cont.	**10 mins.**

Let's take a deeper look into this, write the following on the board:

road is clear	**light is green**	**road is clear & light is green**
yes	no	
no	yes	
yes	yes	
no	no	

This is a Truth table, it shows the values, relationships, and the results of performing logical operations on logical expressions. Each column represents a condition, and in this case we want to see what happens in the different situations in our scenario.

Part 1: 15 Minutes

Introduction Cont.

Activity Cont.	**10 mins.**

Go over each line and ask your students what should be written in the third column in each line. In the end all the lines will say “no” except for the third one, like so:

road is clear	**light is green**	**road is clear & light is green**
yes	no	no
no	yes	no
yes	yes	yes
no	no	no

Part 1: 15 Minutes

Introduction Cont.

Activity Cont.	**10 mins.**

We can work this test with every condition, even if it is not defined:

A	B	A and B
true	false	false
false	true	false
true	true	true
false	false	false

A and B are called the *logical conjunction*. If A is true and B is true, then the conjunction "A and B" is true; under all other possible assignments of truth values to A and B, the conjunction is false.

Part 2: 25 Minutes

Playtime

Log-in	**1 min.**
To review the log-in instructions, please click here.	
Playtime	**9 mins.**
All students should complete challenges 125-130 with at least two stars. (Students from the age of 12 and up should get three stars). Use the classroom dashboard to keep track of students' achievements. Use this time to walk around the class and help students who are struggling.	

Part 2: 25 Minutes

Playtime Cont.

Walk-through	**5 mins.**

Open challenge #128 and delete the code.
Solve the challenge using the following code:

```
until bear.sleeping()
    wait()
until tiger.sleeping()
    wait()
goto banana
```

Observe that it only gets 1 star.

Ask your students if they know how to get the other 2 stars. If not, guide them to use the and operator. Reset the code to see the initial code that shows an example of how the and operator is used. Solve the challenge together to get 3 stars:

```
until bear.sleeping() and tiger.sleeping()
    wait()
goto banana
```

Tell your students to make sure they remember how the syntax of **and** is written because in the next challenges they will have to use it.

Part 2: 25 Minutes

Playtime Cont.

Playtime	**10 mins.**
Students continue their work on challenges 125-130.	
Practice	
Encourage students who finish early to open skill mode on the map and complete unlocked challenges. No new skill challenges are unlocked after completing challenges 125-130.	

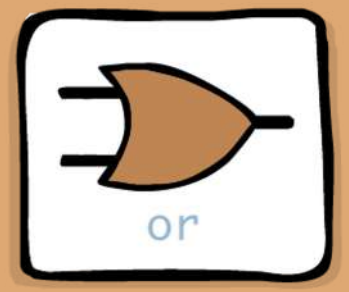
Part 3: 5 Minutes

Debriefing

Explanation	**5 mins.**
Let's define the concept of an *operator*. We have shown conjunctions of two conditions in the tables earlier. What makes up the conjunction A and B? Answer: The two conditions and the word and. The word and, in this context, is called an *operator*. It performs a certain operation: generating the result value (yes or no) according to the two conditions. Generally, an operator is a programming construct that performs an operation, similarly to a function, but differs from functions in the way that it is written. We have seen operators before without calling them by name, such as the assignment operator (=) and the field access operator (.). There are many kinds of operators, and operators are classified in several ways. More specifically, the and operator is a *logical operator*: it performs a logical operation, i.e. an operation that has to do with the laws of valid reasoning. It is also called the logical conjunction operator.	

Lesson 27 – Take it or Leave it!

In this lesson we will continue doing logic. This time we will learn about the “or” operator, clearly the natural next step after “and”.



Objectives

In this lesson, students will:

- Use the “or” operator
- Complete challenges 131-135

Components

U.S. Standards Addressed

CSTA-K12 Computer Science Standards	
• 1B-AP-9 • 1B-AP-10 • 1B-AP-12 • 1B-AP-15	• 2-AP-10 • 2-AP-11 • 2-AP-12 • 2-AP-14 • 2-AP-16 • 2-AP-17

Part 1: 10 Minutes

Introduction

Activity	**8 mins.**

Ask for a volunteer. Have them walk around the classroom and ask each student the following two questions:

- Is your shirt red?
- Are you wearing jeans?

If a student answers “yes” to at least one of the questions, they should go stand on the right side of the class. If both answers are “no”, then they should stay where they are.

All we need is for one of the answers to be yes in order to get the student to the right side of the class.

Let’s write it in pseudo code:

```
for s in students
    if shirt.red or wearing.jeans
        s.stand right
```

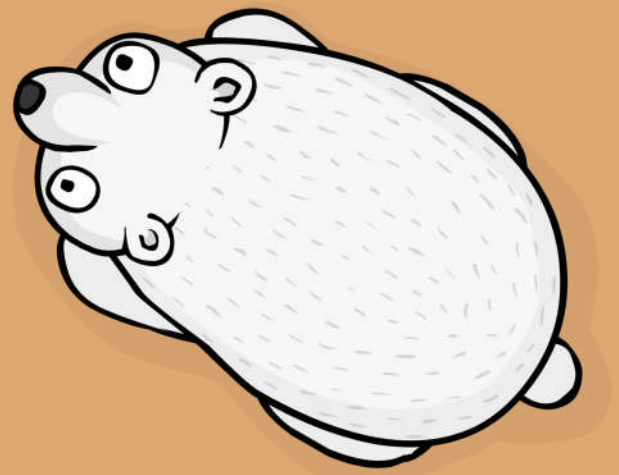
Ask your students how is this different from using the and operator.

If we used and then in order for a student to move to right side, he or she will need to wear both a red shirt and jeans, and what are the odds of that? By using or more kids can move to the right side, because they only have to answer one of the conditions.

Part 1: 10 Minutes

Introduction Cont.

Explanation	**2 mins.**
In Coding Adventure, using the or operator is very simple. Our tigers and bears have another trick up their sleeve, they can also play! Now we can wait for them to sleep or play, making our wait much shorter.	



Part 2: 25 Minutes

Playtime

Log-in	**1 min.**
To review the log-in instructions, please click here.	
Playtime	**7 mins.**
All students should complete challenges 131-135 with at least two stars. (Students from the age of 12 and up should get three stars). Use the classroom dashboard to keep track of students' achievements. Use this time to walk around the class and help students who are struggling.	

Part 2: 25 Minutes

Playtime Cont.

Walk-through	**10 mins.**

Open challenge #133 and reset the code. Run the initial code and see what happens.
Solve the challenge together with your students by following the hints in the comment lines.
The 3-star solution will look like this:

```
for b in bananas
    until tiger.sleeping() and bear.sleeping()
        wait()
    if b.frozen()
        goat.goto b
        goat.hit()
    until tiger.sleeping() and bear.sleeping()
        wait()
    goto b
```

Part 2: 25 Minutes

Playtime Cont.

Walk-through Cont.	**10 mins.**

Open challenge #134 and reset the code.
Show your students how the or operator works.
Ask them why we have to wait until the tiger sleeps **or** plays? Isn't it easier to just wait until the tiger sleeps like we did before? In order to answer that question, try running the following code:

```
until tiger.sleeping()
    wait()
goto banana
```

Show by running the code that it might turn into an infinite loop since the tiger always plays and never goes to sleep. We want to wait for the first opportunity to move safely, but we might not know what will happen first - will the tiger sleep or play? Conclude that if we want to wait for one of the two conditions to happen, but we don't know which will happen, the simplest way to get around that is by using the or operator.

Part 2: 25 Minutes

Playtime Cont.

Playtime	**7 mins.**
Students continue their work on challenges 131-135.	
Practice	
Encourage students who finish early to open skill mode on the map and complete unlocked challenges. After completing challenges 131-135, skill challenges 11-1 – 11-5 are unlocked.	

Part 3: 10 Minutes

Debriefing

Explanation	**10 mins.**

Let's put the or condition into our truth table from the previous lesson. Write the following on the board:

A	B	A and B	A or B
true	false	false	
false	true	false	
true	true	true	
false	false	false	

Part 3: 10 Minutes

Debriefing Cont.

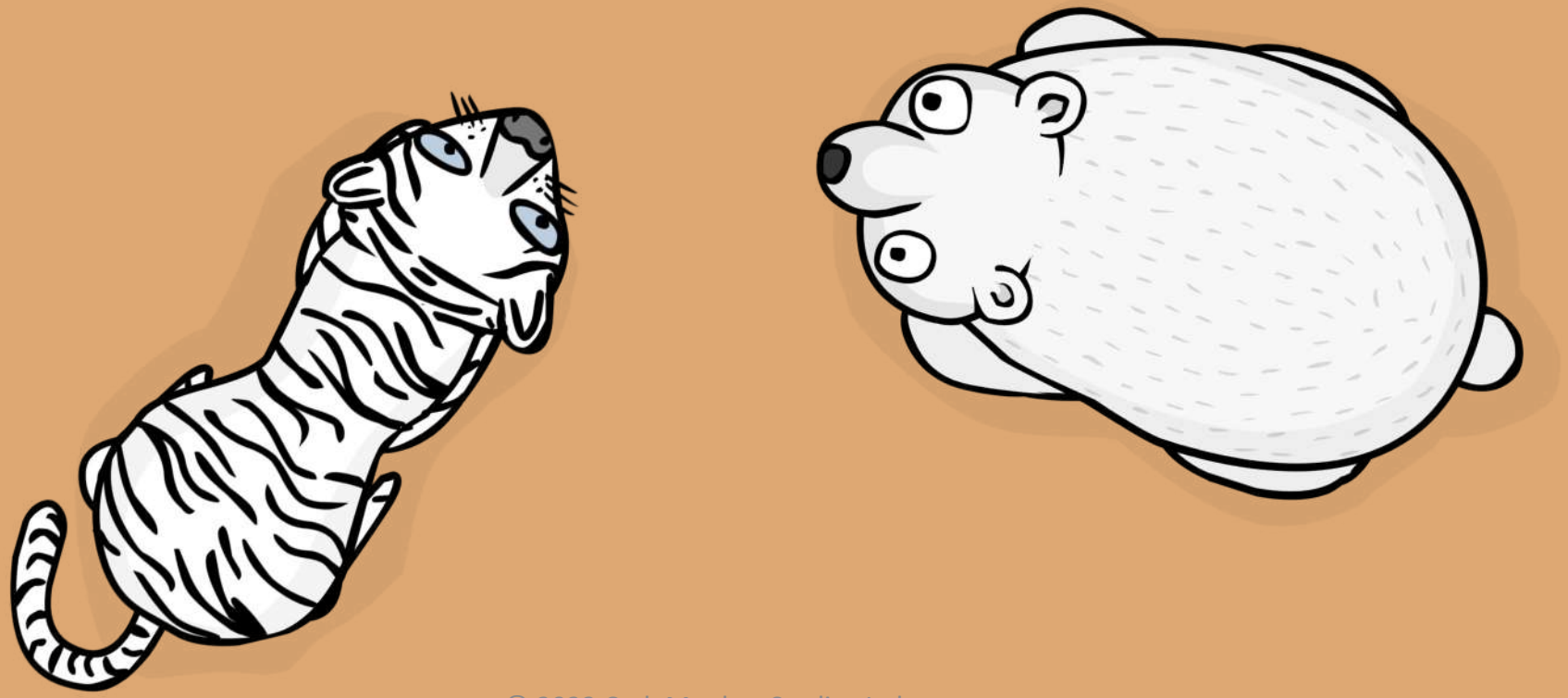
Explanation Cont.	**10 mins.**

Now work with your students to complete the table. Make sure to point out the difference between the results of “and” and “or”. In “and”, only if both conditions are true then the result will be true. Whereas in “or”, each line that has at least one “true” in it is enough to have a true result.

A	B	A and B	A or B
true	false	false	true
false	true	false	true
true	true	true	true
false	false	false	false

Lesson 28 – Mix and Match

In this lesson we will practice our conditionals and logic. We will also learn a new way to use arrays, and the principle of polymorphism.

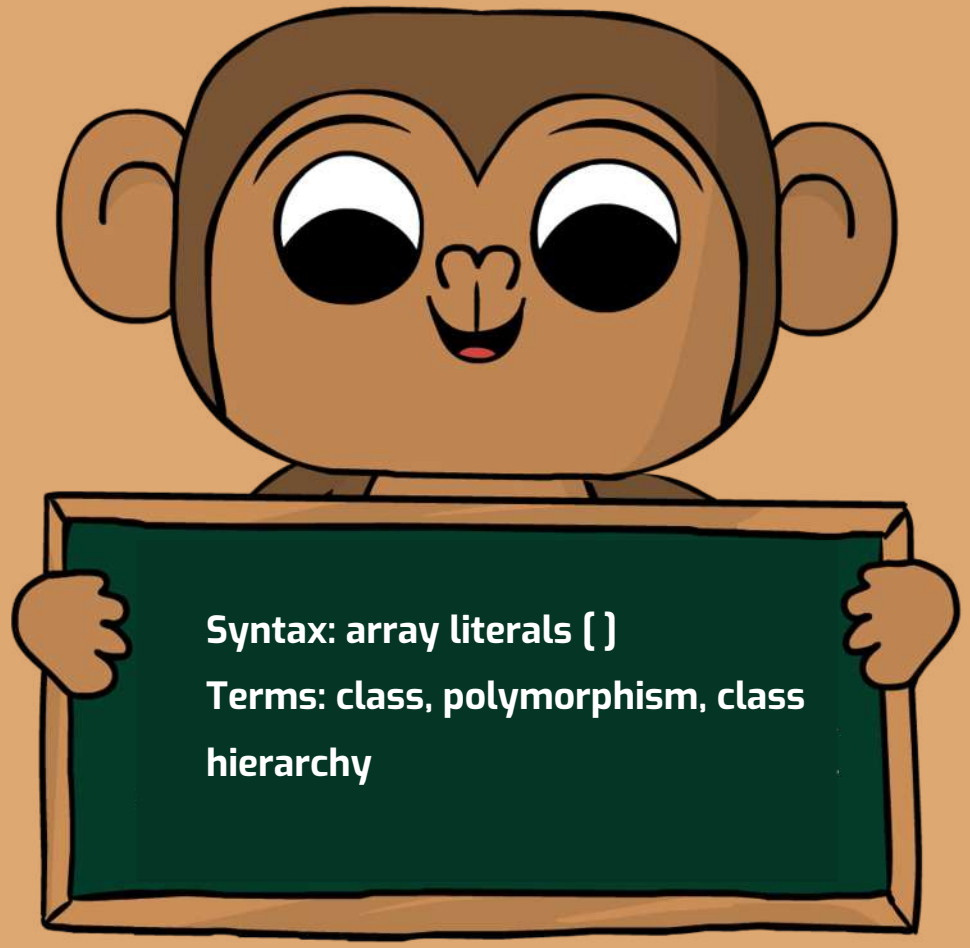


Objectives

In this lesson, students will:

- Practice logical operators
- Learn how to define new arrays
- Work with arrays that contain objects of different types
- Complete challenges 136-140

Components



U.S. Standards Addressed

CSTA-K12 Computer Science Standards	
• 1B-AP-9 • 1B-AP-10 • 1B-AP-12 • 1B-AP-15	• 2-AP-10 • 2-AP-11 • 2-AP-12 • 2-AP-14 • 2-AP-16 • 2-AP-17

Part 1: 15 Minutes

Introduction

Explanation	**7 mins.**

The objects we are programming have different characteristics, for example: Both the monkey and turtle can “step”. However, the monkey can “say” but the turtle cannot. We need to pay attention to that and other differences when we program.
Recall that all the characters and other items on the screen are called objects. Explain that every object belongs to a *class* or *type* that defines what it can and cannot do. For example we can have several crocodiles on the screen. We say that they all belong to the class Crocodile. Crocodiles cannot step, but they can turnTo, just like monkeys and turtles. Sometimes it is useful to use an action that is common to different types of objects, even when the type is unknown. For example, we can write:

```
animal.turnTo banana
```

Without knowing whether animal is a crocodile, monkey or turtle. But as long as the animal is of one of these 3 types, it will know how to turnTo.

Part 1: 15 Minutes

Introduction Cont.

Activity	**8 mins.**
Test your students' understanding of the "and" and "or" conditions. Ask for four volunteers. One will be condition A, the second will be condition B, the third will be the "and" operator, and the fourth will play the "or" operator. Instruct them that on your mark, A and B will have to sit or stand, each can decide on their own what to do. Standing means "true", and sitting means "false". Then, "and" and "or" should sit or stand according to the result. So for example if A is standing (true), B is sitting (false), then "and" should be sitting (false), and "or" should be standing (true). Let them try this a few times to check their understanding.	

Part 2: 25 Minutes

Playtime

Log-in	**1 min.**
To review the log-in instructions, please click here.	
Playtime	**7 mins.**
All students should complete challenges 136-140 with at least two stars. (Students from the age of 12 and up should get three stars). Use the classroom dashboard to keep track of students' achievements. Use this time to walk around the class and help students who are struggling.	

Part 2: 25 Minutes

Playtime Cont.

Walk-through	**10 mins.**

Open challenge #140 and reset the code. Read the initial code carefully together with your students. Start from the beginning of the main code at line 6.

When reading line 7, explain the following:

Writing

```
for stepper in [monkey, goat]
```

is similar to writing

```
for b in bananas
```

In both cases, the object after “in” is an array: bananas is the array of all bananas on screen, and [monkey, goat] is an array of two elements: the monkey and the goat. Both of these animals can step, so it is appropriate to name the loop variable stepper.

Part 2: 25 Minutes

Playtime Cont.

Walk-through Cont.	**10 mins.**

The square brackets [ ] notation is called *array literal* and it is one of the ways to define new arrays.
We can write all sorts of arrays using array literals. Experiment with this a little: delete the code and write the following code. Run it and see what happens.

```
for stepper in [monkey, goat, monkey, goat]
    stepper.turn 180
```

Now try this (ask your students to guess what will happen before you run):

```
for stepper in [monkey, goat, monkey, goat]
    for degrees in [left, right, 180, -180]
        stepper.turn degrees
```

Reset the code again, read lines 6-9 with your students. Ask them to describe what will happen once the code is executed.
Complete the waitFor(t) function with your students, and run the solution to complete the challenge.

Part 2: 25 Minutes

Playtime Cont.

Playtime	**7 mins.**
Students continue their work on challenges 136-140.	
Practice	
Encourage students who finish early to open skill mode on the map and complete unlocked challenges. After completing challenges 136-140, skill challenges 11-6 – 11-16 are unlocked.	

Part 3: 5 Minutes

Debriefing

Explanation	**10 mins.**

Let's talk about the statement stepper.turn which we used earlier. When it is executed, stepper will sometimes be a monkey or goat. In programming lingo, we say stepper will be an object of different types, or of different *classes*.
A class is simply a type of an object. As we learned in previous lessons, different classes can perform different actions. For example, only a goat can hit(), but both goats and monkeys can turn 180. Note that while objects from different classes can perform actions called turn, they might be doing slightly (or entirely) different actions.
The ability to define turn in different ways for different classes is called *polymorphism*. Polymorphism is an important part of object oriented programming languages.

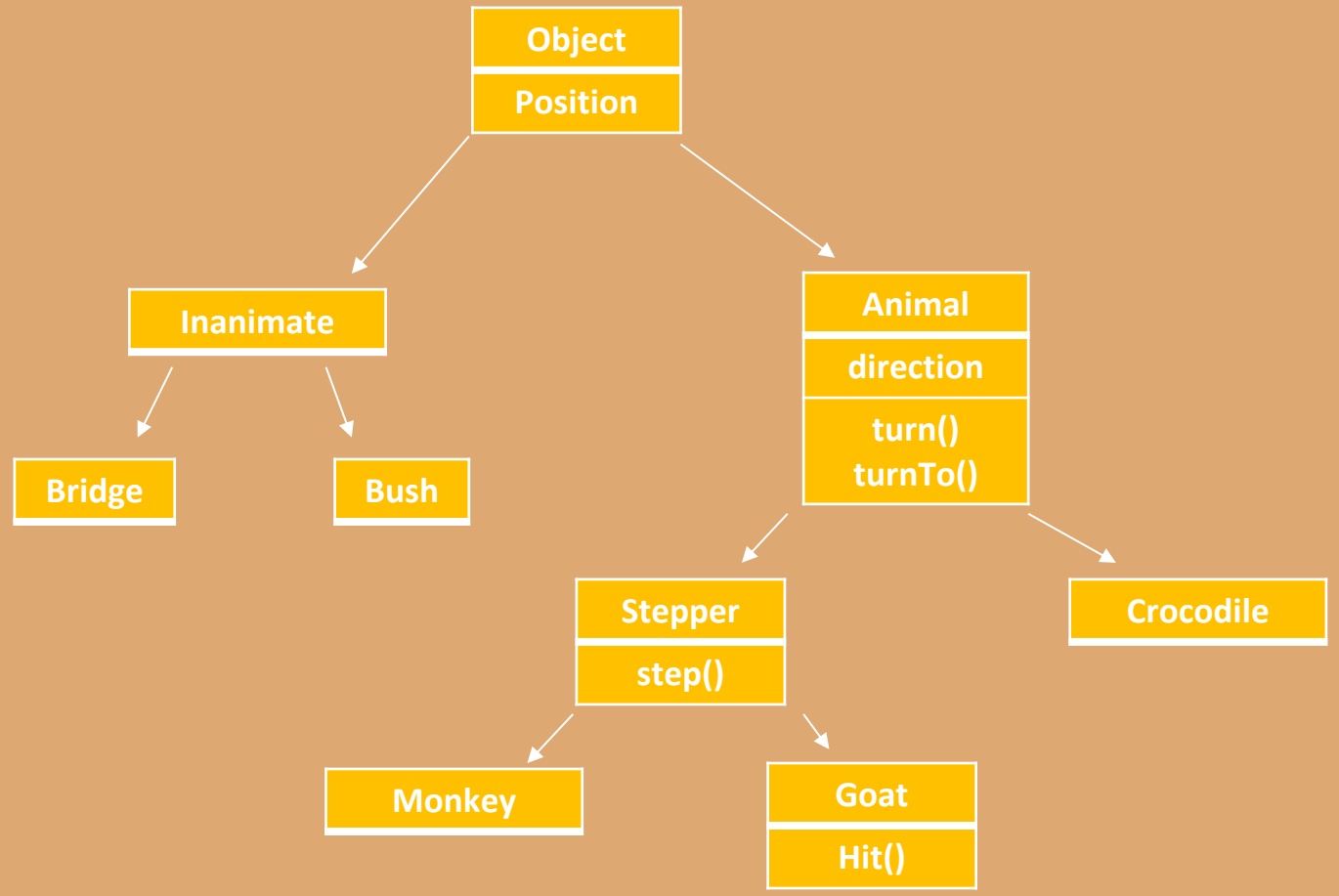
Part 3: 5 Minutes

Debriefing Cont.

Explanation Cont.	**5 mins.**

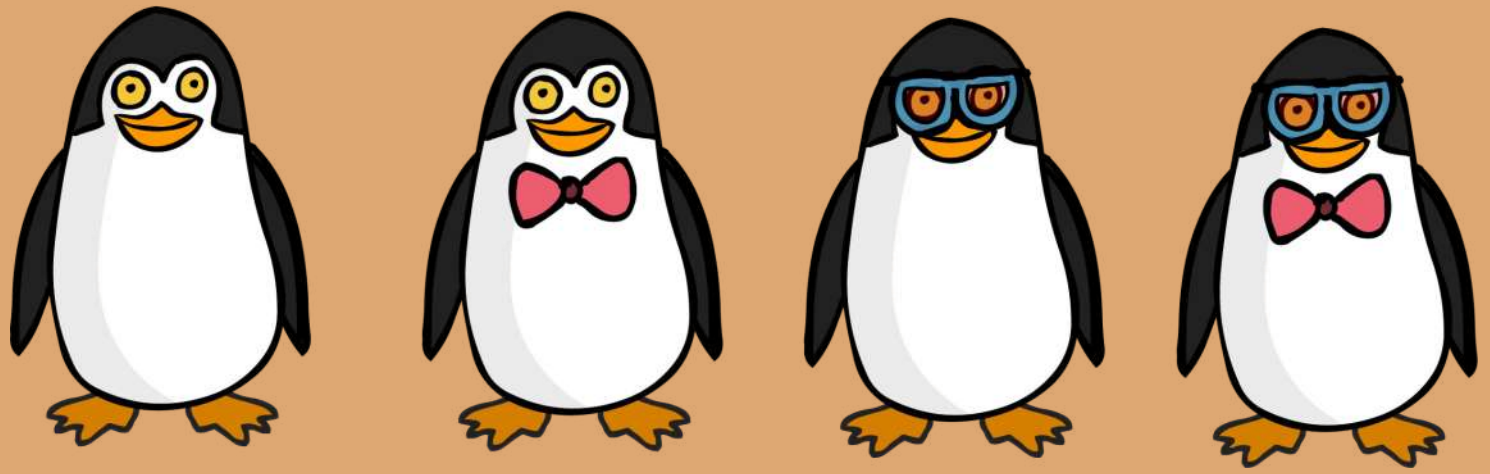
When computer programs involve many different classes of objects, programmers often come up with class diagrams that describe the relationships between classes, and the different features of each class. Show the following diagram to your students as an example.

Explain that every class can do the actions listed in its box, but also the ones listed in the boxes above it that point to it, directly or indirectly. For example, goats can hit(), but also step() and turn().



Lesson 29 – Fashion Alert!

In this lesson we will continue practicing logic operators. We will practice “and” and “or” operators. In this lesson your students will meet a new character - the penguins. The students will use “and” and “or” to move the penguins to form a bridge for the monkey.



Objectives

In this lesson, students will:

- Practice logical operators
- Complete challenges 141-145

Components

U.S. Standards Addressed

CSTA-K12 Computer Science Standards	
• 1B-AP-9 • 1B-AP-10 • 1B-AP-12 • 1B-AP-15	• 2-AP-10 • 2-AP-11 • 2-AP-12 • 2-AP-14 • 2-AP-16 • 2-AP-17

Part 1: 5 Minutes

Introduction

Explanation	**5 mins.**

In the previous lessons the students used “and” and “or” with an “until” loop.
In this lesson, the students will use the “and”/”or” with an “if” statement. These challenges includes a new character - the penguin.
The penguin can have:

- Glasses
- Bow tie
- Both
- None

Part 1: 5 Minutes

Introduction Cont.

Explanation Cont.	**5 mins.**

The students will use two new functions:

- hasGlasses()
  - returns yes - if the penguin is wearing glasses
  - returns no - otherwise
- hasBowTie()
  - returns yes - if the penguin is wearing a bow tie
  - returns no - otherwise

The purpose of the penguins is to form a bridge so that the monkey can get to the banana. Of course, not all the penguins needs to move! If all will move, then there will be holes in the bridge.

In each of the challenges solved in this lesson, the students will need to decide which of the penguins should move.

Those with:

- glasses **and** a bow tie
- glasses **or** a bow tie

Part 2: 30 Minutes

Playtime

Log-in	**1 min.**
To review the log-in instructions, please click here.	
Playtime	**4 mins.**
All students should complete challenges 141-145 with at least two stars. (Students from the age of 12 and up should get three stars). Use the progress tab in the classroom dashboard to keep track of student achievements. Use this time to walk around the class and help students who are struggling.	

Part 2: 30 Minutes

Playtime Cont.

Walk-through	**10 mins.**

Open challenge #141.
This challenge introduces the penguins and the functions “hasGlasses” and “hasBowTie”. In this challenge, the students do not need to use “and” or ”or”.
Remind the students:

- hasGlasses will return yes if the penguin has glasses and no otherwise
- hasBowTie will return yes if the penguin has a bow tie and no otherwise

Before running the code ask your students which penguins wear what?

- glasses:
  - penguins[0]
  - penguins[4]
- bow tie:
  - penguins[1]
  - penguins[3]

Part 2: 30 Minutes

Playtime Cont.

Walk-through Cont.	**10 mins.**

Ask your students what will happen when you will press run?

1. the monkey will say ‘yes’; penguins[0] will move 15 steps; the monkey will say ‘no’;
2. the monkey will say ‘no’; the monkey will say ‘yes’; penguins[1] will move 10 steps;
3. the monkey will say ‘no’; the monkey will say no’; penguins[2] will not move;
4. the monkey will say ‘no’; the monkey will say ‘yes’; penguins[3] will move 10 steps;
5. the monkey will say ‘yes’; penguins[4] will move 15 steps; the monkey will say ‘no’;

You can write the above on the whiteboard and then when you solve the challenge, show the students each step.
Add the following at the end of the code:

```
goto banana
```

Press run and solve the challenge.

Part 2: 30 Minutes

Playtime Cont.

Walk-through Cont.	**10 mins.**

Ask your students if there is a penguin that will answer ‘yes’ to both questions (has glasses and has a bow tie)?

- In this challenge, no

Now ask your students which penguin will move if you use “and”?

Change the code to:

```
for p in penguins
    if p.hasGlasses() and p.hasBowTie()
        p.step 15
goto banana
```

They should answer that no penguin will move and the monkey will fall into the water. Run the challenge and show that no penguins has moved.

Part 2: 30 Minutes

Playtime Cont.

Walk-through Cont.	**10 mins.**

Now, ask your students:

- How many penguins will answer ‘yes’ to either questions (has glasses or has a bow tie)?
  - In this challenge, four
- Which penguin will move if you use “or”?
  - In this challenge, all penguins except penguins[2]

Change the code to “or” (instead of “and”):

```
for p in penguinss
    if p.hasGlasses() or p.hasBowTie()
        p.step 15
goto banana
```

Before running the code, ask the students if this will solve the challenge?

They should answer, no, because the penguins with the glasses and those with the bow tie are not aligned. So, even when they move one or the other it will not solve the challenge.

Run the challenge and show that although the four penguins moved, they still did not create a bridge.

Part 2: 30 Minutes

Playtime Cont.

Playtime	**15 mins.**
Students continue their work on challenges 141-145.	
Practice	
Encourage students who finish early to open skill mode on the map and complete unlocked challenges. After completing challenges 141-145, skill challenges 11-17 – 11-21 are unlocked.	
Quiz	
After completing challenges 125 – 145, you can assign your class the third quiz – Part 2: And & Or. The quiz includes 4 challenges. You can assign quizzes from the Quizzes tab on your teacher dashboard.	

Part 3: 10 Minutes

Debriefing

Explanation	**10 mins.**

Ask your students the following: When is an expression of two conditions (A, B) with a Boolean operator true?

- If the operator is “and”
  - only when both A and B are true
- If operator is “or”
  - when A is true, when B is true

You can also use the truth table below:

A	B	A and B	A or B
true	true	true	true
false	true	false	true
true	false	false	true
false	false	false	false

Part 3: 10 Minutes

Debriefing Cont.

Explanation Cont.	**10 mins.**

Show your students that the “or” operator includes the “and” operator, since when both conditions are true, both the “and” and “or” operators are true. Open skill challenge 11-21.
There are three bananas in this challenge, the monkey needs to cross the lake twice. This means that two bridges are.
Some penguins will need to move twice.
Ask your students to look at the penguins. Before they decide whether they should use “and” or “or”, tell them to first say how many penguins need to move each time.

1. First, three penguins
2. Then, six penguins

Now, turn off the screen so the students will not see the challenge.
Ask them, according to the number of penguins that need to move each time:

1. Will the same operator be used both times?
    - No, because if yes, than the same number of penguins would have moved
2. Which operator will be used first (to move three penguins)
    - “and”, since it is true in less cases than the “or” operator
3. Which operator will be used the second times (to move six penguins)
    - “or”, since it is true when the “and” operator is true and in more cases as well.

Part 3: 10 Minutes

Debriefing

Explanation Cont.	**10 mins.**

Now, turn the screen back on and solve the challenge. Ask your students how to solve the challenge.

- First, loop on all penguins and move those with glasses **and** a bow tie
- Get the first two bananas
- Then, loop on all penguins and move those with glasses **or** a bow tie backwards
- Get the last banana

```
for p in penguins
    if p.hasGlasses() and p.hasBowTie()
        p.step 10
goto bananas[0]
goto bananas[1]
for p in penguins
    if p.hasBowTie() or p.hasGlasses()
        p.step -15
goto bananas[2]
```

Lesson 30 – Loops are Fun

Now that your students are acquainted with the Challenge Builder, they can start building challenges that focus on specific topics. Today your students will create challenges to demonstrate the different types of loops. This lesson can be modified to any topic learned in Coding Adventure, just change the instructions and the first discussion to fit the new topic.

Objectives

In this lesson, students will:

- Review what they learned about loops
- Create their own Coding Adventure challenges about loops
- Solve challenges created by their classmates

U.S. Standards Addressed

CSTA-K12 Computer Science Standards	
• 1B-AP-9 • 1B-AP-10 • 1B-AP-12 • 1B-AP-15	• 2-AP-10 • 2-AP-11 • 2-AP-12 • 2-AP-14 • 2-AP-16 • 2-AP-17

Part 1: 15 Minutes

Introduction

Discussion	**10 mins.**
Ask your students “What are the different kinds of loops we learned about?” Depending how far your students advanced in CodeMonkey, possible answers are: Times loop, For loop, until loop. Ask for three students volunteers, and ask each of them to explain one of the three loops: • **Simple loop (times loop)** - a sequence of instructions that repeats a specified number of times. • **For loop** - used when we have a collection of objects and we want to repeat an action that relates to each one of them specifically. • **Until loop** - contains a block of code that will continue to run until a specific condition is met. Open challenges number 30, 65 and 96 to demonstrate these topics.	
Walkthrough	**5 mins.**
In today’s lesson your students need to create three challenges, one for each loop. encourage them to be creative and try to create unique challenges.	

Part 2: 20 Minutes

Playtime

Log-in	**1 min.**
To review the log-in instructions, please click here.	
Building time!	**19 mins.**
Students build challenges about loops using the Challenge Builder. Use this time to walk around and assist students who are struggling.	
Inspiration	
Here are some tips for creating challenges using loops. Of course, these are just examples and your students can be creative and think of any idea that requires a loop. **Times loop** - Organize the bananas in a geometric form (square, triangle, circle) or in a repeating pattern. **For loop** - Scatter the bananas on the screen, not in a specific pattern. Use other objects as obstacles. Another option is to use several crocodiles. **Until loop** - Let the cat guard the banana	

Part 3: 10 Minutes

Practice

Discussion	**10 mins.**
Students solve challenges created by their friends to practice loops.	
Optional	
Show a challenge created by a student in your classroom and try to solve it with everyone.	

Lesson 31 – What's the Conditional?

Today your students will create challenges to demonstrate the different conditionals. This lesson can be modified to any topic learned in Coding Adventure, just change the instructions and the first discussion to fit the new topic.

Objectives

In this lesson, students will:

- Review what they learned about conditionals
- Create their own Coding Adventure challenges about conditionals
- Solve challenges created by their classmates

U.S. Standards Addressed

CSTA-K12 Computer Science Standards	
• 1B-AP-9 • 1B-AP-10 • 1B-AP-12 • 1B-AP-15	• 2-AP-10 • 2-AP-11 • 2-AP-12 • 2-AP-14 • 2-AP-16 • 2-AP-17

Part 1: 15 Minutes

Introduction

Discussion	**10 mins.**

Ask your students "What are the different kinds of conditionals we learned about?"
Possible answers are: if, if/else, until loop.
Ask your students "When should we use conditionals?"
Possible answer is: when we want our program to make a decision and execute different lines of code based on the decision.
Recall with your students when we will use the different conditionals:

- **If statement** - when a block of code should be executed only if a condition is met.
  - For example, the goat hits a banana only if it is frozen.
- **If/else statements** - when two different blocks of code should be executed, one is executed if a condition is met and the other is executed if the condition is not met.
  - For example, the goat goes if the banana is green, otherwise the monkey goes.
- **Until loop** - when a block of code should be executed until a condition is met.
  - For example, the rat needs to move until it is near the match.

Part 1: 15 Minutes

Introduction Cont.

Discussion Cont.	**10 mins.**
Open challenges 114, 121 and 94 to demonstrate these topics. Ask your students: "what if we want two conditions to be true before executing a block of code?" Possible answer: we should use the operator "and". When we use "and" in an if statement or until loop, the code will be executed only when both conditions are met. Ask your students: "what if we want at least one out of two conditions to be true before executing a block of code?" Possible answer: we should use the operator "or". When we use "or" in an if statement or until loop, the code will be executed when at least one of the conditions is met.	
Walkthrough	**5 mins.**
In today's lesson your students need to create three challenges, one for each conditional. Encourage them to be creative and try to create unique challenges.	

Part 2: 20 Minutes

Playtime

Log-in	**1 min.**
To review the log-in instructions, please click here.	
Building time!	**19 mins.**
Students build challenges about loops using the Challenge Builder. Use this time to walk around and assist students who are struggling.	

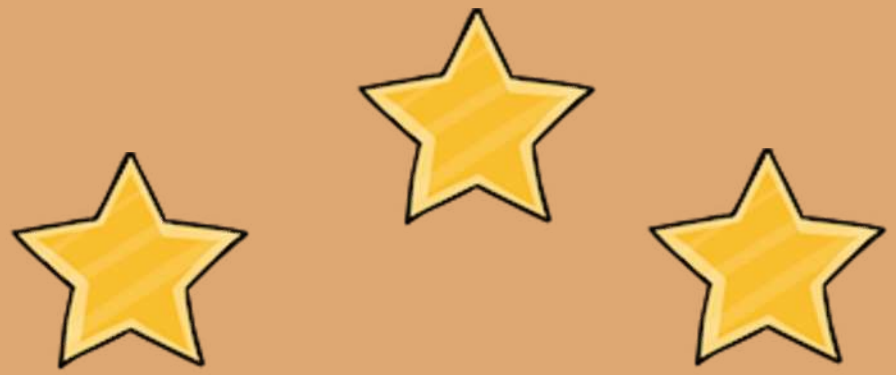
Part 3: 10 Minutes

Practice

Discussion	**10 mins.**
Students solve challenges created by their friends to practice loops.	
Optional	
Show a challenge created by a student in your classroom and try to solve it with everyone.	

Lesson 32 – Functions & Conditions Stars Party!

In this lesson, your students will revisit challenges they have already solved but received only one or two stars for their solution. By the end of this lesson, all of your students should have perfect three-star scores on every challenge in Coding Adventure.



Objectives

In this lesson, students will:

- Revisit challenges in which they received one or two stars
- Solve all challenges with three stars

U.S. Standards Addressed

CSTA-K12 Computer Science Standards	
• 1B-AP-9 • 1B-AP-10 • 1B-AP-12 • 1B-AP-15	• 2-AP-10 • 2-AP-11 • 2-AP-12 • 2-AP-14 • 2-AP-16 • 2-AP-17

Part 1: 15 Minutes

Introduction

Prior to class: Check your classroom dashboard for challenges where a high number of students struggled to get three stars.

Walkthrough	**15 mins.**
Choose two or three challenges that got a relatively high number of blue or red stars, and solve them together in class with your students.	

Part 2: 25 Minutes

Playtime

Log-in	**1 min.**
To review the log-in instructions, please click here.	
Playtime	**24 mins.**
Once they are logged in, instruct them to click the map (upper-right corner) and find challenges where they got one or two stars. At the end of class, all students should aim at having three stars in all the first 145 CodeMonkey challenges. Students who completed all the first 145 challenges with 3 stars should proceed to complete unlocked skill challenges, or get 3 stars in skill challenges they already solved. If any of your students finished all 145 challenges and all skill challenges with 3 stars, ask them to help their fellow classmates.	

Quizzes

Quizzes	
There are three quizzes in this part: 1. Part 2: Functions & Until Loops – 5 challenges 2. Part 2: If & IF-Else – 5 challenges 3. Part 2: And & Or – 4 challenges You can assign quizzes to your class from the Quizzes tab on your teacher dashboard.	

Part 3: 5 Minutes

Debriefing

Explanation	**5 mins.**

Discuss briefly with your students the importance of writing short code.
In Coding Adventure, when we get two stars, it means there is a shorter way to reach the same end result. Either we have lines of code that are unnecessary for reaching the end result, or there is a shorter solution, like using a loop.
Imagine that every time you wanted to go to your next class, you had to first go home and then come back and go to the classroom. It doesn't make any sense to do that. Writing long code is the same. If there is a shorter way to do the same thing, it doesn't make any sense to do it the long way.

Workshop – Escape the Maze

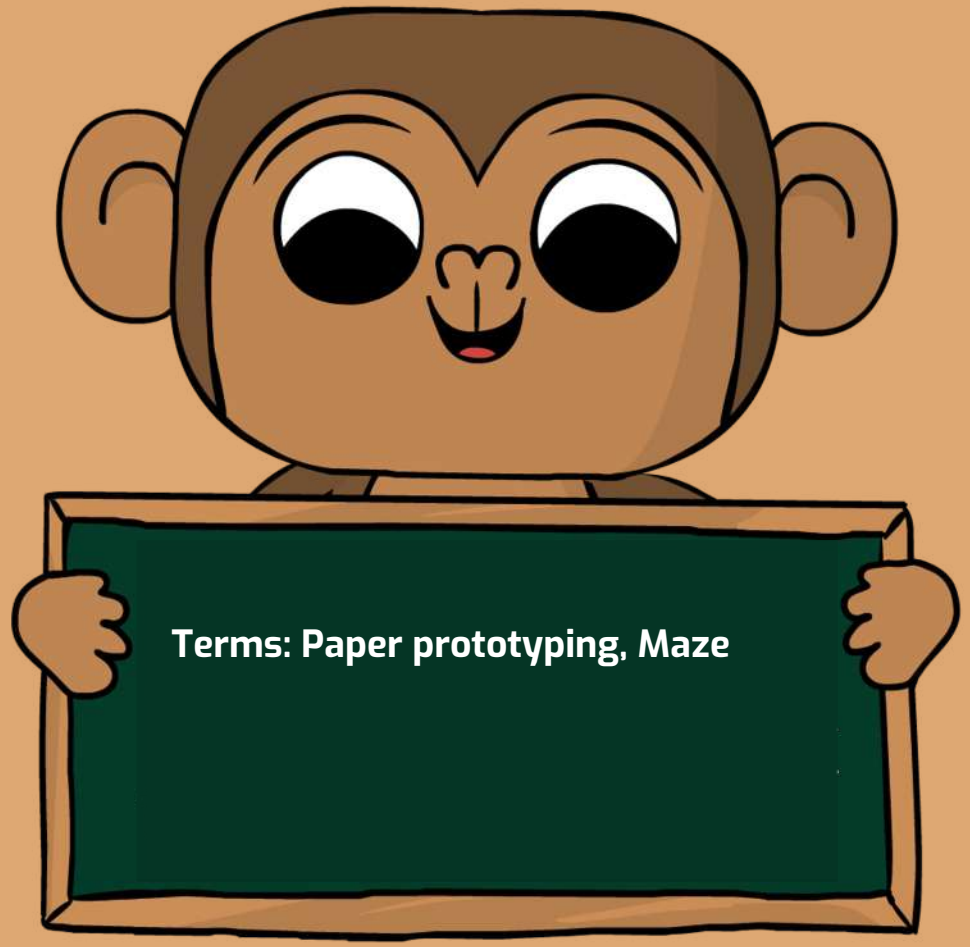
Today your students will create a maze in the Challenge Builder. Creating the maze is a relatively simple task, but making sure it's solvable is harder.

Objectives

In this lesson, students will:

- Learn about paper prototyping
- Create their own maze challenge
- Solve challenges created by their classmates

Components



Part 1: 20 Minutes

Introduction

Explanation	**5 mins.**
Make sure all of your students have a blank page that they can sketch on. Tell your students that today they are going to build a maze in Challenge Builder, but before they can start designing on the computer, we want them to sketch on paper what their maze is going to look like - Including where the monkey will be, where the bananas will be, placement of bushes etc. This technique is called paper prototyping. Before we start designing something on the computer, we can sketch it out to make sure that the people who will use our project understand how to use it. In this case, your students will need to make sure that the maze they build can indeed be solved.	

Part 1: 20 Minutes

Introduction Cont.

Paper Prototyping	**10 mins.**
Each student will create a paper prototype of one challenge representing a maze with the following components: 1. One monkey 2. One or more bananas 3. Bushes and other obstacles that make up the maze	
Paper Prototyping – Switch!	**5 mins.**
Ask students to switch their sketches and let a friend solve what they created. Of course, the solution will be on paper as well because the challenge does not exist yet on the computer.	

Part 2: 20 Minutes

Playtime

Log-in	**1 min.**
To review the log-in instructions, please click here.	
Building Time!	**19 mins.**
Students will build the maze they sketched in the Challenge Builder. Use this time to walk around and assist students who are struggling. Note that the sketch they made probably will not translate exactly the same to the builder. If that happens, tell them to scale down their maze so it can fit on the stage. Encourage students who finish early to try and solve others' challenges.	

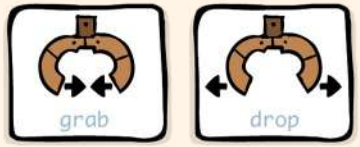
Part 3: 5 Minutes

Debriefing

Discussion	**5 mins.**
• Was it easy or hard to copy the design to the computer? • Was it helpful to plan ahead on paper? • Did you manage to copy it exactly like you planned? • Did you manage to solve your own challenge? • Did you solve other students' challenges?	
Optional	
Show a challenge created by a student in your classroom and try to solve it with everyone.	

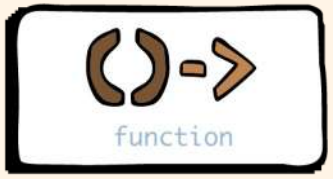
Reference Card

Please note: To see descriptions of previous functions and code constructs please refer to the previous lesson plans file covering challenges 1-75.

Keyword/Button	Description
	“grab()” and “drop()” are functions without an argument that are used in part three to help the rat collect matches. Functions without an argument perform some action, but do not require us to *pass* any input. Pressing the **grab** or **drop** buttons will write the words “grab()” or “drop()” in your code, respectively.
`# This is a comment`	This is a comment line. It is marked in the code with the pound symbol (#) at the beginning. The computer does not treat this line as an instruction. Rather, it is used by the programmers who write and read the code in order to understand each other.

Reference Card Cont.

Keyword/Button	Description

	A function is a set of instructions that performs a specific task. The computer will only execute the function when we call it, meaning use it in our code.
	This is an example of how we define a function:
	`goto = (p) ->`
	`....turnTo p`
	`....step distanceTo p`

	Once we define a function, a new empty button will appear with the name of the function.
	When we want to call the function in the code, we pair the name of the function with the name of the object we want the actions inside of the function to be performed on:
	`goto match`
	`grab()`
	`goto pile`
	`drop()`
	Pressing the function button will write the following text in your code. Note the comment line:
	function_name = (argument) ->
	....# your code here

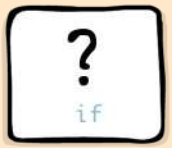
Reference Card Cont.

Keyword/Button	Description
 Example: until near match step 1	The “until” loop contains a block of code that will continue to run *until* a specific condition is met; this condition is called a loop condition. The computer checks the condition at the beginning. If the answer is false, the loop will keep going. It will only stop once the answer is true. Pressing the **until** button will write the following text in your code. Note the comment line: until condition ....# your code here
	We can use the function “near” with a loop condition. The value returned by the function “near” will determine when the “until” loop will stop executing. Pressing the **near** button will write the word “near” in your code.
 	In part three, we are introduced to the cat. The cat will attack the rat if he sees him, so we have to wait for him to fall asleep. “sleeping” is another function we can use with a loop condition. “wait()” is a function without an argument. When using “sleeping”, we need to write: until cat.sleeping() wait()

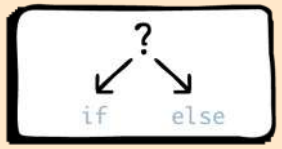
Reference Card Cont.

Keyword/Button	Description
	The goto function makes the monkey “turnTo” an object and “step distanceTo” that certain object. Using the function “goto” will make your code shorter and more readable. Use it with any object on the screen as an argument: banana (“goto banana”), bridge (“goto bridge”) etc. Pressing the **goto** button will write the word “goto” in your code.
	In challenge 106 we are introduced to the goat who will help us break the ice around frozen bananas. The function “hit()” will tell the goat to hit the frozen banana so the monkey can get it, but be careful not to use it on un-frozen bananas. To use “hit()” we have to write “goat.hit()”, note that the two are separated by a dot (.). Pressing the **hit** button will write the word “.hit()” in your code.
	“frozen()” is a function without an argument that is used to ask “is the banana frozen?”, we use it by writing “banana.frozen()”. The function then returns a value that is yes or no according to the state of the object. Use “frozen()” as a condition for the if statement like so: if banana.frozen() to tell the computer to do things only if! Pressing the **frozen** button will write the word “.frozen()” in your code.

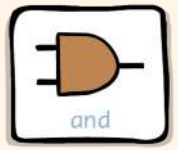
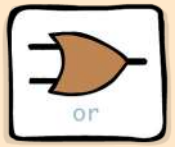
Reference Card Cont.

Keyword/Button	Description
	"if" is used when we want our code to decide what to do based on a certain condition. The state might be unknown to us when we write our code, and the computer will have to decide what to do while the code is executed. Pressing the **if** button will write the following in your code: if condition ....# your code here
	"green()" is a function without an argument that can be used as a condition to the if statement. This function will ask "is the banana green?", then returns a value that is yes or no according to the state of the object. We use "green()" like so: if banana.green() Pressing the **green** button will write the word ".green()" in your code.

Reference Card Cont.

Keyword/Button	Description
	An if statement helps the computer decide what to do based on a certain condition. When we use an if-else statement, we are telling the computer what to do in case the condition is true (under the “if”), and what to do in case the condition wasn’t met, meaning it is false (under the “else”). if [condition] code to be executed if the condition is true else code to be executed if the condition is false The state might be unknown to us when we write our code, but we are telling the computer what to do in both cases, and it will decide what to do while the code is executed. Pressing the **if-else** button will write the following in your code: if condition # Your code here else # And more here

Reference Card Cont.

Keyword/Button	Description
	In challenge 125 we are introduced to the tiger and the bear. They will attack our monkey if they see him, so we have to wait for them to fall asleep or be distracted by playing. “playing()” is another function we can use with a loop condition. When using “playing”, we need to write: *until tiger.playing()* *wait()* Pressing the **playing** button will write the word “.playing()” in your code.
	and is a *logical operator*: it performs a logical operation. We use it with an until loop to make a loop condition that is made out of two different conditions. That way we can make the computer wait until both conditions are true, before executing the code below. Pressing the **and** button will write the word “and” in your code.
	or is a *logical operator*: it performs a logical operation. We use it with an until loop to make a loop condition that is made out of two different conditions. That way we can make the computer wait until at least one of the conditions is true, before executing the code below. Pressing the **or** button will write the word “or” in your code.

Reference Card Cont.

Keyword/Button	Description
	The penguins appear in challenge #141. Some of them are wearing a bow tie. Certain penguins needs to move in order to form a bridge for the monkey. Sometimes, the ones with a bow tie and glasses need to move. Other times, those with a bow tie or glasses need to move. “hasBowTie” is a function we can use with a loop or if condition. When using “hasBowTie”, we need to write: if penguin.hasBowTie() penguin.step 10 Pressing the **hasBowTie** button will write the word “.hasBowTie()” in your code.
	Some of the penguins have glasses. Certain penguins needs to move in order to form a bridge for the monkey. Sometimes, the ones with a bow tie and glasses need to move. Other times, those with a bow tie or glasses. “hasGlasses” is another function we can use with a loop or if condition. When using “hasGlasses”, we need to write: if penguin.hasGlasses() penguin.step 10 Pressing the **hasGlasses** button will write the word “.hasGlasses()” in your code.

Character Review

Character	Description
	We are introduced to the rats a few times in the game, but it's not until challenge #76 that we can really control them. Rats love to collect items. In part three, we help them collect matches. In order to do so, we will use these simple functions: "grab()" and "drop()".
	The ants are introduced in challenge #94 to help demonstrate the "until" loop. The ants are dragging our precious matches, and we need to "step" until we reach them and take the matches back.
	The cat is introduced in challenge #101 to demonstrate the "until" loop with the function "wait()". The cat will attack the rat if he sees him, so we have to wait for him to fall asleep before we go out to collect matches.
	In challenge #106 we are introduced to the goat who will help us break the ice around frozen bananas. The function "hit()" will tell the goat to hit the frozen banana so the monkey can get it, but be careful not to use it on un-frozen bananas.

Character Review Cont.

Character	Description
	Frozen bananas are playing hard to get. In order to collect them we first need to send the goat to break the ice, and only then we can collect them.
	Monkeys don't like green bananas, but goats do! Let the goat collect green bananas to keep her satisfied.
	The tiger is introduced in challenge 125, it will attack the monkey if it sees him, so we have to wait for it to fall asleep or play before we go out to collect bananas.
	The bear is introduced in challenge 126, it will attack the monkey if it sees him, so we have to wait for it to fall asleep or play before we go out to collect bananas.
	The penguins are introduced in challenge 141. The need to form a bridge for the monkey. The penguins can wear nothing (1st image), a bow tie (2nd image), glasses (3rd image) or both (4th image). Not all the penguins should move, this is decided in each challenge. In some challenges the penguins with the glasses and a bow tie should move. In other challenges, the penguins with glasses or a bow tie should move.

Great Job!

You have Completed

CODING ADVENTURE

FUNCTIONS & CONDITIONS