Hour of Code

Banana Tales Teacher Notes

The Goal

The goal of each challenge in Banana Tales is to feed the banana to the baby monkey. The little car containing the banana is not smart; once started all it will do is move towards the baby via the most direct route possible. As programmers, the students will write code not to control the car but to control the environment to make sure that the path between the car and the baby monkey is clear and safe.

Platform Hints

Basic Flow

For each challenge students will follow the same basic cycle to develop and test a solution.

1. Write code in the area on the left
2. Click the Run! button to execute the code
3. Click the banana car to set it in motion

If the banana reaches the monkey, great! The student can move on to the next challenge. If not they should click Rewind and try again.

Fixing Mistakes

Sometimes students may realize that they have a mistake in their code even before clicking on the banana. If they want to modify their code after they have clicked Run! but before clicking the banana, they will need to click the Rewind button first.

To the left of the Run! button is a button with a counterclockwise arrow. This can be used to reset all the changes that have been made to the code.

Drag and Drop

On certain challenges students may have to drag and drop additional animals on screen in order to create a solution. This only applies to challenges 6, 9, 10, 11, 16,

and 19. If draggable animals are available they will appear in the upper left corner of the play area.

Python Concepts

Objects

In challenge 2 students are introduced to a concept that will be absolutely central to everything they do in Banana Tales, the concept of *objects*. To begin to understand what objects are and how they work, consider the line of code that solves this challenge, `giraffe.height = 6`.

In this statement, `giraffe` represents an *object*. Object has a special meaning in computer science. The following definition is a bit oversimplified, but it will do for right now: An object is a collection of code and data intended to represent something. In this case, the `giraffe` object represents, well, a giraffe, or at least a version of a giraffe that works in the Banana Tales game world. The `giraffe` object includes code that does things like telling the car not to fall when it drives over the giraffe's head and data like the image files used to draw the giraffe on the screen.

Most of the code and data that makes up the `giraffe` object is hidden away inside the CodeMonkey platform where we can't get to it, but we do have access to part of it. That's where the `.height` comes in. `height` is a *property* of the `giraffe` object. A property of an object is a piece of its data that we, the programmers, have access to. In this case, the `height` property of the `giraffe` object represents the height of the giraffe we see on screen. The statement `giraffe.height = 6` says to change the height of the giraffe to 6 units, and when it executes we can see the change happen.

The dot between `giraffe` and `height` is important. The name of the object goes before the dot, the name of the property goes after the dot. The analogy is not perfect, but the dot in Python works a lot like the 's for possessives in English. `giraffe.height` essentially means "giraffe's height".

In some challenges there will be multiple animals of the same type on the screen. Until lists are introduced in challenge 13, each animal will have its own name, generally something like `giraffe_1`. Students can see these names by hovering their mouse over the object on screen, and if they click on the object its name will be automatically be copied into the code window. A statement like `giraffe_1.height = 4` only changes the height of that specific giraffe, not any of the others on the screen.

The snake object, introduced in challenge 5, has a `length` property instead of `height` and stretches horizontally instead of vertically but otherwise works very much like the giraffe.

Methods

In challenge 12 the whale object is introduced. Unlike giraffes and snakes, whale objects don't have properties to change their appearance. Instead, a whale object has a *method* called `blow()`. A method is a specific piece of code that is part of an object that makes it do something. In our case, the `blow()` method instructs the whale object to spray a waterspout above its head.

Notice the syntax for a method. Like a property, it is connected to the object by a dot. Unlike a property, the method name is followed by parentheses. Inside the parentheses is a value called the *argument* of the method. Argument probably seems like kind of a strange word here, but in this context it just means an input that tells the method how to do its job. The argument to the `blow()` method is a number that tells the whale how tall to make the water spout.

Lists

Challenge 13 introduces lists. Notice how the two whales are referred to here. In earlier challenges when there were multiple objects of the same type, each one had its own name, like `giraffe_2` and `giraffe_3`.

Here the plural word `whales` refers to a new kind of thing, a *list*. A list is pretty much just what it sounds like: an ordered collection of things. In this case, the `whales` list contains two separate whale objects. We can refer to them individually by using an *index* in square brackets following the name `whales`. The index counts from the beginning of the list, except the counting starts at 0. So the first whale in the `whales` list is `whales[0]` and the second whale is `whales[1]`.

Notice this distinction: In a name like `whale_2`, the underscore and the number have no special meaning to Python. `whale_2` is just a name that happens to have an underscore and a number. But for something like `whales[2]`, the square brackets and number do have a special meaning. They mean that we are referring to the third item (count 0, 1, 2) in a list called `whales`.

(To be clear, it is not the fact that `whales` is plural that makes it a list. But using a plural name is a good way for a programmer to remind her or himself that `whales` is a list that contains multiple objects.)

For Loops

Challenge 17 contains the first example of a `for` loop. In computer science, a loop is a control structure that tells the computer to repeat certain code over and over. There are different types of loop structures, and they determine how many times to repeat the loop in different ways. In Python, a `for` loop is used to repeat the code one time for each element in the list.

Look at the first line of the code: `for whale in whales:` The keywords `for` and `in` and the colon and the end of the line are part of Python syntax and will be the same for every `for` loop. The `whales` after the word `in` refers to list we intend to loop through, and `whale` is the name of a *variable* that changes each time the loop repeats. In other words, each time through the loop, `whale` will represent a different object that is part of the `whales` list.

The second line is the code that will be repeated by the loop. Notice how it is indented several spaces relative to the `for`. In Python, indentation is used to group code together. If we wanted to have several lines of code repeat within the loop, we would have to indent each one the same amount.

The Cast

It's a jungle out there! Fortunately, Banana Tales programmers can enlist the assistance of several types of animal friends to help create a safe path from the banana to the baby monkey.

Baby Monkey

The baby monkey got separated from his twin and now needs our help to feed him bananas.

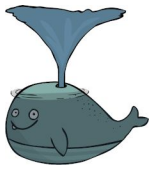
Giraffes

Giraffes are (generally) tall and skinny, though how tall can be controlled using their `height` property. Giraffes' flat heads are perfect for filling in one square wide gaps in the banana's path.

Snakes

Snakes are good for filling in wide horizontal gaps. They can be stretched or shrunk via their `length` property.

Whales



Whales have a `blow()` method that causes them to emit a waterspout that will lift the banana car like an elevator. The `blow()` method takes one argument that tells how high the water spout should go.