CodeMonkey

Write code. Catch bananas. Save the world.

BANANA TALES

PART ONE

Lesson Plans
1-12

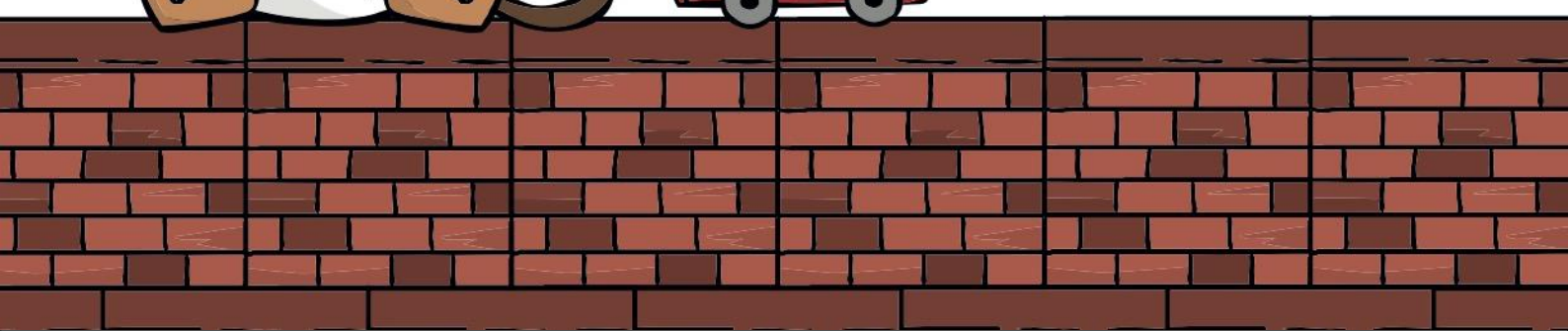


Table of Contents

Lesson 9 - While Loops

CSTA Standards

Elementary (3 - 5)	Middle (6 - 8)	High (9 - 12)
1B-AP-09 1B-AP-10 1B-AP-15 1B-AP-17	2-AP-11 2-AP-12 2-AP-17 2-AP-19	3A-AP-14 3A-AP-15 3A-AP-23 3B-AP-12 3B-AP-21

Objectives

- Express a conditional with arithmetic comparison operators
- Use a `while` loop to repeat an action as long as a certain condition is true
- Complete Challenges 62 - 68

Components

PYTHON

- `while` statements

PLATFORM

- Elephant and Well objects
- `water_level` and `max_water_level` properties
- `spray_at()` method

Introduction - 10 min

ACTIVITY - 9 MIN

For the activity you will need at least one container and a supply of counters. A glass, cup, or jar is fine for the container (transparent is better) and the counters can be chips, marbles, coins, or

something similar. Ideally you can provide a container and some counters to every student, but the activity can work with just one. You will also need some way to play music that is audible to the entire class.

Provide cups and counters to as many students as possible then display the following pseudo-code:

```
repeat these steps while the music is playing:
    add 1 counter to the cup
    wait 2 seconds
```

If any students ask how they will know how long two seconds is, tell them just to estimate. You may wish to demonstrate counting off “one-one-thousand, two-one-thousand” but don’t let concerns about exact timing bog down the activity.

Tell the students to follow the code and then start the music. They should begin adding counters to their cups at approximately two second intervals. After 20 or so seconds stop the music.

Briefly discuss the activity. If you noticed any student who only added one counter to the cup or who kept adding counters after the music had stopped, point out that the first statement tells them to repeat the actions below, but only while the music is playing. If any students were adding counters continually without any pause remind them that because both instructions were indented both should be repeated - add counter, wait; add counter, wait; add counter, wait; etc.

Now have students empty their cups and reset. If you don’t have enough supplies for everyone, switch off now and let a new group of students take a turn. This time the students will be following this pseudo-code:

```
repeat these steps while the number of counters in the cup is less than 6:
    add 1 counter to the cup
    wait 2 seconds
```

Tell the students to start. It should only take them 20 seconds or so to complete the procedure. Ask students to count the number of counters in their cup. If they followed the procedure correctly, they should each have six, but it is fairly likely that some of them will have only five and some may possibly have seven.

Ask one or more students who carried out the procedure correctly to explain their thinking. The key idea is that you have to check the number of counters in the cup before you repeat each time. If the number of counters in the cup is five, yes, that is less than six, so you do the steps again and add another counter. When there are six counters in the cup, six is *not* less than six so you do not repeat anymore and the procedure is over.

EXPLANATION - 1 MIN

Ask students what you call a structure in computer programming that repeats the same steps over and over. The answer is loop, though if a student says `for` loop you can point out that there are other types of loops that use different rules to determine how many times to repeat the steps. Today’s lesson is about a new kind of loop called a `while` loop.

Playtime

LOG-IN - 2 MIN

Students should log-in to their accounts as usual.

Playtime (1) - 4 min

Let students work through Challenges 62 and 63 on their own. These challenges introduce the elephant and well objects and the `spray_at()` method that causes an elephant to add water to a well. It shouldn't take students very long to complete these challenges, but before you move on make sure they understand the following ideas.

- `spray_at()` does not automatically fill the well; it only adds enough water to raise the level one unit. That is why it is necessary to `spray_at()` multiple times.
- Trying to `spray_at()` a well that is already full causes an error that forces you to Rewind without completing the challenge.

Explanation - 9 min

Before students move on from Challenge 63, discuss the properties of wells that are introduced there. `max_water_level` never changes. It represents the amount of water the well can hold without causing an error. `water_level` is the current water level. It increases by one each time an elephant sprays water at the well.

Now look at Challenge 64 as a class. This is a two-level challenge. Remind the students that for this type of challenge they have to write code that works for both levels without any changes and ask them why that makes this challenge harder. They should be able to recognize that because the well on the second level is deeper than the well on the first, it will require more `spray_at()` statements to fill. Completing this challenge depends on finding a way to write code that will spray the right number of times for each well.

Ask a student to read the comment and supply the missing statement. To solve the challenge all that is needed is `elephant.spray_at(well)` under the `while` statement. But have students first try this lengthier solution:

```
print("Max water level is:")
print(well.max_water_level)
print("Water level is:")
print(well.water_level)
while well.water_level < well.max_water_level:
    # The elephant should spray at the well.
    elephant.spray_at(well)
    print("Water level is:")
    print(well.water_level)
```

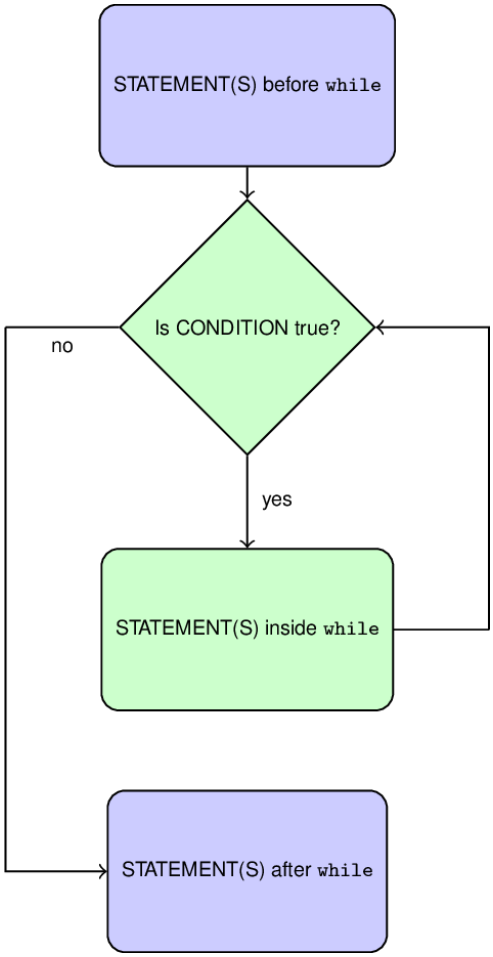
This won't earn three stars, but that doesn't matter. Students can go back and delete extra `print()` statements later. The extra output is helpful for explaining how the `while` statement works.

The general idea of a `while` statement is simple; it is a new kind of loop that repeats a set of steps over and over as long as a certain condition is true. In this case the condition is a mathematical one: The loop will repeat as long as the water level in the well is less than the maximum possible water level for the well. That means that when water level is *equal to* the maximum water level then the

loop will *not* repeat again. That is exactly what we want, because if the well is already full, we don't want to add any more water.

It is important to understand that the condition of the `while` statement is checked before every time through the loop (including the first time), but not in the middle of the loop. In the code above, `well.water_level` changes as soon as the `spray_at()` method is executed, but the next two `print()` statements happen no matter what since the condition is only checked again once all of the statements have been executed.

The flowchart below illustrates how `while` loops work in general:



Once the discussion of `while` loops is complete, let students delete the `print()` statements to earn their three stars on Challenge 44 and move on to the last two in this lesson.

PLAYTIME (2) - 15 MIN

Challenges 45 and 46 are a bit more difficult than usual, at least as far as earning three stars goes. But these are good challenges for students to grapple with on their own. They will need to use `while` loops as in the previous challenge to fill each well the right amount, but they will also need to use a `for` loop to write the code to fill multiple wells in the fewest number of lines.

Challenges 65 - 68 gradually reduce the scaffolding and increase the difficulty as students practice setting up and using `while` loops.

- Challenge 65 can be solved by simply duplicating the given code and changing the list indices to 0, but in order to earn three stars students will have to use a `for` loop. If they get hung up on this tell them to move on then come back later to pick up the third star.
- Challenge 66 provides a template for using the elephant `while` loop inside of a `for`. Point out the use of the `len()` function inside of `range()`. This makes the code more flexible; it will work for lists of any length.
- Challenge 67 is similar to 66 except students have to write all the code themselves. They should refer back as necessary.
- On Challenge 68, the order in which the draggable elephants are added matters. The first elephant should be added to the central ledge, the second to the ledge on the right. This is important so the indices of elephants and wells match.

Debrief - 5 min

Display each of the following code snippets:

IF

```
if CONDITION:
    # One or more statements
    STATEMENT
    STATEMENT
    # etc.
```

FOR

```
for VARIABLE in LIST:
    # One or more statements
    STATEMENT
    STATEMENT
    # etc.
```

WHILE

```
while CONDITION:
    # One or more statements
    STATEMENT
    STATEMENT
    # etc.
```

Ask students to compare and contrast these types of statements. Similarities they might point out are that they all use a colon, they all use indentation, and they all involve one or more statements. Going deeper, they might observe that each of them is a way to control if and how many times the statement(s) are executed. `if` executes the statements or not depending on the condition. `for` executes the statements for each element in the list. And `while` executes the statements over and over again as long as the condition is true.

Explain that all of these are examples of what are called **control structures**. Every programming language has some kinds of control structures. While there are a few more control structures in Python, at this point students have met all of the most important ones.