

# Compute It | Teaching Guide

## Overview

**Compute It** - Code Hour is a self-led activity that allows students to understand how to analyze and follow code instructions

Students will progress through a level-based format, in which they will gradually be introduced to different coding concepts

1. **Loops**
2. **Conditional Statements**
3. **Nested Statements**
4. **Functions**
5. **Recursion**

After a few levels of understanding it will introduce a new concept

## **Important Things to Note:**

- In order to complete a level, students must correctly follow the level's code. They must complete the current level to unlock the next level.
- “Compute it” is designed to be usable by educators with zero coding knowledge, or even without educators. It is key to not explain too much or just tell students the answers. Rather, allow them to get comfortable with the feeling of not knowing an answer right away. Remind them that it is okay if a problem seems difficult at first.
- It's easier for the beginners to be introduced to abstract concepts by practicing than by using complex words. And figuring it out will reward them with a “eureka” moment, and have more impact.
- We witnessed it several times in our test sessions that beginners, assisted by an expert friend, typically have a slower progression.
- To have a positive effect as a guide, only provide small hints only when the player is asking for it or seems stuck.

**Link to the activity :** <https://compute-it.toxicode.fr/>

## Vocab

**Loops** - A sequence of instructions that are repeated until a certain condition is reached. ('while', 'repeat') A while loop continues to execute a block of code as long as the specified condition is true. A repeat is used for iterating a given number of times.

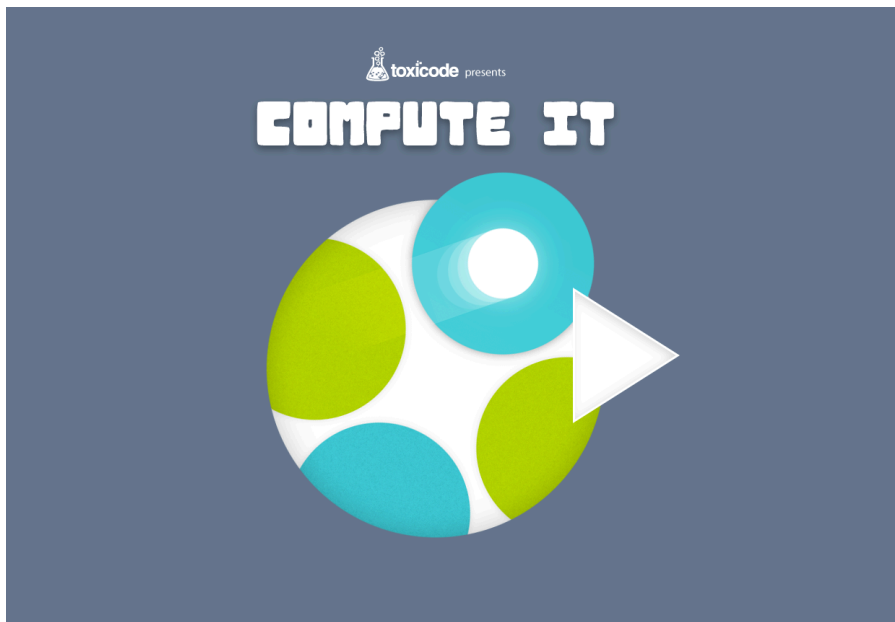
**Conditional Statements** - Sections of code that are only carried out if the given condition is true ('if', 'else'). if statements are used to make decisions in code. If the condition is true, the indented block of code is executed. Else statements provide an alternative block of code to execute if the initial condition is false. Example: "If color is red, move right; else move left".

**Functions** - A set of program instructions grouped together to perform a specific task, serving as a self-contained unit within the larger program.

## Jumping in to the activity

For this guide we will demonstrate the flow of moving through this activity :

Once your student has clicked/copied the link, they will need to hit the **play** button to start the activity

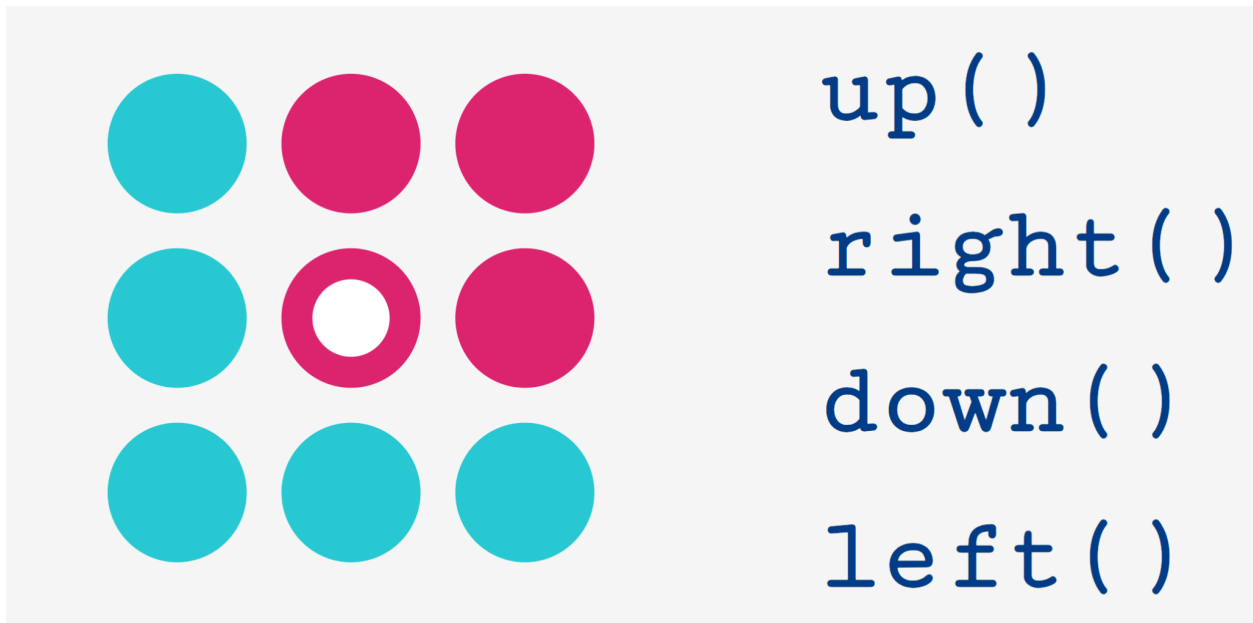


Students will then start will level 0:



The goal of each level is to correctly move the white circle based on the instructions on the right. In this first level, for example, you need to move the white circle by **pressing the right arrow key** three times.

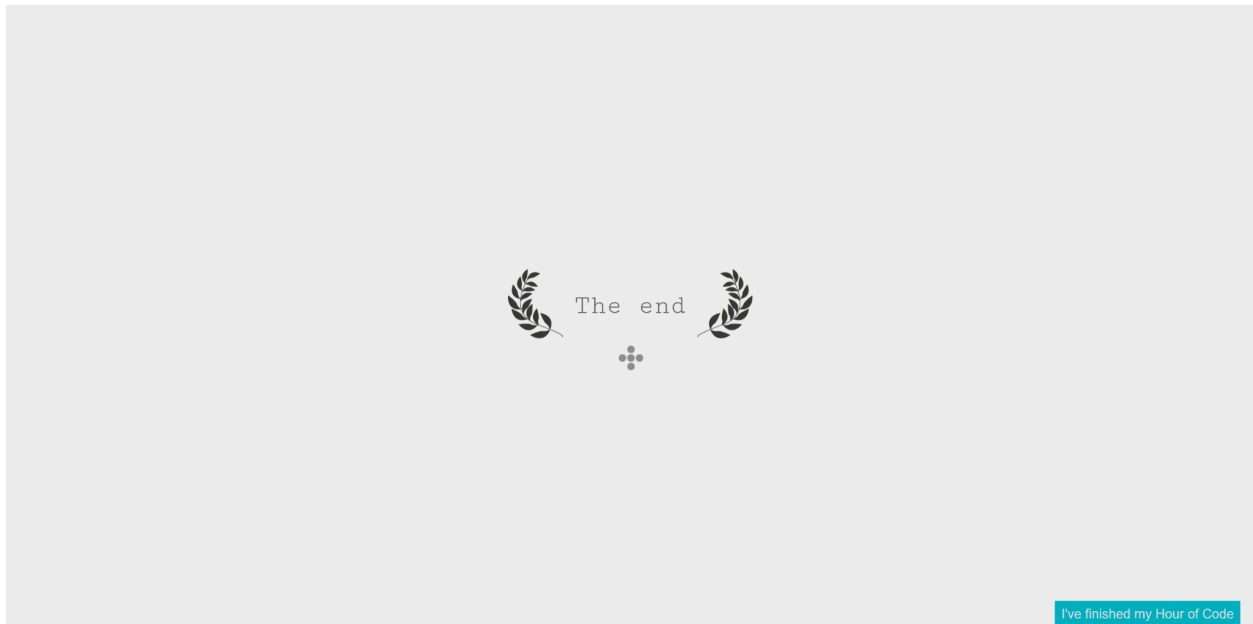
The following levels will have that same theme of moving directionally.



As the student progresses the levels will increase in difficulty and concepts introduced. For a more in-depth guide, see the section titled 'Key'.

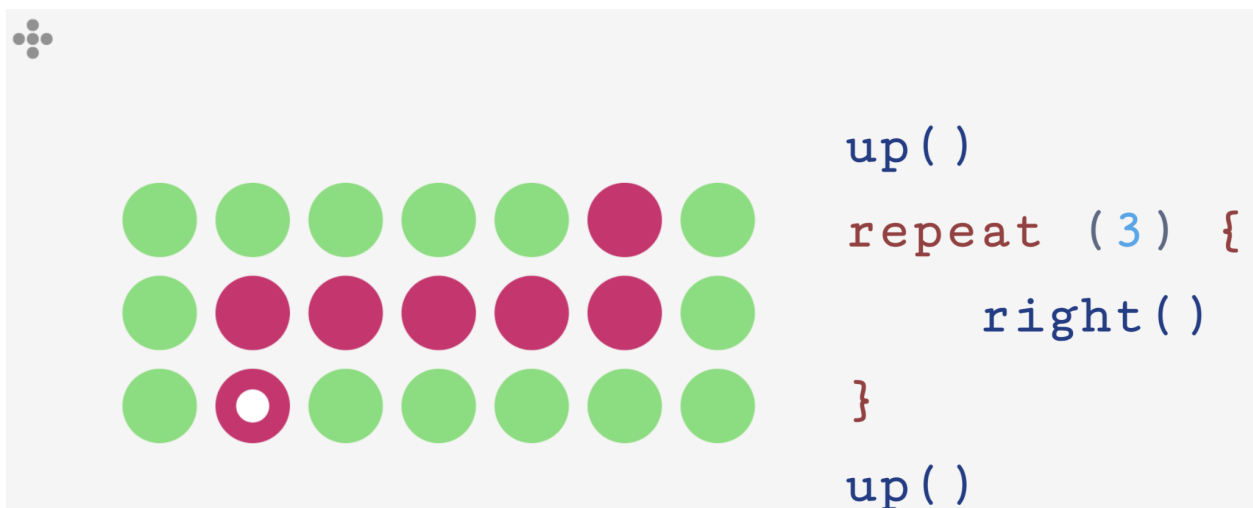
## Completed activity

Once all levels have been completed, the following screen will appear:



## KEY

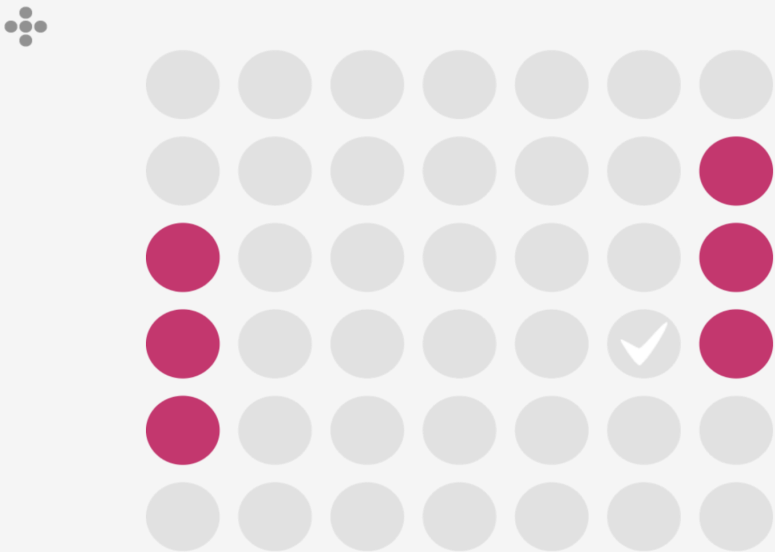
The first concept introduced is the 'repeat' loop.



Code inside the 'repeat' loop is to be repeated as many times as the statement specifies, in this case, 3.

For example "repeat (3)" means that the code in that statement (indentation) is to be repeated 3 times. They will move up and then move right 3 times and then move up to complete the level shown above.

It is potentially helpful to remember that all code is read from top to bottom, as illustrated in the following level:



```
repeat (3) {  
  down()  
  left()  
}  
repeat (2) {  
  right()  
  down()  
}  
repeat (2) {  
  up()  
  right()  
}
```

Though the levels might begin to look intimidating in the increase of code, it is still simply one set of instructions following another.

The notion explained above can be seen in this level:

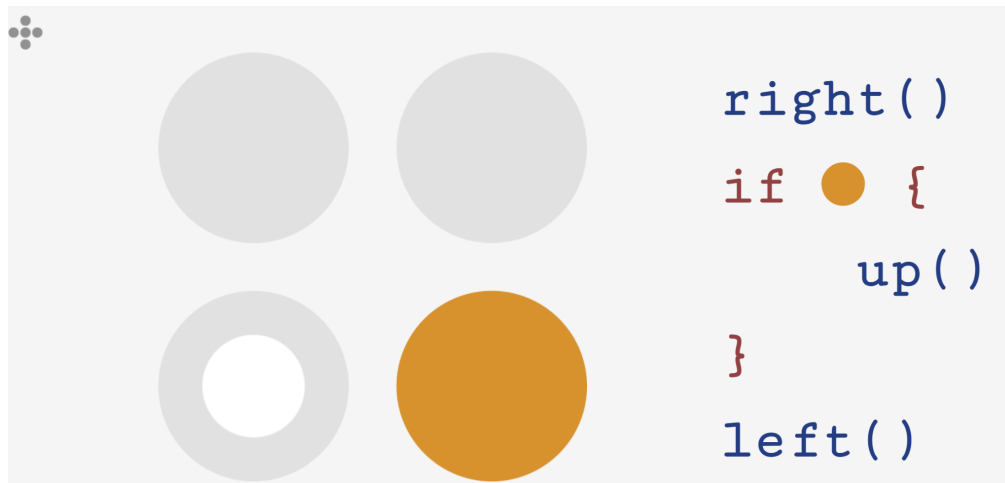


```
repeat (4) {  
  repeat (4) {  
    right()  
  }  
  up()  
}
```

This level introduces the concept of 'nesting' code/loops. As mentioned they must first read from top to bottom, doing so they will enter the first repeat loop. This just states that anything inside it is to be repeated 4 times.

Inside the loop is another loop that says to move right 4 times, after that move up. So together the student must - (move right 4 times, and then move up) and repeat that 4 times

The next concept introduced is the 'if' statement:



Instructions inside the 'if' are only to be done if the if statement condition is true. If the statement is not true, that section of code is simply ignored.

In the example above, if the white circle is on an orange circle, at the time of reaching that line of code, then you move up. If not, it gets ignored and the next line of code gets looked at.

This idea is seen especially in this level:



It is important to stop and think after each action, breaking down each piece into an individual step.

The else statement is just as intuitive. If the 'if' clause is false, the code will move to the else block.

```

left()
if ● {
    left()
} else {
    up()
}

```

It is important to note that each if can only be paired (correspond to) with one else.

```

if ● {
    up()
    right()
} else {
    left()
}

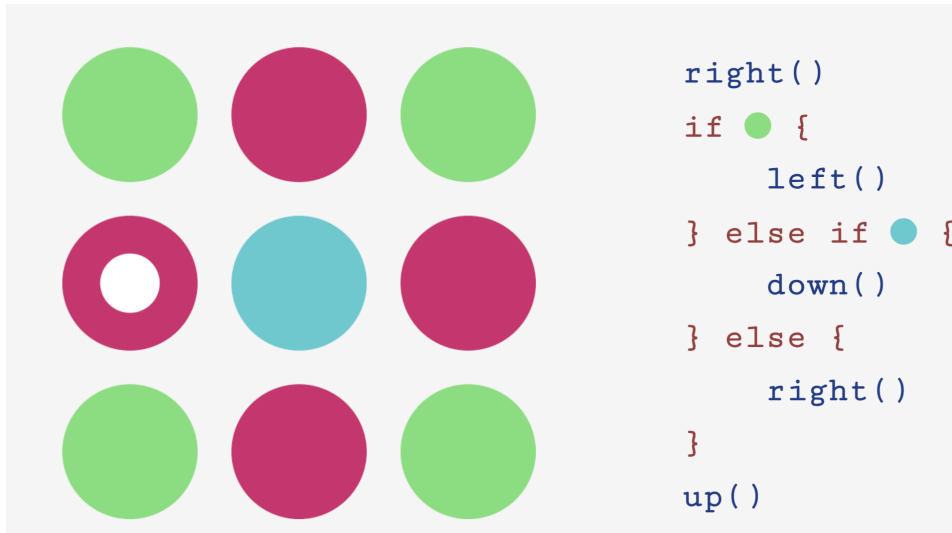
if ● {
    right()
    right()
} else {
    down()
}
left()

```

In this example, each if/else should be seen as a pairing.

When completing this level, after doing the instructions in the first if section, the student must know which section to look at next. One common mistake is to go straight to the 'left ()' line, but reading top to bottom one would find that they are to simply evaluate the next if/else statement.

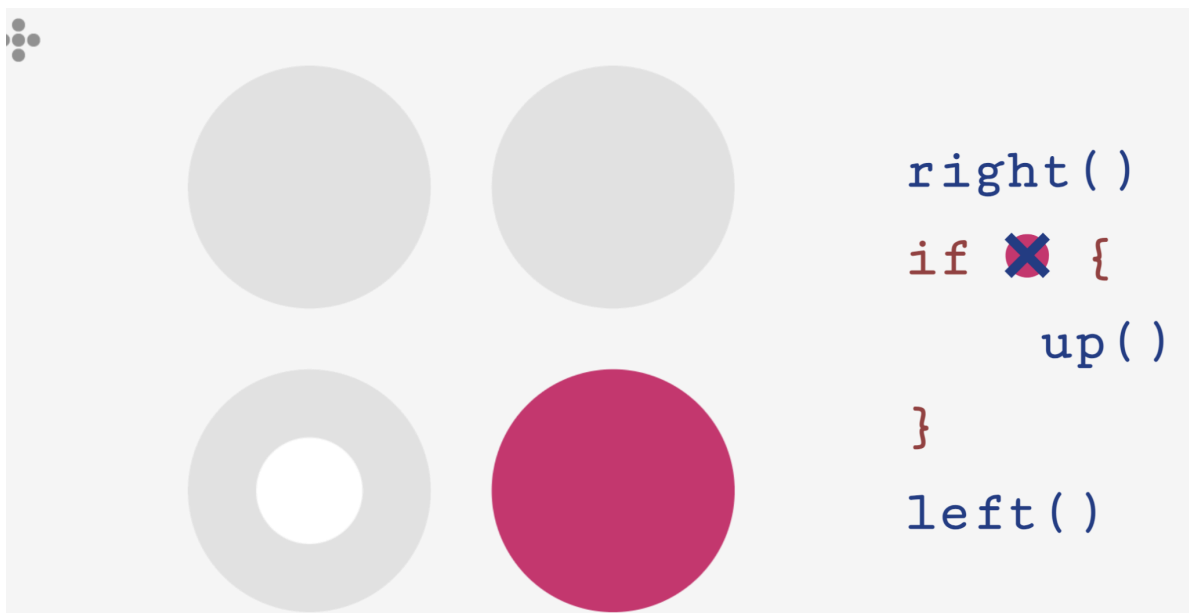
To 'chain' multiple if's to a single else statement, the else if is used.



"The if-else if-else structure allows your program to make decisions. It's like a flowchart that starts at the top with right() and then encounters a fork in the road. At each fork (the if and else if), your program checks if the path (condition) is clear (true). If it is, it takes that path, executing the code inside the block. If not, it moves on to the next fork (the else if). If all paths are blocked (all conditions are false), it takes the default path (the else block). After passing through the forks (conditions), it continues on its way with up()."

When looking at this code, one can assess and ask the question each time, seeing the else if as a "if not, how about if ...". The else in this case corresponds to both of the previous statements

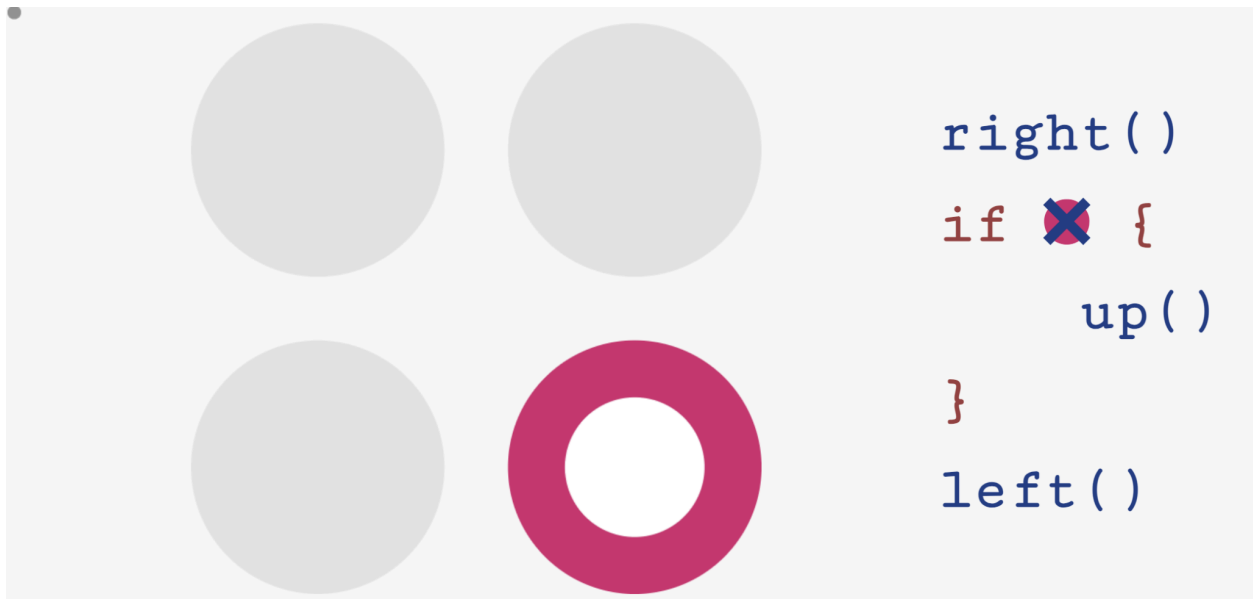
The level below introduces the 'not' operator, indicated by the blue 'X' over the red circle.



In this example, the if statement means "If the white circle is **not** on a red circle, move up".



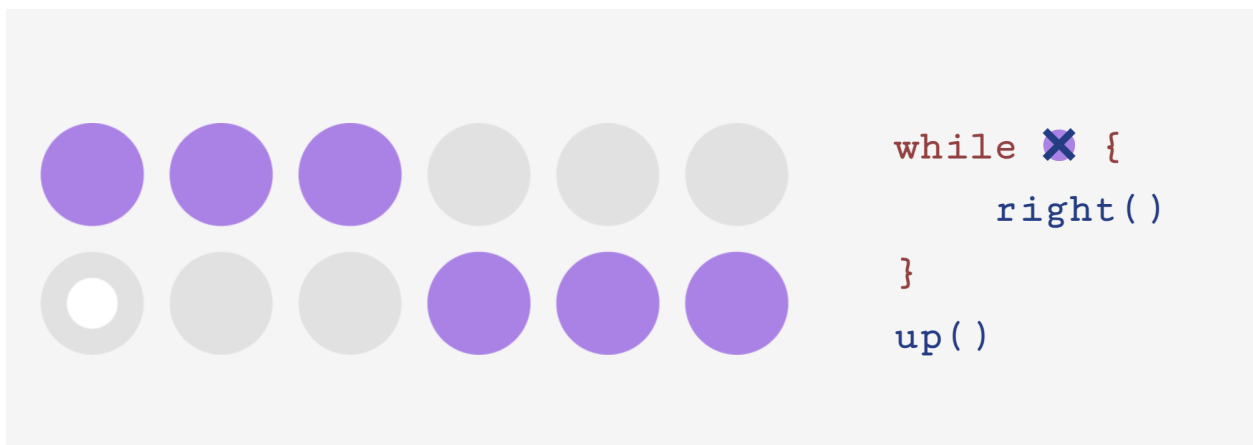
When completing this level the student would first move right, moving them to the red circle.



```
right()  
if X {  
    up()  
}  
left()
```

The student will now read the if statement mentioned earlier. However it states that they will only move up if they are not on a red circle, because they are currently on a red circle they will not move up and instead will just move left.

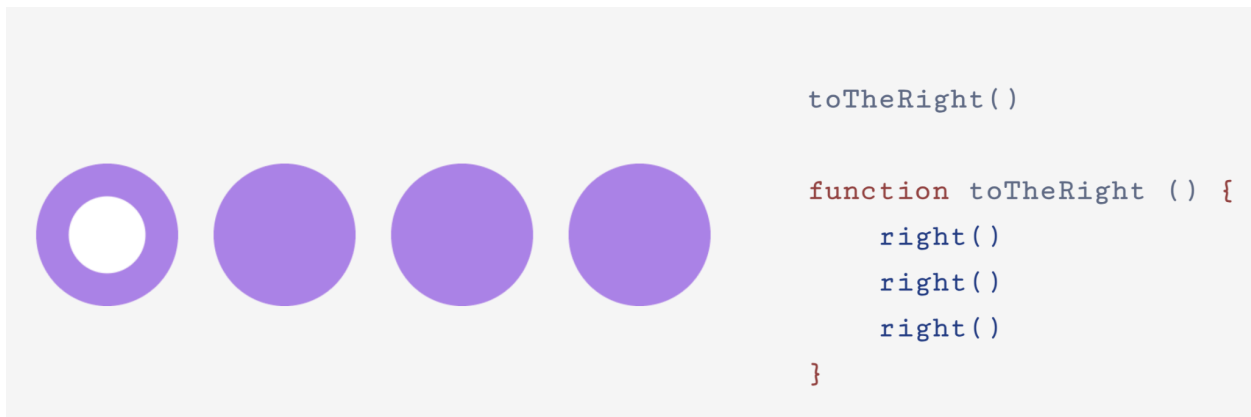
The 'not' can also be applied to loops as well:



```
while X {  
    right()  
}  
up()
```

In this example the student will move right each time they are not on a purple circle, then they will move up.

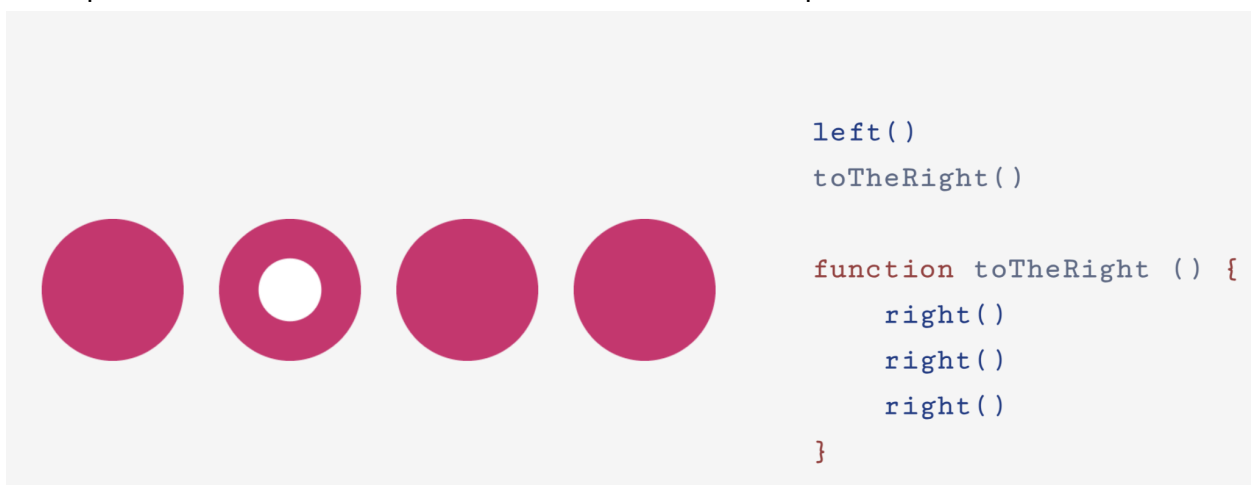
The next and final concept introduced is the 'function', denoted by the *function* keyword.




'function toTheRight()' is the declaration of the function. This is like defining a set of instructions or a recipe that you can use whenever you need it. In this example, you see right() repeated three times.

This means that every time toTheRight() is used, the student will move to the right three times.

It is important to note that the code should still be read from top to bottom.



When the student sees 'left()' they will move to the left as usual then they will read toTheRight(). When reading this, they are to look at the instructions for toTheRight(). Meaning they will then move to the right three times. One common mistake might be to then read function toTheRight(), and move three more times, but the function keyword means that section of code is only to be done when the function name (in this case, toTheRight()) is used by itself.



```

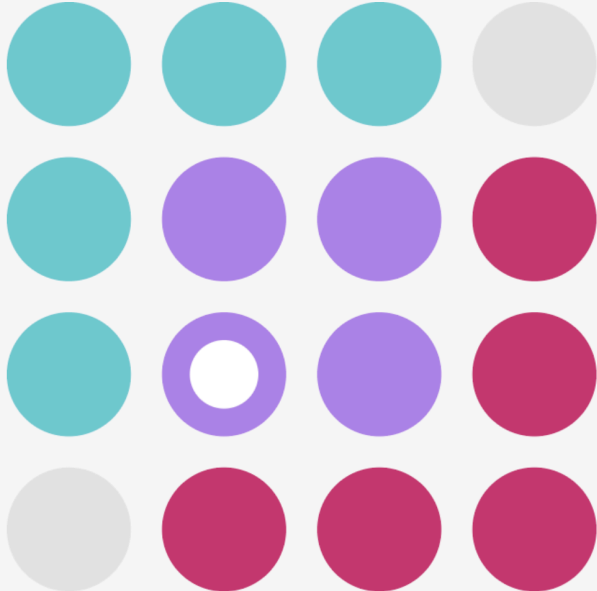
function toTheRight () {
  right()
  right()
  right()
}

left()
toTheRight()

```

As seen in this example, the location of where the function is defined does not matter. This is identical to the previous example.

One thing that the students can do is associate the name of the function with that function's instructions.



```

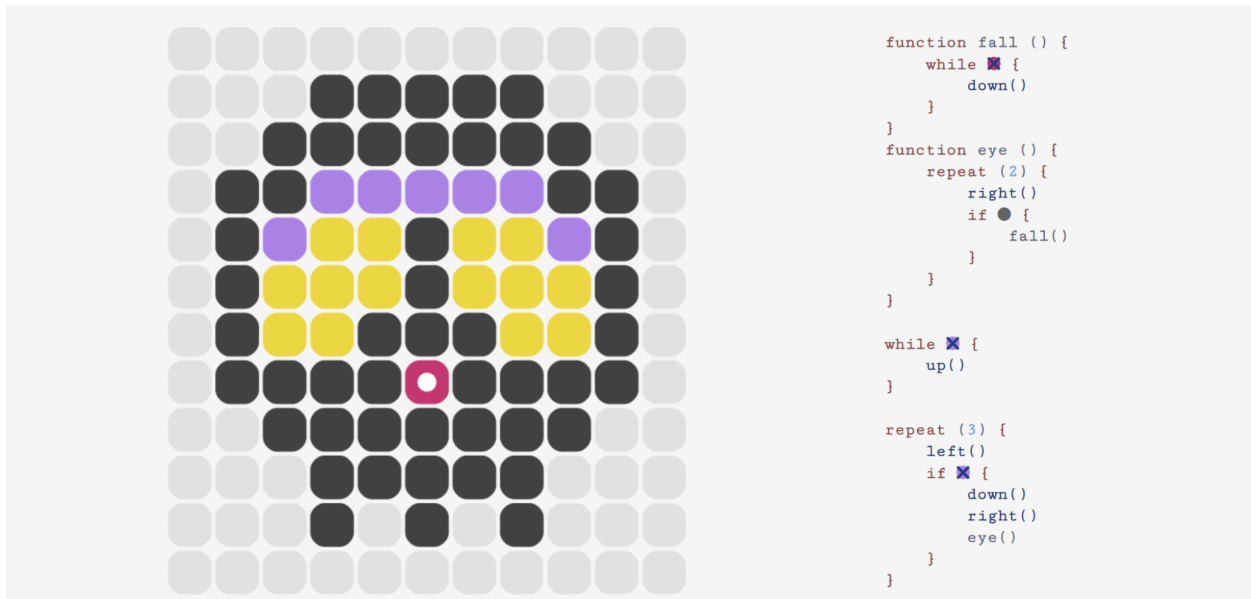
right()
square()
left()
square()

function square () {
  up()
  right()
  down()
  left()
}

```

In this example, that would mean associating square() with moving up, right, down, then left. Every time square() is seen the student will move up, right, and then left.

This last example, the final level, will demonstrate everything learned. Let's break it down.



First, two functions are being defined, `fall()` and `eye()`:

```
function fall () {
  while X {
    down()
  }
}
function eye () {
  repeat (2) {
    right()
    if ● {
      fall()
    }
  }
}
```

From these definitions, everytime `fall()` and `eye()` are seen the students will do the respective functions instructions.

Every time `fall()` is seen the student will move down as long as they are not on a red circle. Every time `eye()` is seen the student will move to the right. If they are on a black circle they will do the `fall()` instructions. That sequence of moving to the right, and doing `fall()` if they are on a

black circle will be repeated twice (as indicated by the repeat(2)). Now that the functions have been defined, let's move on to the next section of code.

```
while ✖ {  
  up()  
}  
  
repeat (3) {  
  left()  
  if ✖ {  
    down()  
    right()  
    eye()  
  }  
}
```

While the student is not on a purple circle they will keep moving up, once they are on a purple circle they will move to the next line of code. The repeat(3) indicates that the following will be repeated 3 times: the student will move left then check if they are on a purple circle, if they are not on a purple circle they will move down then right, and then they will follow the eye() function's instructions. If they are on a purple circle they will skip the if statement for that iteration of the repeat loop.

After completing the level they will see the end screen



This concludes the Compute It Teaching Guide. For any additional questions, comments, or overall feedback, please feel free to reach out via our feedback form. Thank you!