Introducing Swift Playgrounds

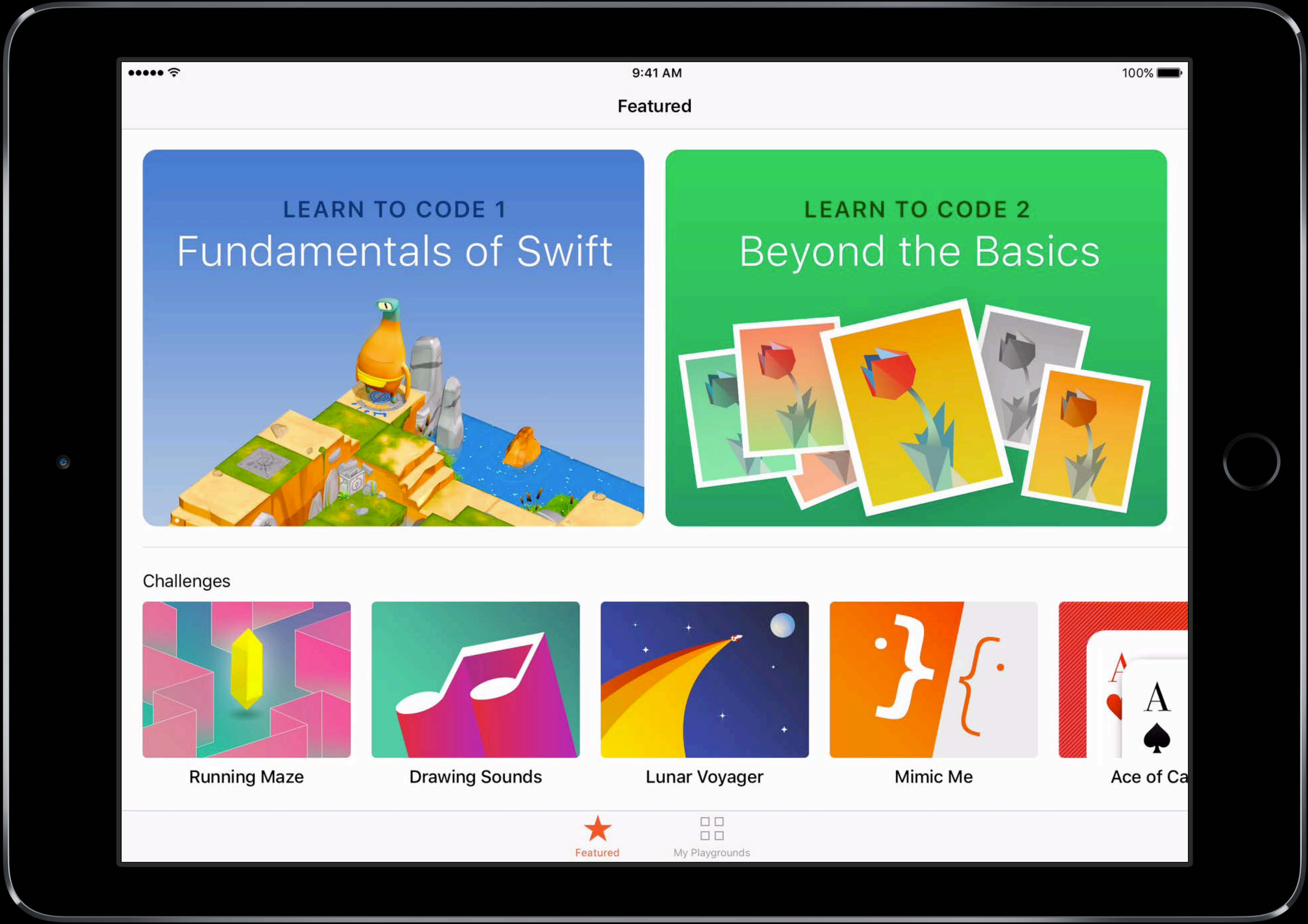
Exploring with Swift on iPad

Session 408

Matt Patenaude Playgrounds Engineer
Maxwell Swadling Playgrounds Engineer
Jonathan Penn Playgrounds Engineer
Izzy Fraimow Playgrounds Engineer

9:41 AM
100%
Tuesday
7
FaceTime
Calendar
Photos
Camera
Contacts
Clock
Maps
Home
Videos
Notes
Reminders
News
iTunes Store
App Store
iBooks
Settings
Playgrounds
Messages
Safari
Mail
Music

9:41 AM
100%
FaceTime
Tuesday
7
Calendar
Photos
Camera
Contacts
Clock
Maps
Home
Videos
Notes
Reminders
News
iTunes Store
App Store
iBooks
Settings
Playgrounds
Messages
Safari
Mail
Music



9:41 AM

100%

Featured

LEARN TO CODE 1

Fundamentals of Swift

LEARN TO CODE 2

Beyond the Basics

Challenges

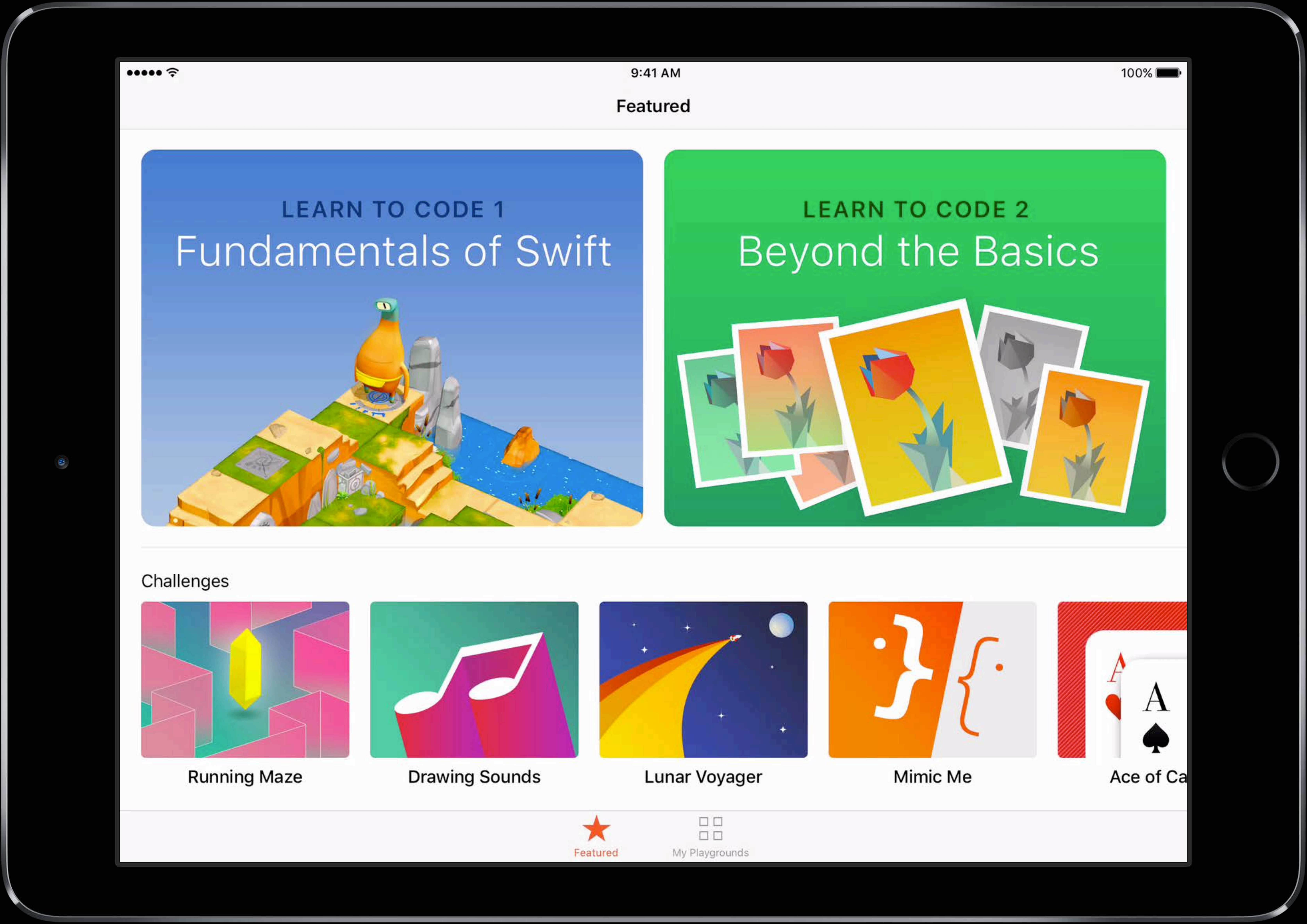
Running Maze

Drawing Sounds

Lunar Voyager

Mimic Me

Ace of Ca



9:41 AM 100%

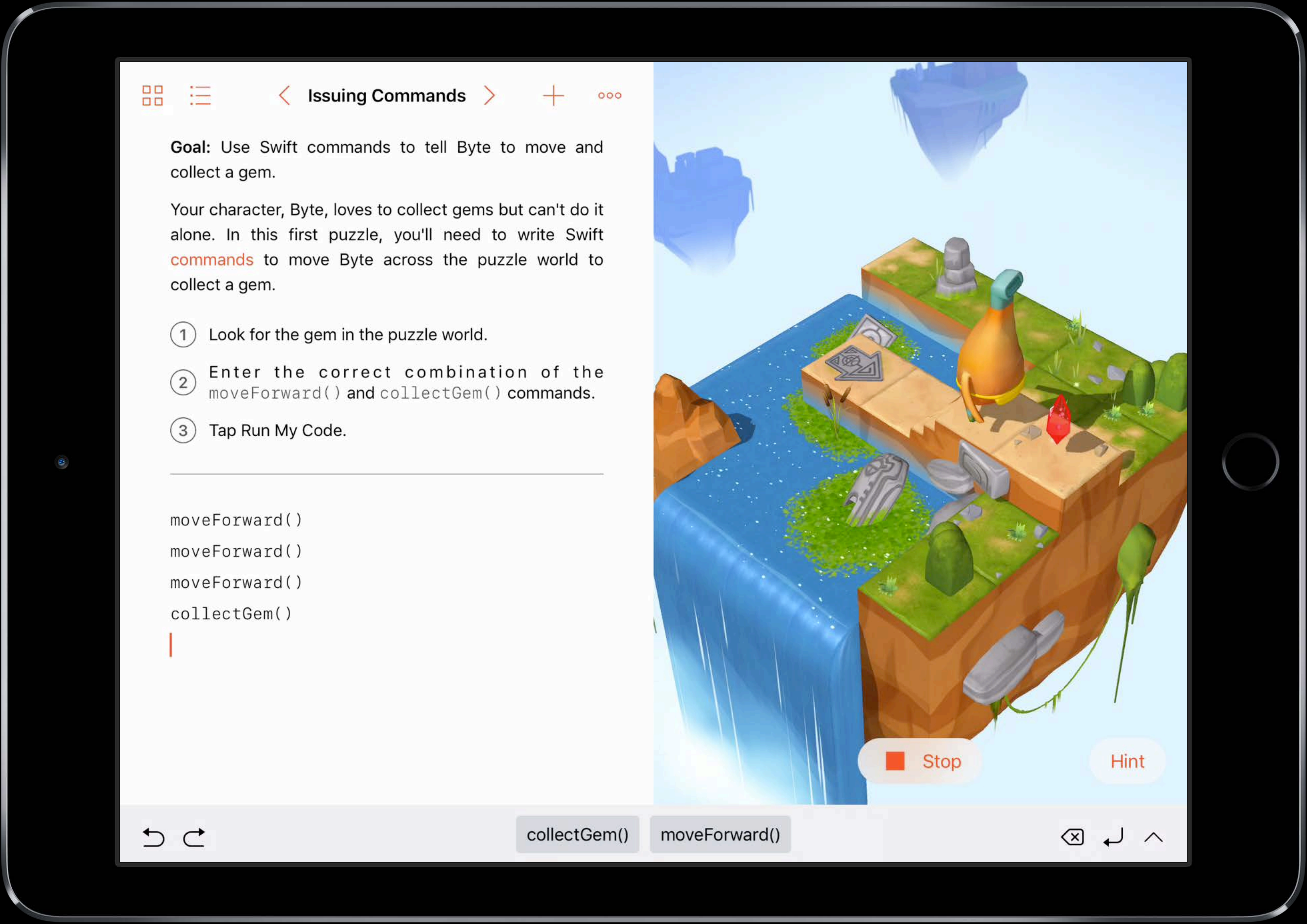
Featured

LEARN TO CODE 1
Fundamentals of Swift

LEARN TO CODE 2
Beyond the Basics

Challenges

Running Maze
Drawing Sounds
Lunar Voyager
Mimic Me
Ace of Ca

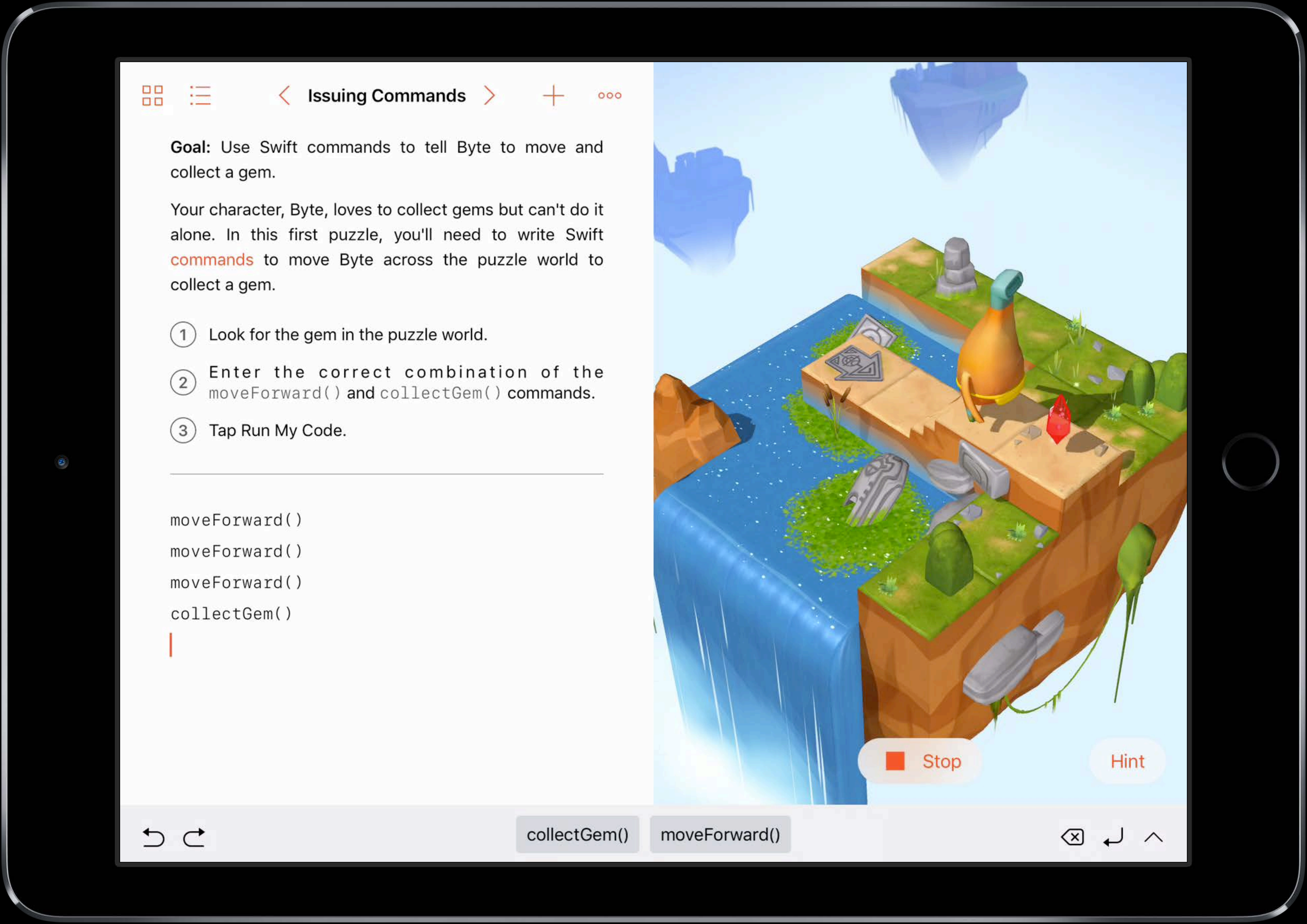


Goal: Use Swift commands to tell Byte to move and collect a gem.

Your character, Byte, loves to collect gems but can't do it alone. In this first puzzle, you'll need to write Swift commands to move Byte across the puzzle world to collect a gem.

1. Look for the gem in the puzzle world.
2. Enter the correct combination of the `moveForward()` and `collectGem()` commands.
3. Tap Run My Code.

```
moveForward()
moveForward()
moveForward()
collectGem()
```

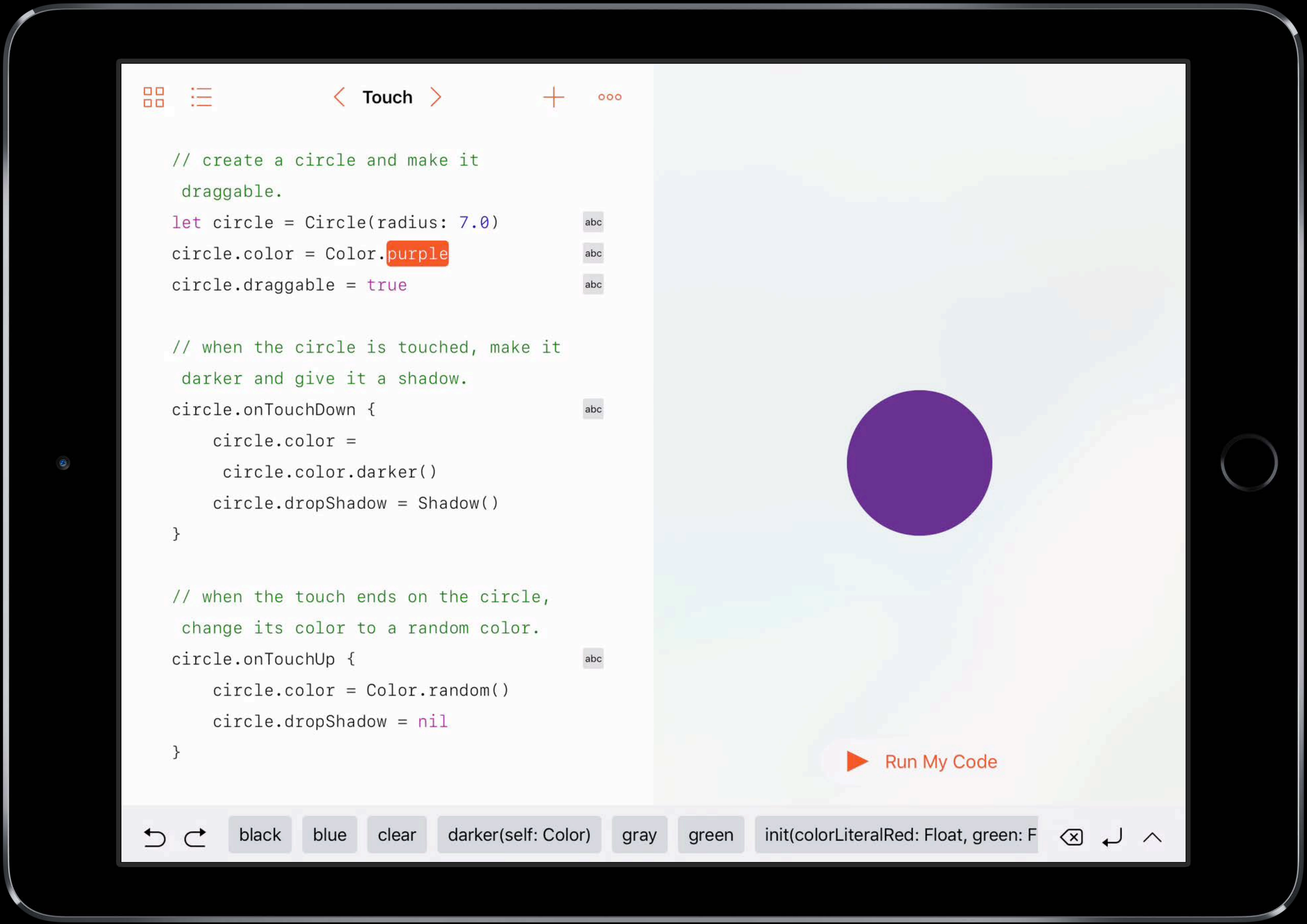


Goal: Use Swift commands to tell Byte to move and collect a gem.

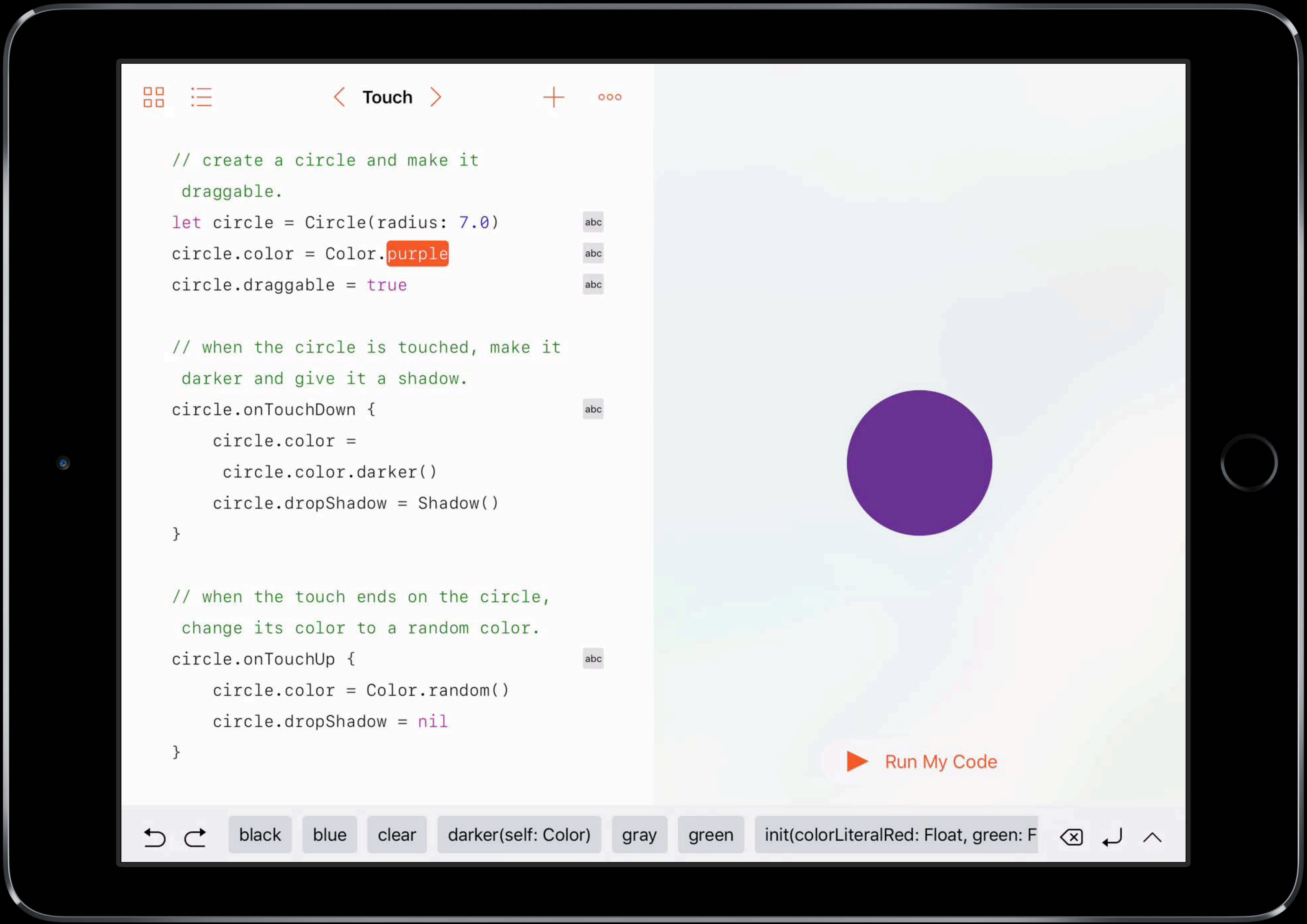
Your character, Byte, loves to collect gems but can't do it alone. In this first puzzle, you'll need to write Swift commands to move Byte across the puzzle world to collect a gem.

1. Look for the gem in the puzzle world.
2. Enter the correct combination of the `moveForward()` and `collectGem()` commands.
3. Tap Run My Code.

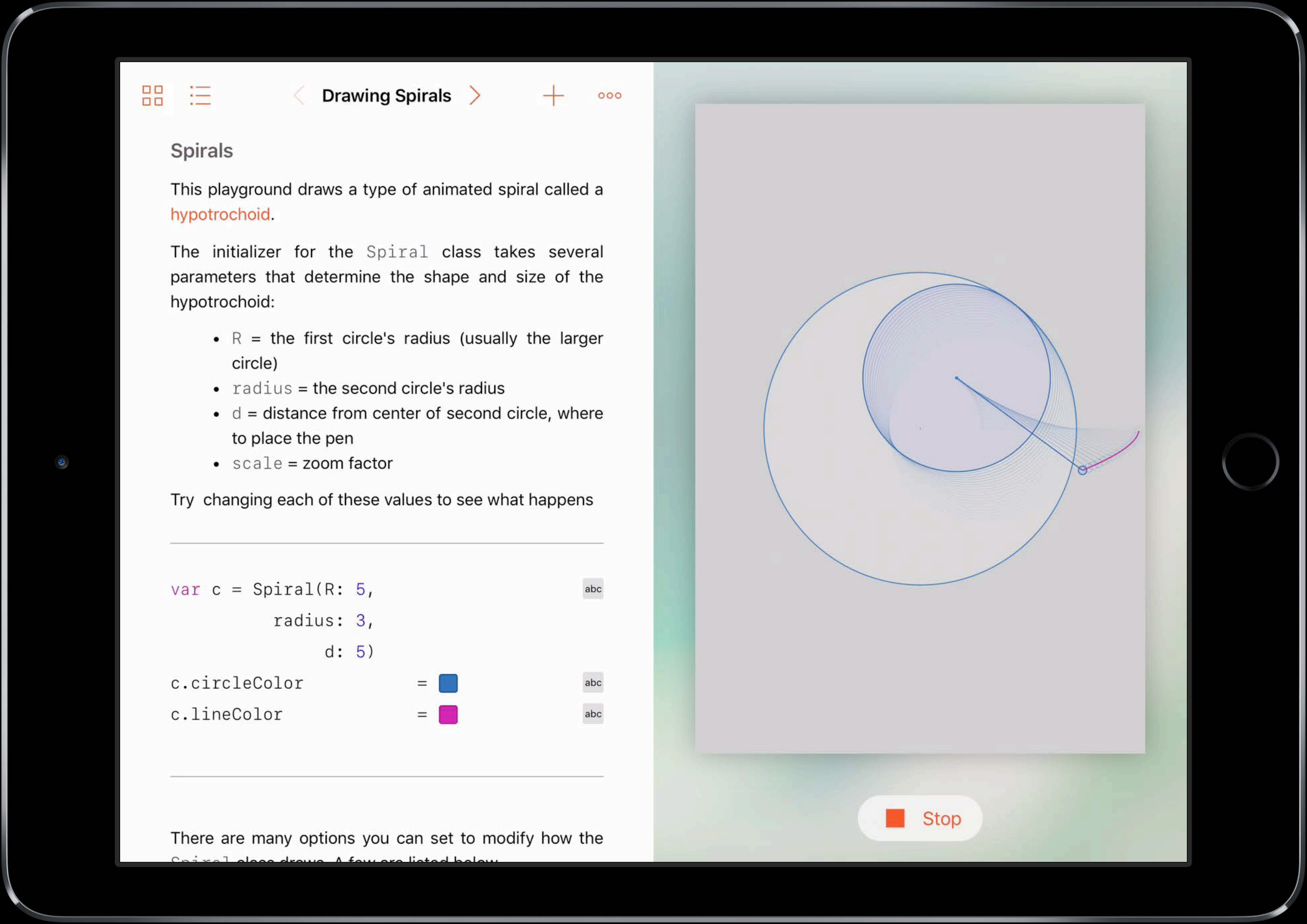
```
moveForward()
moveForward()
moveForward()
collectGem()
```



```
// create a circle and make it
 draggable.
let circle = Circle(radius: 7.0)
circle.color = Color.purple
circle.draggable = true

// when the circle is touched, make it
 darker and give it a shadow.
circle.onTouchDown {
    circle.color =
     circle.color.darker()
    circle.dropShadow = Shadow()
}

// when the touch ends on the circle,
 change its color to a random color.
circle.onTouchUp {
    circle.color = Color.random()
    circle.dropShadow = nil
}
```



```
// create a circle and make it
 draggable.
let circle = Circle(radius: 7.0)
circle.color = Color.purple
circle.draggable = true

// when the circle is touched, make it
 darker and give it a shadow.
circle.onTouchDown {
    circle.color =
     circle.color.darker()
    circle.dropShadow = Shadow()
}

// when the touch ends on the circle,
 change its color to a random color.
circle.onTouchUp {
    circle.color = Color.random()
    circle.dropShadow = nil
}
```



Spirals

This playground draws a type of animated spiral called a hypotrochoid.

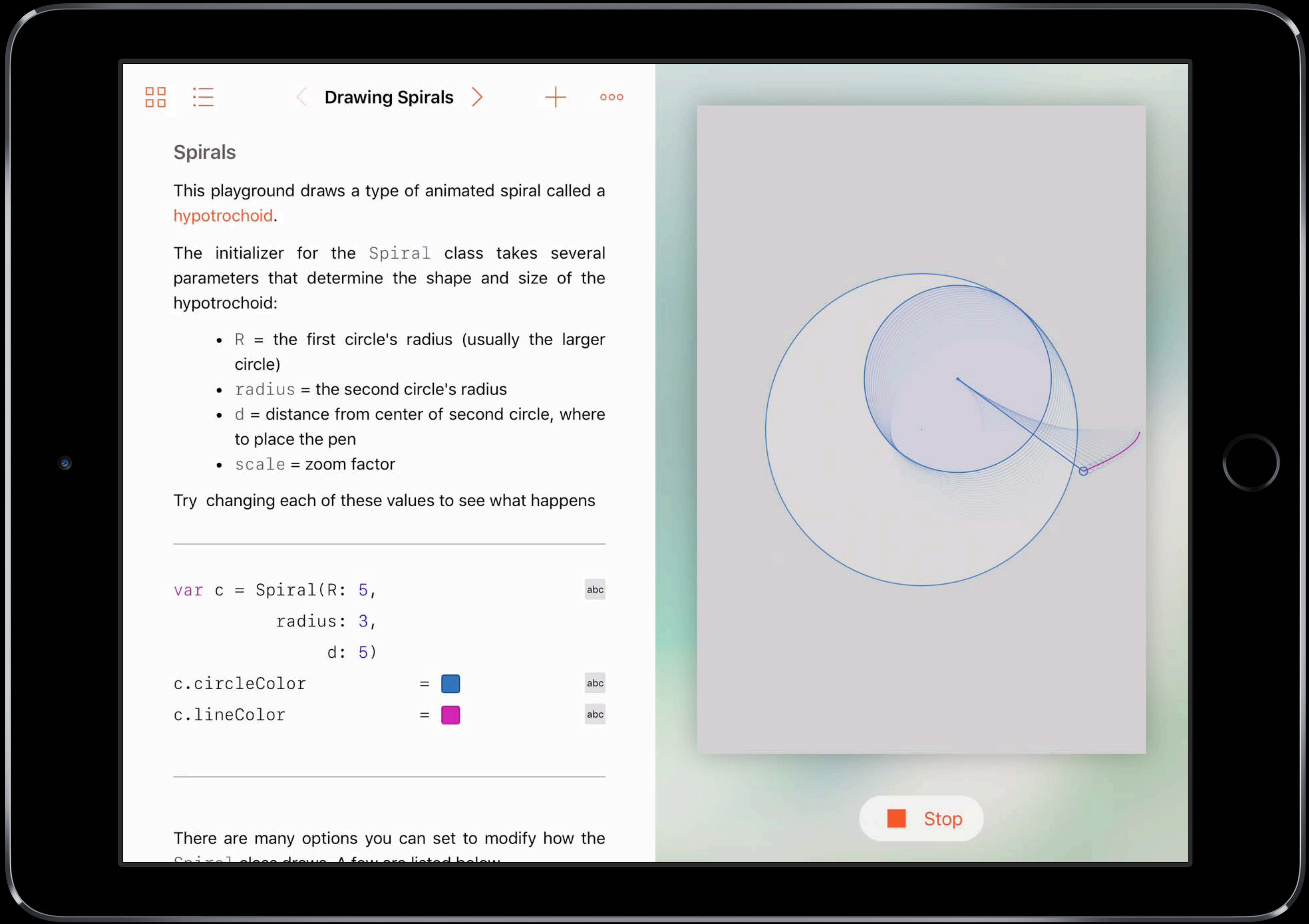
The initializer for the `Spiral` class takes several parameters that determine the shape and size of the hypotrochoid:

- `R` = the first circle's radius (usually the larger circle)
- `radius` = the second circle's radius
- `d` = distance from center of second circle, where to place the pen
- `scale` = zoom factor

Try changing each of these values to see what happens

```
var c = Spiral(R: 5,
          radius: 3,
               d: 5)
c.circleColor          = 🟦
c.lineColor            = 🟪
```

There are many options you can set to modify how the



Spirals

This playground draws a type of animated spiral called a hypotrochoid.

The initializer for the `Spiral` class takes several parameters that determine the shape and size of the hypotrochoid:

- `R` = the first circle's radius (usually the larger circle)
- `radius` = the second circle's radius
- `d` = distance from center of second circle, where to place the pen
- `scale` = zoom factor

Try changing each of these values to see what happens

```
var c = Spiral(R: 5,
          radius: 3,
               d: 5)
c.circleColor          = 🟦
c.lineColor            = 🟪
```

There are many options you can set to modify how the `Spiral` class draws. A few are listed below.

Using Swift Playgrounds

Using Swift Playgrounds

Authoring for Swift Playgrounds

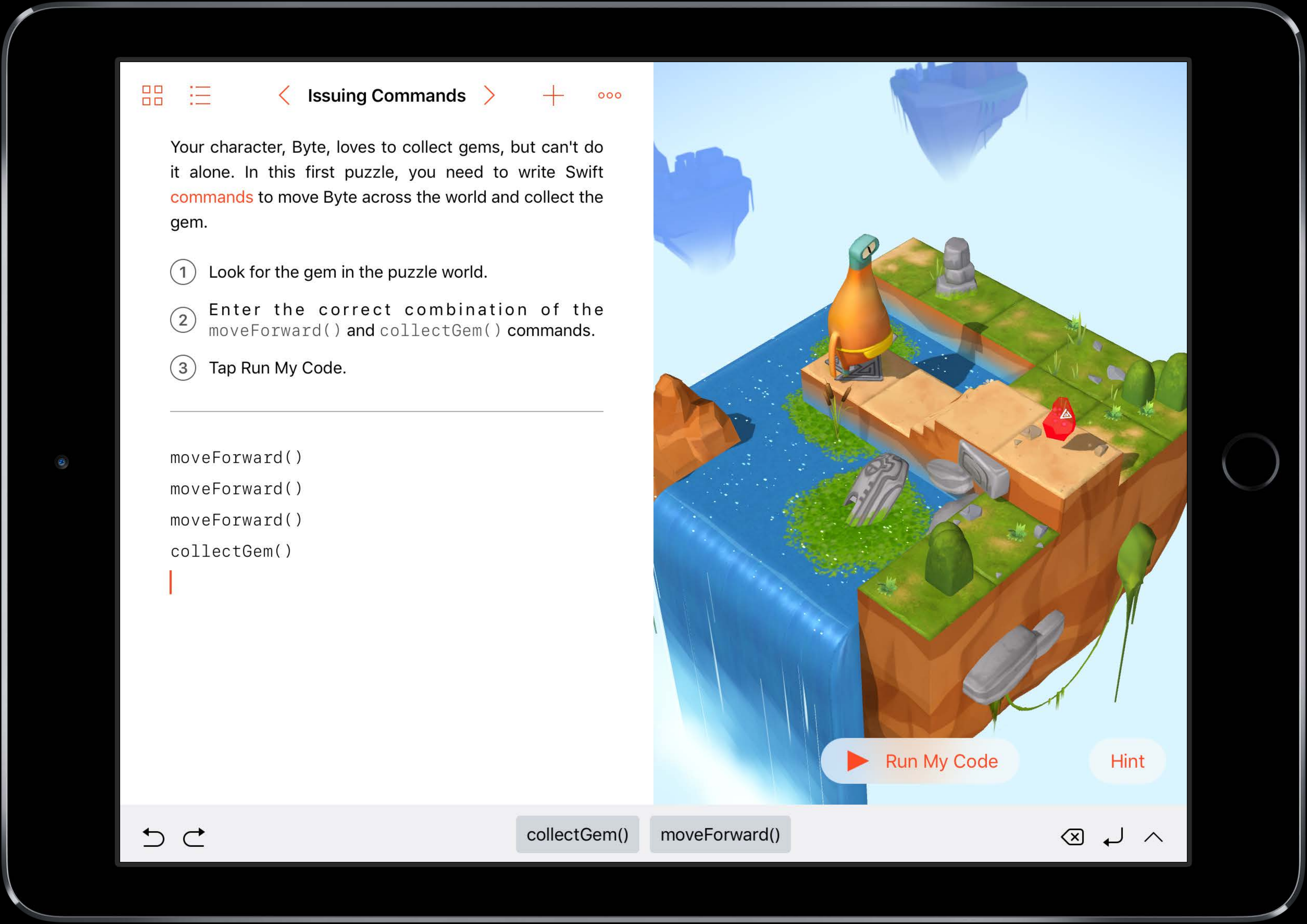
Using Swift Playgrounds

Authoring for Swift Playgrounds

Growing and Exploring

Using Swift Playgrounds

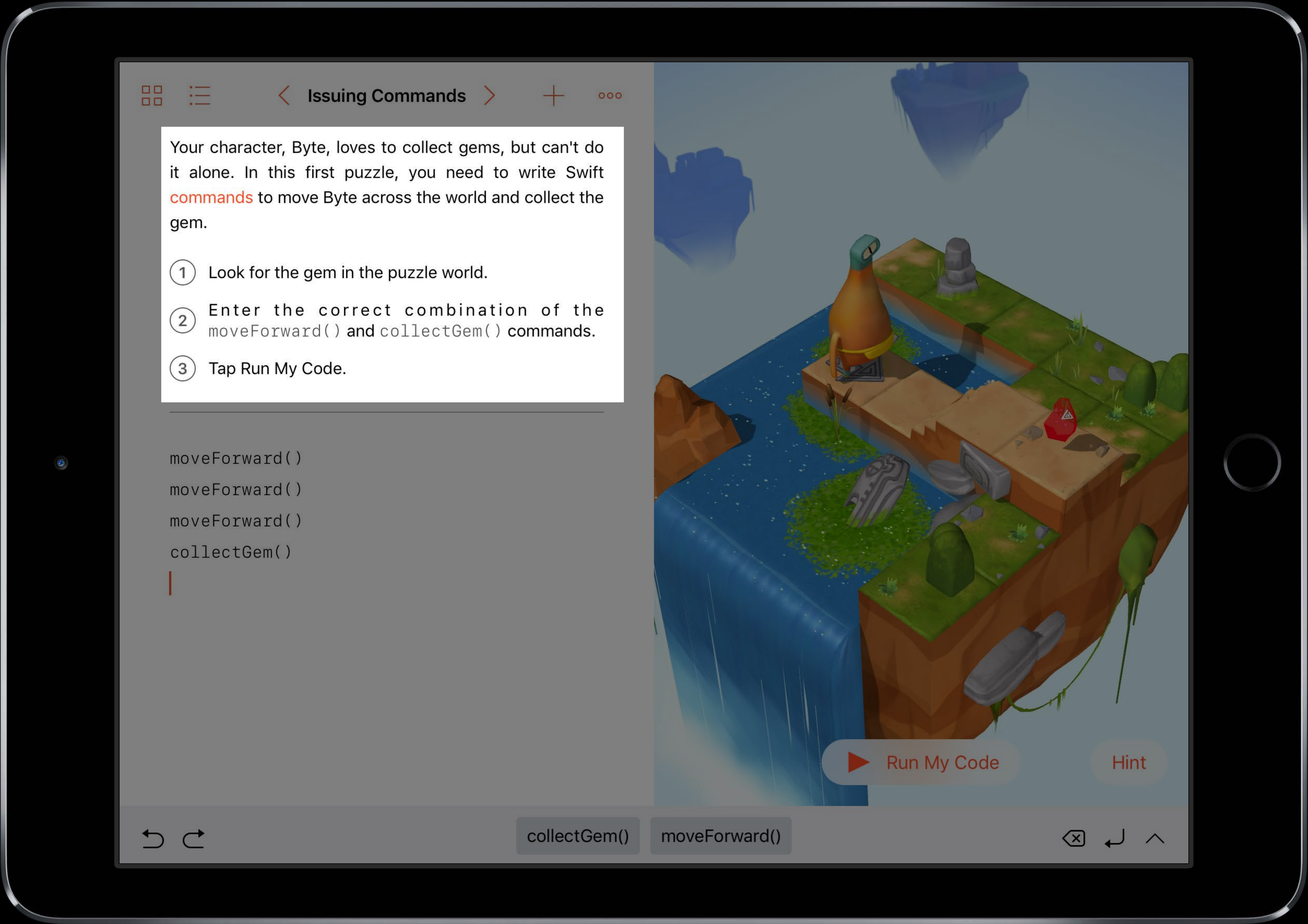
Maxwell Swadling
Playgrounds Engineer



Your character, Byte, loves to collect gems, but can't do it alone. In this first puzzle, you need to write Swift commands to move Byte across the world and collect the gem.

1. Look for the gem in the puzzle world.
2. Enter the correct combination of the `moveForward()` and `collectGem()` commands.
3. Tap Run My Code.

```
moveForward()
moveForward()
moveForward()
collectGem()
```

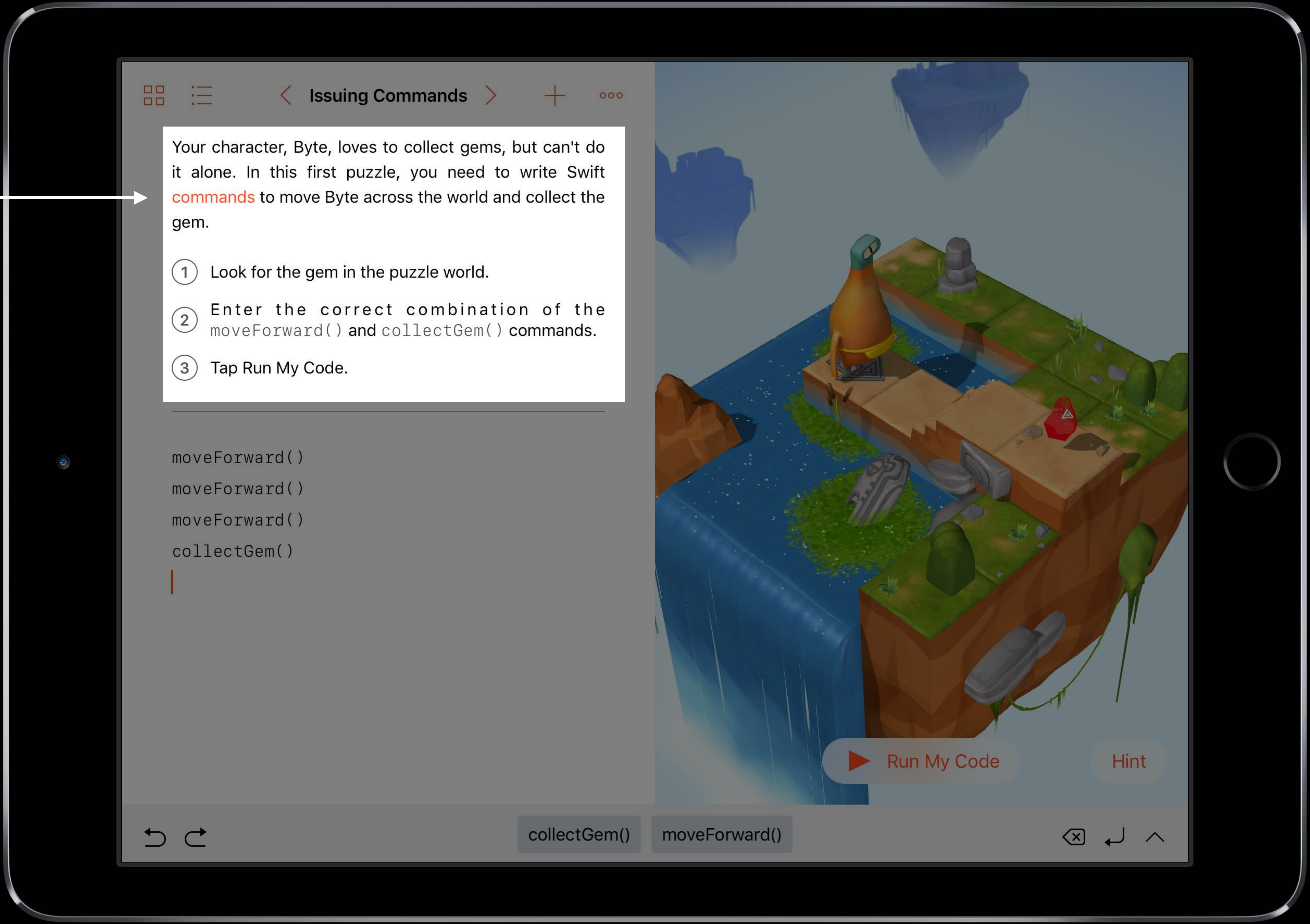

Issuing Commands

Your character, Byte, loves to collect gems, but can't do it alone. In this first puzzle, you need to write Swift commands to move Byte across the world and collect the gem.

1. Look for the gem in the puzzle world.
2. Enter the correct combination of the `moveForward()` and `collectGem()` commands.
3. Tap Run My Code.

```
moveForward()
moveForward()
moveForward()
collectGem()
```

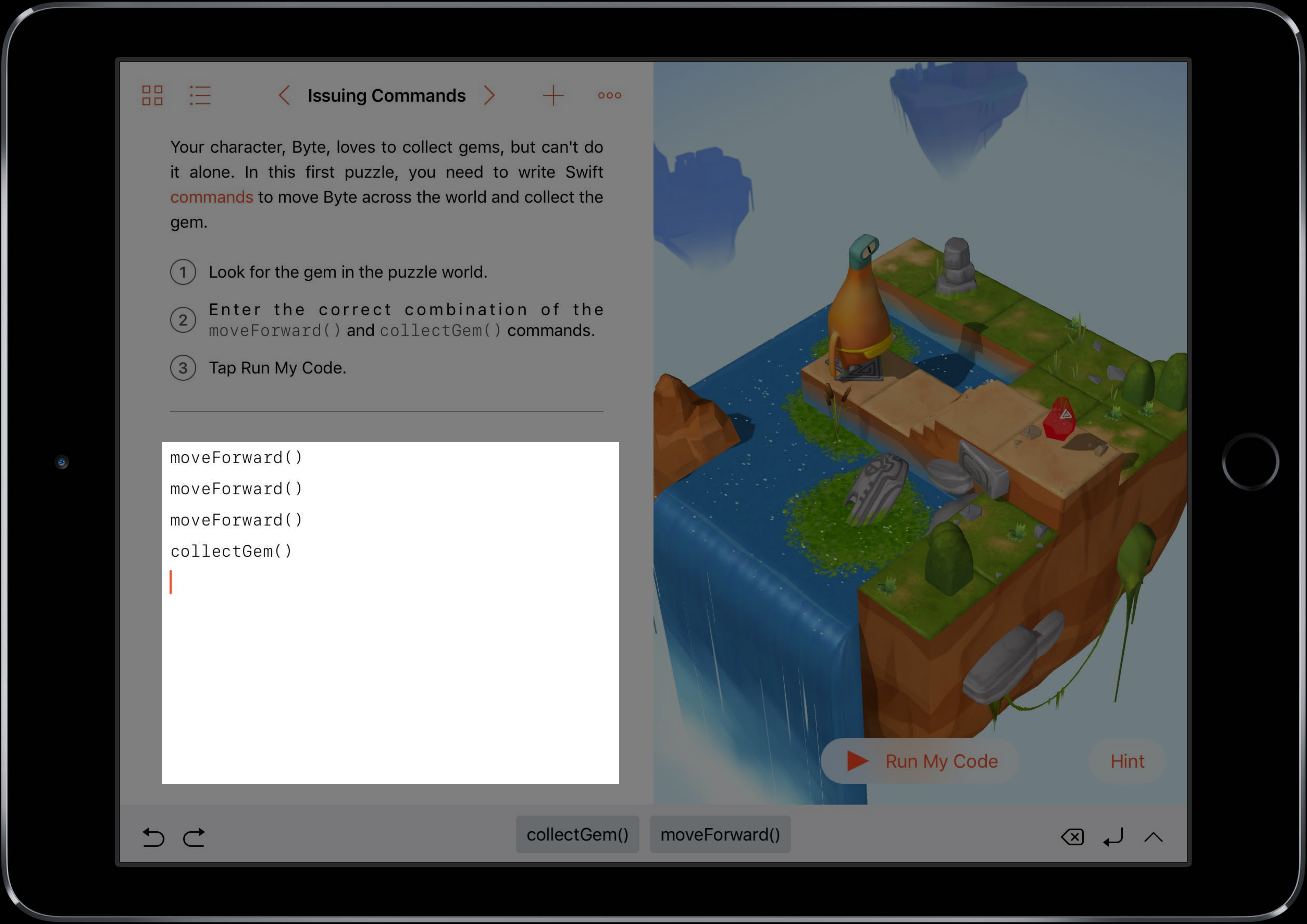
Playground Markup



Your character, Byte, loves to collect gems, but can't do it alone. In this first puzzle, you need to write Swift commands to move Byte across the world and collect the gem.

1. Look for the gem in the puzzle world.
2. Enter the correct combination of the `moveForward()` and `collectGem()` commands.
3. Tap Run My Code.

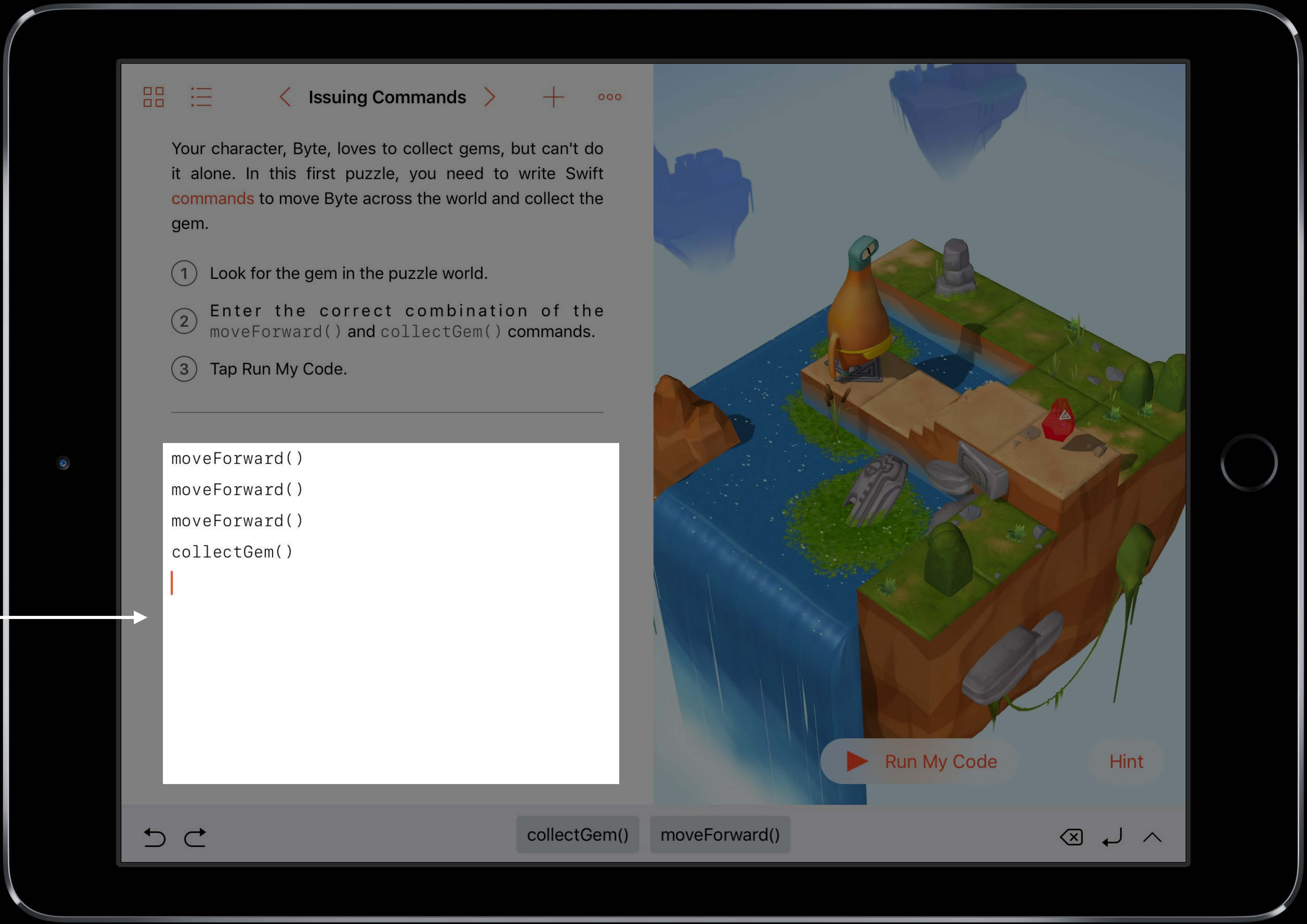
```
moveForward()
moveForward()
moveForward()
collectGem()
```



Your character, Byte, loves to collect gems, but can't do it alone. In this first puzzle, you need to write Swift commands to move Byte across the world and collect the gem.

1. Look for the gem in the puzzle world.
2. Enter the correct combination of the `moveForward()` and `collectGem()` commands.
3. Tap Run My Code.

```
moveForward()
moveForward()
moveForward()
collectGem()
```

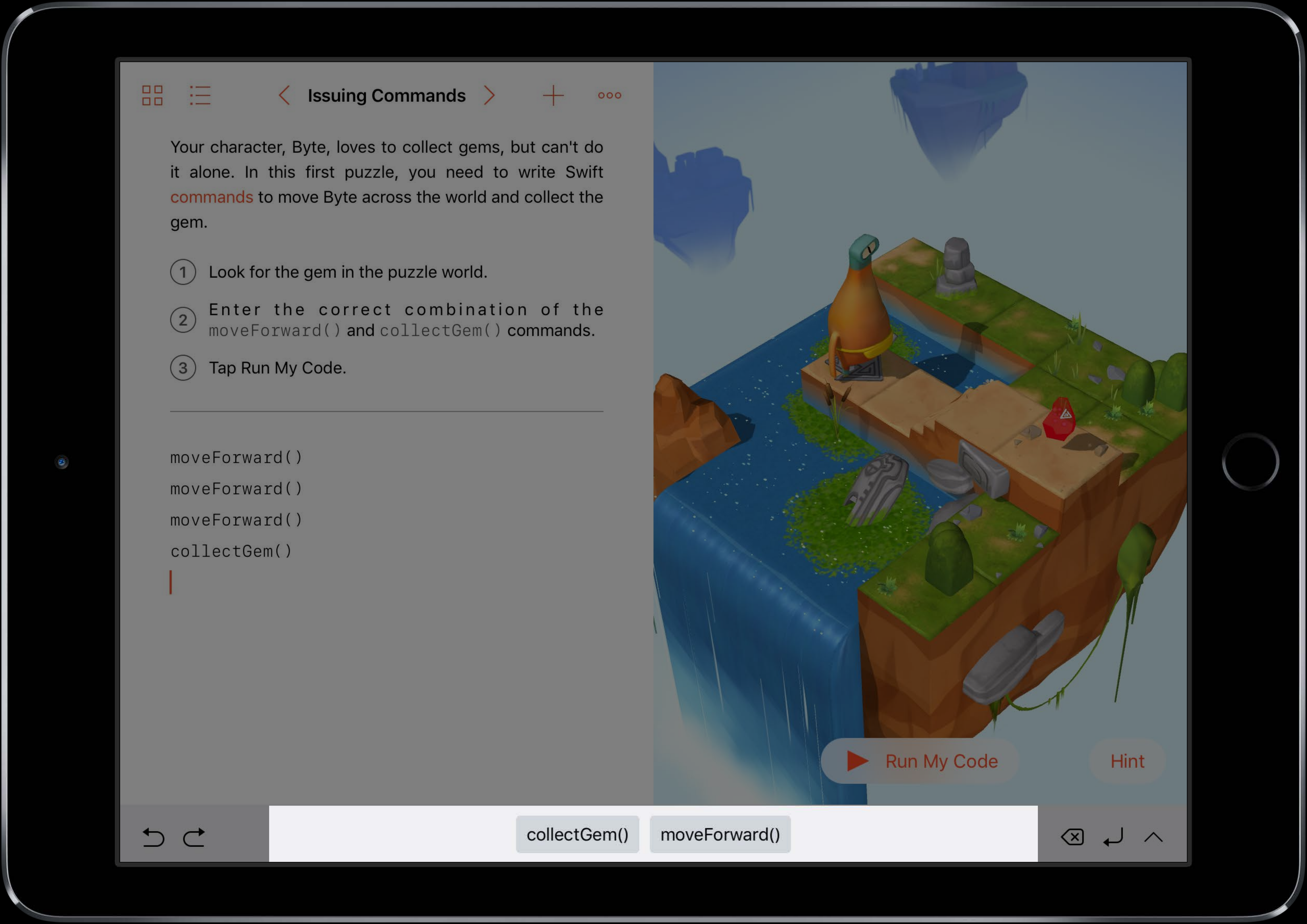


Your character, Byte, loves to collect gems, but can't do it alone. In this first puzzle, you need to write Swift commands to move Byte across the world and collect the gem.

1. Look for the gem in the puzzle world.
2. Enter the correct combination of the `moveForward()` and `collectGem()` commands.
3. Tap Run My Code.

```
moveForward()
moveForward()
moveForward()
collectGem()
```

Source Code

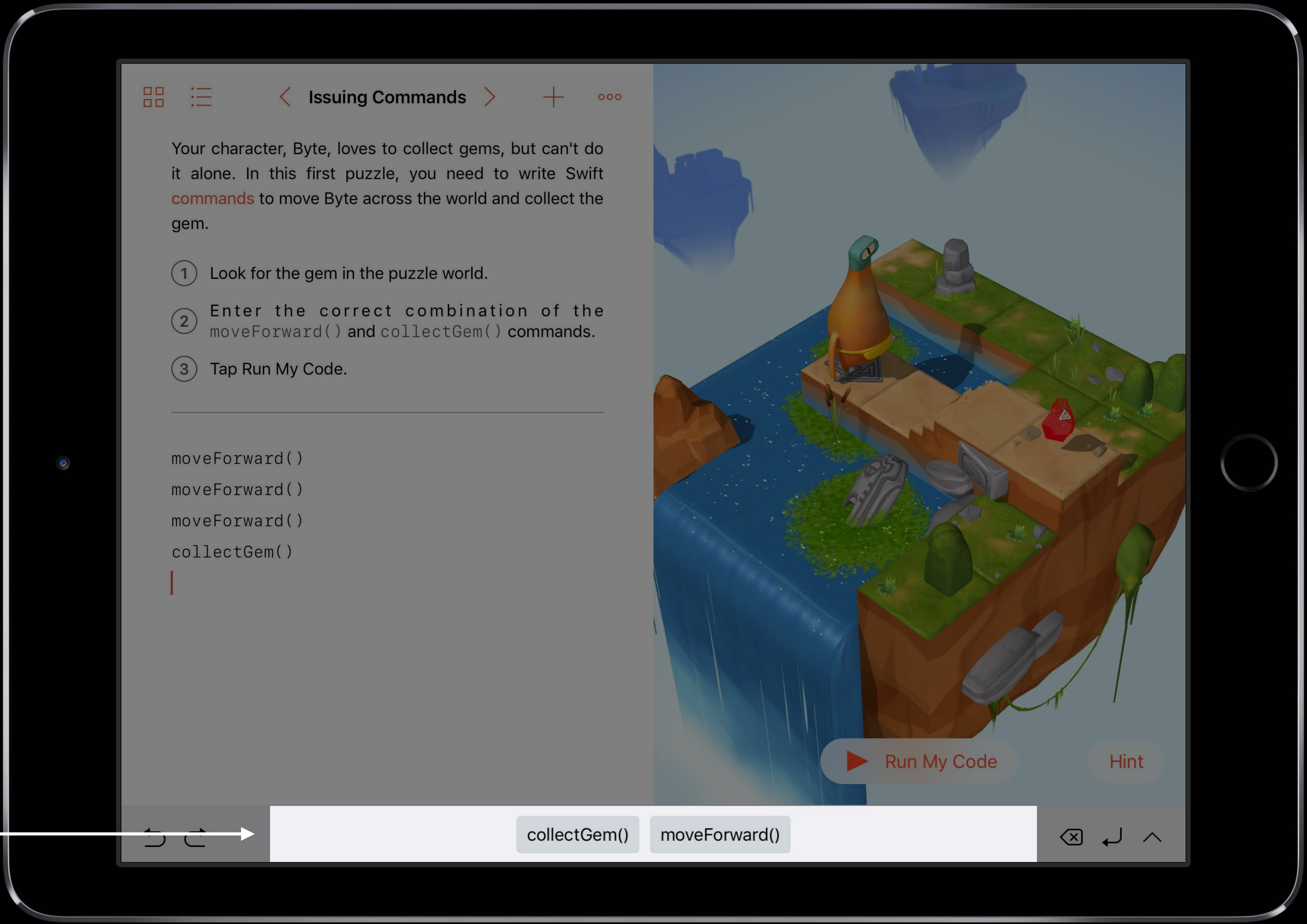


Issuing Commands

Your character, Byte, loves to collect gems, but can't do it alone. In this first puzzle, you need to write Swift commands to move Byte across the world and collect the gem.

1. Look for the gem in the puzzle world.
2. Enter the correct combination of the `moveForward()` and `collectGem()` commands.
3. Tap Run My Code.

```
moveForward()
moveForward()
moveForward()
collectGem()
```

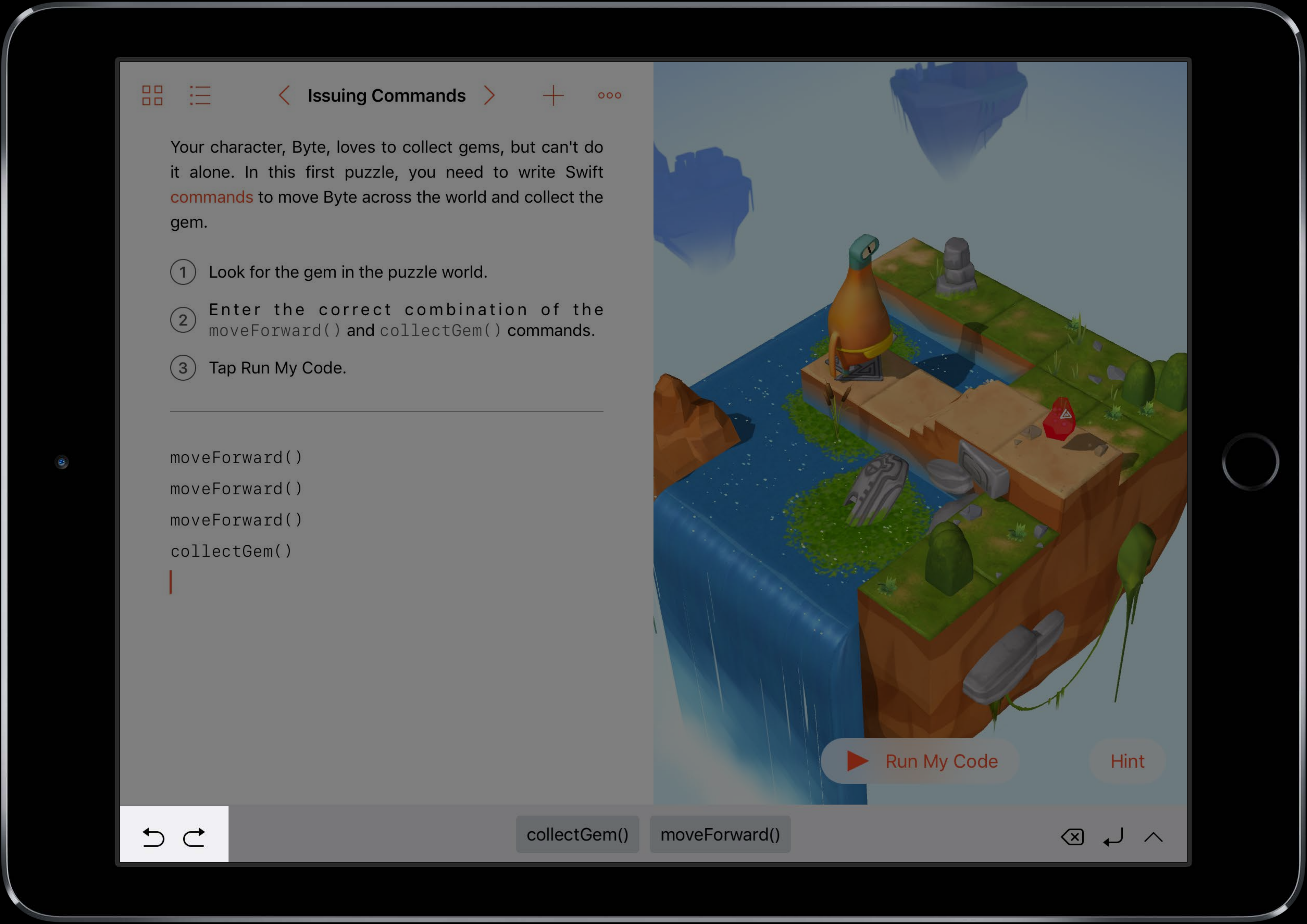


Your character, Byte, loves to collect gems, but can't do it alone. In this first puzzle, you need to write Swift commands to move Byte across the world and collect the gem.

1. Look for the gem in the puzzle world.
2. Enter the correct combination of the `moveForward()` and `collectGem()` commands.
3. Tap Run My Code.

```
moveForward()
moveForward()
moveForward()
collectGem()
```

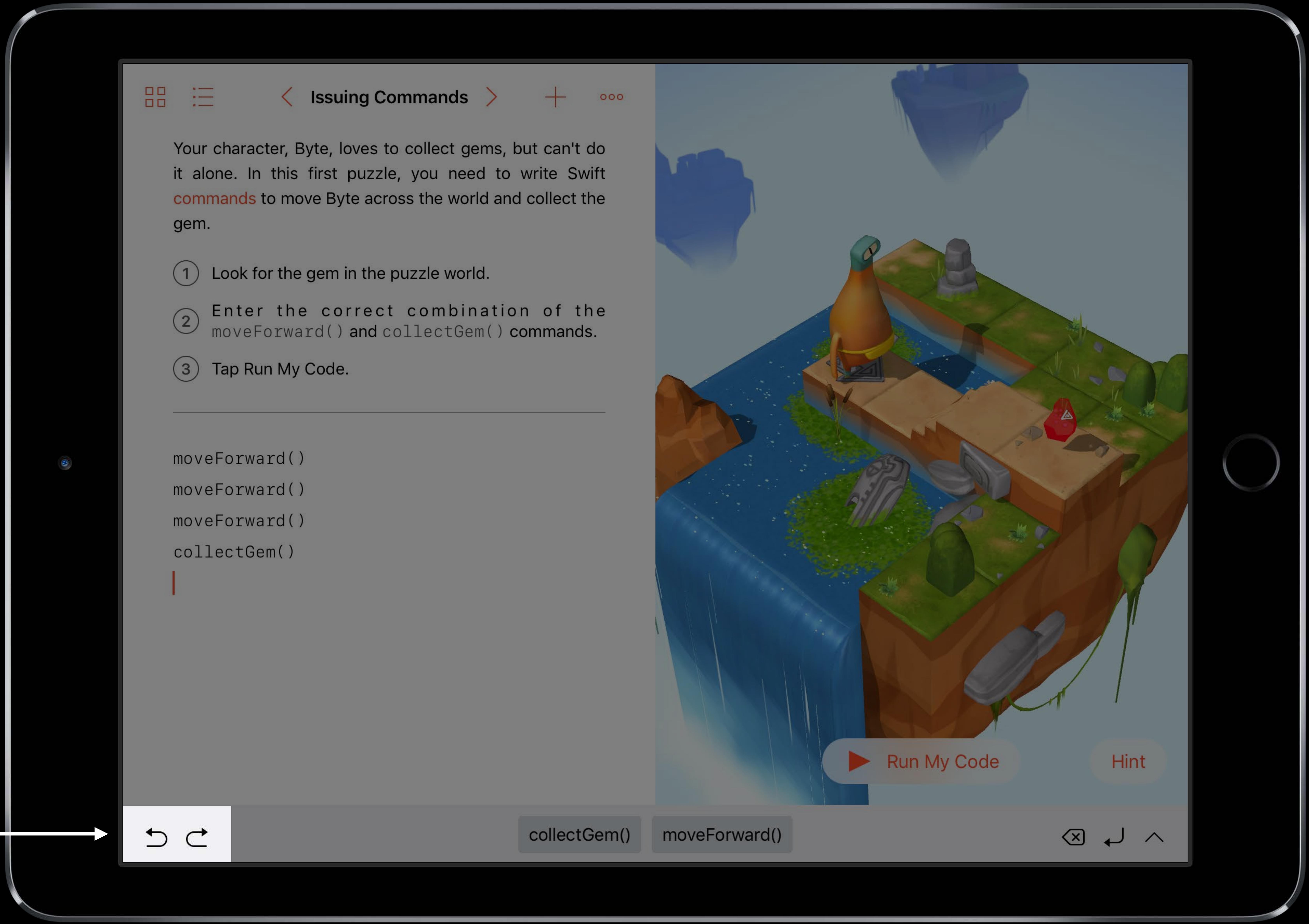
Code Completion



Your character, Byte, loves to collect gems, but can't do it alone. In this first puzzle, you need to write Swift commands to move Byte across the world and collect the gem.

1. Look for the gem in the puzzle world.
2. Enter the correct combination of the `moveForward()` and `collectGem()` commands.
3. Tap Run My Code.

```
moveForward()
moveForward()
moveForward()
collectGem()
```

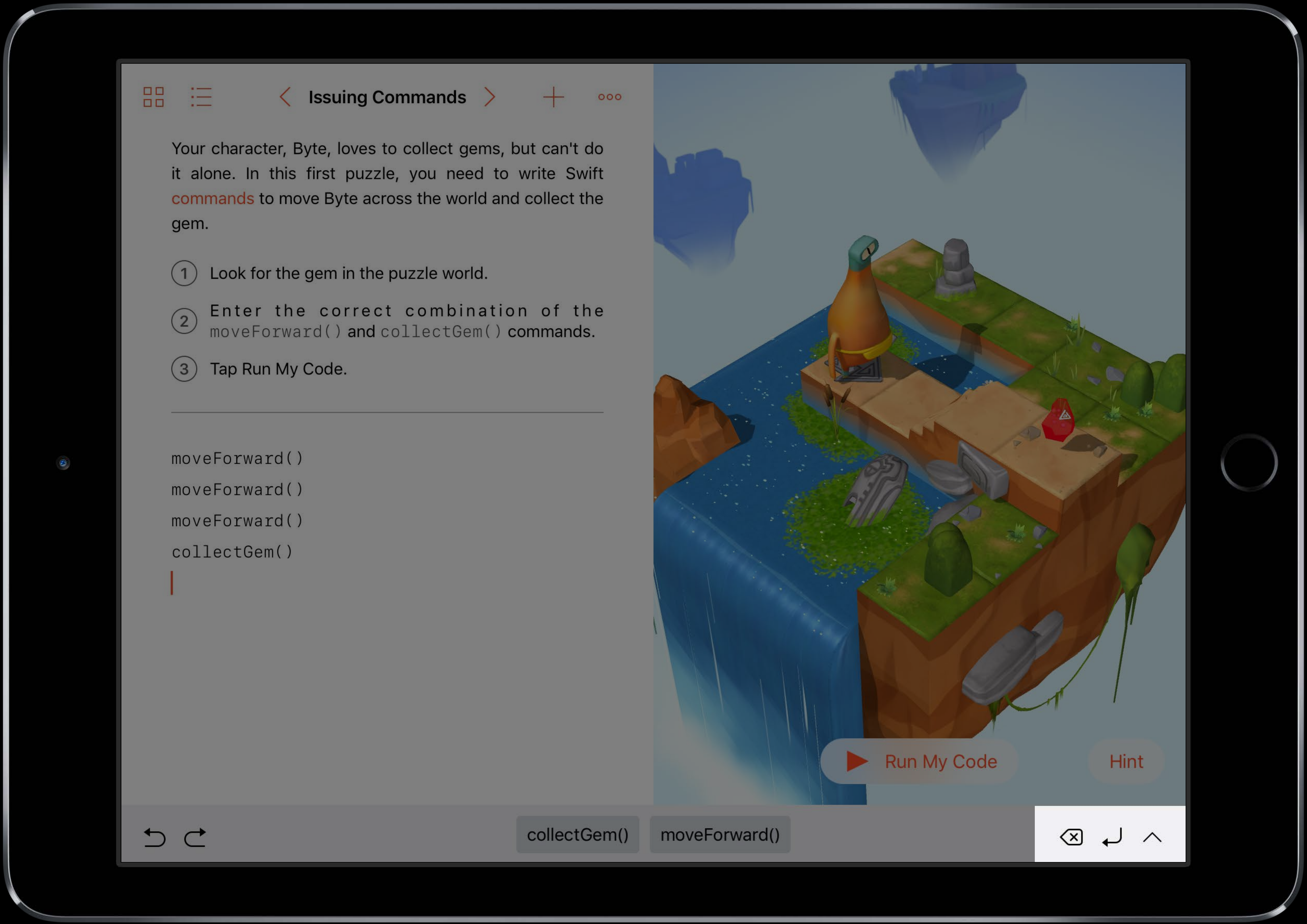


Your character, Byte, loves to collect gems, but can't do it alone. In this first puzzle, you need to write Swift commands to move Byte across the world and collect the gem.

1. Look for the gem in the puzzle world.
2. Enter the correct combination of the `moveForward()` and `collectGem()` commands.
3. Tap Run My Code.

```
moveForward()
moveForward()
moveForward()
collectGem()
```

Undo/Redo

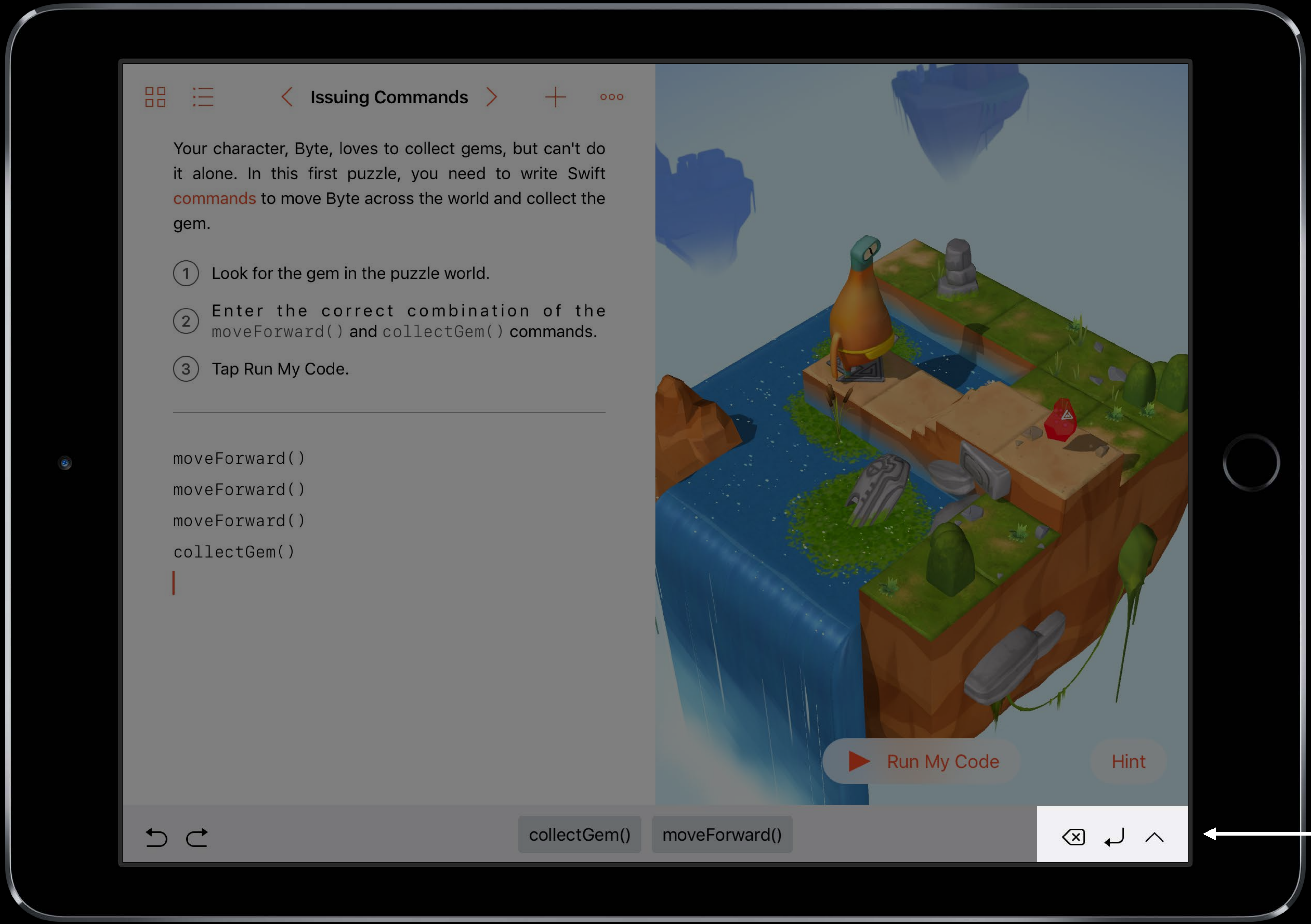


Issuing Commands

Your character, Byte, loves to collect gems, but can't do it alone. In this first puzzle, you need to write Swift commands to move Byte across the world and collect the gem.

1. Look for the gem in the puzzle world.
2. Enter the correct combination of the `moveForward()` and `collectGem()` commands.
3. Tap Run My Code.

```
moveForward()
moveForward()
moveForward()
collectGem()
```



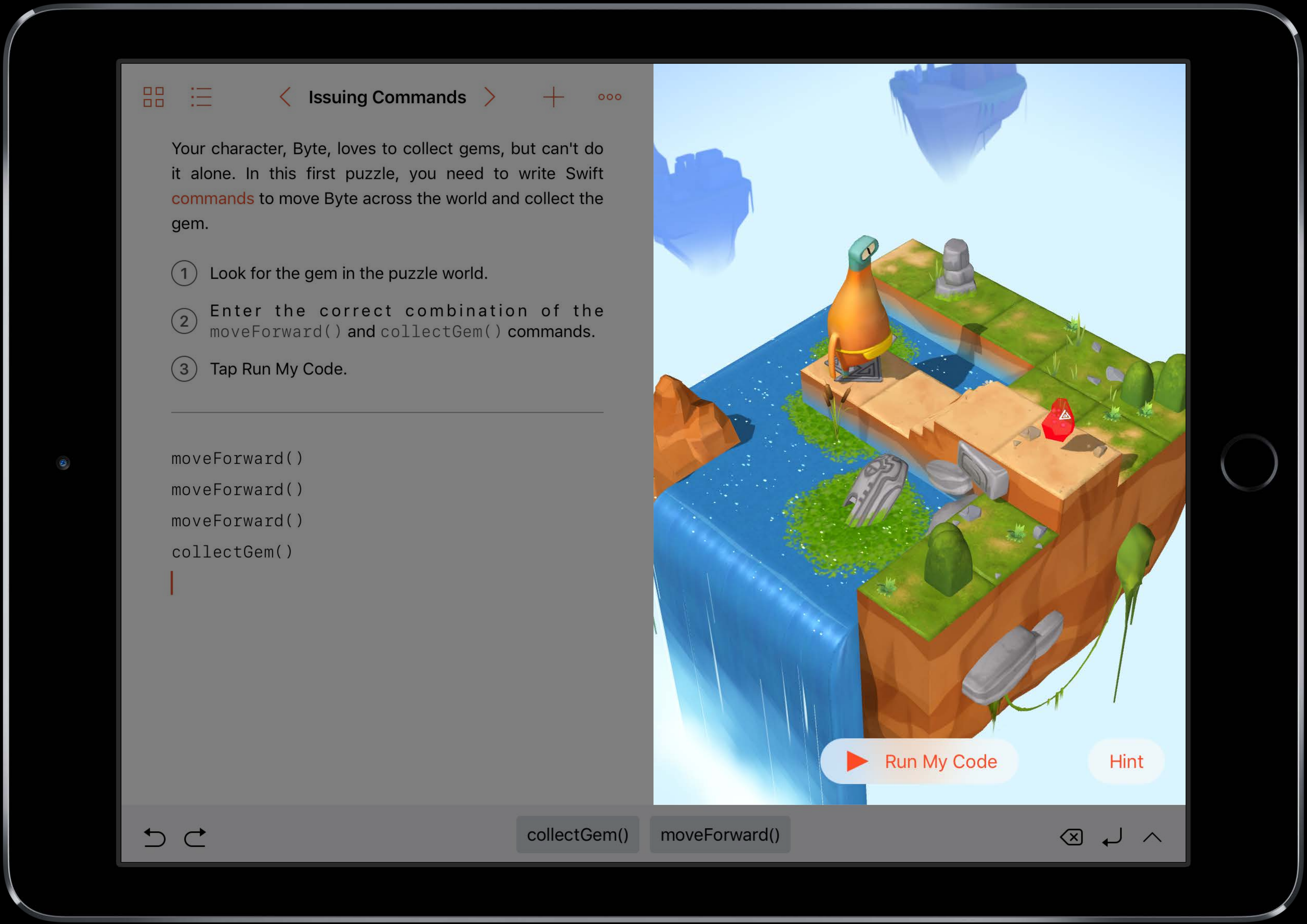
Issuing Commands

Your character, Byte, loves to collect gems, but can't do it alone. In this first puzzle, you need to write Swift commands to move Byte across the world and collect the gem.

1. Look for the gem in the puzzle world.
2. Enter the correct combination of the `moveForward()` and `collectGem()` commands.
3. Tap Run My Code.

```
moveForward()
moveForward()
moveForward()
collectGem()
```

Shortcuts

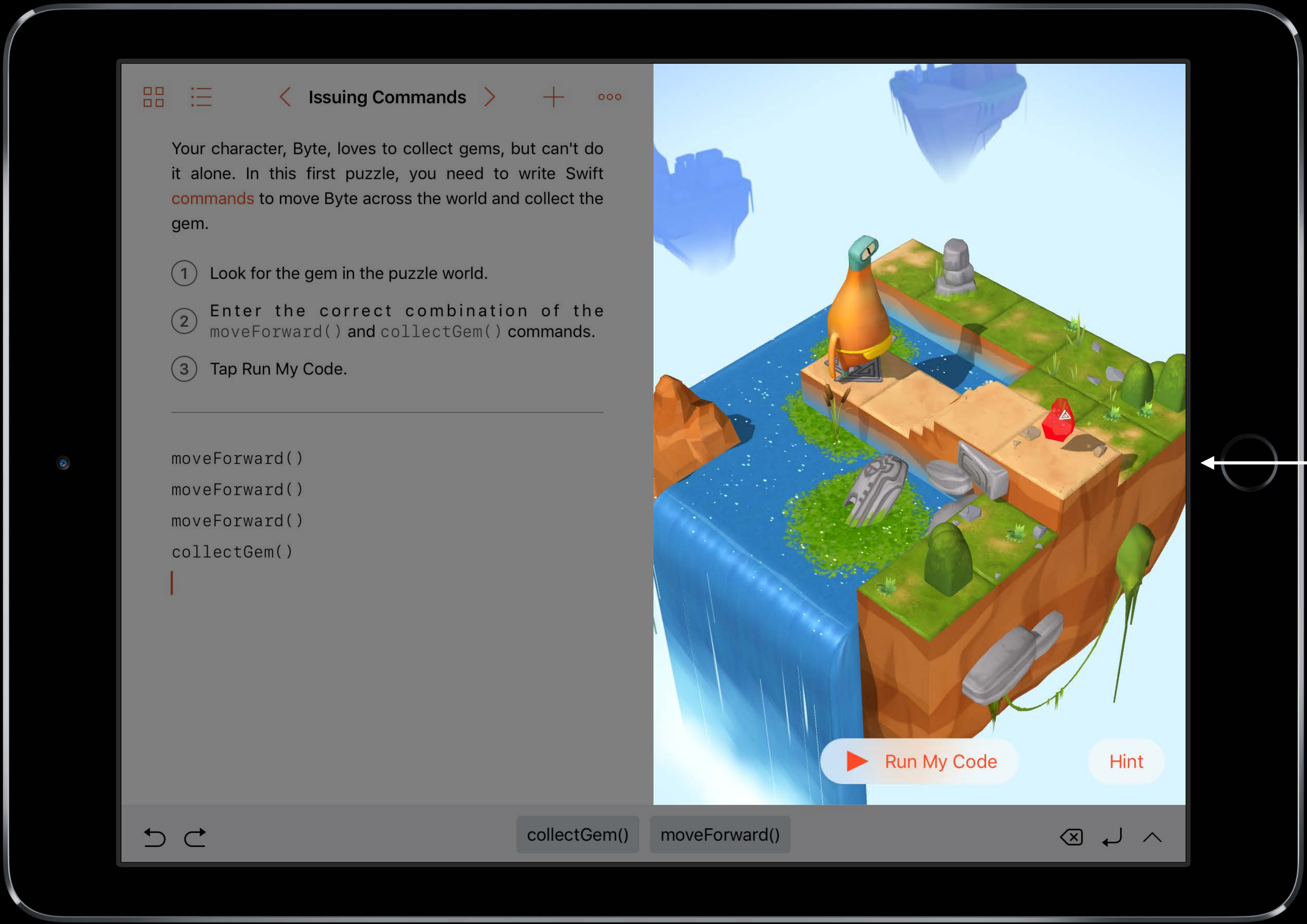


Issuing Commands

Your character, Byte, loves to collect gems, but can't do it alone. In this first puzzle, you need to write Swift commands to move Byte across the world and collect the gem.

1. Look for the gem in the puzzle world.
2. Enter the correct combination of the `moveForward()` and `collectGem()` commands.
3. Tap Run My Code.

```
moveForward()
moveForward()
moveForward()
collectGem()
```



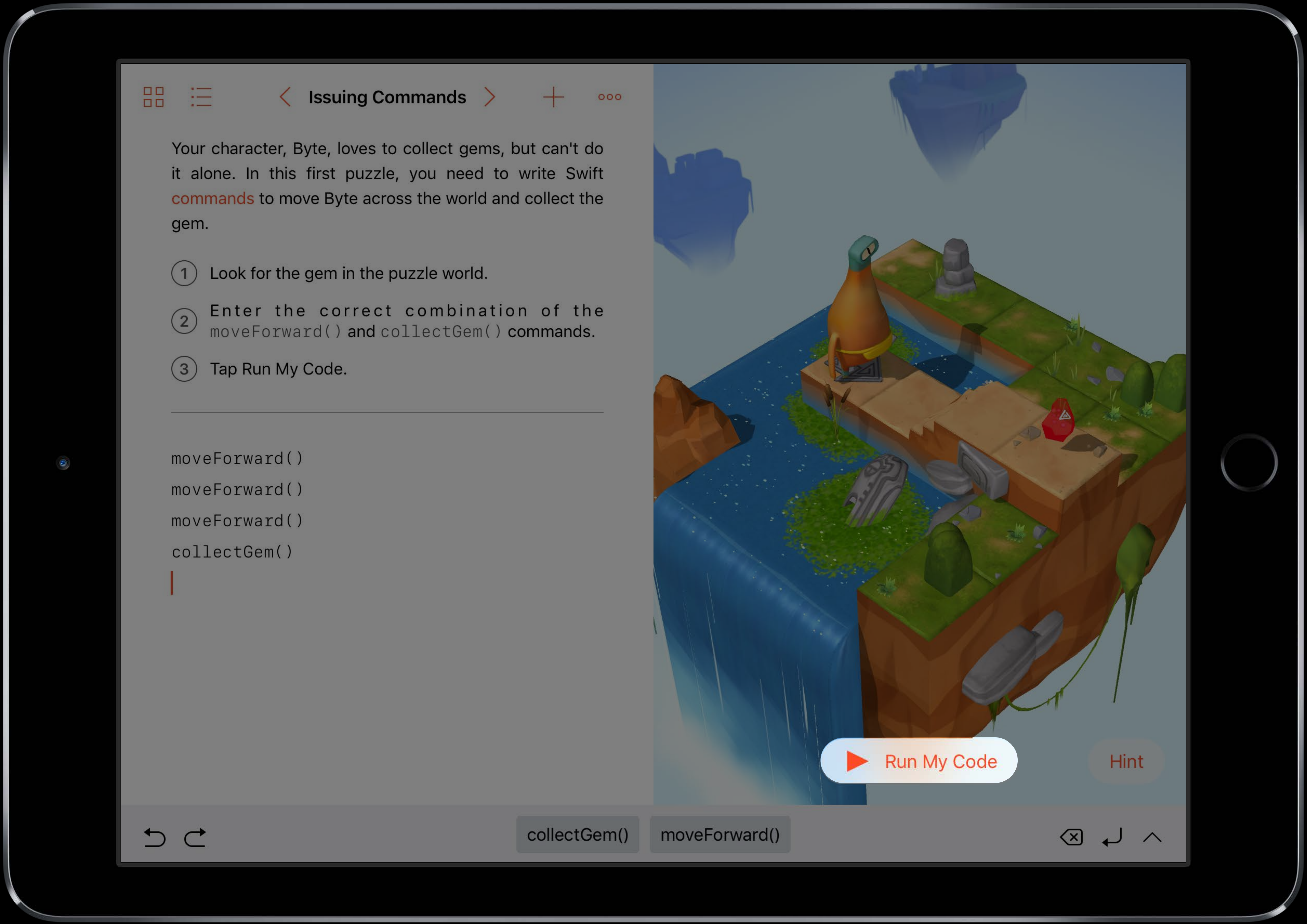
Issuing Commands

Your character, Byte, loves to collect gems, but can't do it alone. In this first puzzle, you need to write Swift commands to move Byte across the world and collect the gem.

1. Look for the gem in the puzzle world.
2. Enter the correct combination of the `moveForward()` and `collectGem()` commands.
3. Tap Run My Code.

```
moveForward()
moveForward()
moveForward()
collectGem()
```

Live View

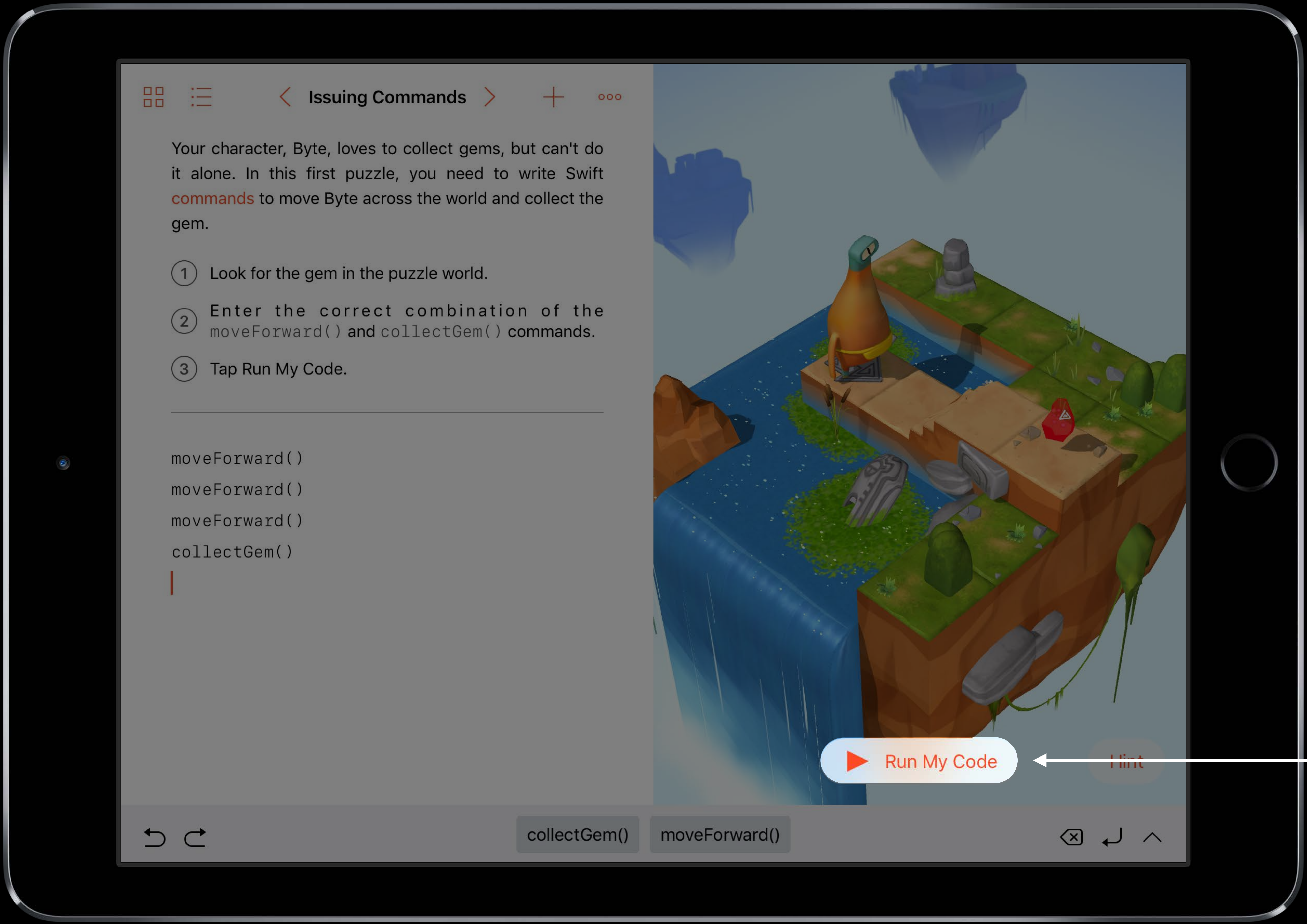


Issuing Commands

Your character, Byte, loves to collect gems, but can't do it alone. In this first puzzle, you need to write Swift commands to move Byte across the world and collect the gem.

1. Look for the gem in the puzzle world.
2. Enter the correct combination of the `moveForward()` and `collectGem()` commands.
3. Tap Run My Code.

```
moveForward()
moveForward()
moveForward()
collectGem()
```

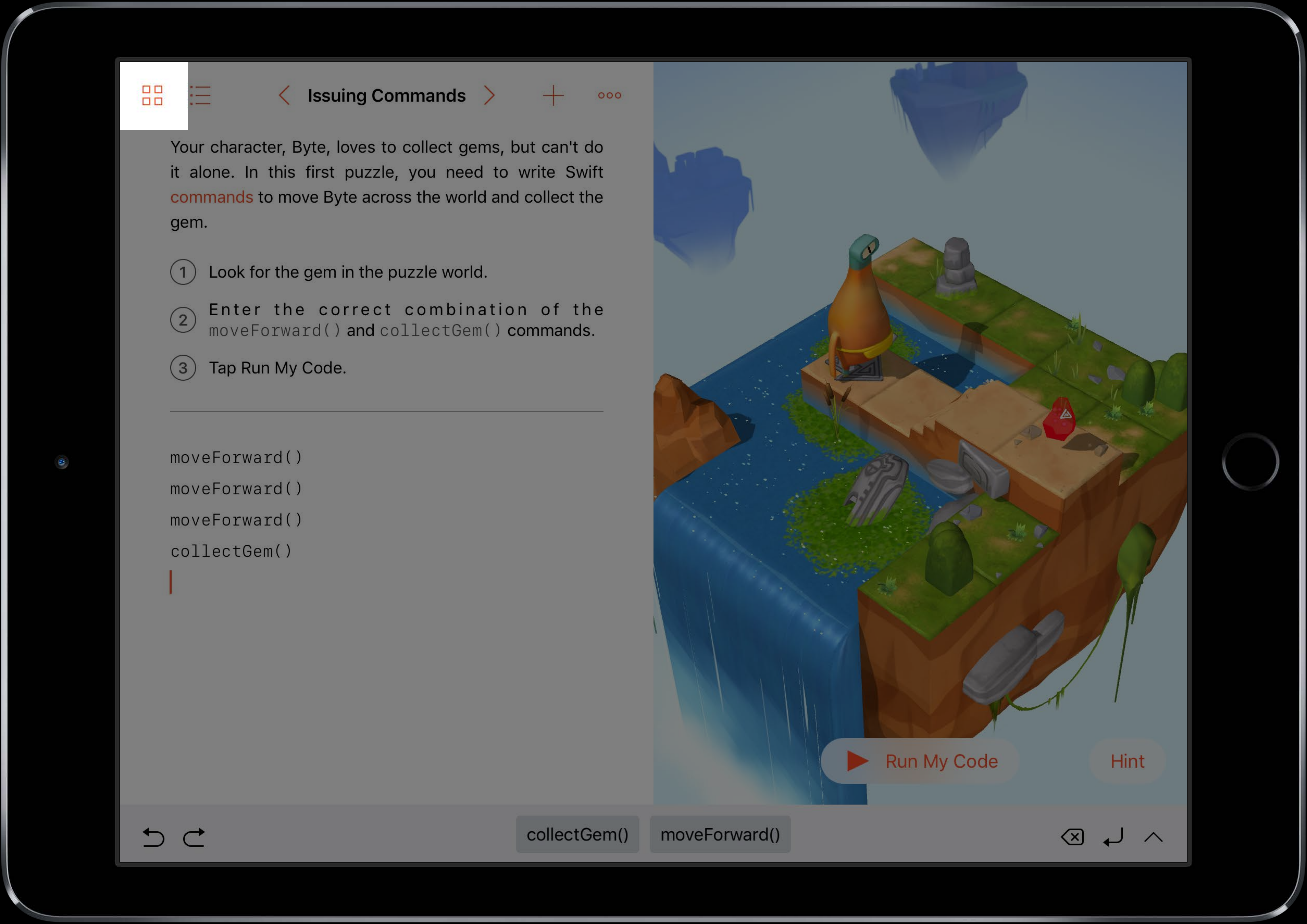


Your character, Byte, loves to collect gems, but can't do it alone. In this first puzzle, you need to write Swift commands to move Byte across the world and collect the gem.

1. Look for the gem in the puzzle world.
2. Enter the correct combination of the `moveForward()` and `collectGem()` commands.
3. Tap Run My Code.

```
moveForward()
moveForward()
moveForward()
collectGem()
```

Run Button

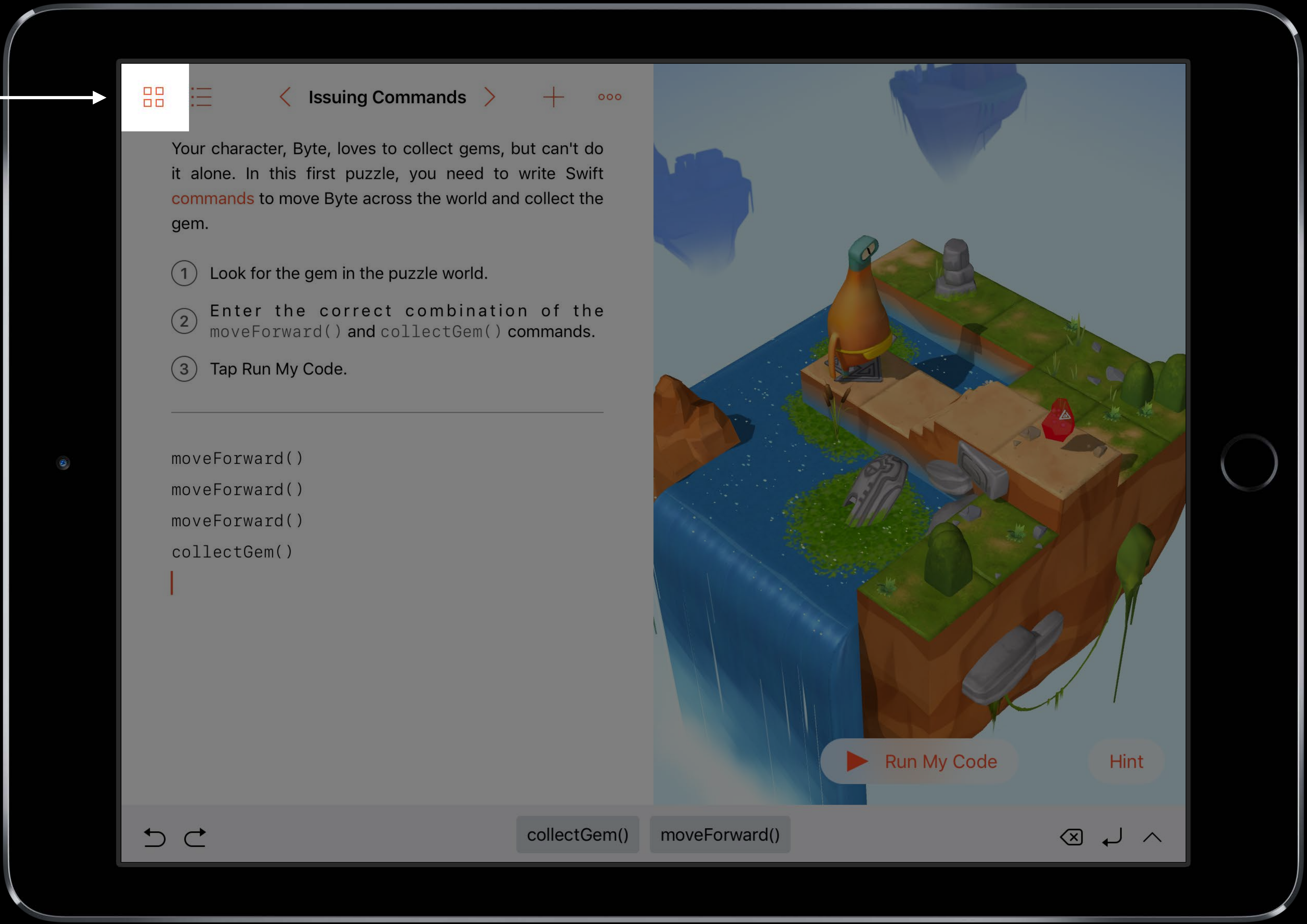


Your character, Byte, loves to collect gems, but can't do it alone. In this first puzzle, you need to write Swift commands to move Byte across the world and collect the gem.

1. Look for the gem in the puzzle world.
2. Enter the correct combination of the `moveForward()` and `collectGem()` commands.
3. Tap Run My Code.

```
moveForward()
moveForward()
moveForward()
collectGem()
```

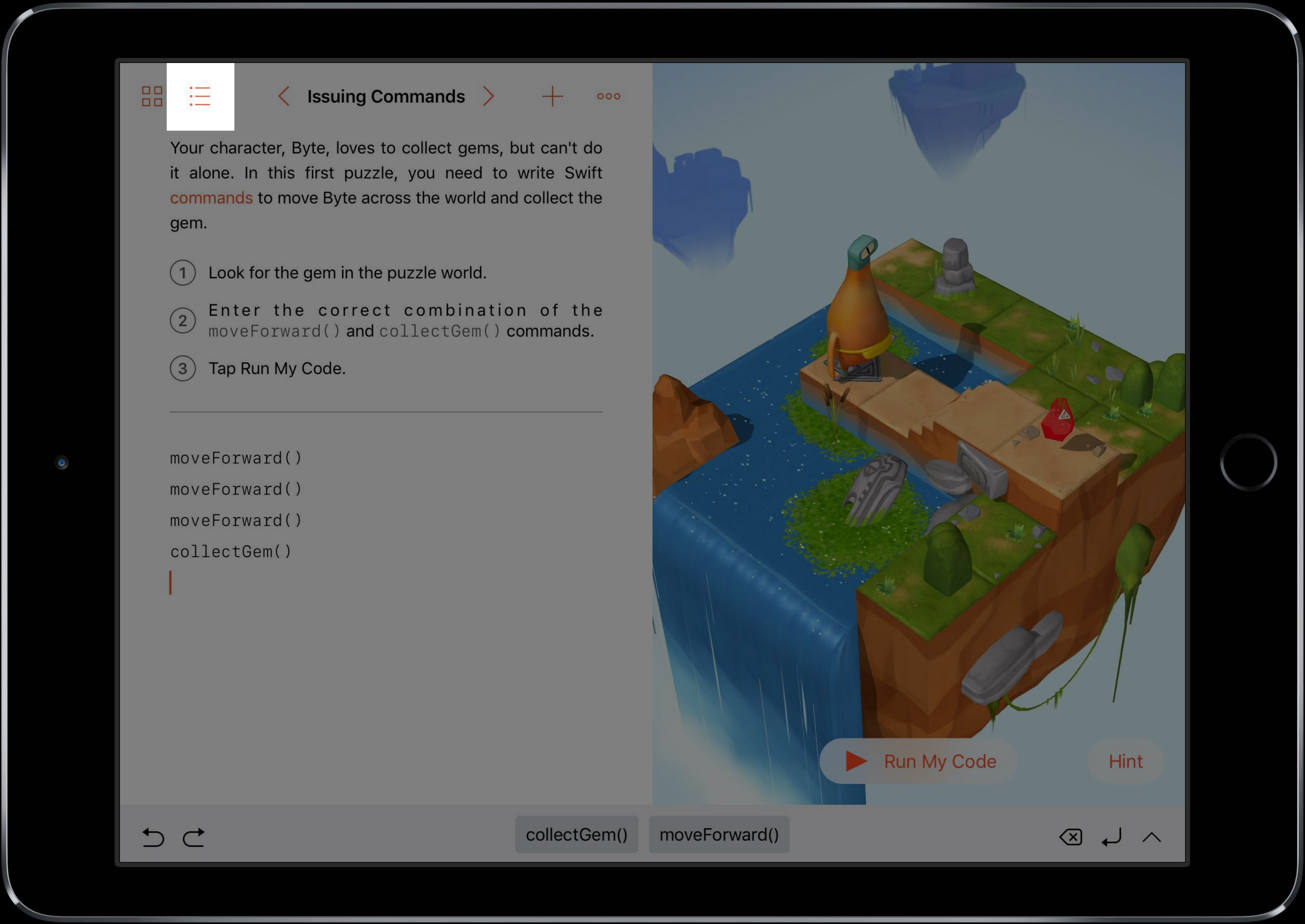
Documents



Your character, Byte, loves to collect gems, but can't do it alone. In this first puzzle, you need to write Swift commands to move Byte across the world and collect the gem.

1. Look for the gem in the puzzle world.
2. Enter the correct combination of the `moveForward()` and `collectGem()` commands.
3. Tap Run My Code.

```
moveForward()
moveForward()
moveForward()
collectGem()
```

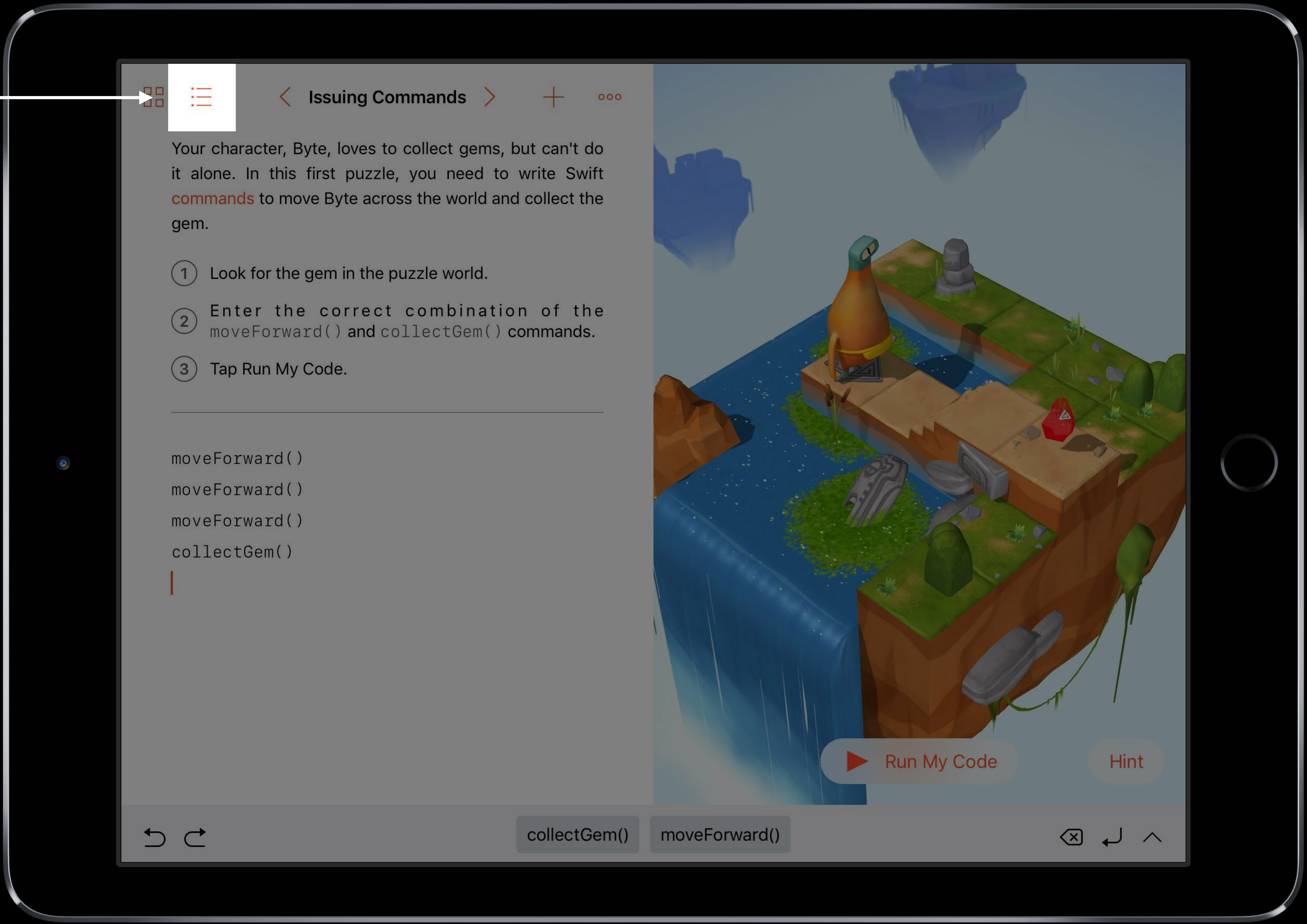

Issuing Commands

Your character, Byte, loves to collect gems, but can't do it alone. In this first puzzle, you need to write Swift commands to move Byte across the world and collect the gem.

1. Look for the gem in the puzzle world.
2. Enter the correct combination of the `moveForward()` and `collectGem()` commands.
3. Tap Run My Code.

```
moveForward()
moveForward()
moveForward()
collectGem()
```

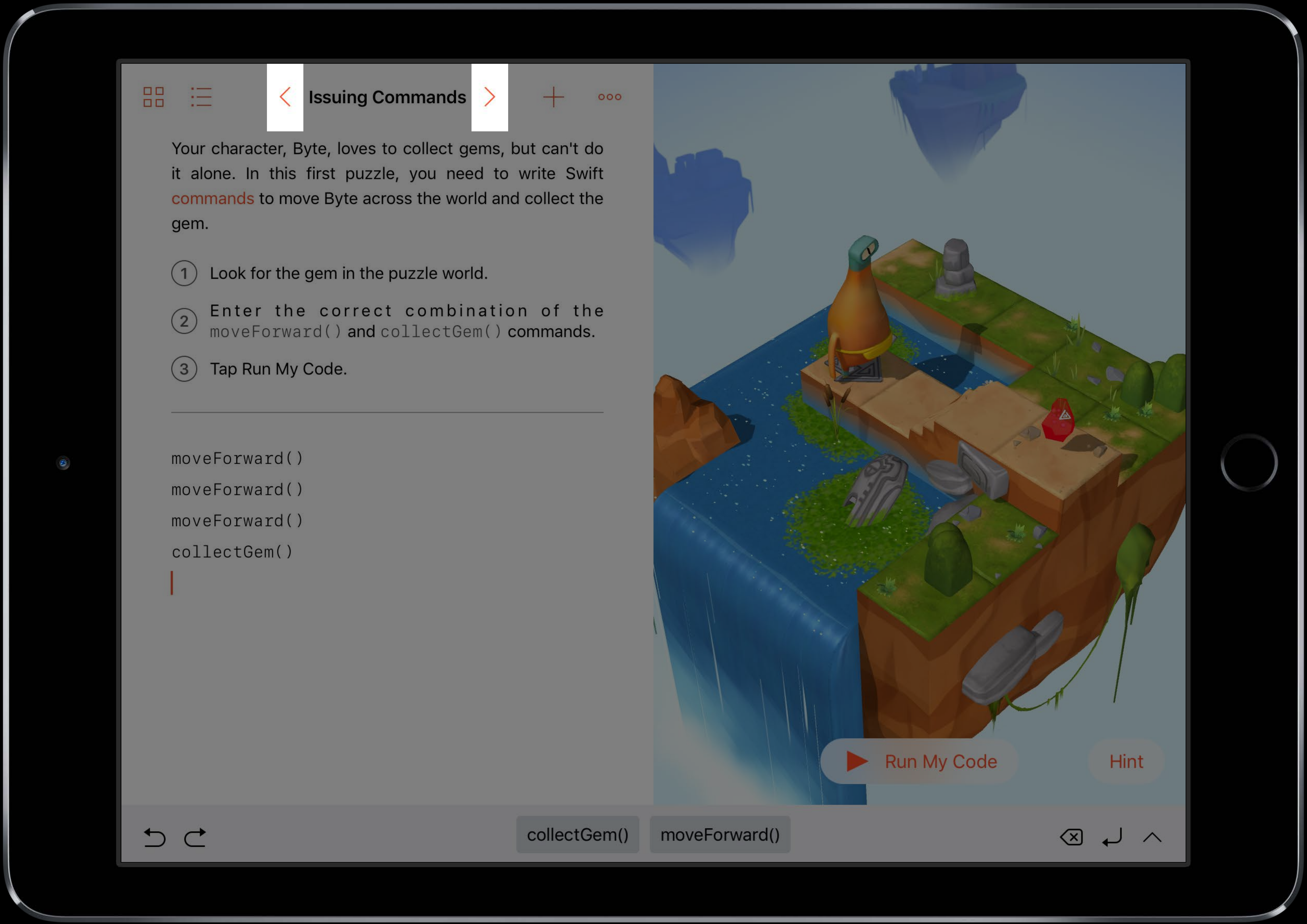
Table of Contents



Your character, Byte, loves to collect gems, but can't do it alone. In this first puzzle, you need to write Swift commands to move Byte across the world and collect the gem.

1. Look for the gem in the puzzle world.
2. Enter the correct combination of the `moveForward()` and `collectGem()` commands.
3. Tap Run My Code.

```
moveForward()
moveForward()
moveForward()
collectGem()
```

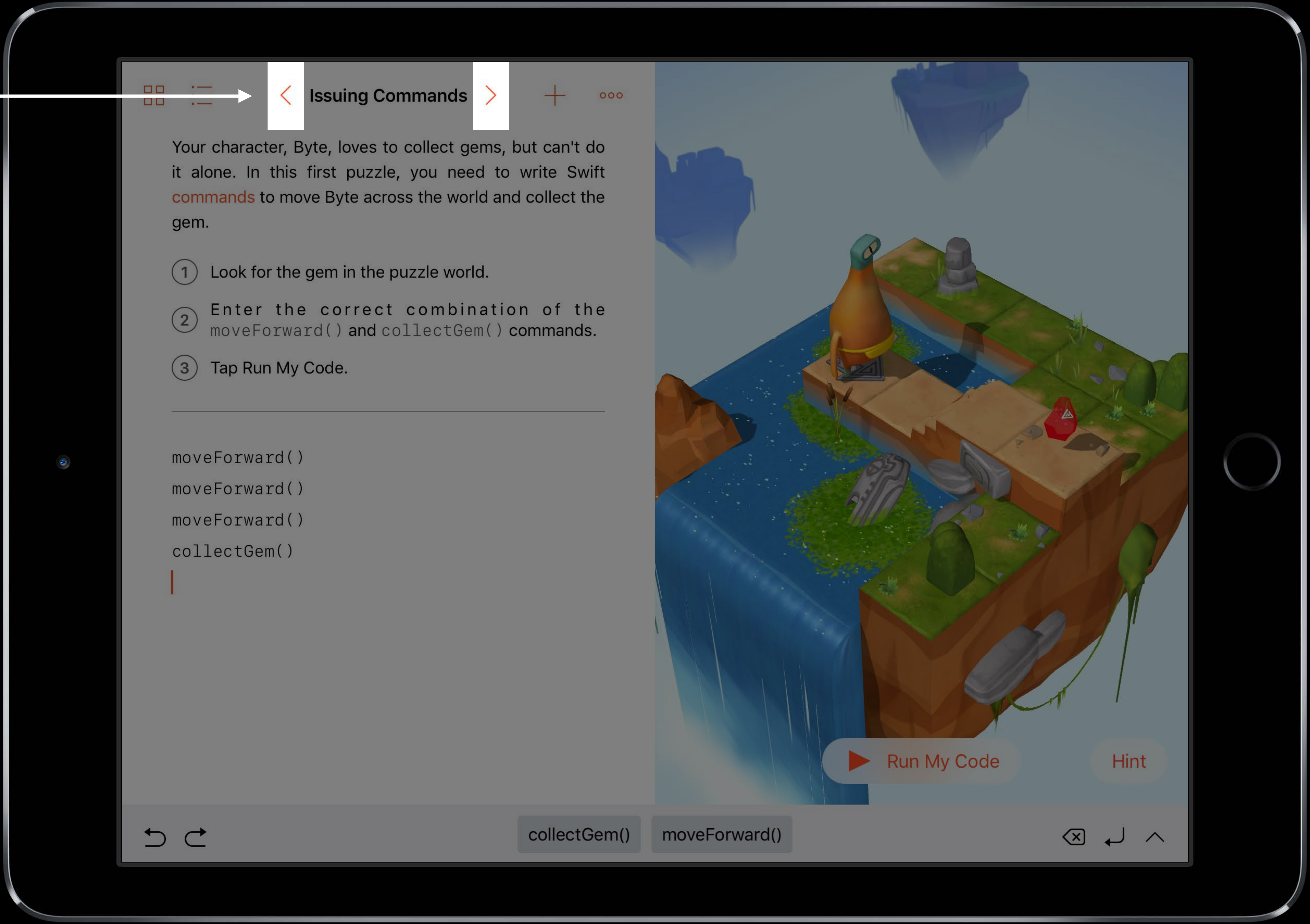


Your character, Byte, loves to collect gems, but can't do it alone. In this first puzzle, you need to write Swift commands to move Byte across the world and collect the gem.

1. Look for the gem in the puzzle world.
2. Enter the correct combination of the `moveForward()` and `collectGem()` commands.
3. Tap Run My Code.

```
moveForward()
moveForward()
moveForward()
collectGem()
```

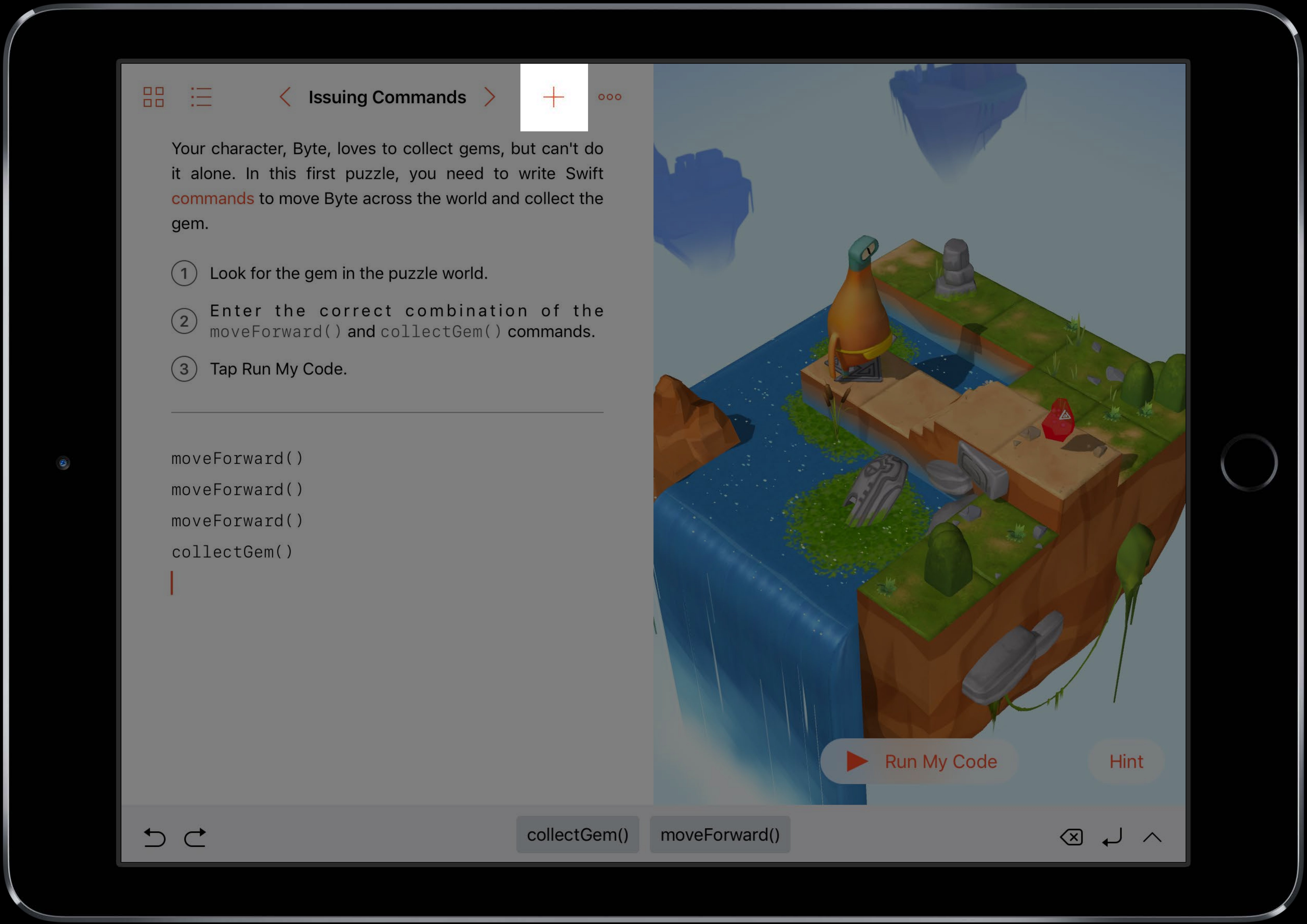
Page Navigation



Your character, Byte, loves to collect gems, but can't do it alone. In this first puzzle, you need to write Swift commands to move Byte across the world and collect the gem.

1. Look for the gem in the puzzle world.
2. Enter the correct combination of the `moveForward()` and `collectGem()` commands.
3. Tap Run My Code.

```
moveForward()
moveForward()
moveForward()
collectGem()
```

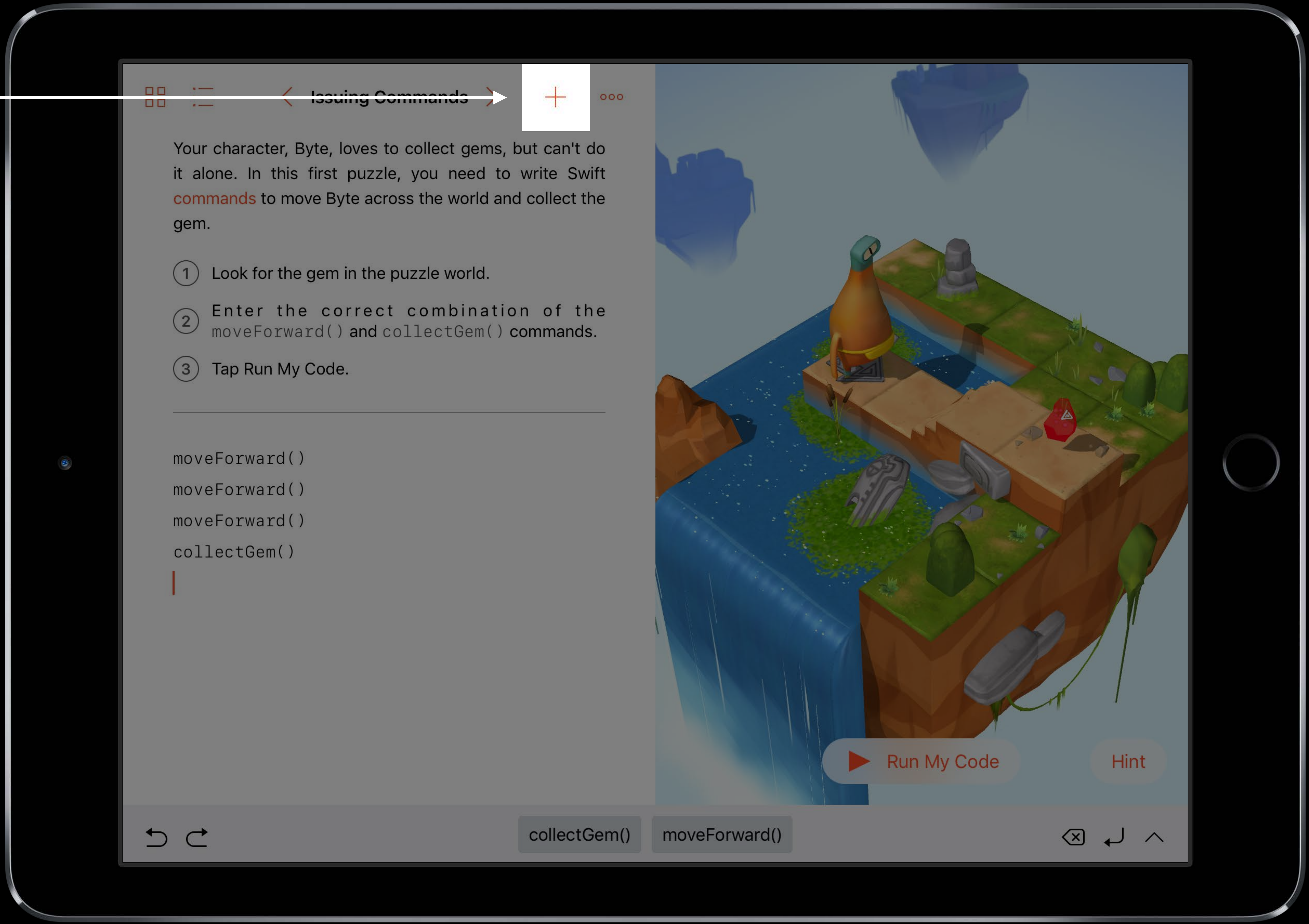


Your character, Byte, loves to collect gems, but can't do it alone. In this first puzzle, you need to write Swift commands to move Byte across the world and collect the gem.

1. Look for the gem in the puzzle world.
2. Enter the correct combination of the `moveForward()` and `collectGem()` commands.
3. Tap Run My Code.

```
moveForward()
moveForward()
moveForward()
collectGem()
```

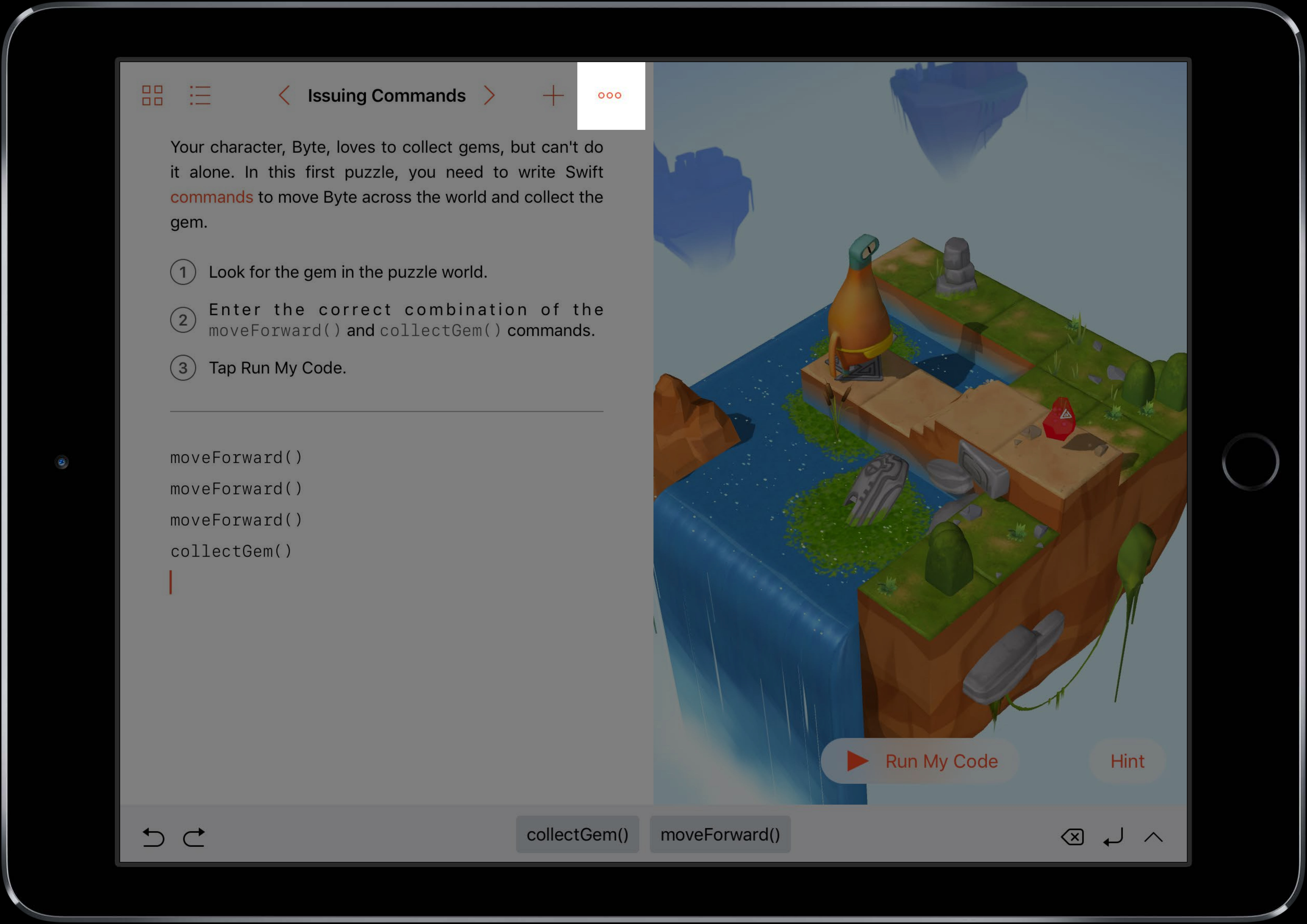
Library



Your character, Byte, loves to collect gems, but can't do it alone. In this first puzzle, you need to write Swift commands to move Byte across the world and collect the gem.

1. Look for the gem in the puzzle world.
2. Enter the correct combination of the `moveForward()` and `collectGem()` commands.
3. Tap Run My Code.

```
moveForward()
moveForward()
moveForward()
collectGem()
```

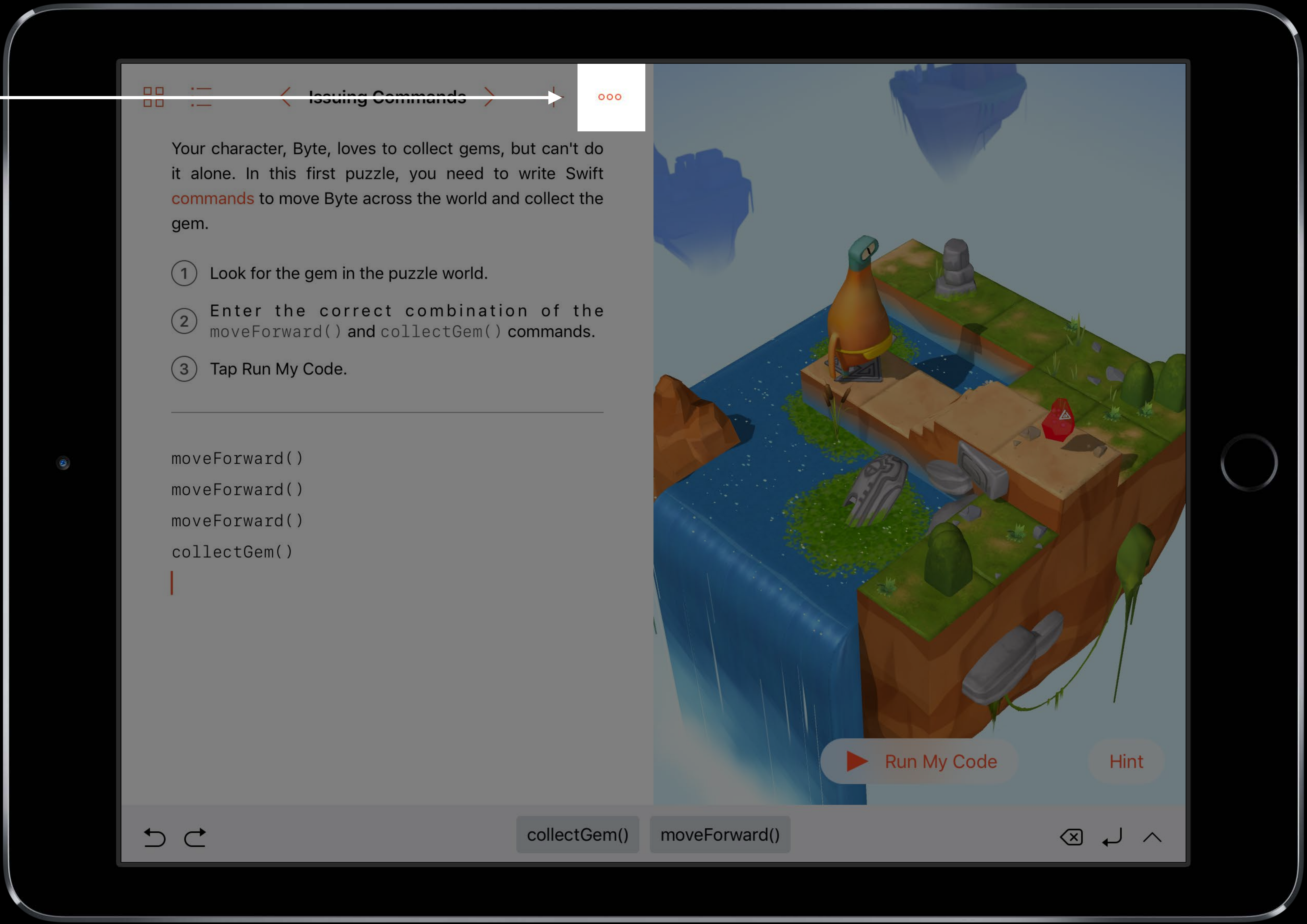


Your character, Byte, loves to collect gems, but can't do it alone. In this first puzzle, you need to write Swift commands to move Byte across the world and collect the gem.

1. Look for the gem in the puzzle world.
2. Enter the correct combination of the `moveForward()` and `collectGem()` commands.
3. Tap Run My Code.

```
moveForward()
moveForward()
moveForward()
collectGem()
```

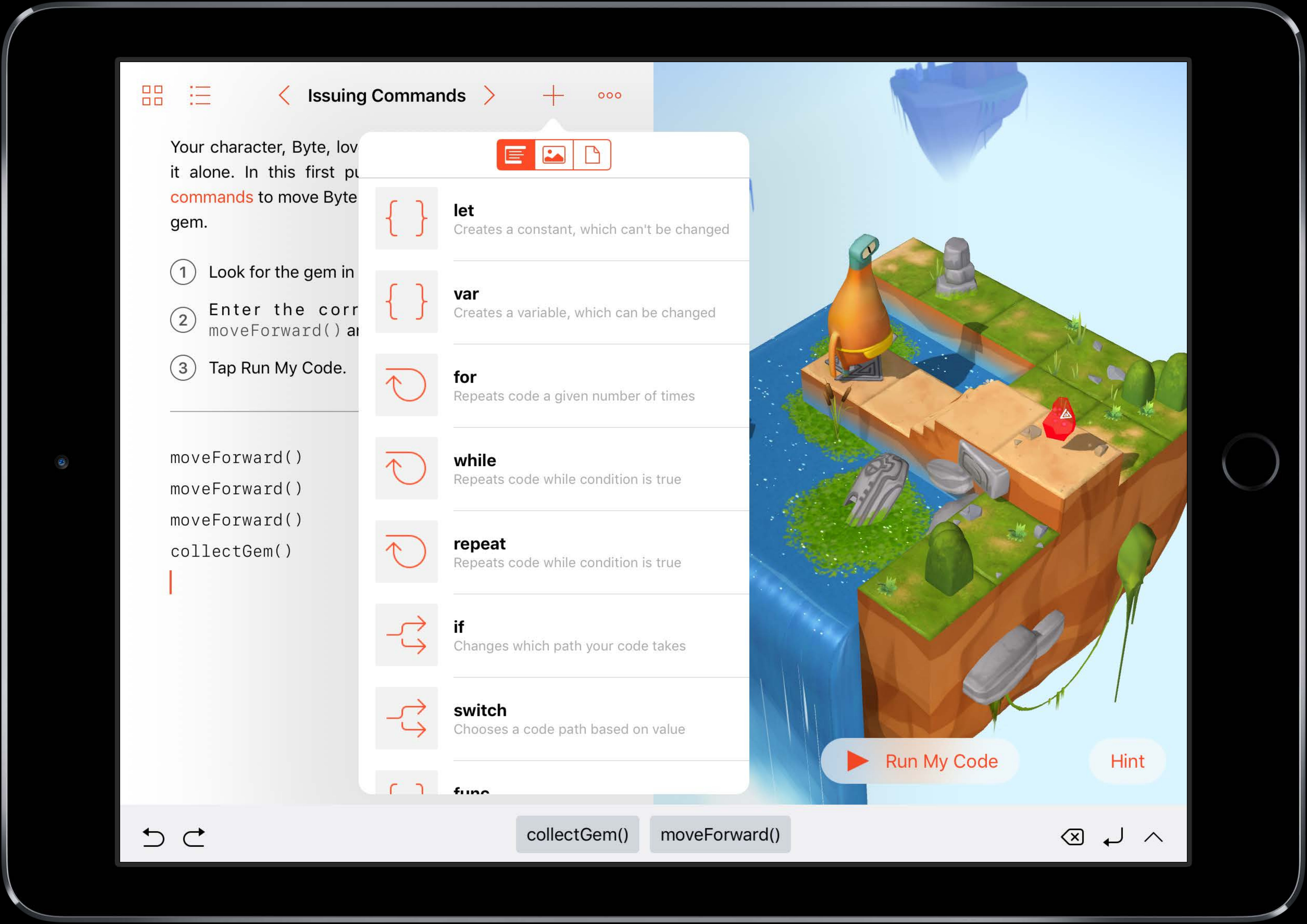
Tools Menu



Your character, Byte, loves to collect gems, but can't do it alone. In this first puzzle, you need to write Swift commands to move Byte across the world and collect the gem.

1. Look for the gem in the puzzle world.
2. Enter the correct combination of the `moveForward()` and `collectGem()` commands.
3. Tap Run My Code.

```
moveForward()
moveForward()
moveForward()
collectGem()
```



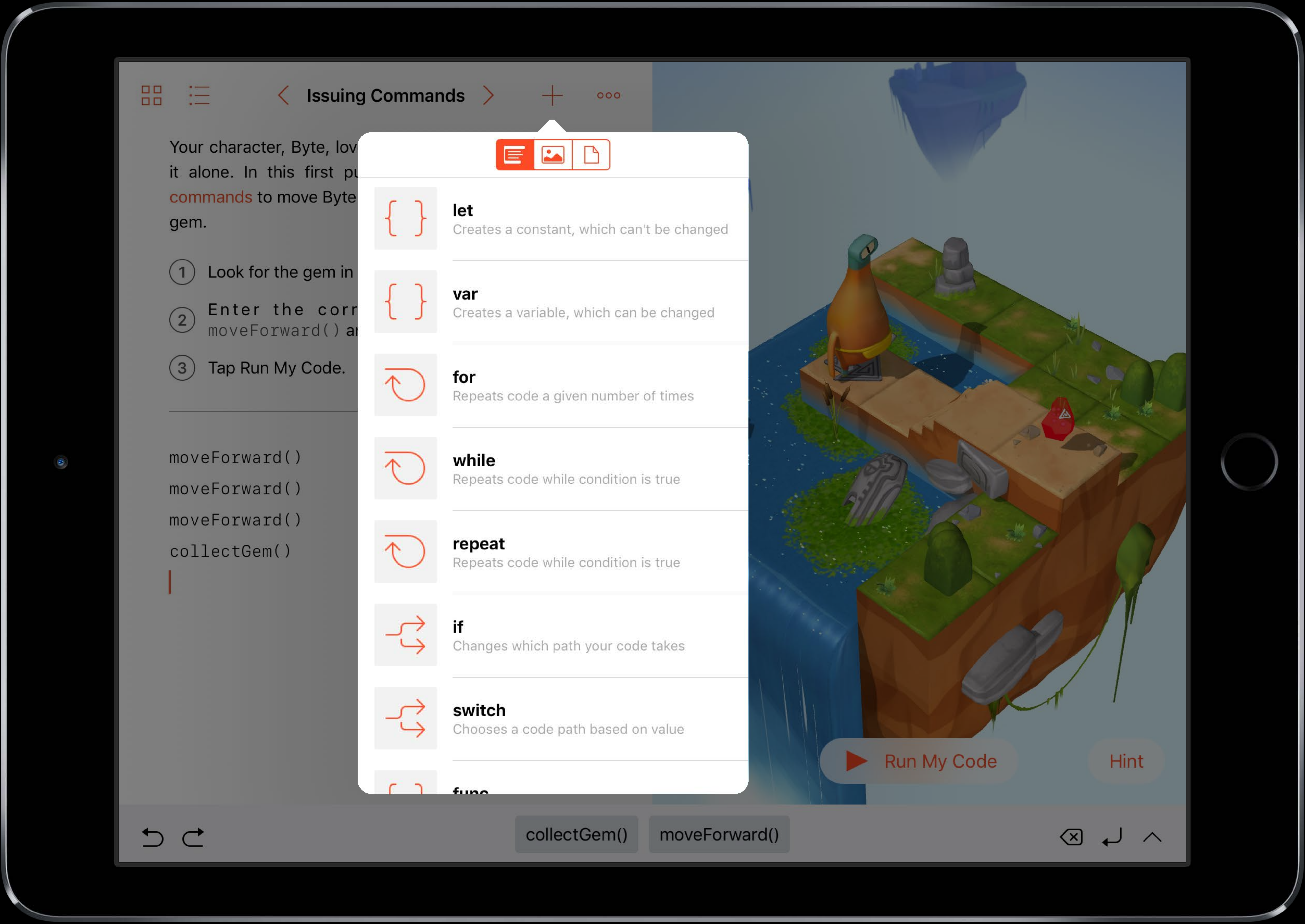
Issuing Commands

Your character, Byte, lov
it alone. In this first pu
commands to move Byte
gem.

1. Look for the gem in
2. Enter the corr
   moveForward() ar
3. Tap Run My Code.

```
moveForward()
moveForward()
moveForward()
collectGem()
```

let	Creates a constant, which can't be changed
var	Creates a variable, which can be changed
for	Repeats code a given number of times
while	Repeats code while condition is true
repeat	Repeats code while condition is true
if	Changes which path your code takes
switch	Chooses a code path based on value
func	



Your character, Byte, lov
it alone. In this first pu
commands to move Byte
gem.

1 Look for the gem in
2 Enter the corr
moveForward() ar
3 Tap Run My Code.

```
moveForward()
moveForward()
moveForward()
collectGem()
```

let
Creates a constant, which can't be changed

var
Creates a variable, which can be changed

for
Repeats code a given number of times

while
Repeats code while condition is true

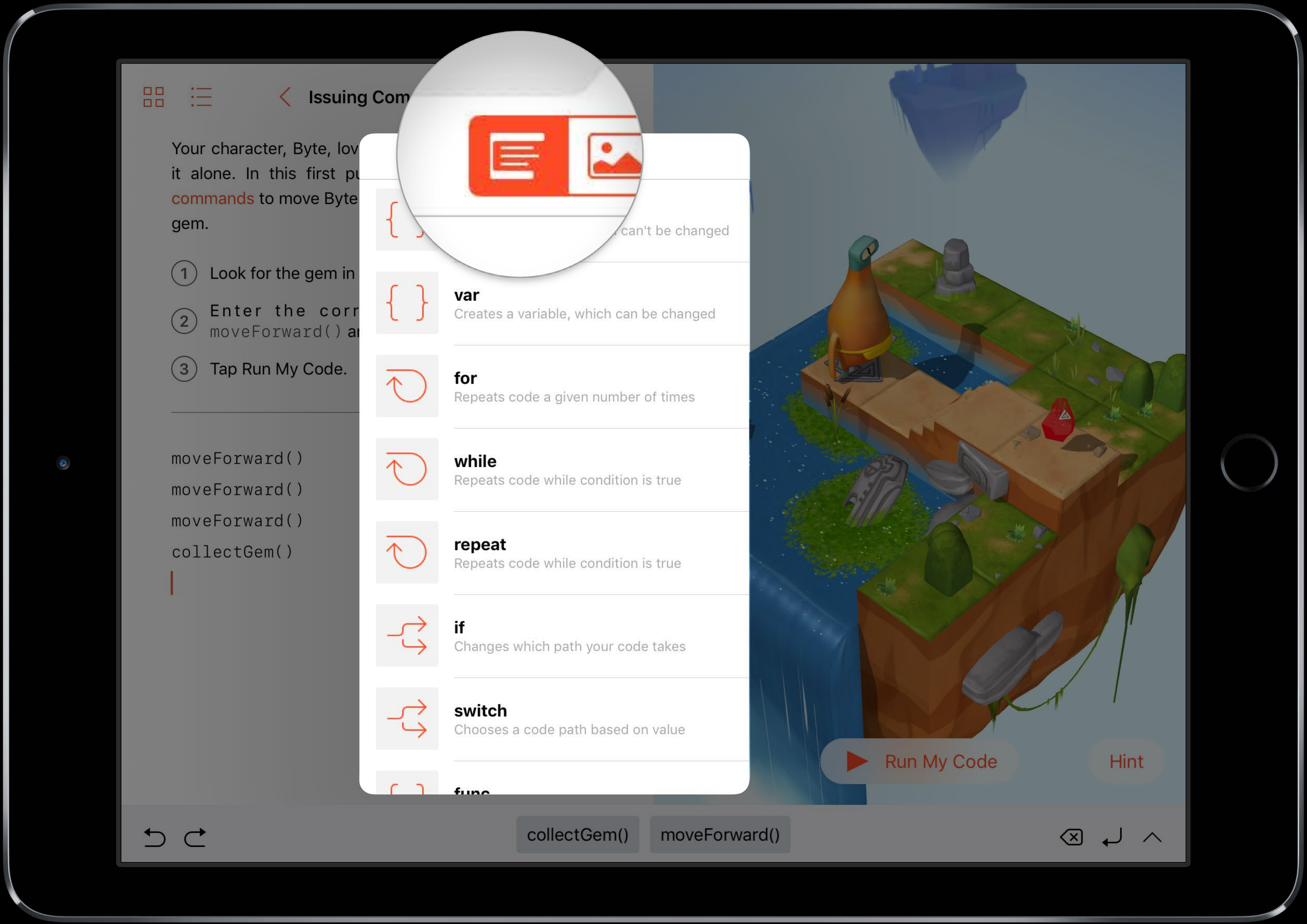
repeat
Repeats code while condition is true

if
Changes which path your code takes

switch
Chooses a code path based on value

func

Snippets



Issuing Com

Your character, Byte, lov
it alone. In this first pu
commands to move Byte
gem.

1 Look for the gem in

2 Enter the corr
moveForward() ar

3 Tap Run My Code.

```
moveForward()
moveForward()
moveForward()
collectGem()
```

can't be changed

var
Creates a variable, which can be changed

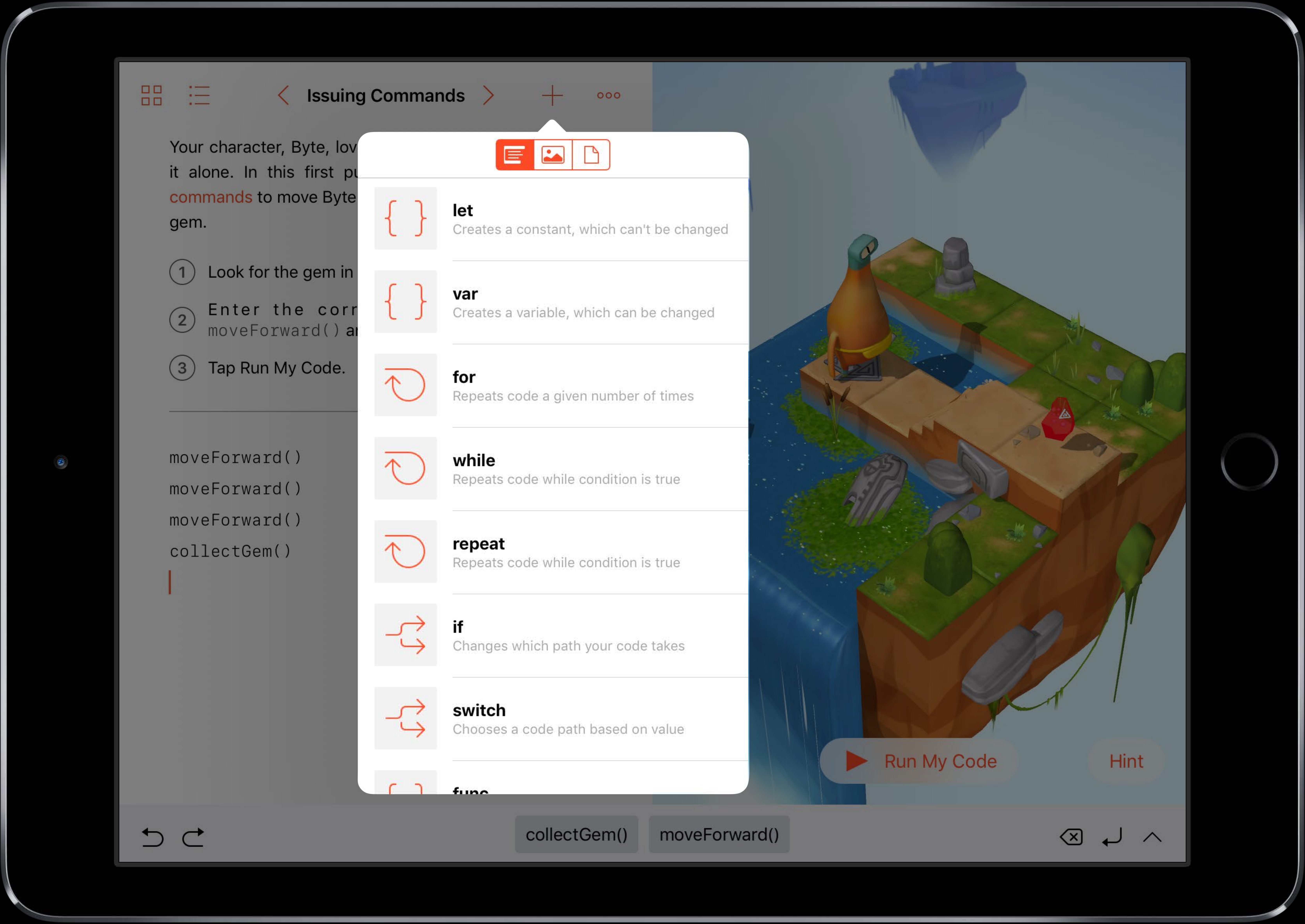
for
Repeats code a given number of times

while
Repeats code while condition is true

repeat
Repeats code while condition is true

if
Changes which path your code takes

switch
Chooses a code path based on value



Issuing Commands

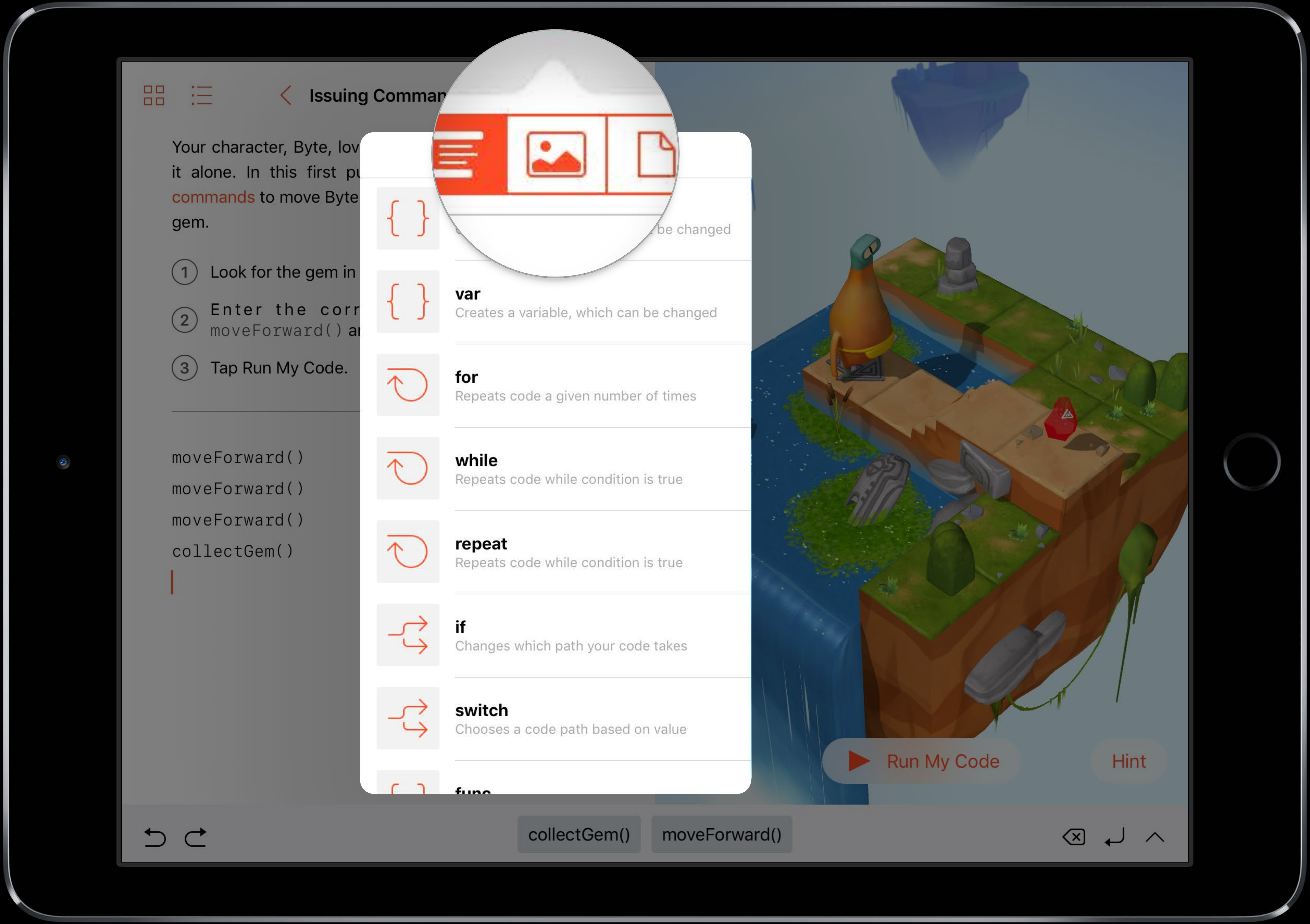
Your character, Byte, lov
it alone. In this first pu
commands to move Byte
gem.

1. Look for the gem in
2. Enter the corr
moveForward() ar
3. Tap Run My Code.

```
moveForward()
moveForward()
moveForward()
collectGem()
```

let	Creates a constant, which can't be changed
var	Creates a variable, which can be changed
for	Repeats code a given number of times
while	Repeats code while condition is true
repeat	Repeats code while condition is true
if	Changes which path your code takes
switch	Chooses a code path based on value
func	

Images



Issuing Comman

Your character, Byte, lov
it alone. In this first pu
commands to move Byte
gem.

1 Look for the gem in

2 Enter the corr
moveForward() ar

3 Tap Run My Code.

```
moveForward()
moveForward()
moveForward()
collectGem()
```

be changed

var
Creates a variable, which can be changed

for
Repeats code a given number of times

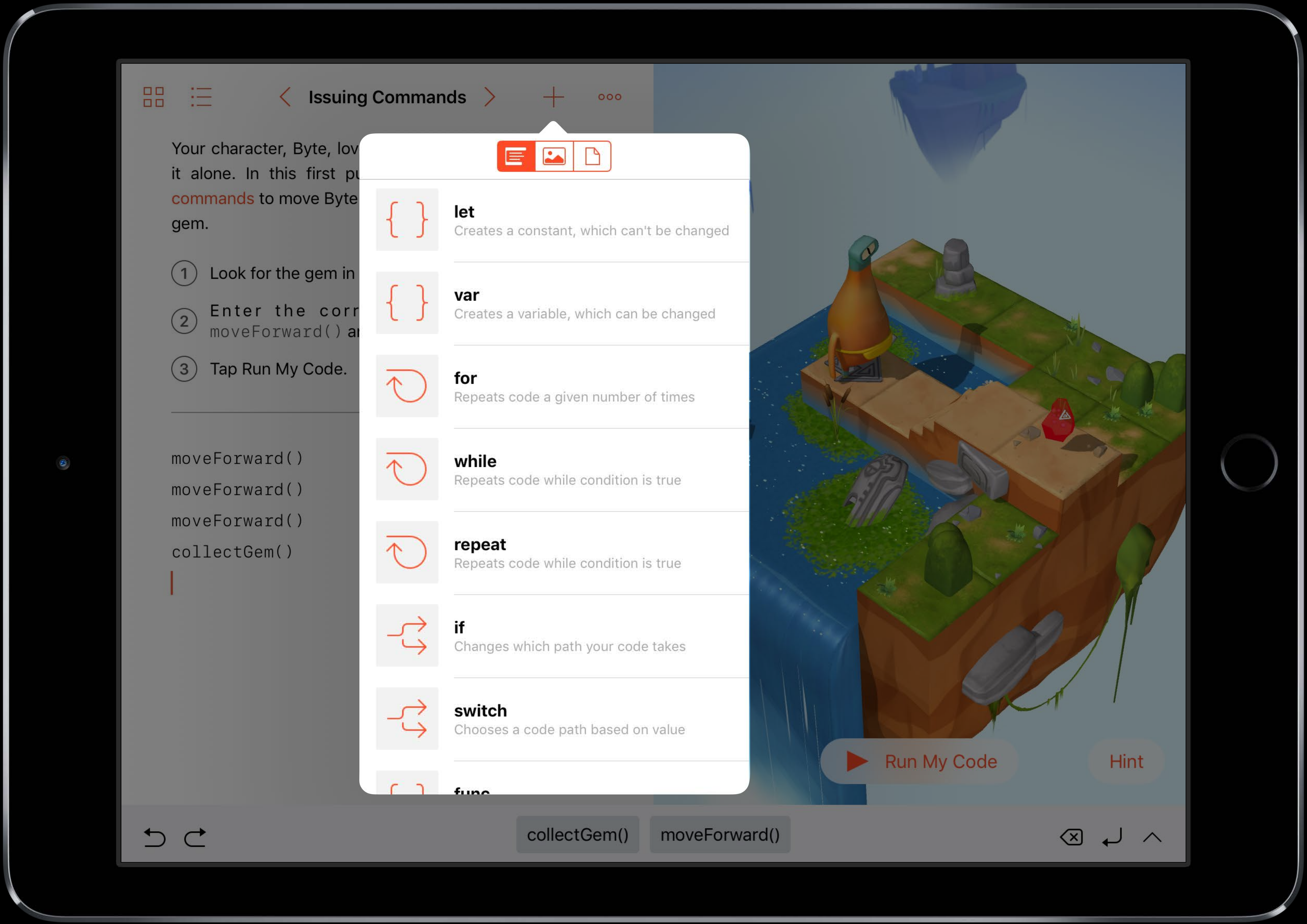
while
Repeats code while condition is true

repeat
Repeats code while condition is true

if
Changes which path your code takes

switch
Chooses a code path based on value

func



Issuing Commands

Your character, Byte, lov
it alone. In this first pu
commands to move Byte
gem.

1 Look for the gem in

2 Enter the corr
moveForward() ar

3 Tap Run My Code.

```
moveForward()
moveForward()
moveForward()
collectGem()
```

let
Creates a constant, which can't be changed

var
Creates a variable, which can be changed

for
Repeats code a given number of times

while
Repeats code while condition is true

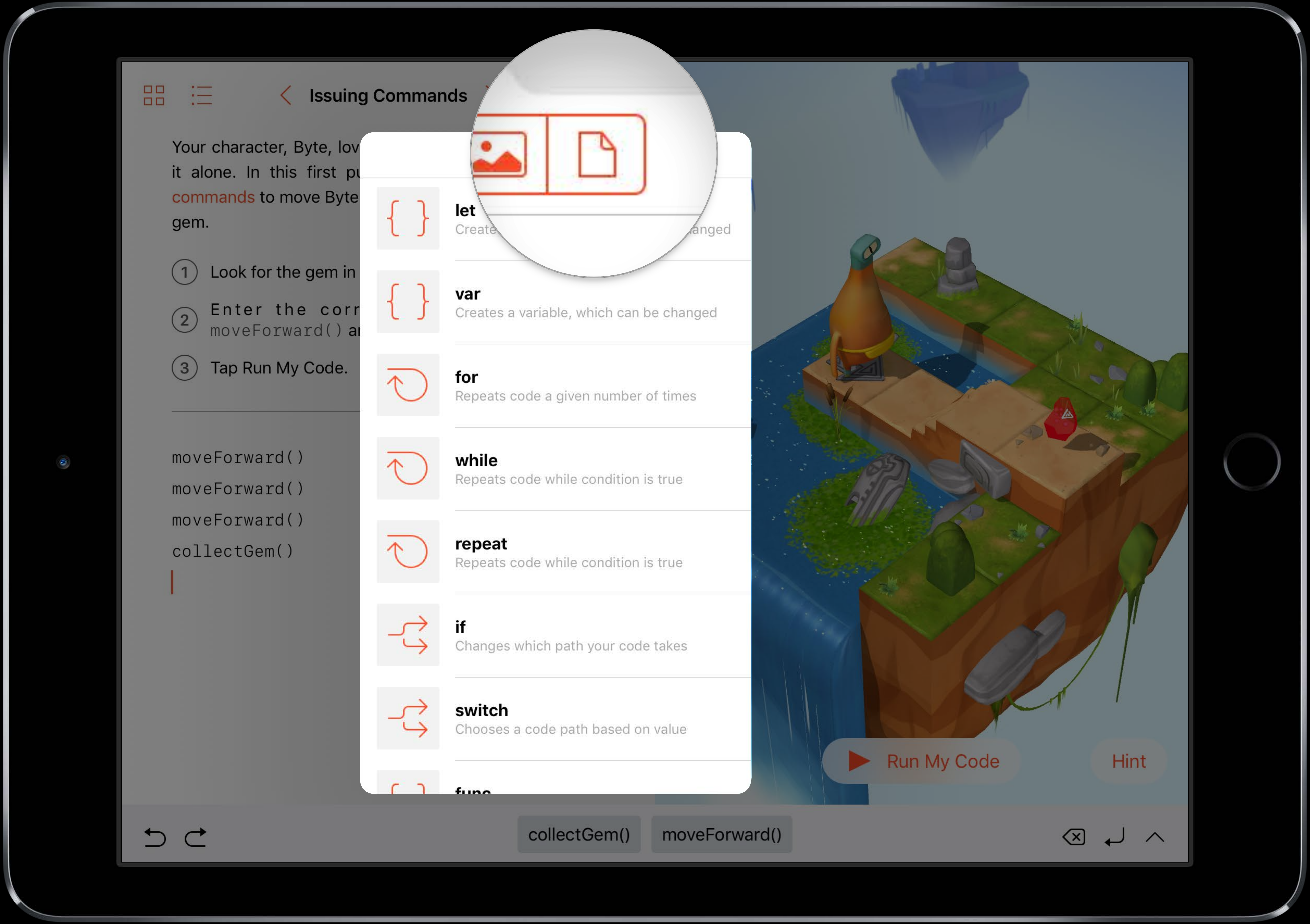
repeat
Repeats code while condition is true

if
Changes which path your code takes

switch
Chooses a code path based on value

func

Files



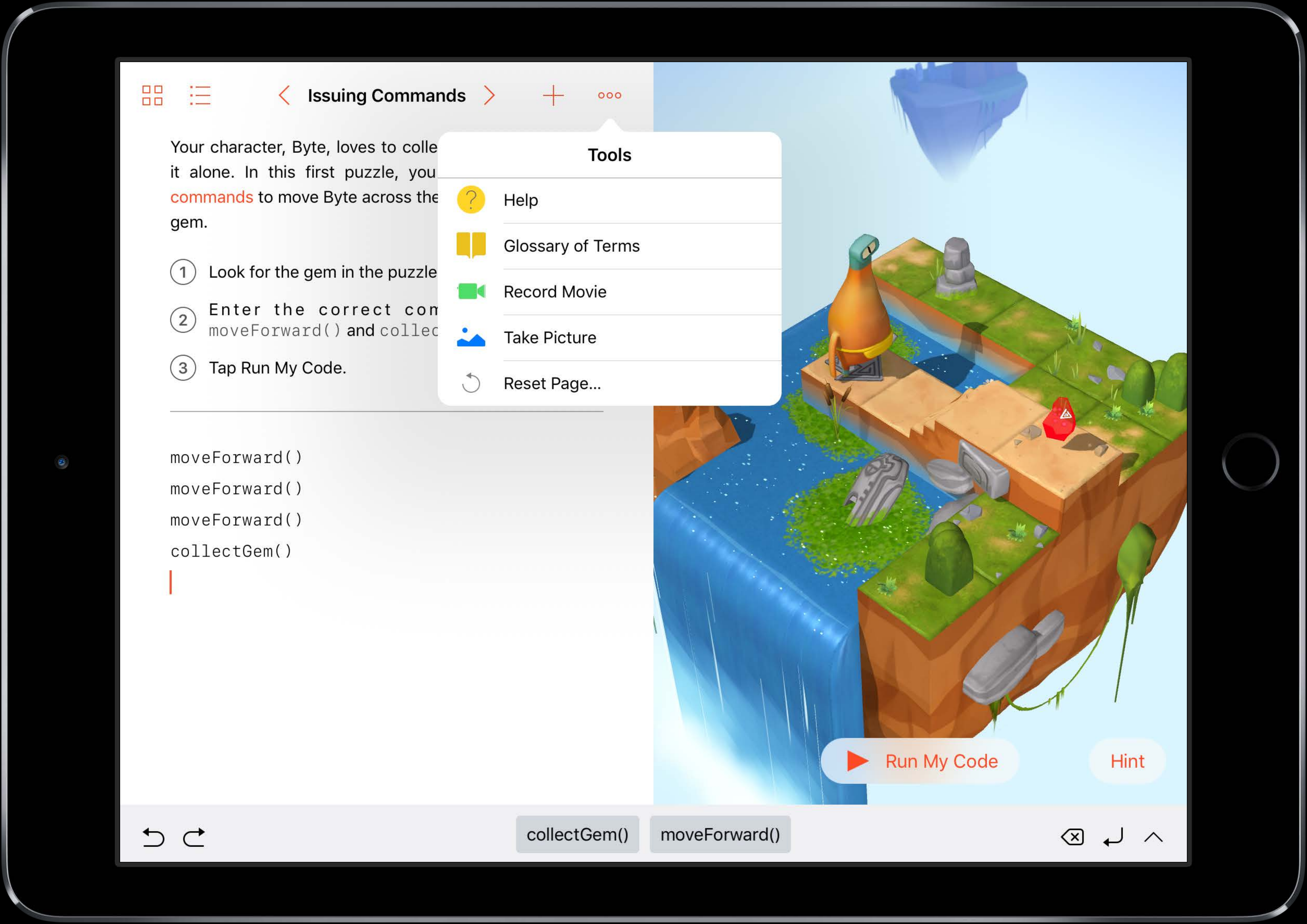
Issuing Commands

Your character, Byte, lov
it alone. In this first pu
commands to move Byte
gem.

1. Look for the gem in
2. Enter the corr
moveForward() ar
3. Tap Run My Code.

```
moveForward()
moveForward()
moveForward()
collectGem()
```

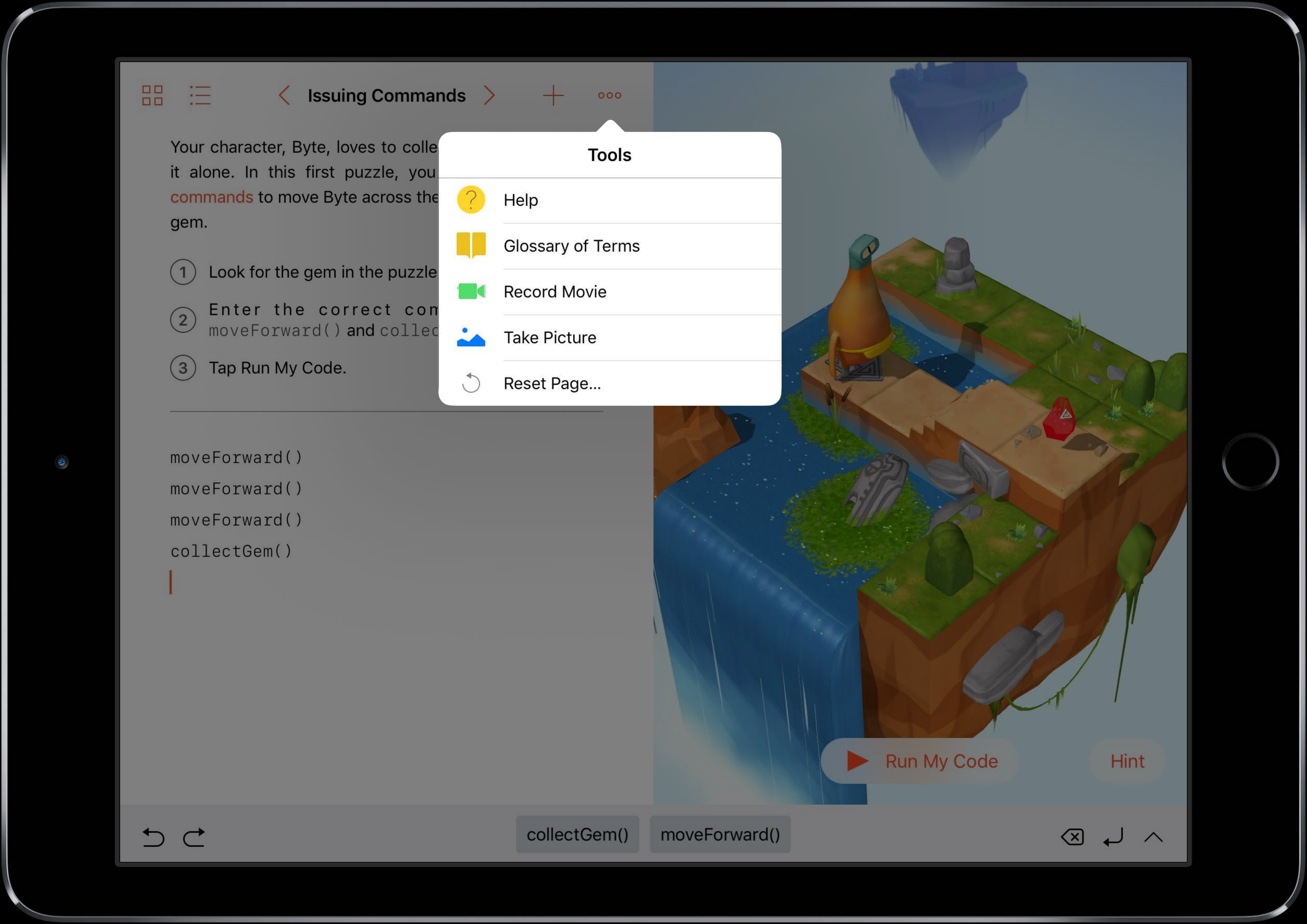
let	Create … anged
var	Creates a variable, which can be changed
for	Repeats code a given number of times
while	Repeats code while condition is true
repeat	Repeats code while condition is true
if	Changes which path your code takes
switch	Chooses a code path based on value
func	



Your character, Byte, loves to colle
it alone. In this first puzzle, you
commands to move Byte across the
gem.

1. Look for the gem in the puzzle
2. Enter the correct com
`moveForward()` and collec
3. Tap Run My Code.

```
moveForward()
moveForward()
moveForward()
collectGem()
```



Issuing Commands

Your character, Byte, loves to colle
it alone. In this first puzzle, you
commands to move Byte across the
gem.

1. Look for the gem in the puzzle
2. Enter the correct com
moveForward() and collec
3. Tap Run My Code.

```
moveForward()
moveForward()
moveForward()
collectGem()
```

Demo

Using Swift Playgrounds

Authoring for Swift Playgrounds

Jonathan Penn
Playgrounds Engineer

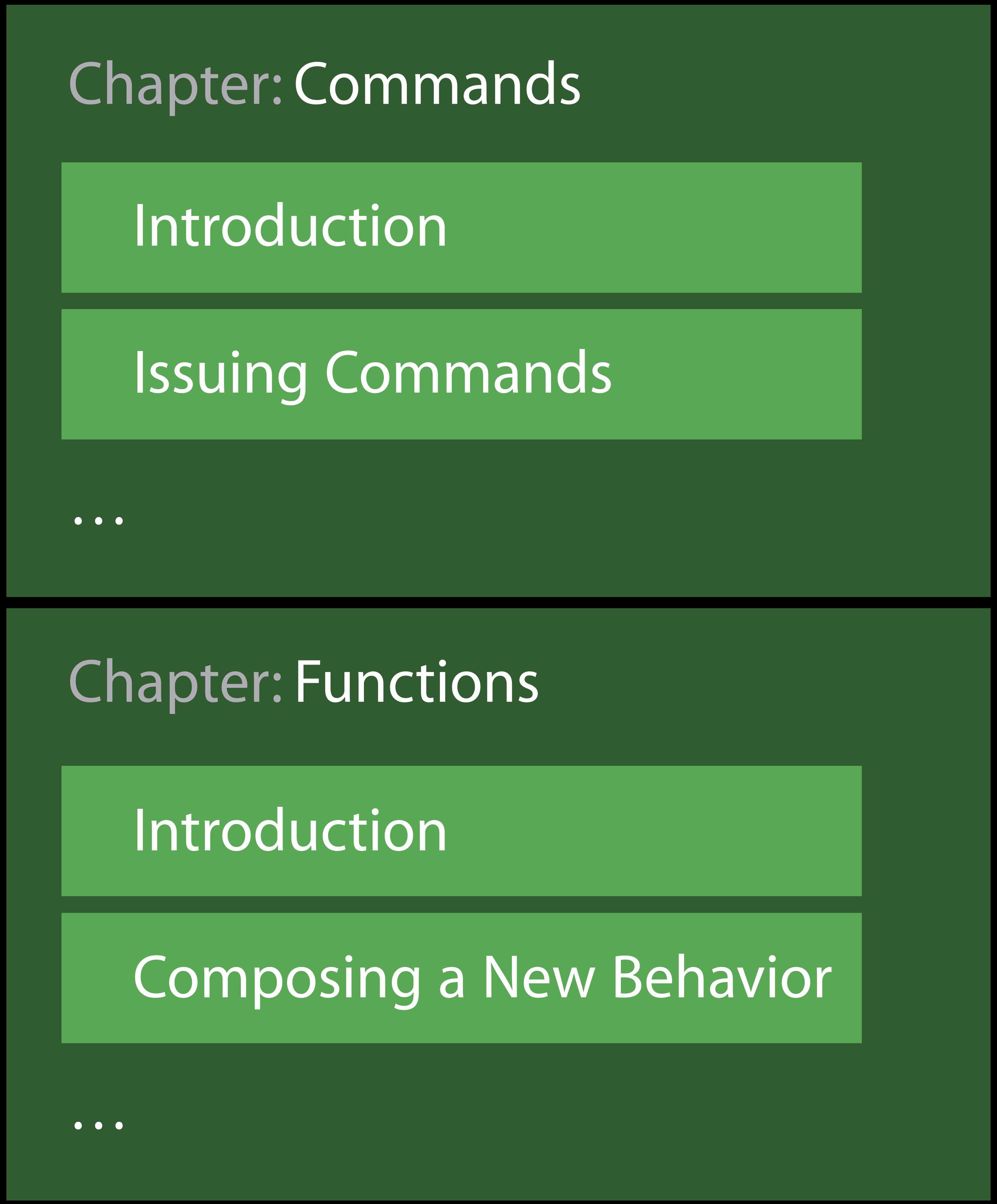
PLAYGROUND

PLAYGROUND

PLAYGROUND

BOOK

Chapters / Pages



Chapters / Pages

Chapter: Commands

Introduction

Issuing Commands

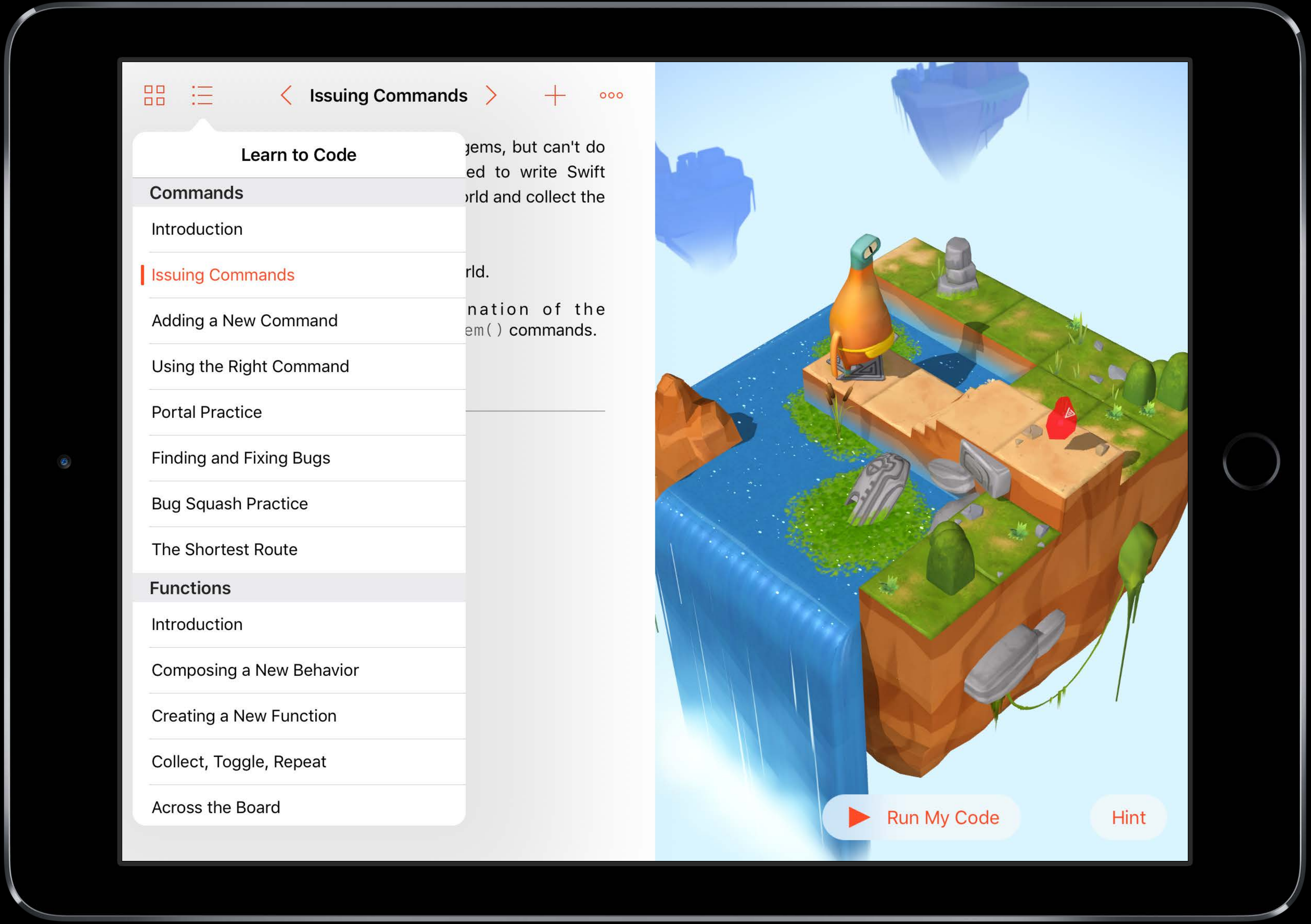
...

Chapter: Functions

Introduction

Composing a New Behavior

...



Issuing Commands

Learn to Code

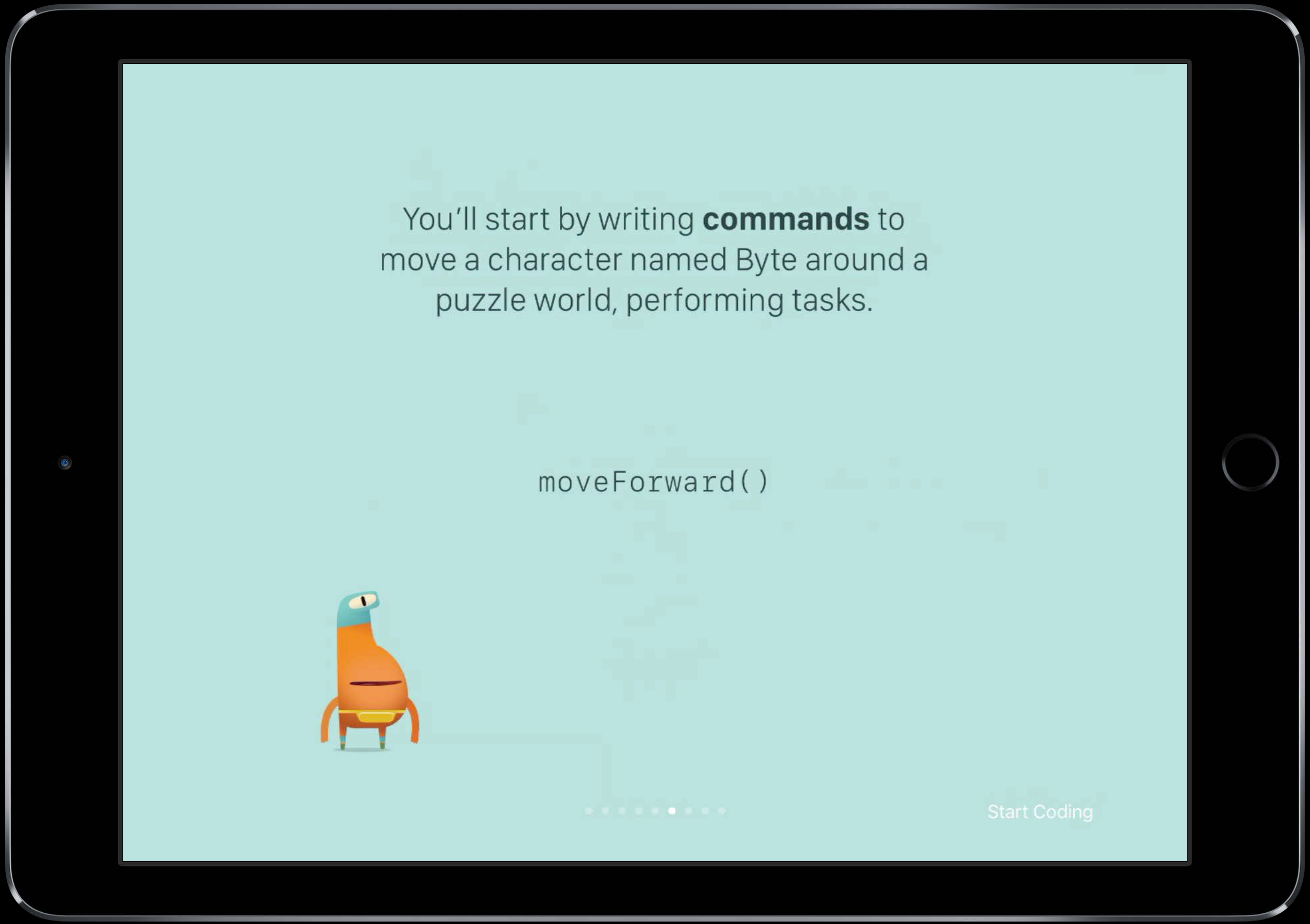
Commands

- Introduction
- Issuing Commands
- Adding a New Command
- Using the Right Command
- Portal Practice
- Finding and Fixing Bugs
- Bug Squash Practice
- The Shortest Route

Functions

- Introduction
- Composing a New Behavior
- Creating a New Function
- Collect, Toggle, Repeat
- Across the Board

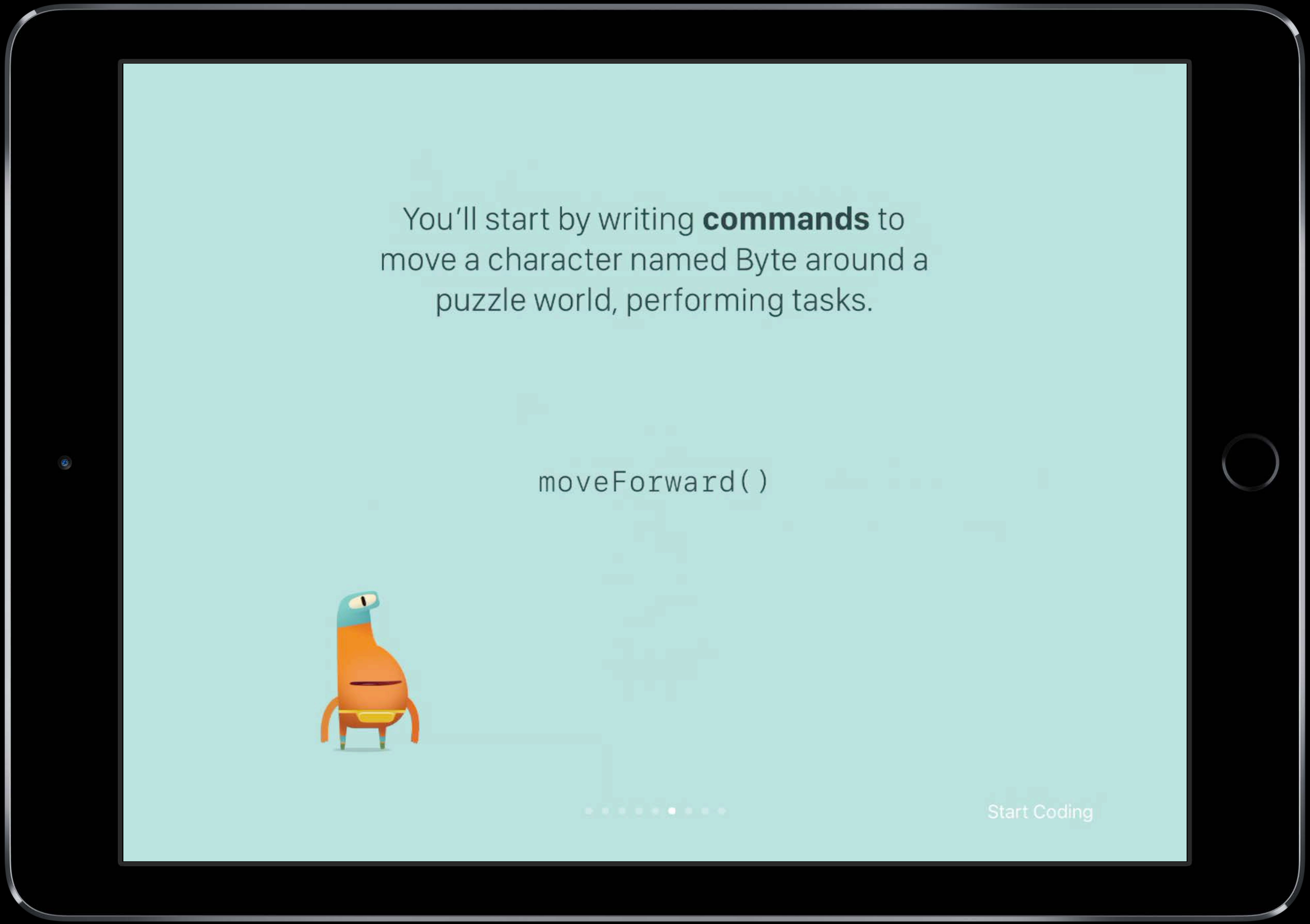
Cutscenes



You'll start by writing **commands** to move a character named Byte around a puzzle world, performing tasks.

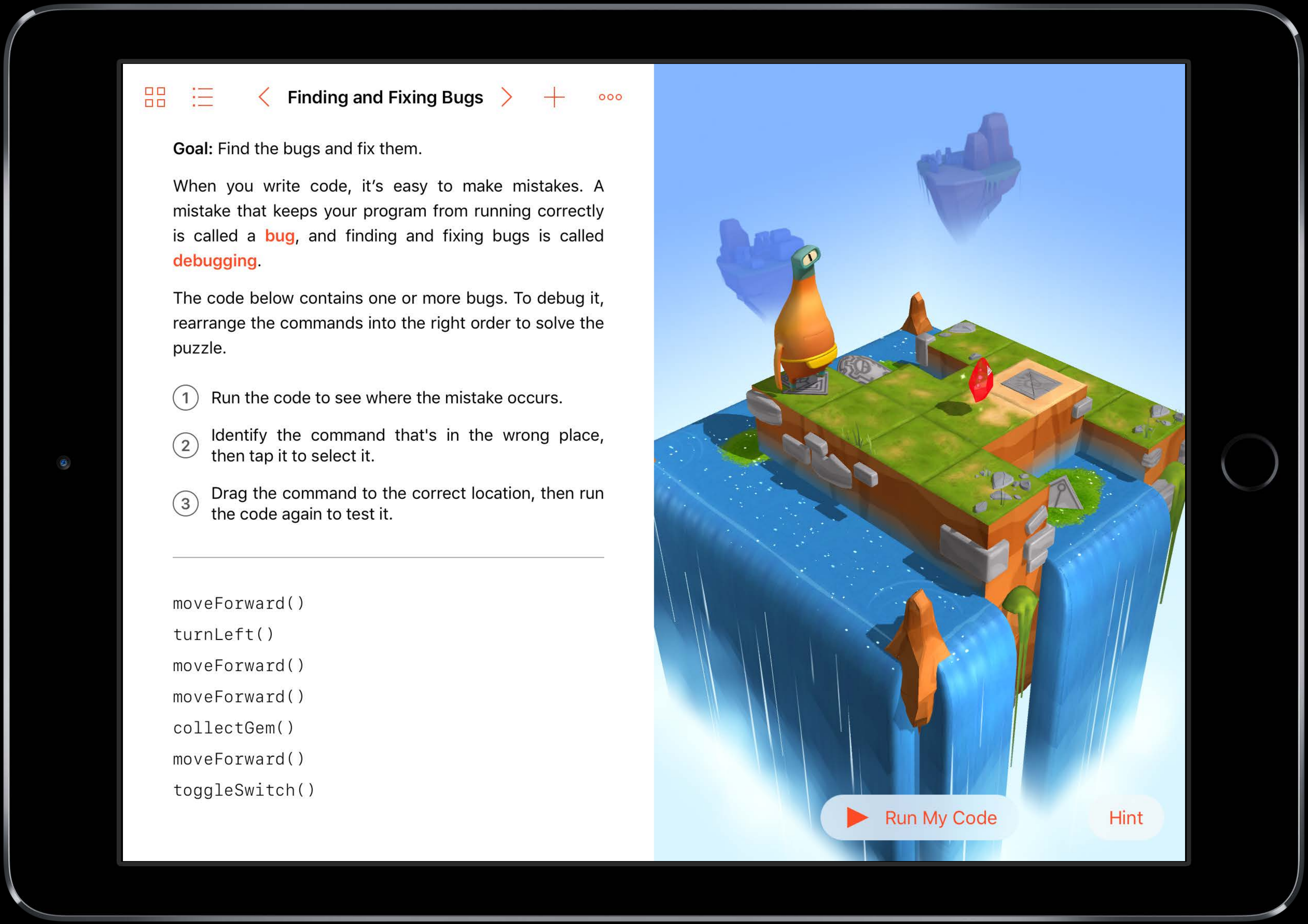
```
moveForward()
```

Cutscenes


You'll start by writing **commands** to move a character named Byte around a puzzle world, performing tasks.

```
moveForward()
```

Glossary



Goal: Find the bugs and fix them.

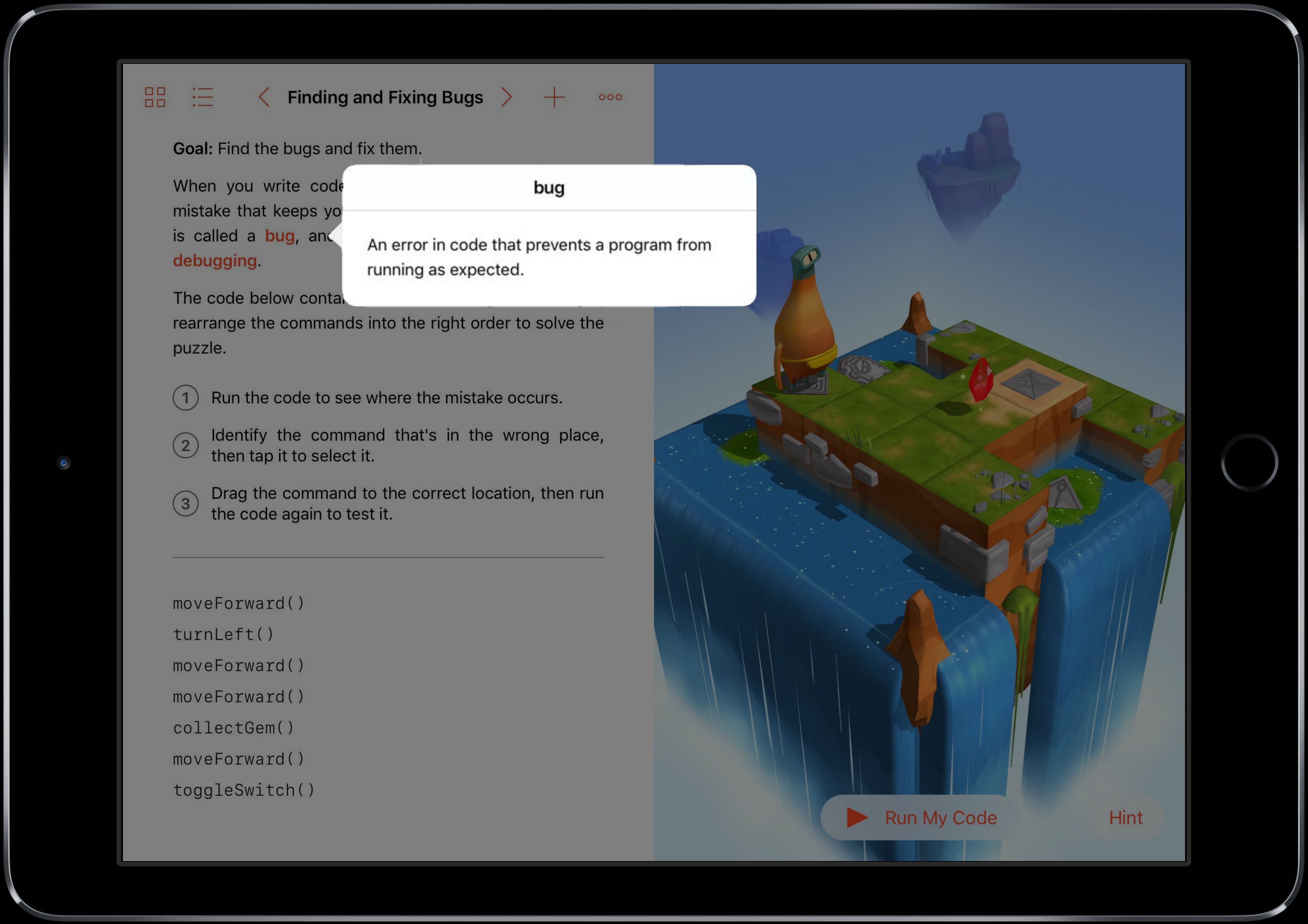
When you write code, it's easy to make mistakes. A mistake that keeps your program from running correctly is called a bug, and finding and fixing bugs is called debugging.

The code below contains one or more bugs. To debug it, rearrange the commands into the right order to solve the puzzle.

1. Run the code to see where the mistake occurs.
2. Identify the command that's in the wrong place, then tap it to select it.
3. Drag the command to the correct location, then run the code again to test it.

```
moveForward()
turnLeft()
moveForward()
moveForward()
collectGem()
moveForward()
toggleSwitch()
```

Glossary



Goal: Find the bugs and fix them.

When you write code … mistake that keeps yo… is called a bug, an… debugging.

The code below conta… rearrange the commands into the right order to solve the puzzle.

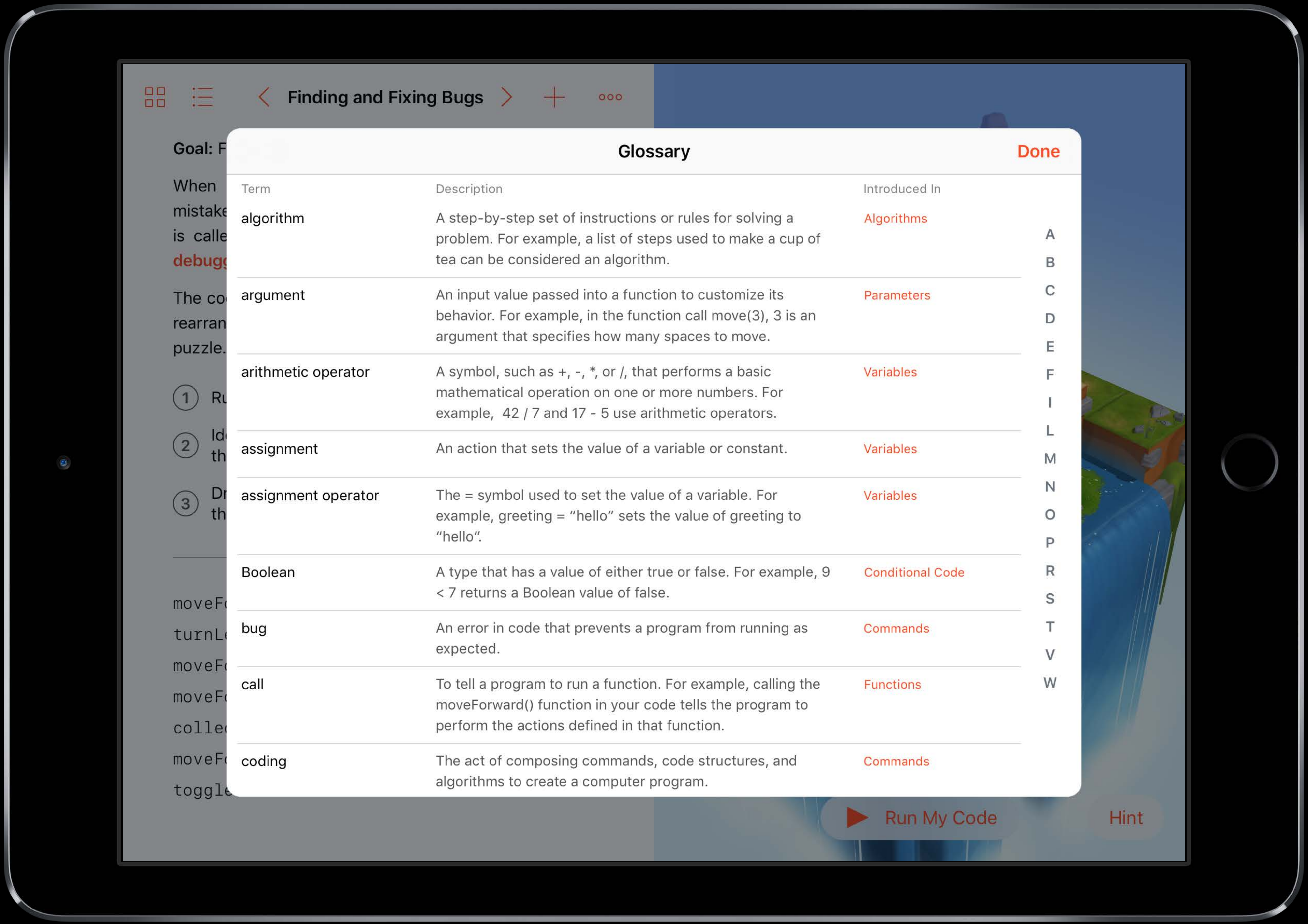
1. Run the code to see where the mistake occurs.
2. Identify the command that's in the wrong place, then tap it to select it.
3. Drag the command to the correct location, then run the code again to test it.

```
moveForward()
turnLeft()
moveForward()
moveForward()
collectGem()
moveForward()
toggleSwitch()
```

bug

An error in code that prevents a program from running as expected.

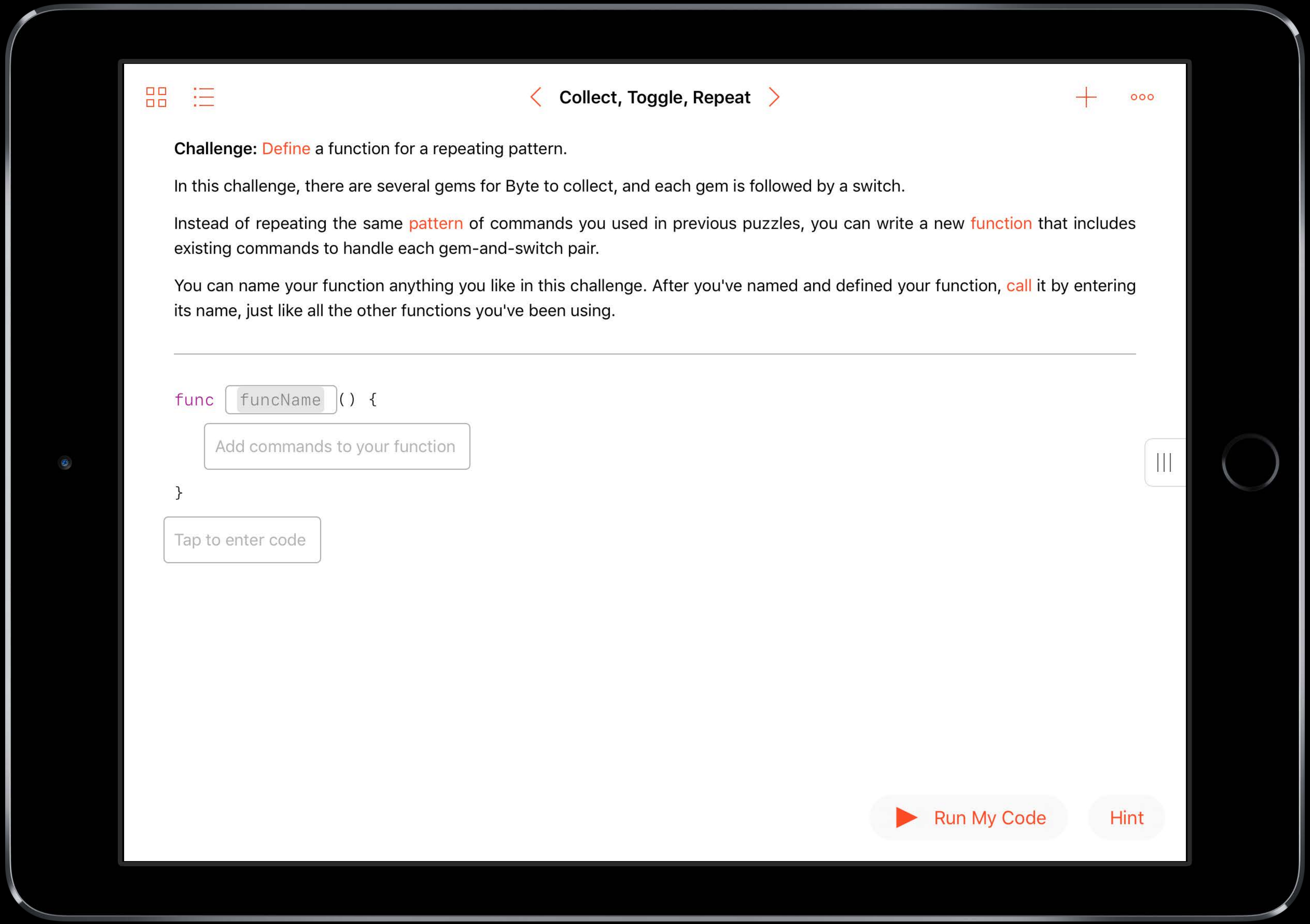
Glossary



Glossary

Term	Description	Introduced In
algorithm	A step-by-step set of instructions or rules for solving a problem. For example, a list of steps used to make a cup of tea can be considered an algorithm.	Algorithms
argument	An input value passed into a function to customize its behavior. For example, in the function call move(3), 3 is an argument that specifies how many spaces to move.	Parameters
arithmetic operator	A symbol, such as +, -, *, or /, that performs a basic mathematical operation on one or more numbers. For example, 42 / 7 and 17 - 5 use arithmetic operators.	Variables
assignment	An action that sets the value of a variable or constant.	Variables
assignment operator	The = symbol used to set the value of a variable. For example, greeting = "hello" sets the value of greeting to "hello".	Variables
Boolean	A type that has a value of either true or false. For example, 9 < 7 returns a Boolean value of false.	Conditional Code
bug	An error in code that prevents a program from running as expected.	Commands
call	To tell a program to run a function. For example, calling the moveForward() function in your code tells the program to perform the actions defined in that function.	Functions
coding	The act of composing commands, code structures, and algorithms to create a computer program.	Commands

Editable Regions



Challenge: Define a function for a repeating pattern.

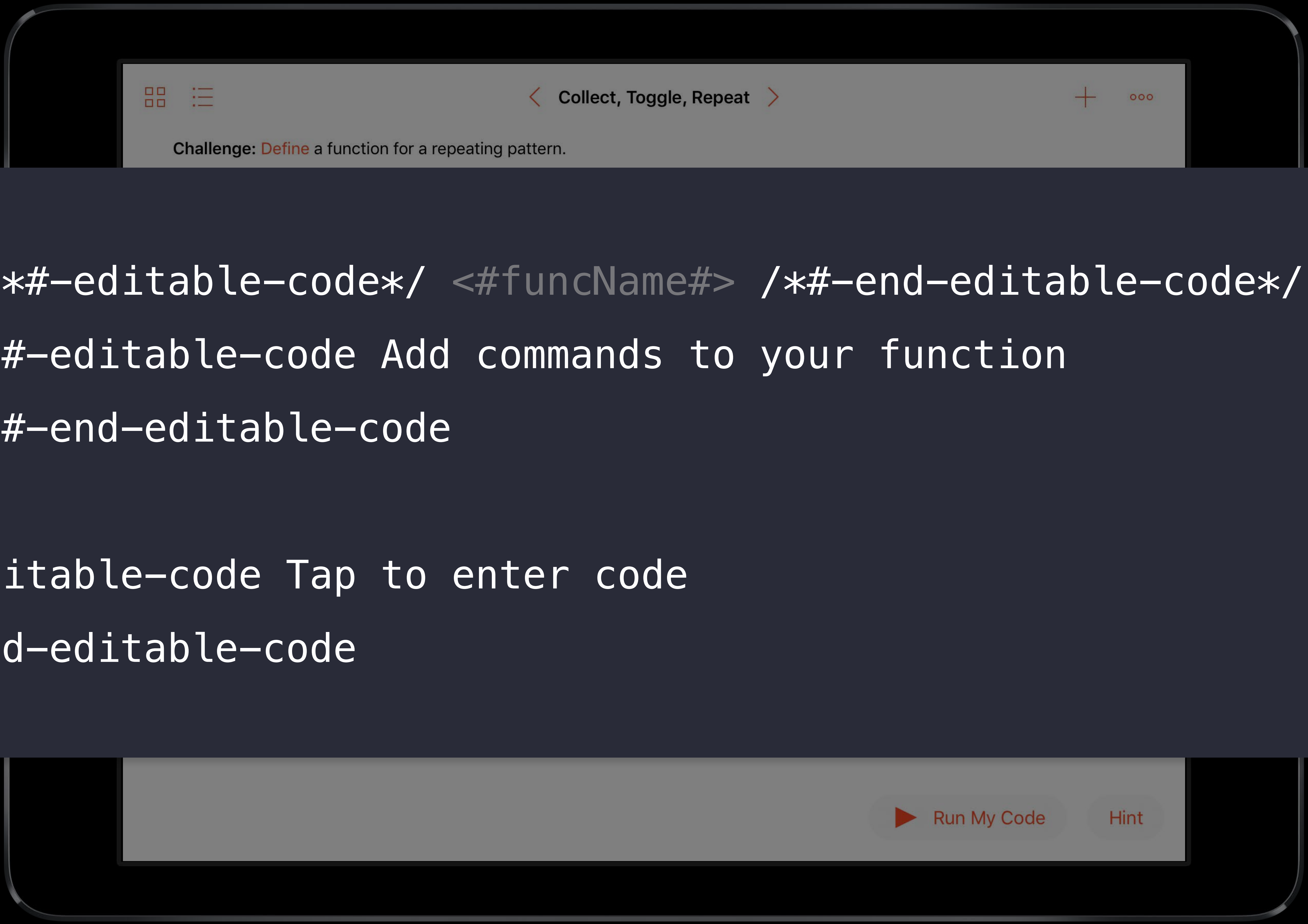
In this challenge, there are several gems for Byte to collect, and each gem is followed by a switch.

Instead of repeating the same pattern of commands you used in previous puzzles, you can write a new function that includes existing commands to handle each gem-and-switch pair.

You can name your function anything you like in this challenge. After you've named and defined your function, call it by entering its name, just like all the other functions you've been using.

```
func funcName() {
    Add commands to your function
}
Tap to enter code
```

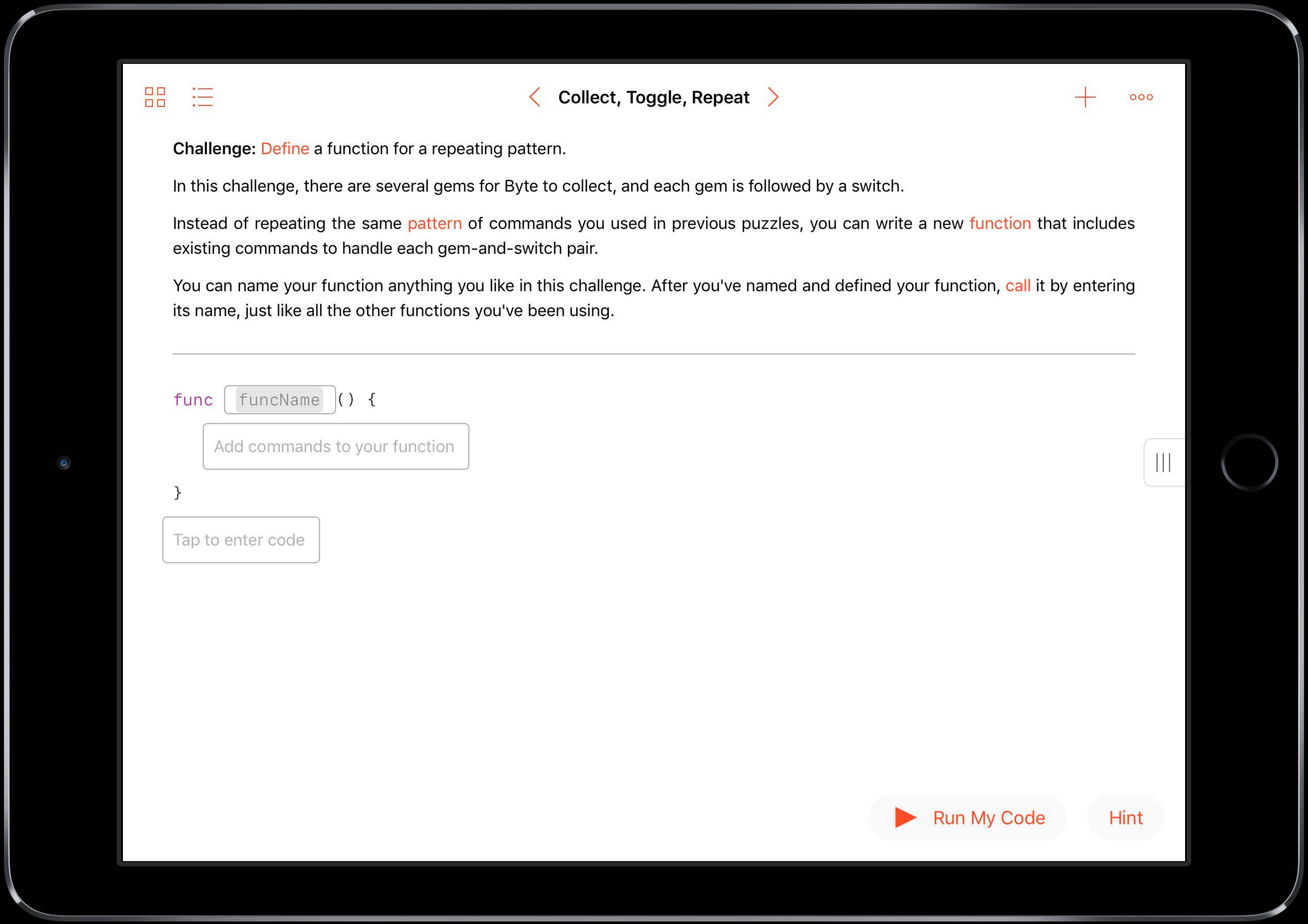
Editable Regions



Challenge: Define a function for a repeating pattern.

```
func /*#-editable-code*/ <#funcName#> /*#-end-editable-code*/() {
    //#-editable-code Add commands to your function
    //#-end-editable-code
}
//#-editable-code Tap to enter code
//#-end-editable-code
```

Hidden Code



Challenge: Define a function for a repeating pattern.

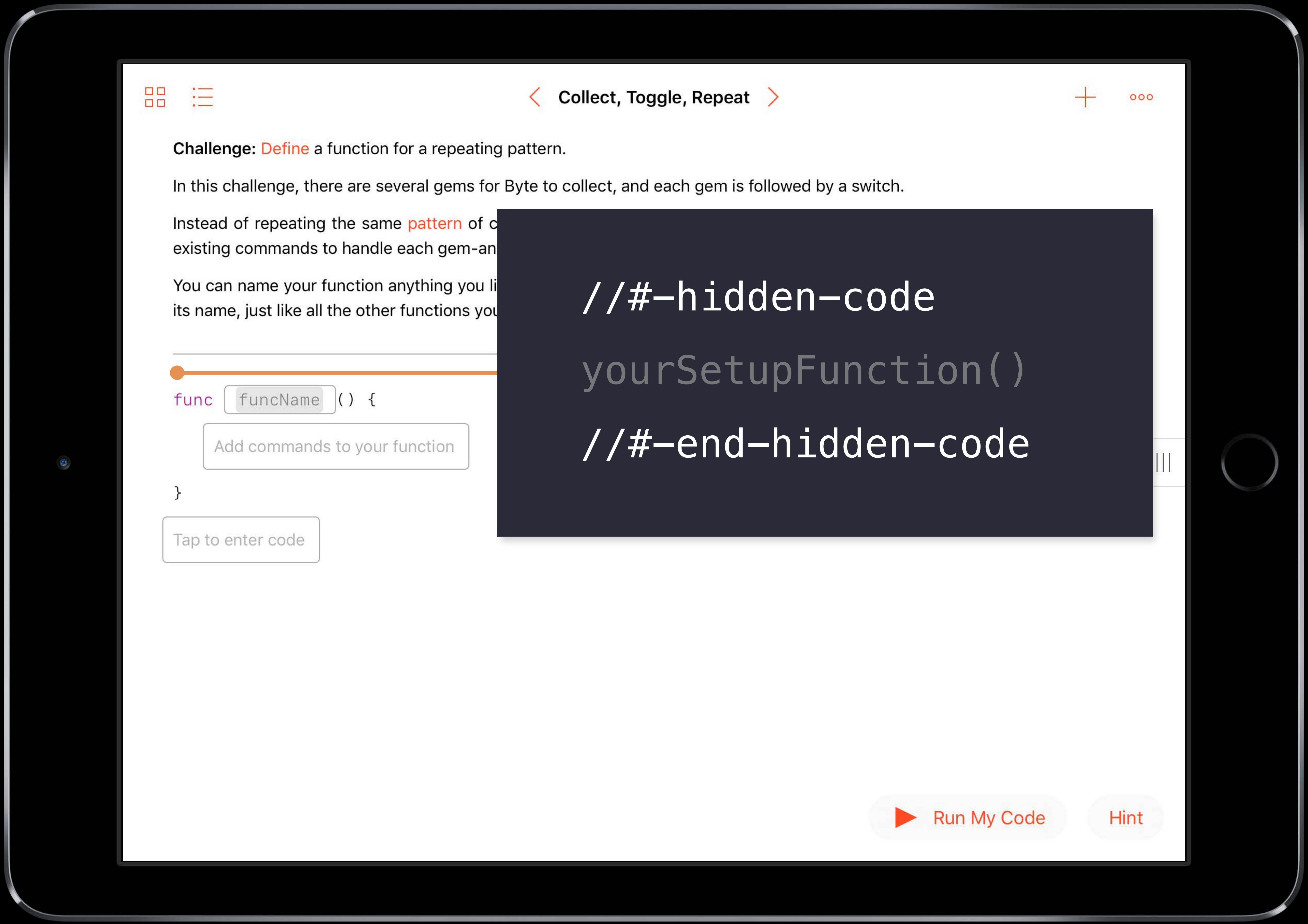
In this challenge, there are several gems for Byte to collect, and each gem is followed by a switch.

Instead of repeating the same pattern of commands you used in previous puzzles, you can write a new function that includes existing commands to handle each gem-and-switch pair.

You can name your function anything you like in this challenge. After you've named and defined your function, call it by entering its name, just like all the other functions you've been using.

```
func funcName() {
    Add commands to your function
}
Tap to enter code
```

Hidden Code



Collect, Toggle, Repeat

Challenge: Define a function for a repeating pattern.

In this challenge, there are several gems for Byte to collect, and each gem is followed by a switch.

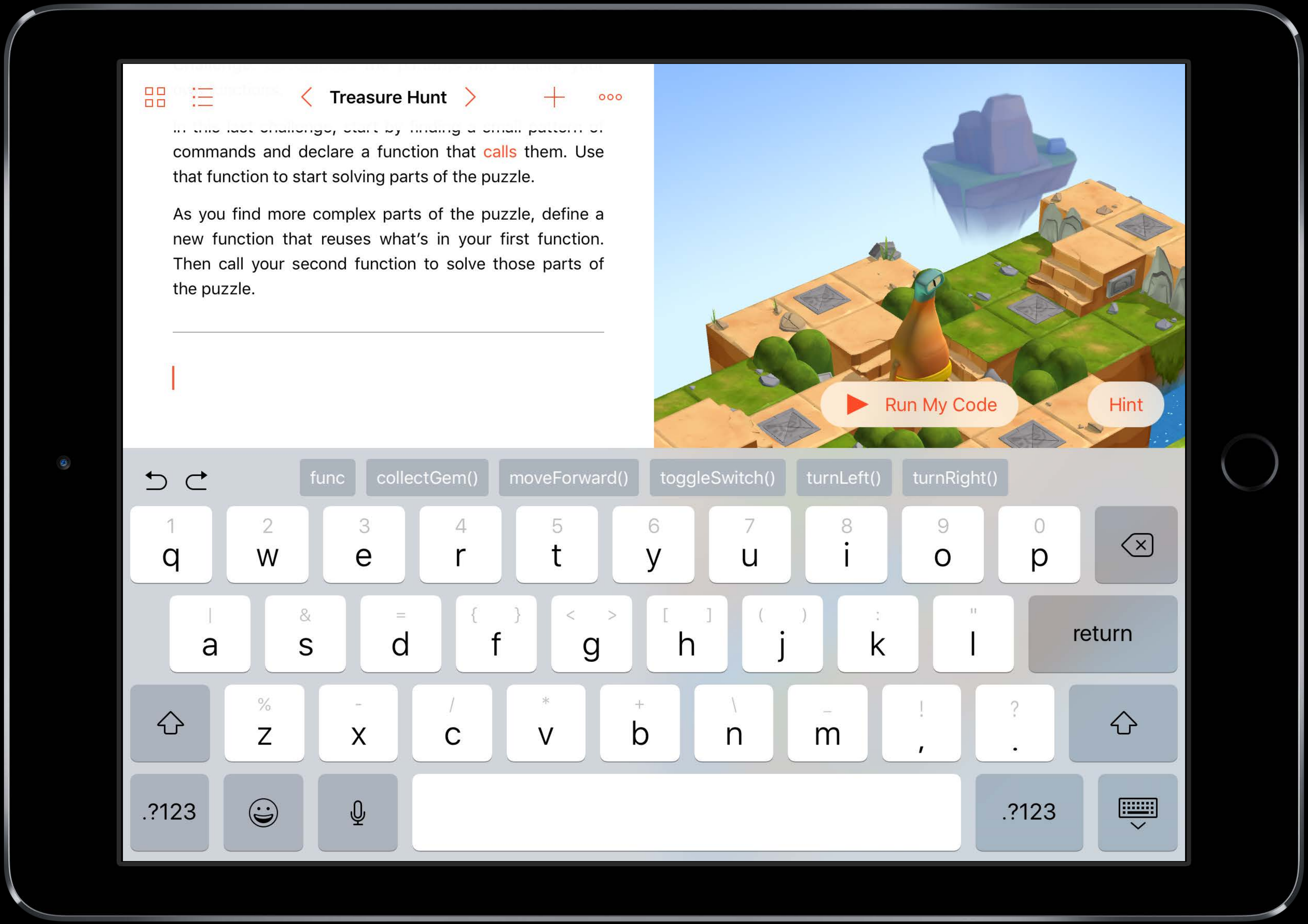
Instead of repeating the same pattern of c
existing commands to handle each gem-an

You can name your function anything you li
its name, just like all the other functions you

```
func funcName() {
    Add commands to your function
}
Tap to enter code
```

```
//#-hidden-code
yourSetupFunction()
//#-end-hidden-code
```

Configurable Code Completion



commands and declare a function that calls them. Use that function to start solving parts of the puzzle.

As you find more complex parts of the puzzle, define a new function that reuses what's in your first function. Then call your second function to solve those parts of the puzzle.

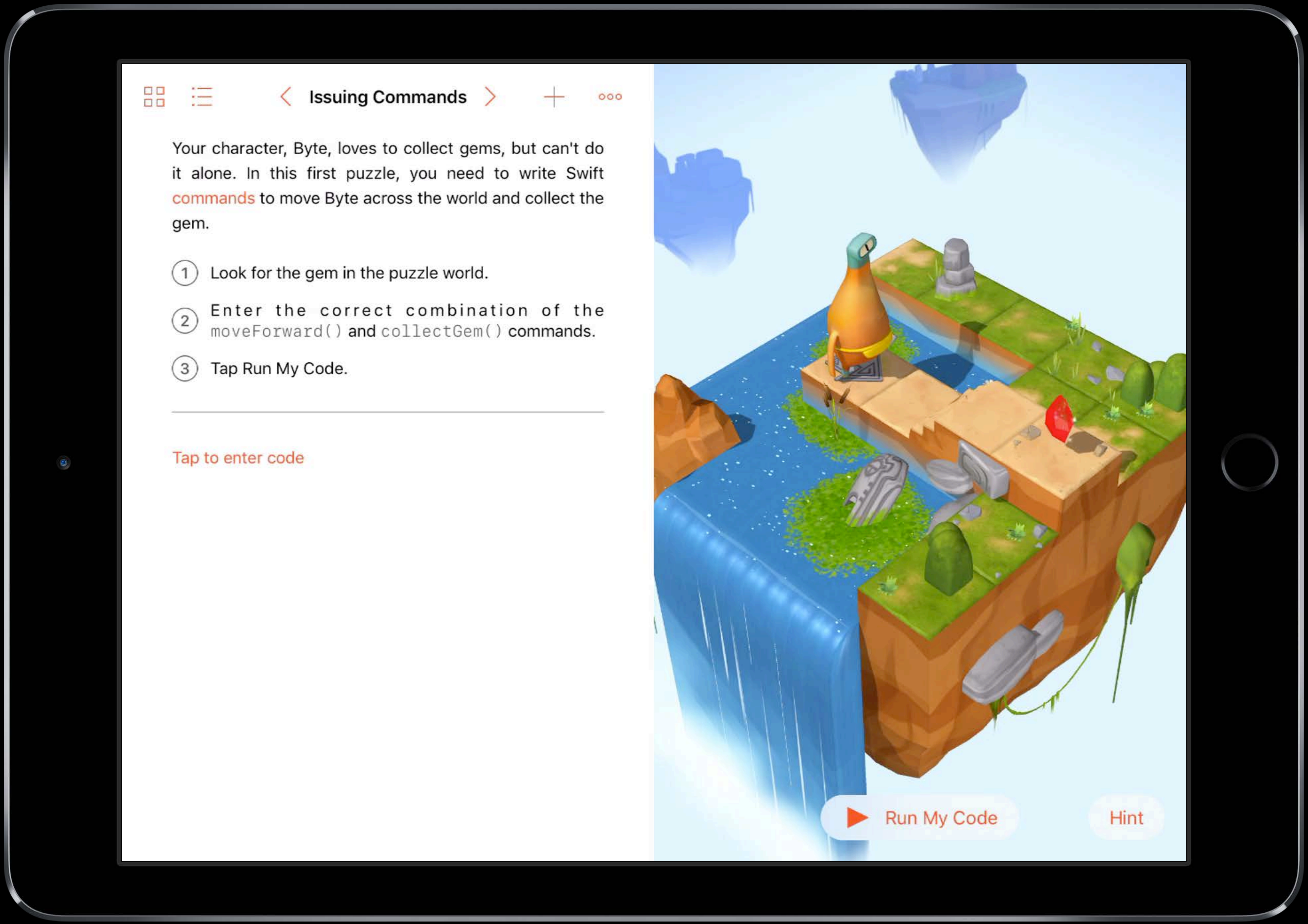
Configurable Code Completion

commands and declare a function that calls them. Use that function to start solving parts of the puzzle.

As you find more complex parts of the puzzle, define a new function that reuses what's in your first function. Then call your second function to solve those parts of the puzzle.

```
//#-code-completion(everything, hide)
//#-code-completion(currentmodule, show)
//#-code-completion(identifier, show, moveForward(), turnLeft())
```

“Always-On” Live View



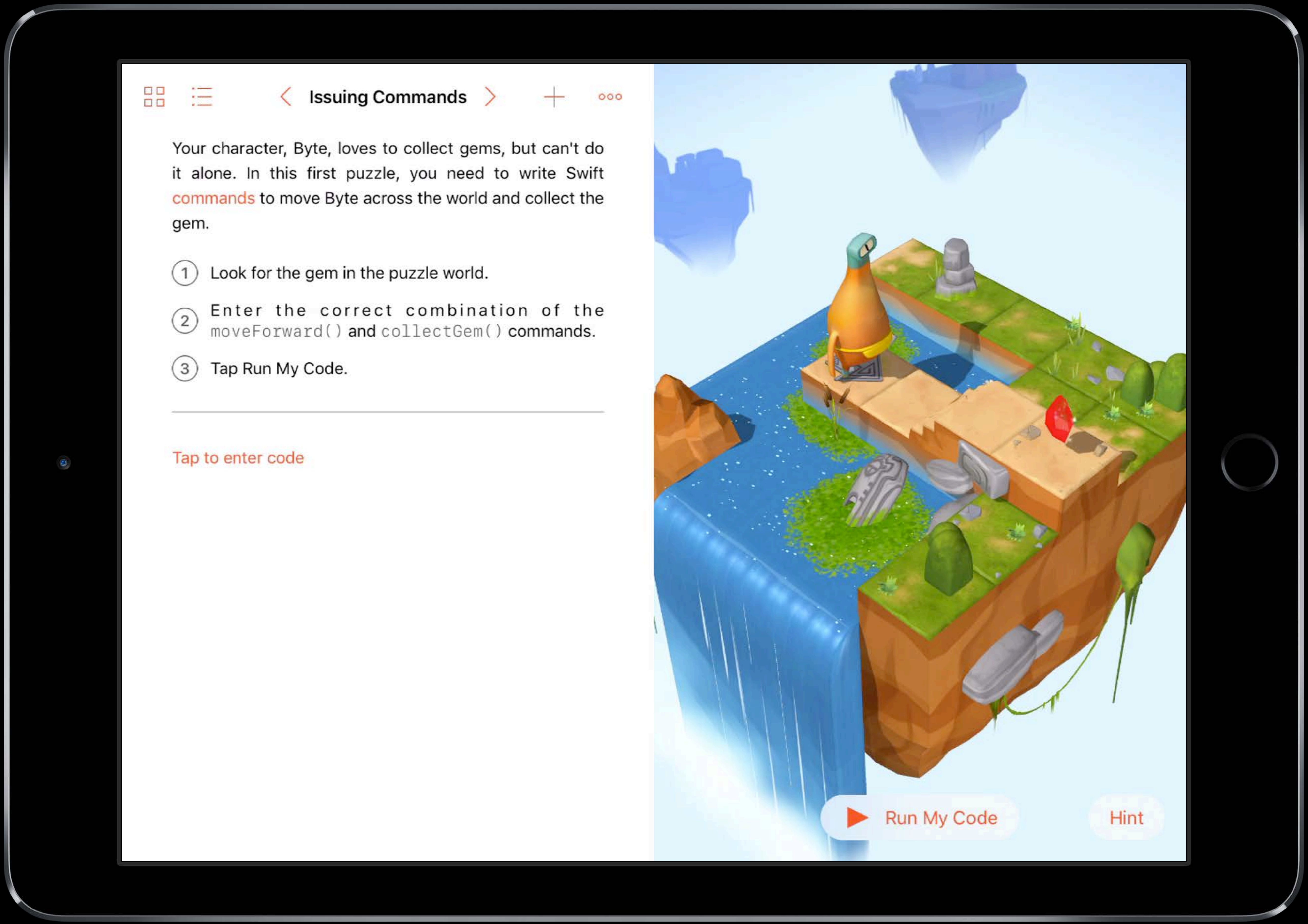
Issuing Commands

Your character, Byte, loves to collect gems, but can't do it alone. In this first puzzle, you need to write Swift commands to move Byte across the world and collect the gem.

1. Look for the gem in the puzzle world.
2. Enter the correct combination of the `moveForward()` and `collectGem()` commands.
3. Tap Run My Code.

Tap to enter code

"Always-On" Live View



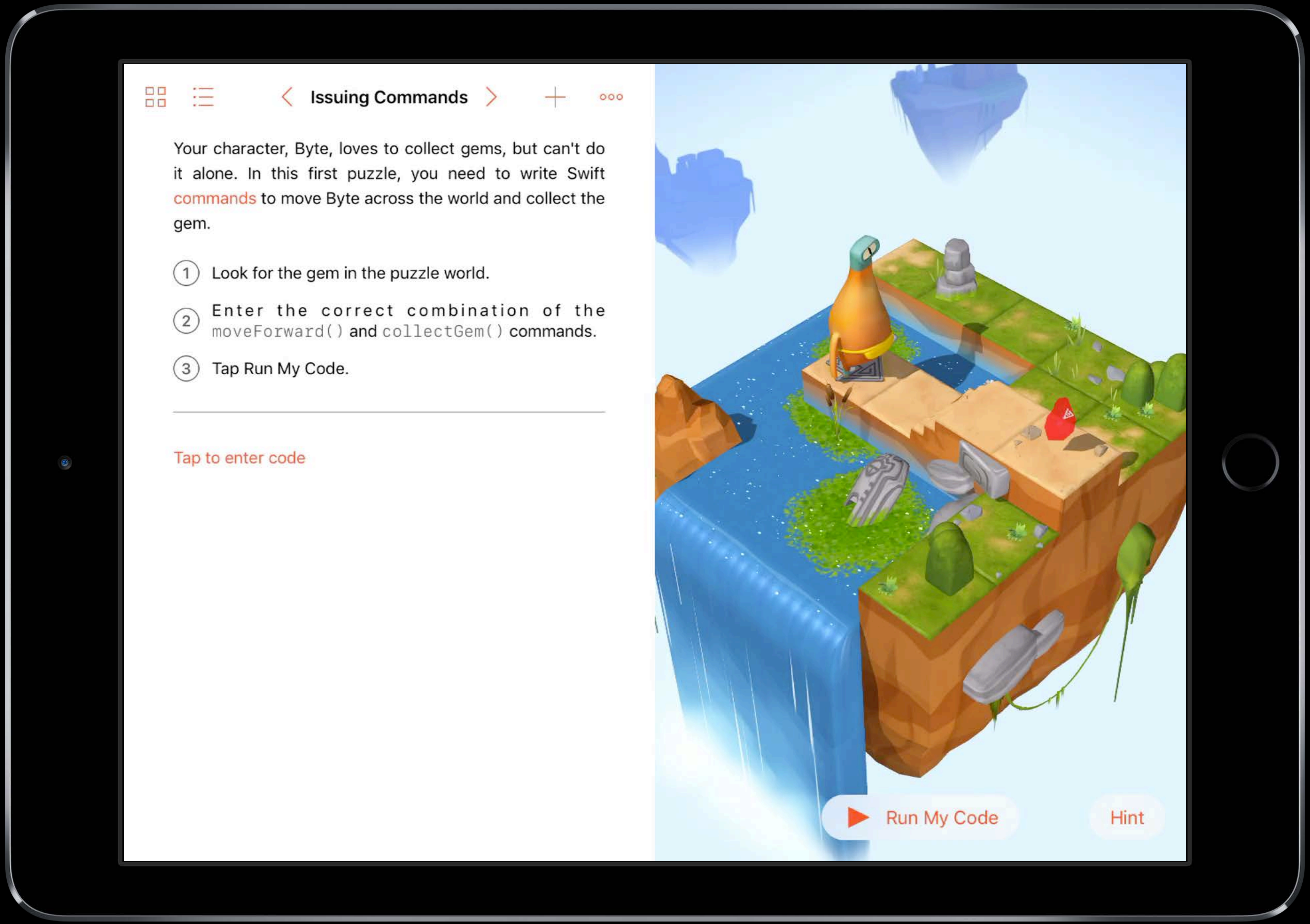
Issuing Commands

Your character, Byte, loves to collect gems, but can't do it alone. In this first puzzle, you need to write Swift commands to move Byte across the world and collect the gem.

1. Look for the gem in the puzzle world.
2. Enter the correct combination of the `moveForward()` and `collectGem()` commands.
3. Tap Run My Code.

Tap to enter code

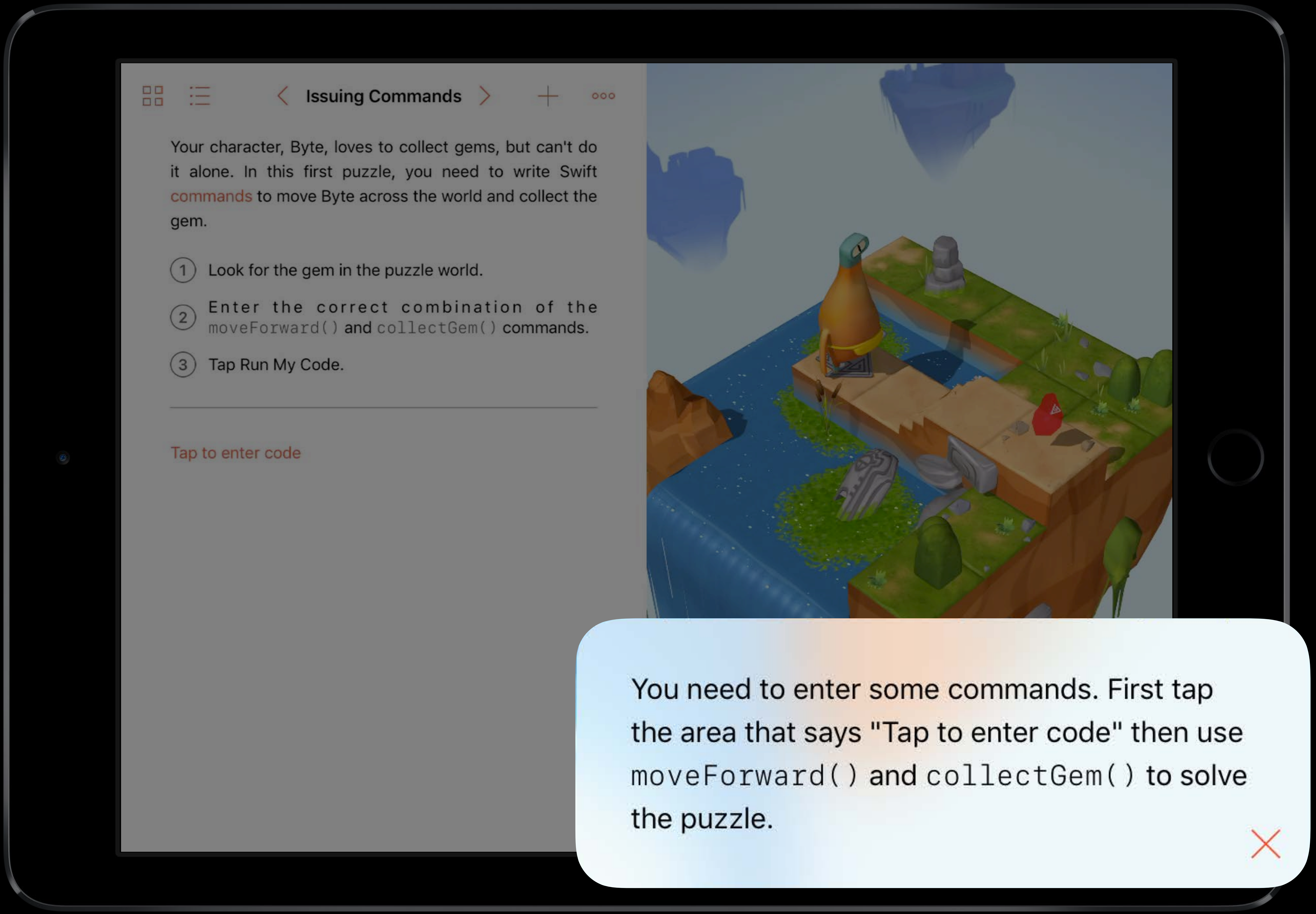
Hints



Your character, Byte, loves to collect gems, but can't do it alone. In this first puzzle, you need to write Swift commands to move Byte across the world and collect the gem.

1. Look for the gem in the puzzle world.
2. Enter the correct combination of the `moveForward()` and `collectGem()` commands.
3. Tap Run My Code.

Hints


Issuing Commands

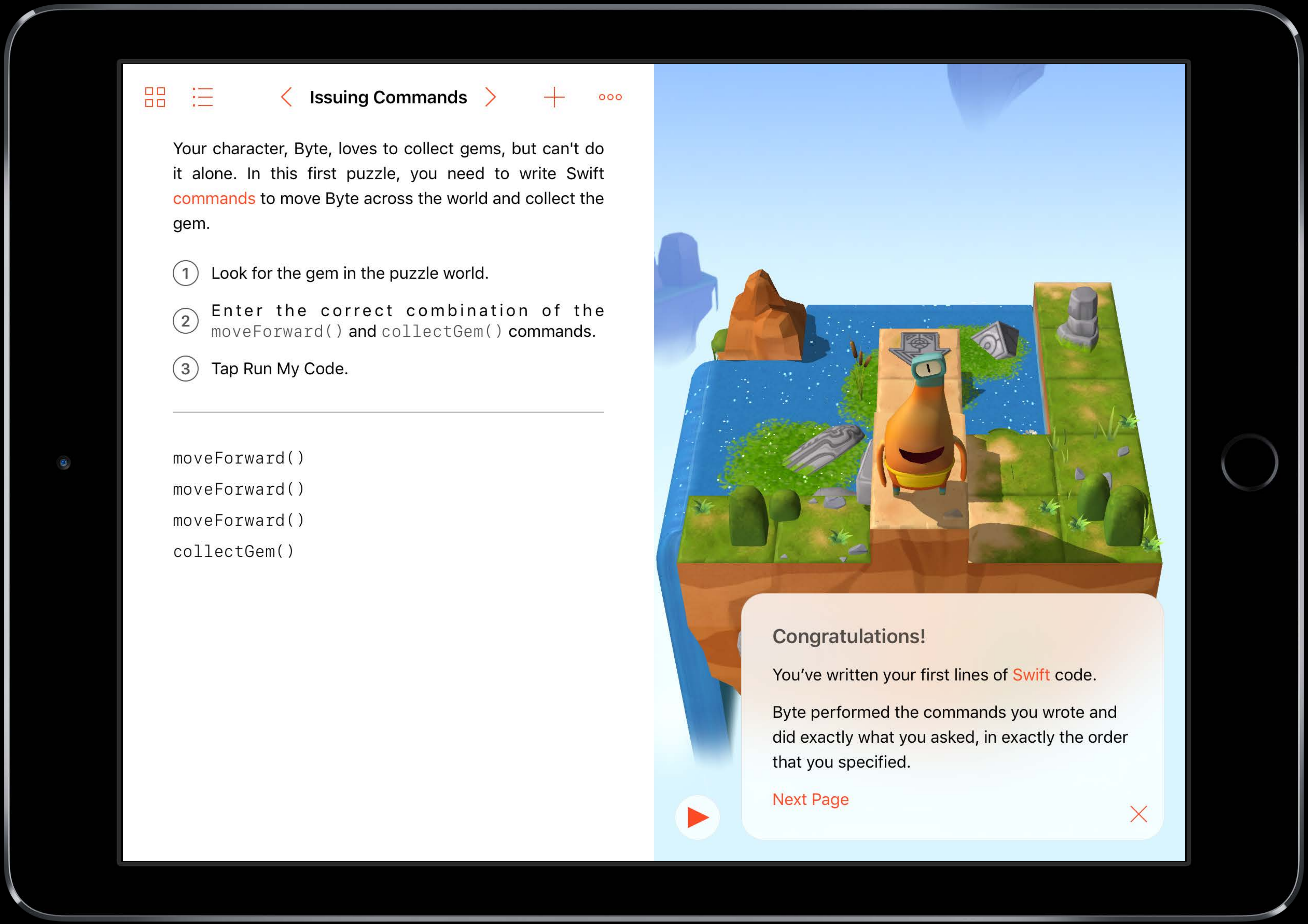
Your character, Byte, loves to collect gems, but can't do it alone. In this first puzzle, you need to write Swift commands to move Byte across the world and collect the gem.

1. Look for the gem in the puzzle world.
2. Enter the correct combination of the `moveForward()` and `collectGem()` commands.
3. Tap Run My Code.

Tap to enter code

You need to enter some commands. First tap the area that says "Tap to enter code" then use `moveForward()` and `collectGem()` to solve the puzzle.

Assessment



Your character, Byte, loves to collect gems, but can't do it alone. In this first puzzle, you need to write Swift commands to move Byte across the world and collect the gem.

1. Look for the gem in the puzzle world.
2. Enter the correct combination of the `moveForward()` and `collectGem()` commands.
3. Tap Run My Code.

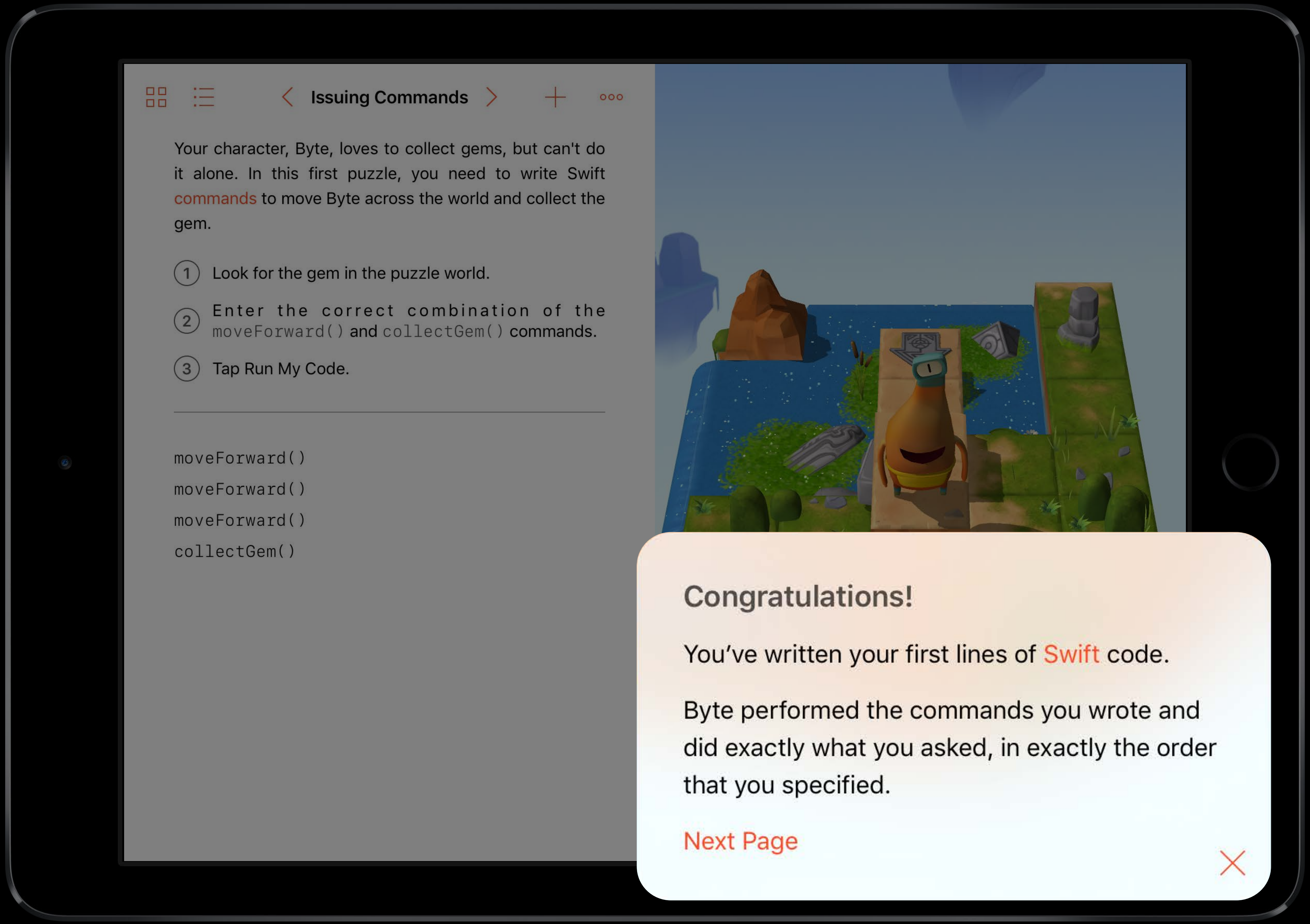
```
moveForward()
moveForward()
moveForward()
collectGem()
```

Congratulations!

You've written your first lines of Swift code.

Byte performed the commands you wrote and did exactly what you asked, in exactly the order that you specified.

Assessment



Your character, Byte, loves to collect gems, but can't do it alone. In this first puzzle, you need to write Swift commands to move Byte across the world and collect the gem.

1. Look for the gem in the puzzle world.
2. Enter the correct combination of the `moveForward()` and `collectGem()` commands.
3. Tap Run My Code.

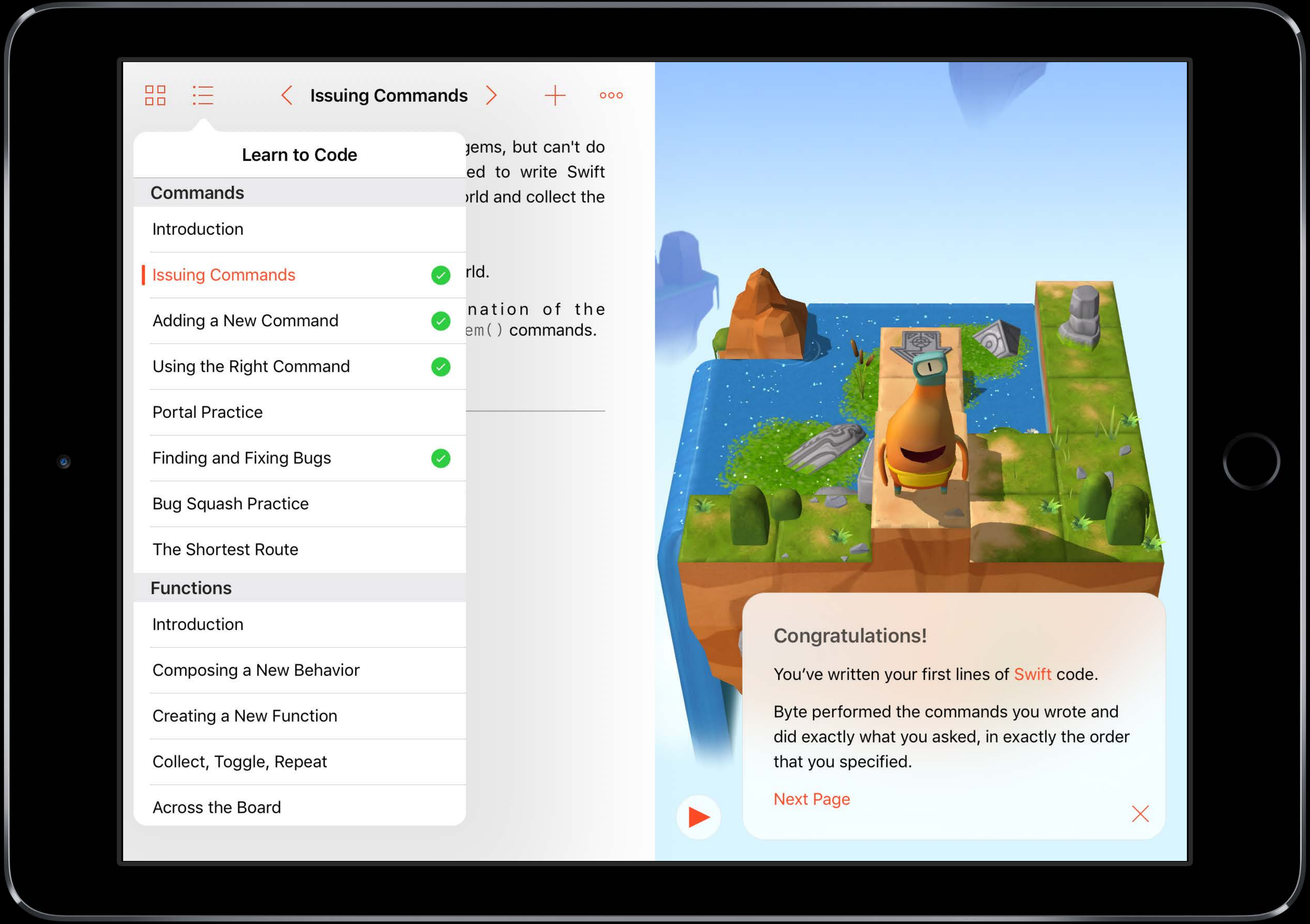
```
moveForward()
moveForward()
moveForward()
collectGem()
```

Congratulations!

You've written your first lines of Swift code.

Byte performed the commands you wrote and did exactly what you asked, in exactly the order that you specified.

Assessment



Issuing Commands

Learn to Code

Commands
- Introduction
- Issuing Commands ✓
- Adding a New Command ✓
- Using the Right Command ✓
- Portal Practice
- Finding and Fixing Bugs ✓
- Bug Squash Practice
- The Shortest Route

Functions
- Introduction
- Composing a New Behavior
- Creating a New Function
- Collect, Toggle, Repeat
- Across the Board

gems, but can't do
ed to write Swift
rld and collect the

rld.

nation of the
em() commands.

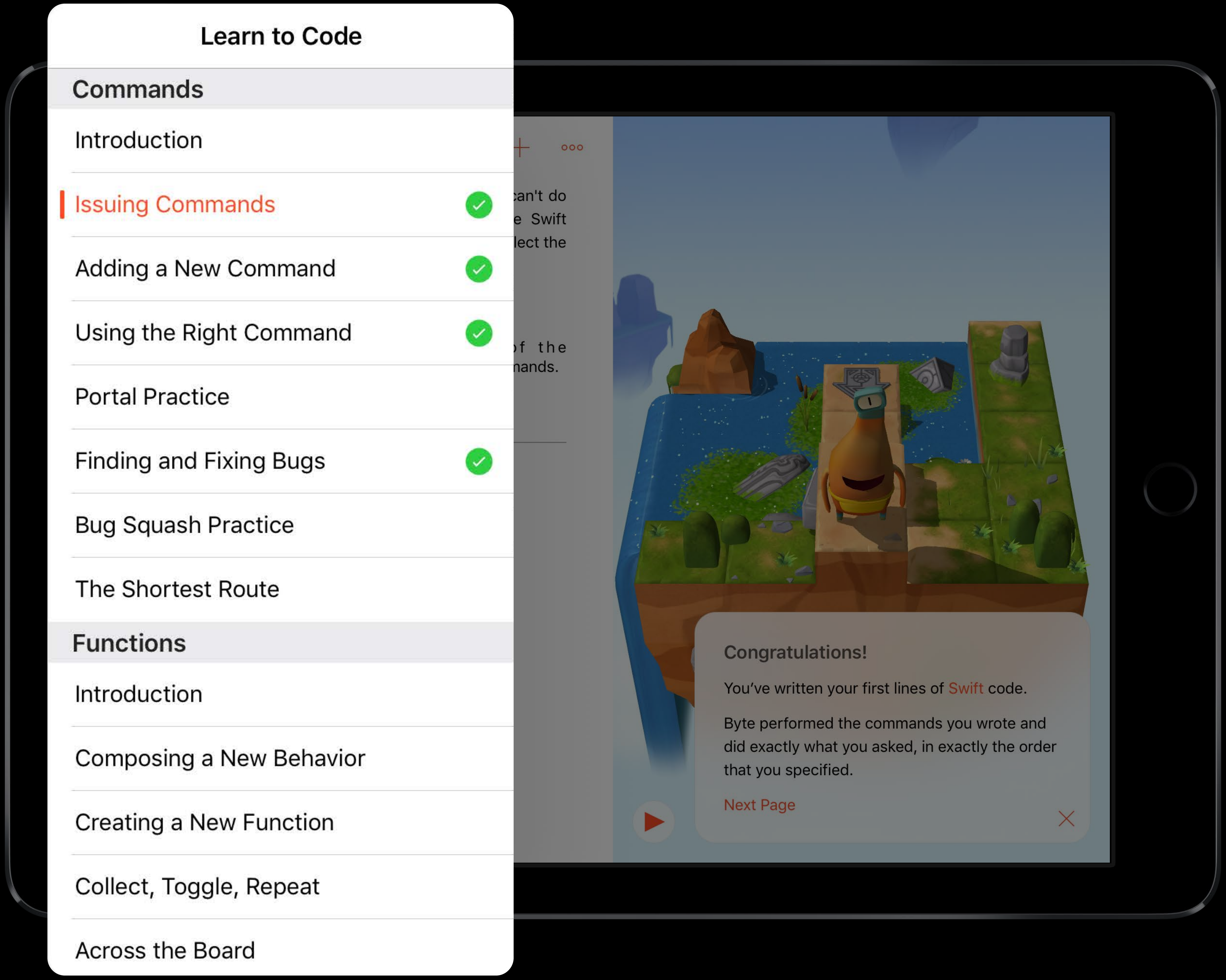
Congratulations!

You've written your first lines of Swift code.

Byte performed the commands you wrote and did exactly what you asked, in exactly the order that you specified.

Next Page

Assessment



Learn to Code

Commands

- Introduction
- Issuing Commands ✓
- Adding a New Command ✓
- Using the Right Command ✓
- Portal Practice
- Finding and Fixing Bugs ✓
- Bug Squash Practice
- The Shortest Route

Functions

- Introduction
- Composing a New Behavior
- Creating a New Function
- Collect, Toggle, Repeat
- Across the Board

Congratulations!

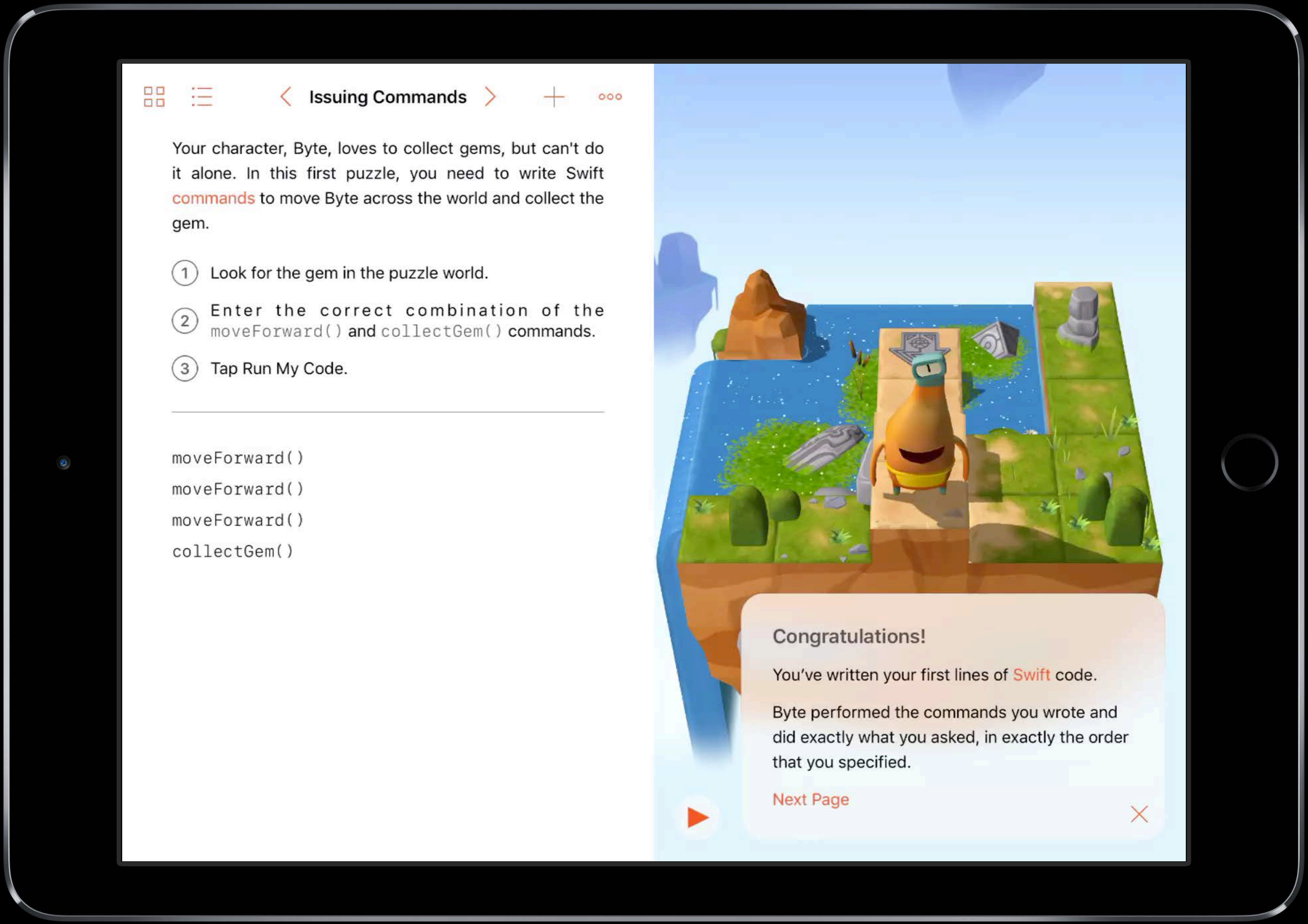
You've written your first lines of Swift code.

Byte performed the commands you wrote and did exactly what you asked, in exactly the order that you specified.

Next Page

```swift
// Key/Value Store

import PlaygroundSupport

let store = PlaygroundPage.current.keyValueStore

store["Greeting"] = .string("Hello, WWDC!")

if case let .string(greeting)? = store["Greeting"] {
    print(greeting)     // "Hello, WWDC!"
}
```

Resettable



Your character, Byte, loves to collect gems, but can't do it alone. In this first puzzle, you need to write Swift commands to move Byte across the world and collect the gem.

1. Look for the gem in the puzzle world.
2. Enter the correct combination of the `moveForward()` and `collectGem()` commands.
3. Tap Run My Code.

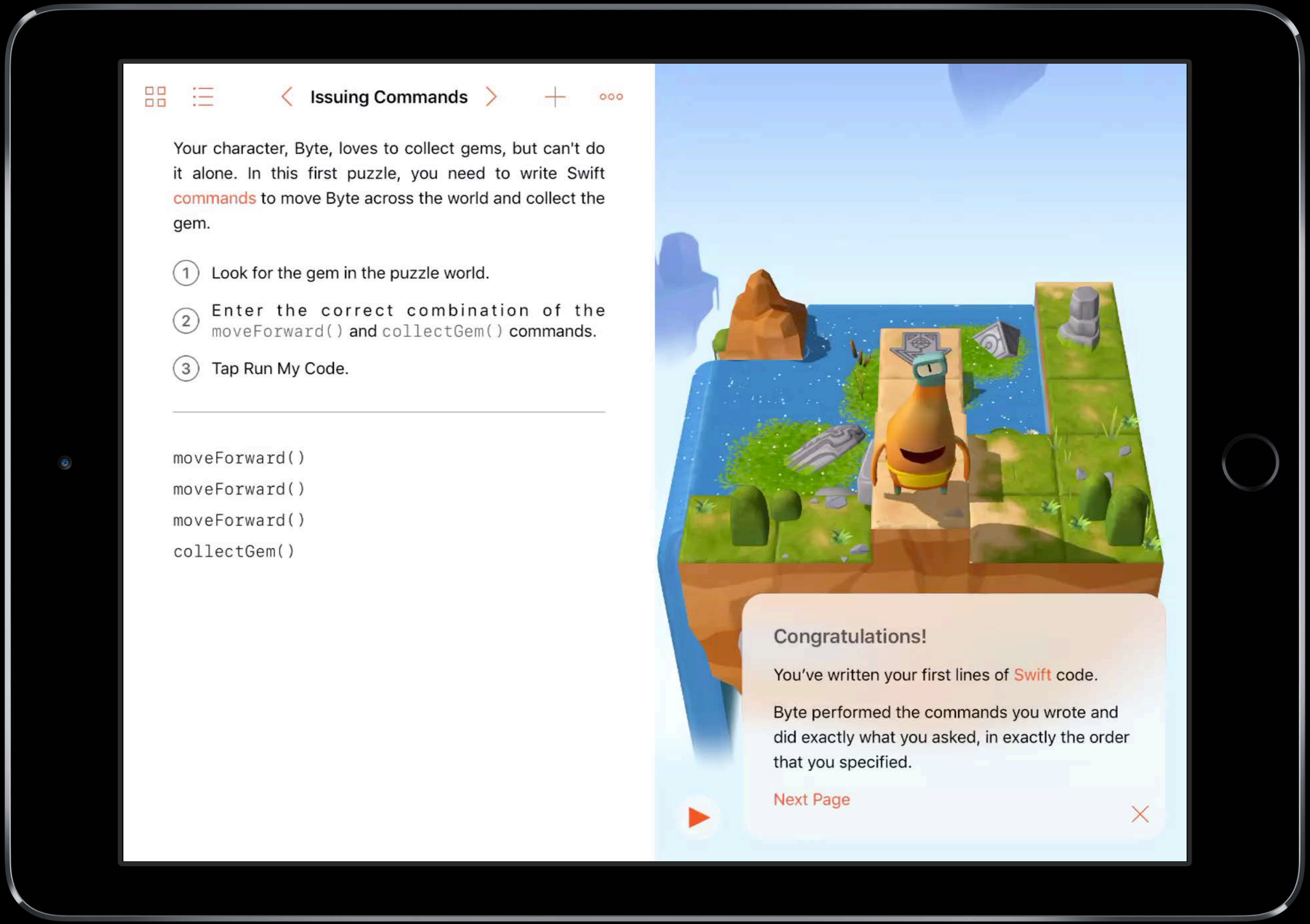
```
moveForward()
moveForward()
moveForward()
collectGem()
```

Congratulations!

You've written your first lines of Swift code.

Byte performed the commands you wrote and did exactly what you asked, in exactly the order that you specified.

Resettable



Your character, Byte, loves to collect gems, but can't do it alone. In this first puzzle, you need to write Swift commands to move Byte across the world and collect the gem.

1. Look for the gem in the puzzle world.
2. Enter the correct combination of the `moveForward()` and `collectGem()` commands.
3. Tap Run My Code.

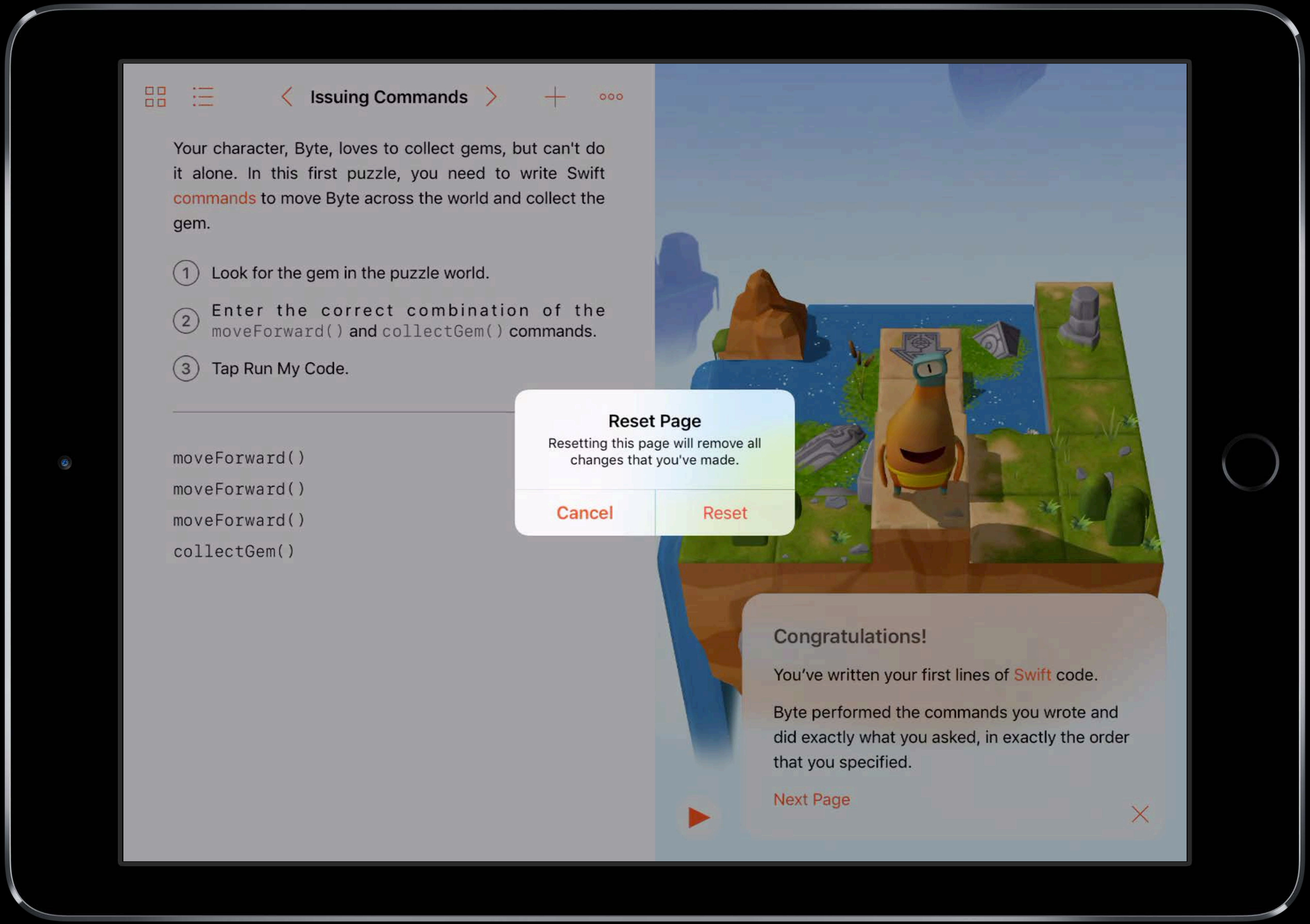
```
moveForward()
moveForward()
moveForward()
collectGem()
```

Congratulations!

You've written your first lines of Swift code.

Byte performed the commands you wrote and did exactly what you asked, in exactly the order that you specified.

Resettable



Issuing Commands

Your character, Byte, loves to collect gems, but can't do it alone. In this first puzzle, you need to write Swift commands to move Byte across the world and collect the gem.

1. Look for the gem in the puzzle world.
2. Enter the correct combination of the `moveForward()` and `collectGem()` commands.
3. Tap Run My Code.

```
moveForward()
moveForward()
moveForward()
collectGem()
```

Reset Page

Resetting this page will remove all changes that you've made.

Cancel | Reset

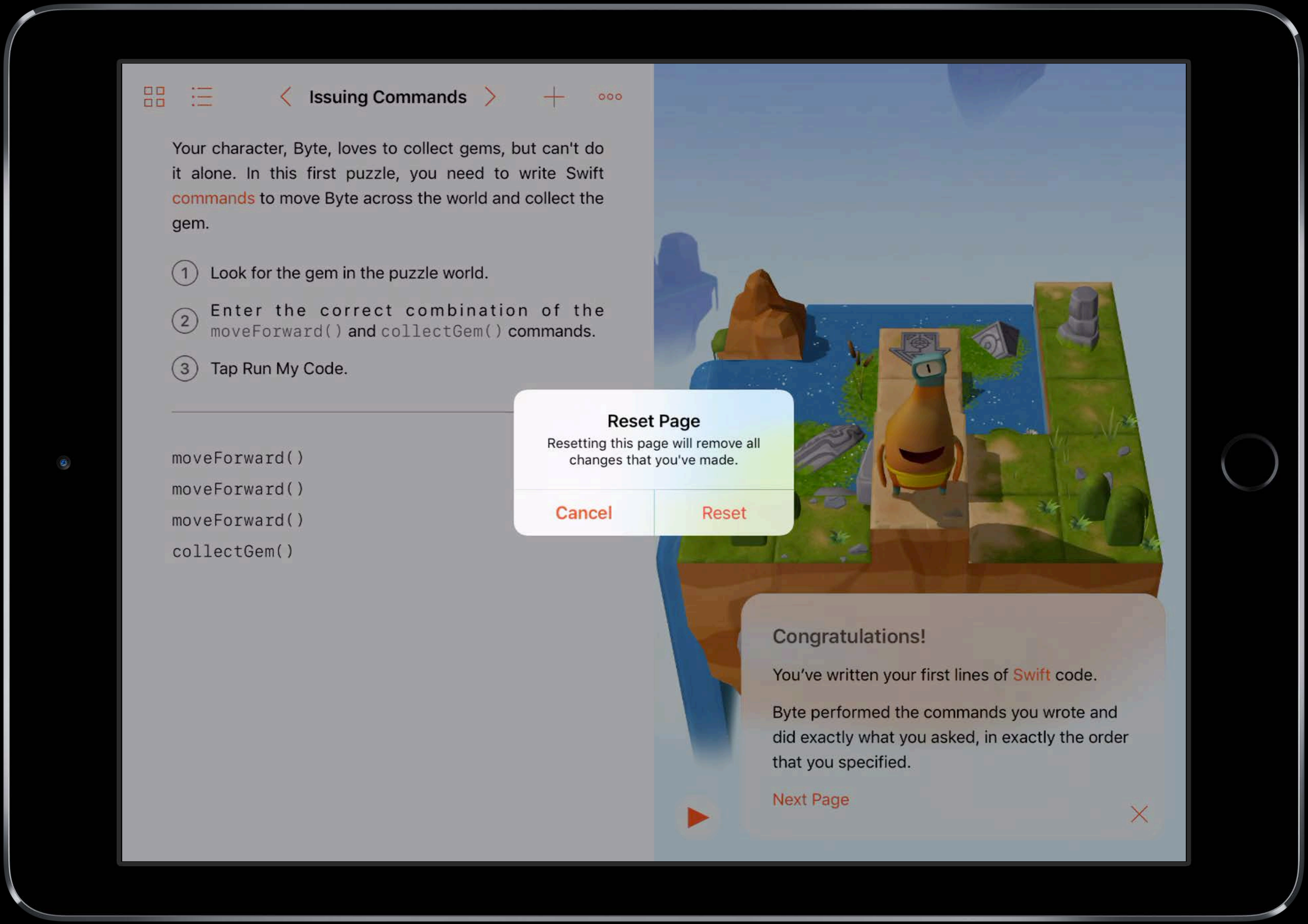
Congratulations!

You've written your first lines of Swift code.

Byte performed the commands you wrote and did exactly what you asked, in exactly the order that you specified.

Next Page

Resettable



Your character, Byte, loves to collect gems, but can't do it alone. In this first puzzle, you need to write Swift commands to move Byte across the world and collect the gem.

1. Look for the gem in the puzzle world.
2. Enter the correct combination of the `moveForward()` and `collectGem()` commands.
3. Tap Run My Code.

```
moveForward()
moveForward()
moveForward()
collectGem()
```

Reset Page

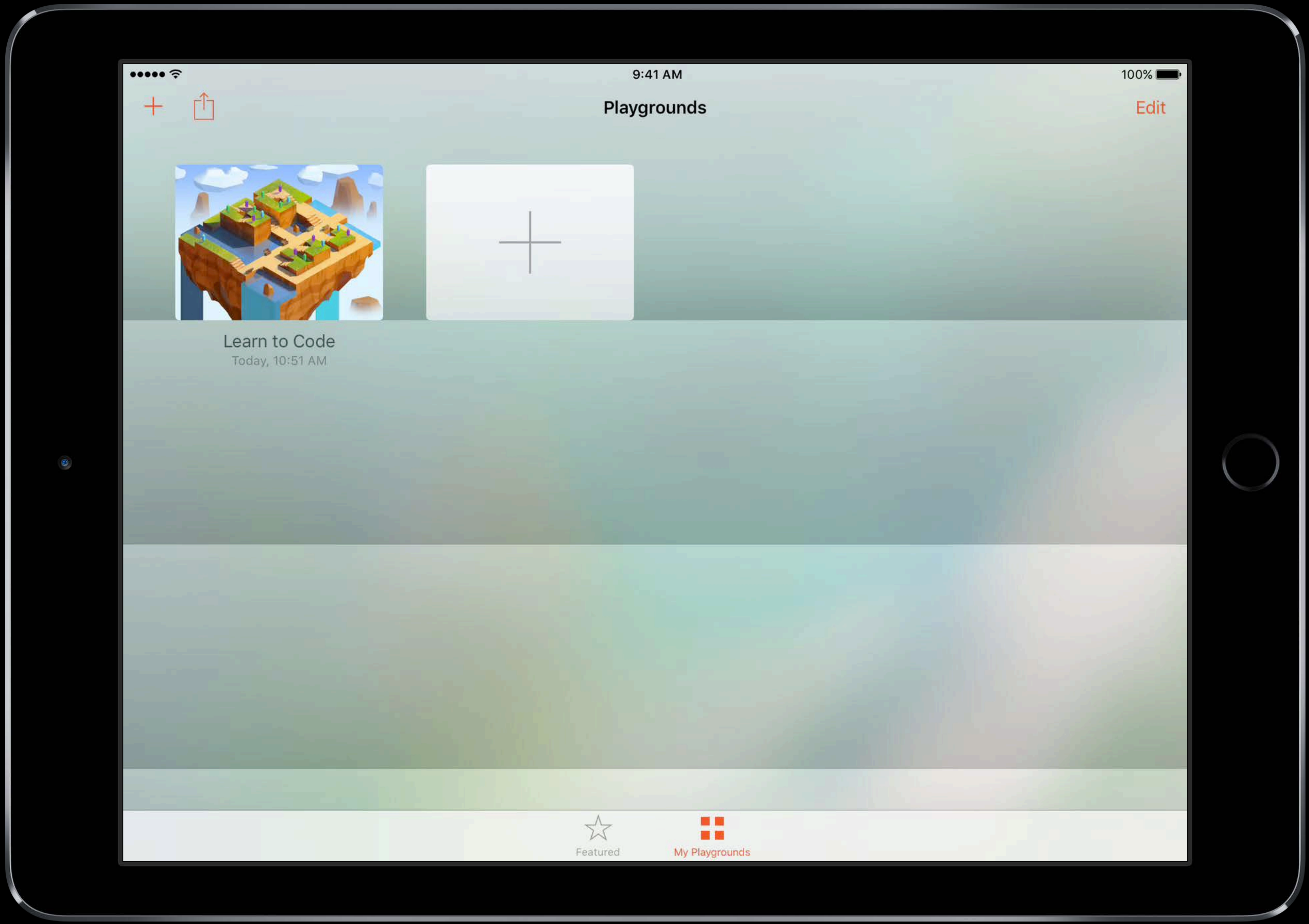
Resetting this page will remove all changes that you've made.

Congratulations!

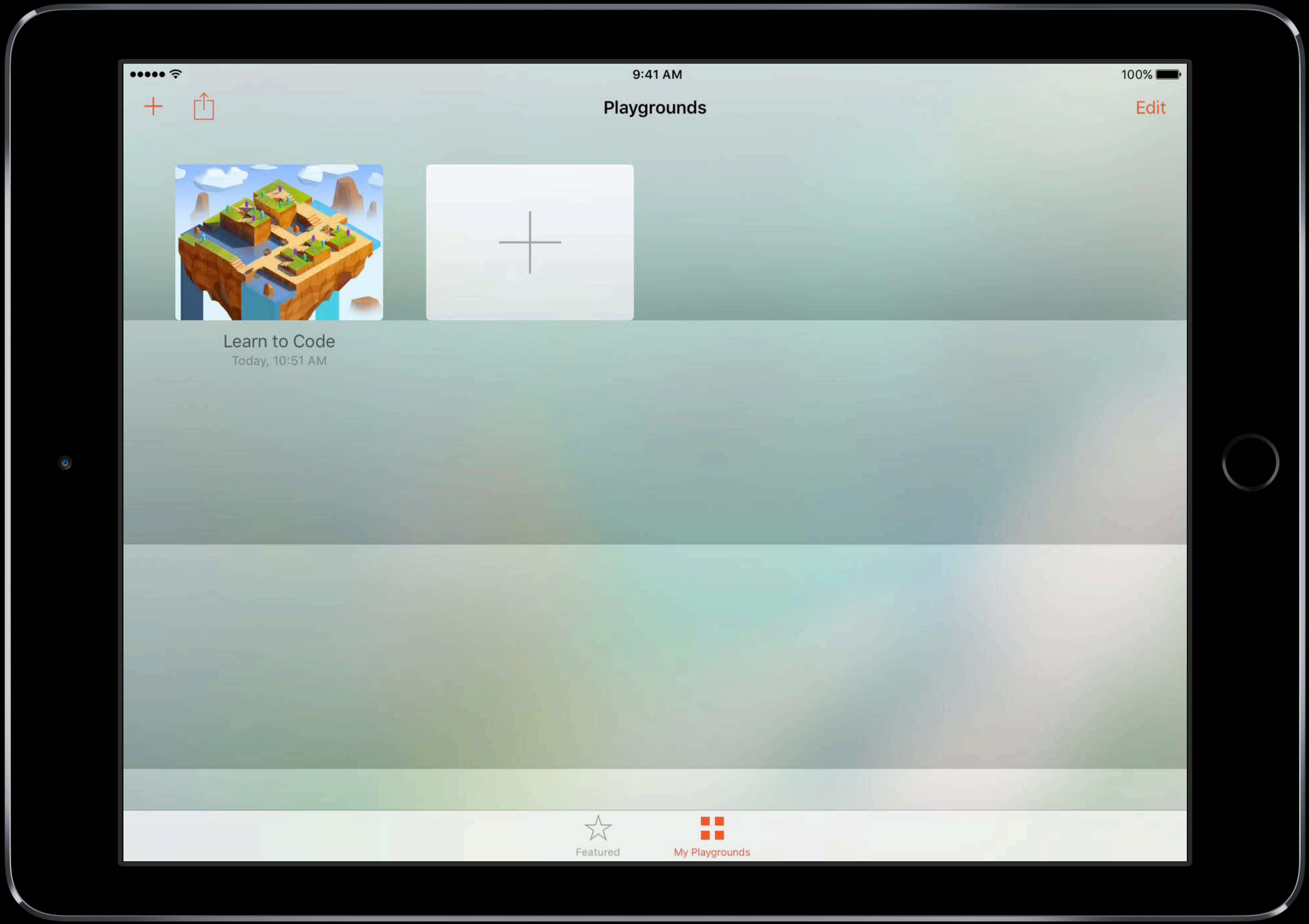
You've written your first lines of Swift code.

Byte performed the commands you wrote and did exactly what you asked, in exactly the order that you specified.

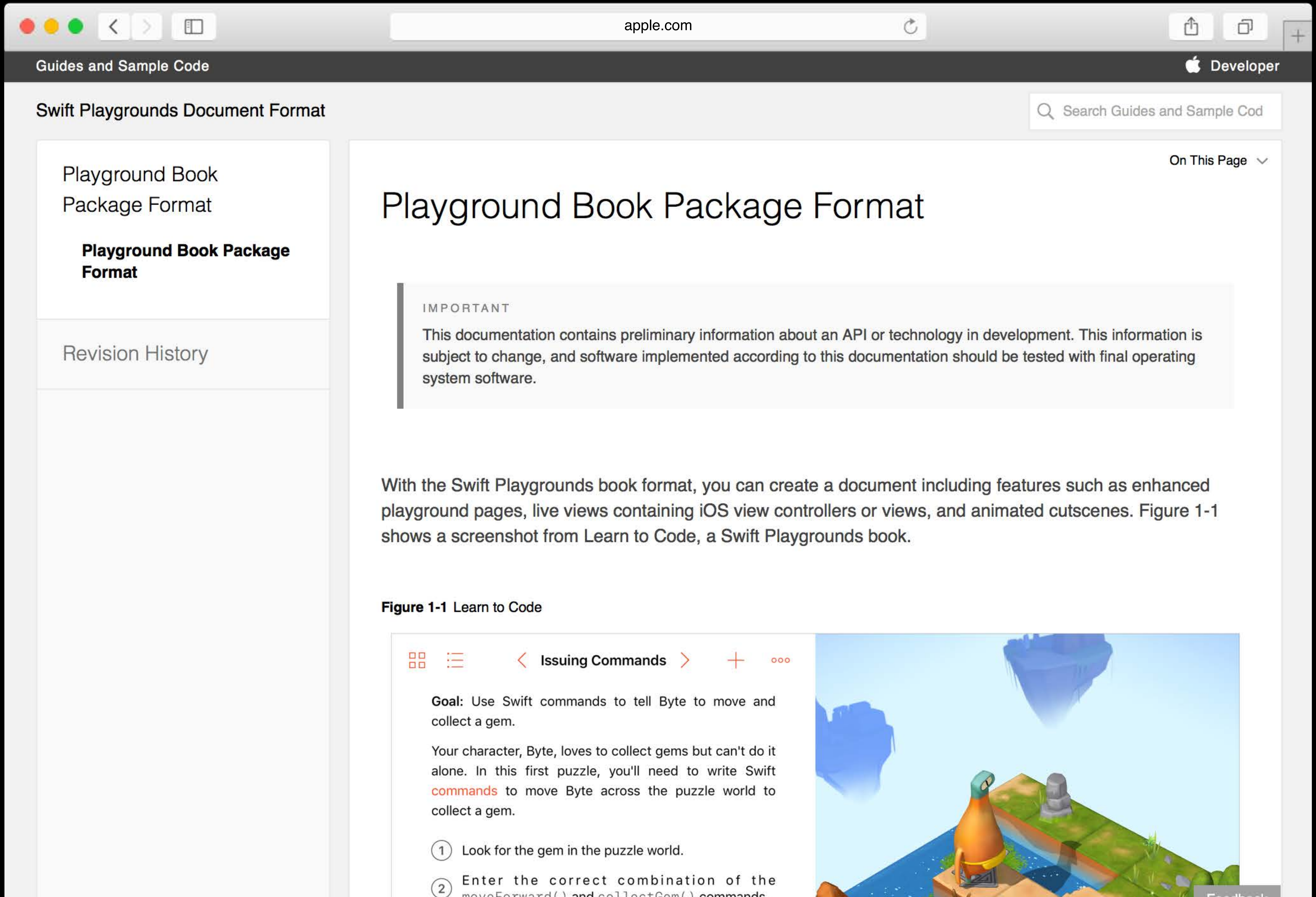
Resettable


9:41 AM
100%
Playgrounds
Edit
Learn to Code
Today, 10:51 AM
Featured
My Playgrounds

Resettable


9:41 AM
100%
Playgrounds
Edit
Learn to Code
Today, 10:51 AM
Featured
My Playgrounds

Documented



Swift Playgrounds Document Format

Playground Book Package Format

Playground Book Package Format

Revision History

Playground Book Package Format

IMPORTANT

This documentation contains preliminary information about an API or technology in development. This information is subject to change, and software implemented according to this documentation should be tested with final operating system software.

With the Swift Playgrounds book format, you can create a document including features such as enhanced playground pages, live views containing iOS view controllers or views, and animated cutscenes. Figure 1-1 shows a screenshot from Learn to Code, a Swift Playgrounds book.

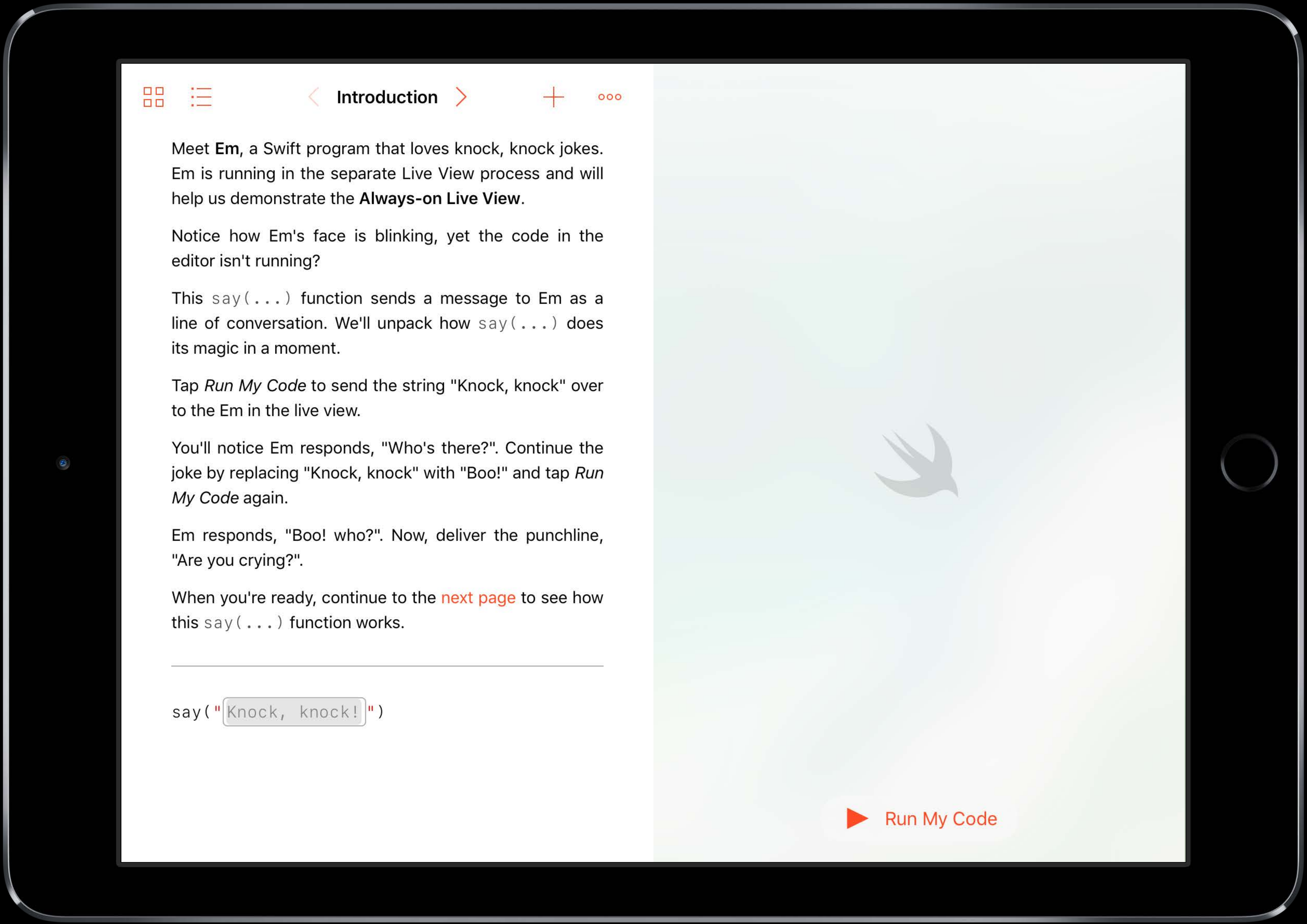
Figure 1-1 Learn to Code

Goal: Use Swift commands to tell Byte to move and collect a gem.

Your character, Byte, loves to collect gems but can't do it alone. In this first puzzle, you'll need to write Swift commands to move Byte across the puzzle world to collect a gem.

1. Look for the gem in the puzzle world.
2. Enter the correct combination of the `moveForward()` and `collectGem()` commands.

BOOK



Meet **Em**, a Swift program that loves knock, knock jokes. Em is running in the separate Live View process and will help us demonstrate the **Always-on Live View**.

Notice how Em's face is blinking, yet the code in the editor isn't running?

This `say(...)` function sends a message to Em as a line of conversation. We'll unpack how `say(...)` does its magic in a moment.

Tap *Run My Code* to send the string "Knock, knock" over to the Em in the live view.

You'll notice Em responds, "Who's there?". Continue the joke by replacing "Knock, knock" with "Boo!" and tap *Run My Code* again.

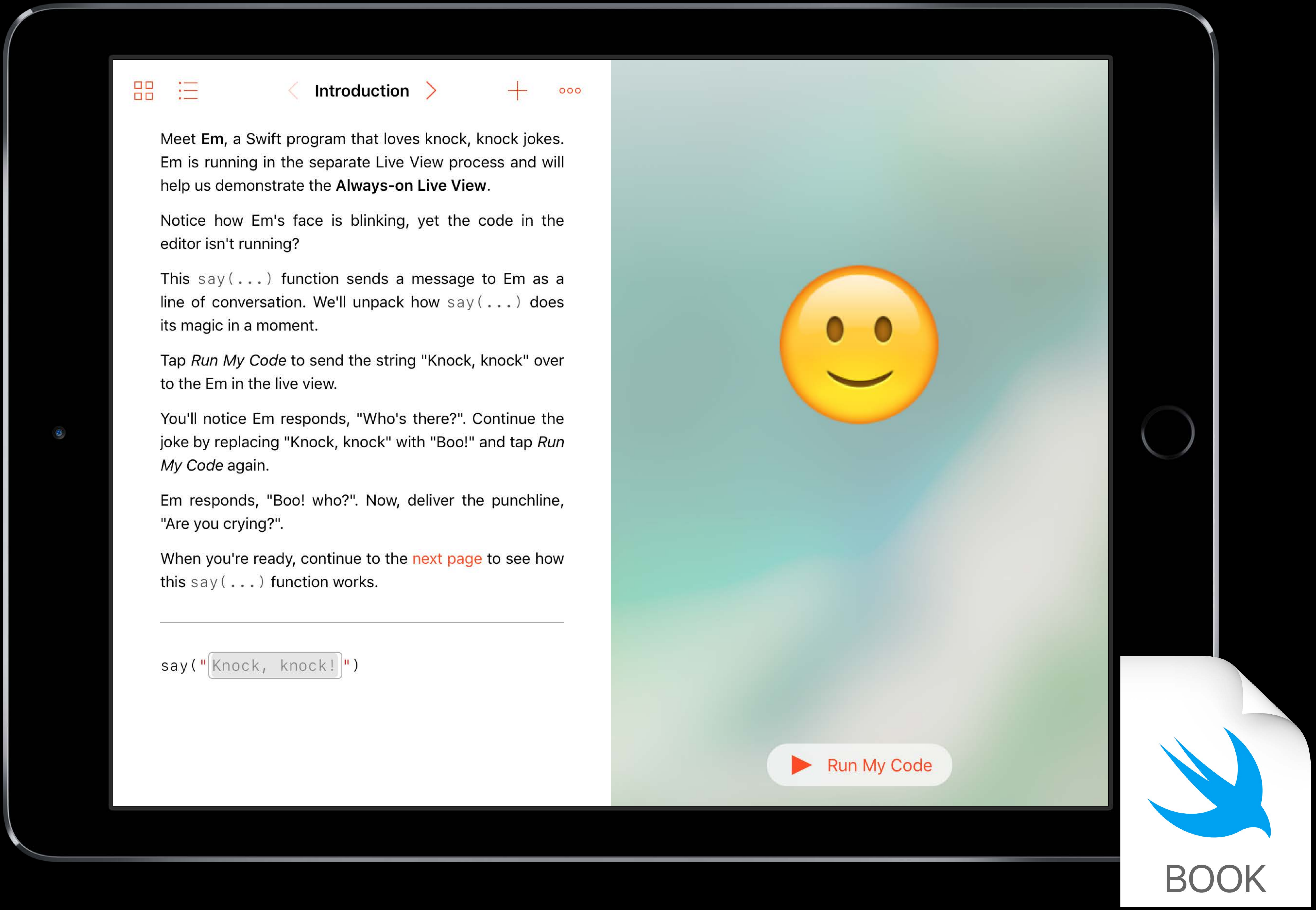
Em responds, "Boo! who?". Now, deliver the punchline, "Are you crying?".

When you're ready, continue to the next page to see how this `say(...)` function works.

```
say("Knock, knock!")
```

Demo

Authoring for Swift Playgrounds



Meet **Em**, a Swift program that loves knock, knock jokes. Em is running in the separate Live View process and will help us demonstrate the **Always-on Live View**.

Notice how Em's face is blinking, yet the code in the editor isn't running?

This `say(...)` function sends a message to Em as a line of conversation. We'll unpack how `say(...)` does its magic in a moment.

Tap *Run My Code* to send the string "Knock, knock" over to the Em in the live view.

You'll notice Em responds, "Who's there?". Continue the joke by replacing "Knock, knock" with "Boo!" and tap *Run My Code* again.

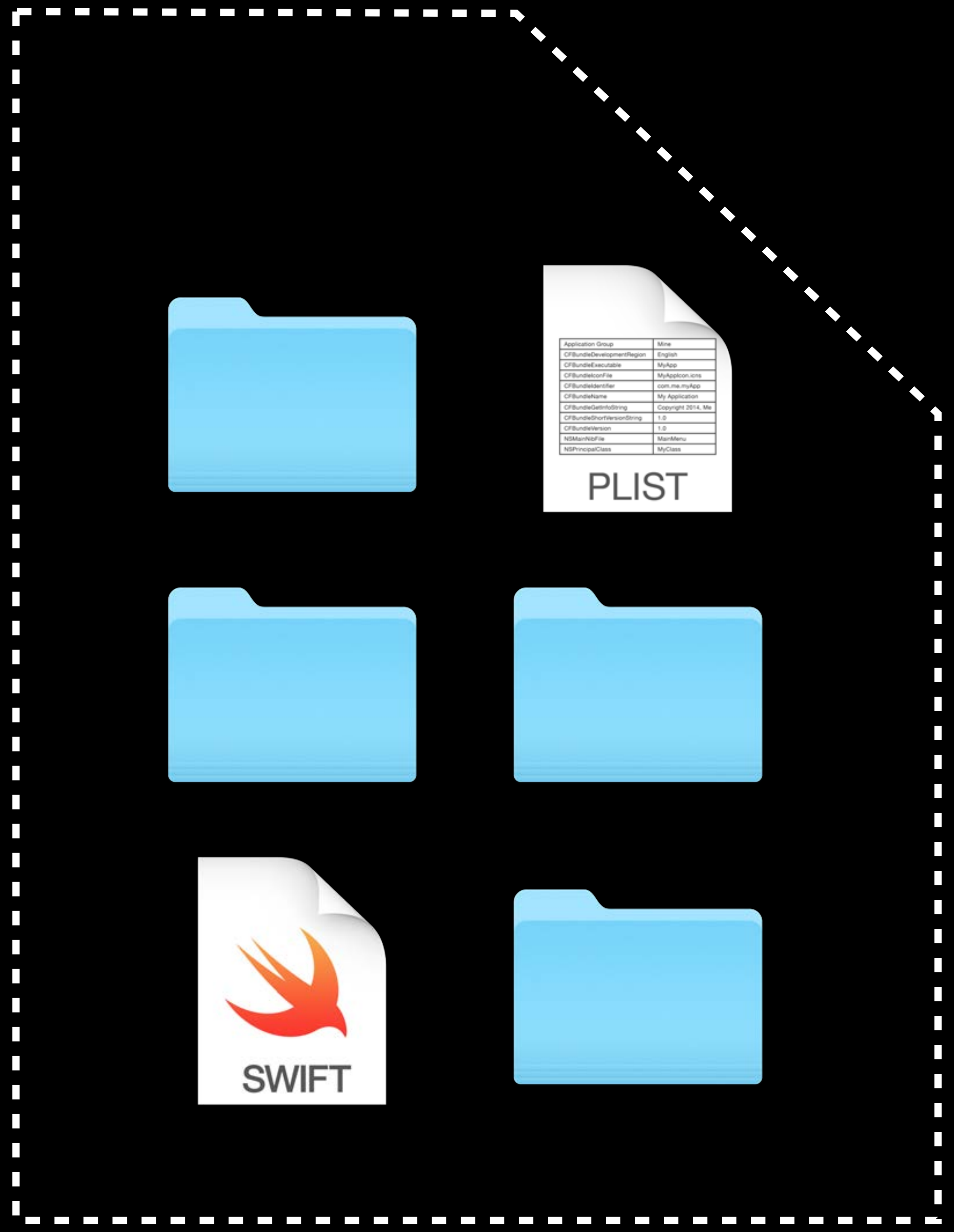
Em responds, "Boo! who?". Now, deliver the punchline, "Are you crying?".

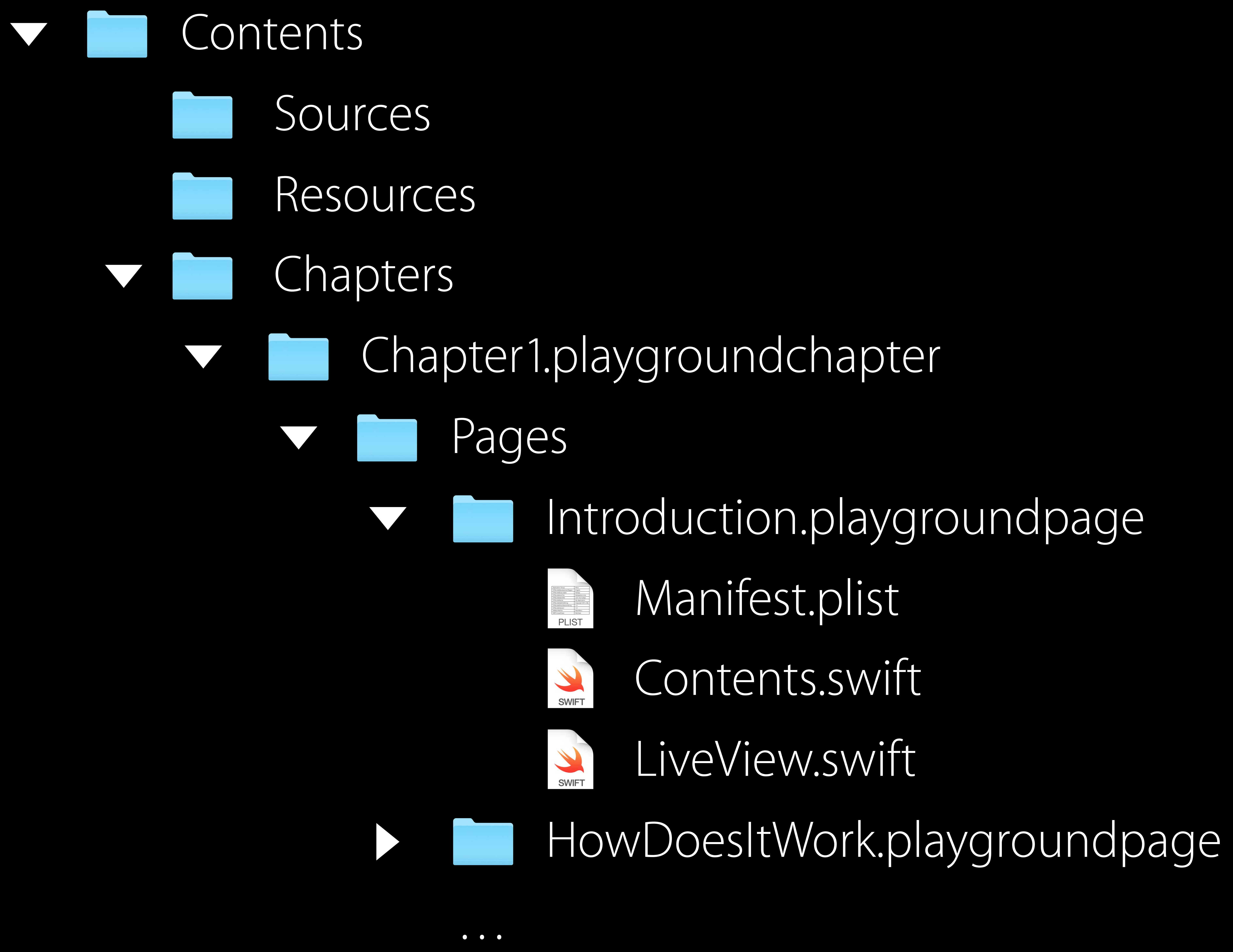
When you're ready, continue to the next page to see how this `say(...)` function works.

```
say("Knock, knock!")
```

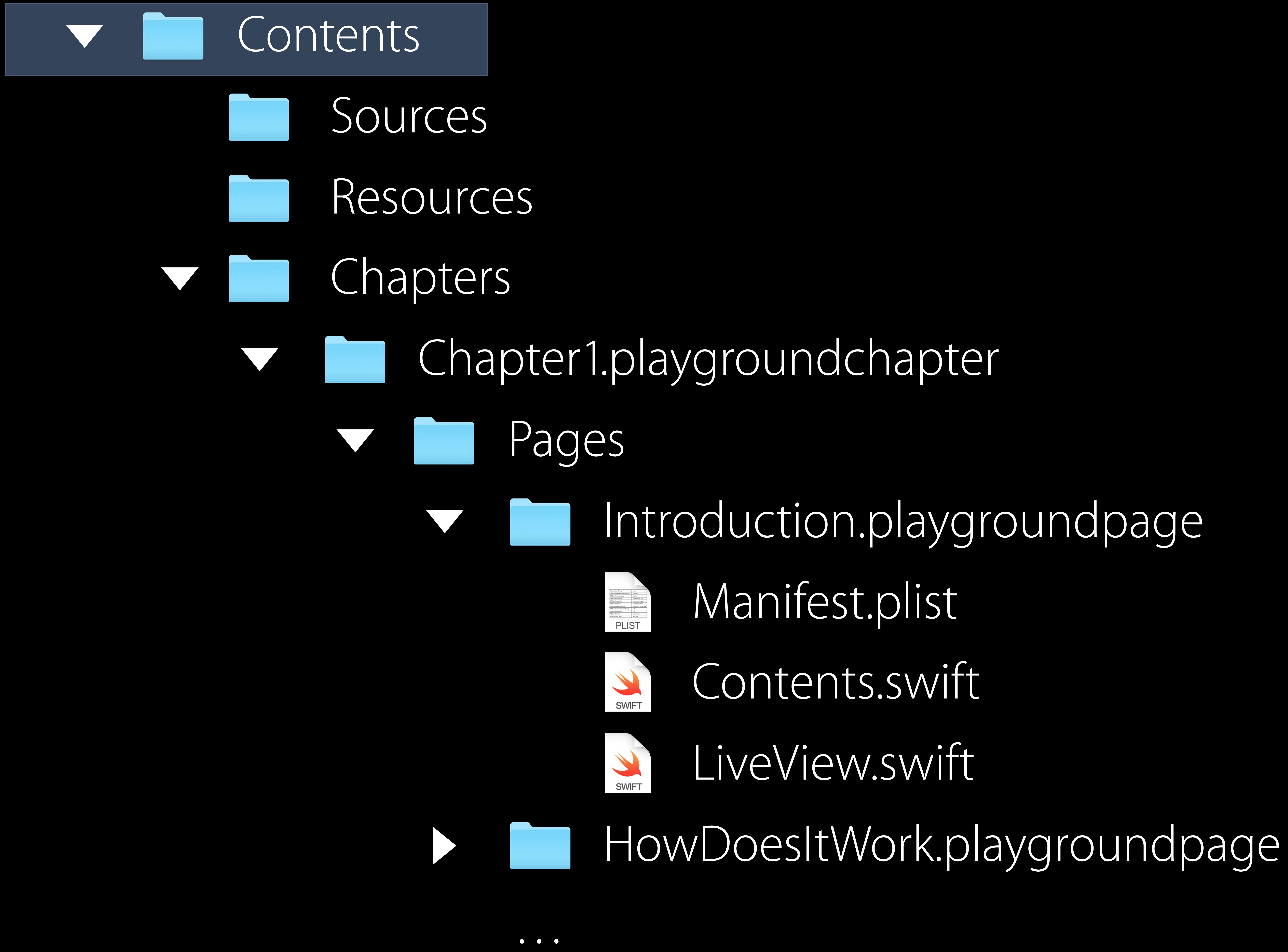
BOOK

MyFirst.playgroundbook

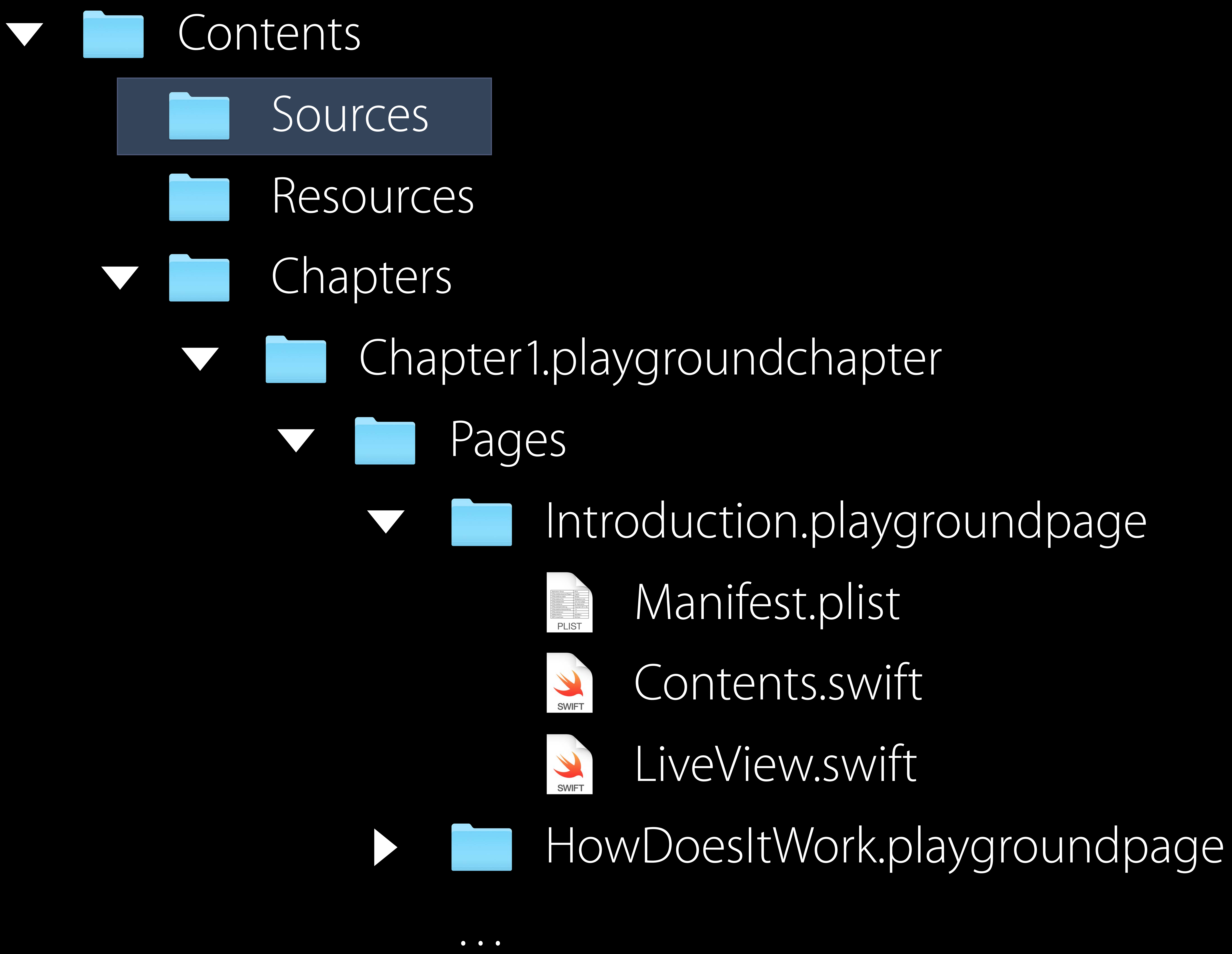

PLIST
SWIFT



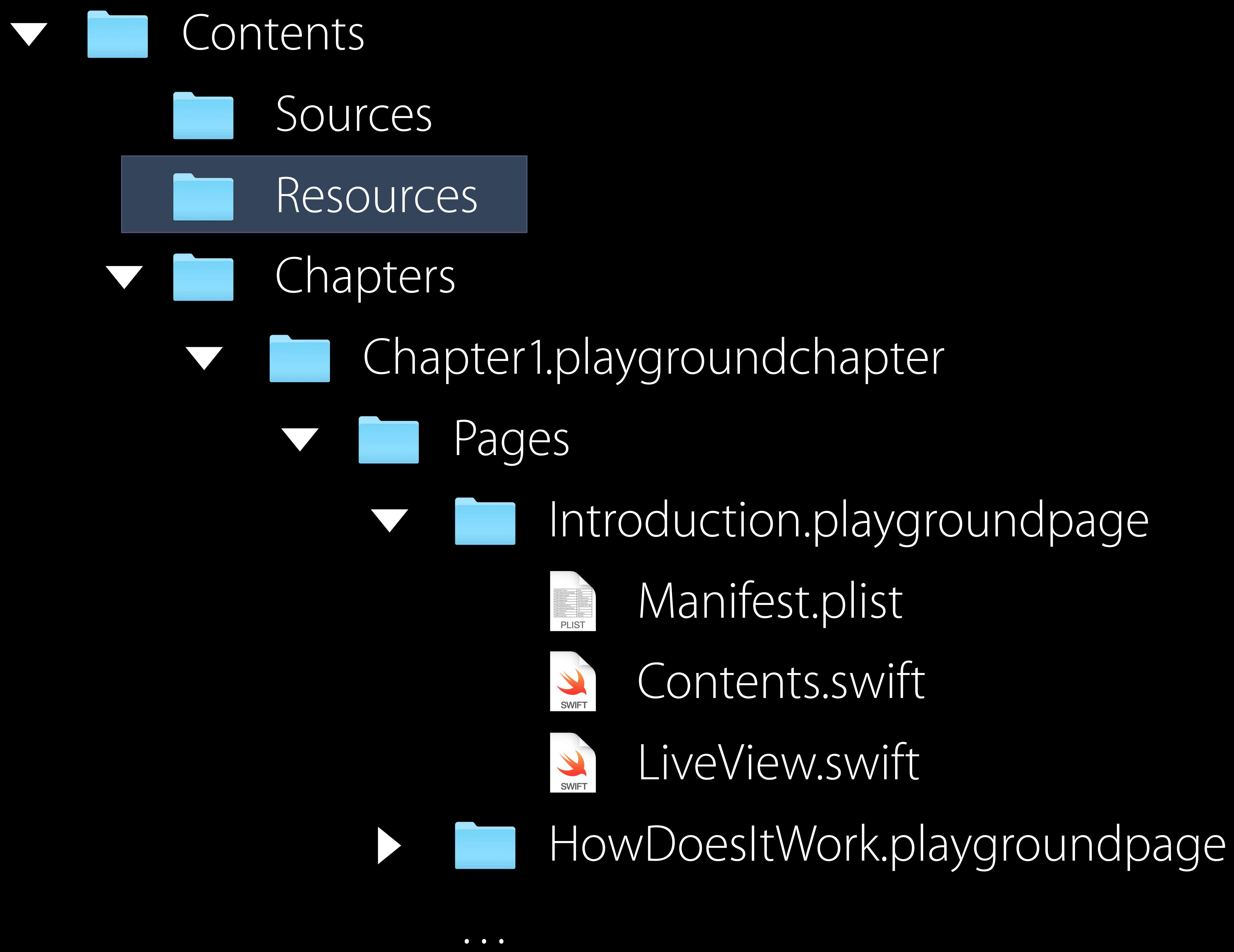
```
▼ Contents
      Sources
      Resources
  ▼ Chapters
    ▼ Chapter1.playgroundchapter
      ▼ Pages
        ▼ Introduction.playgroundpage
              Manifest.plist
              Contents.swift
              LiveView.swift
        ▶ HowDoesItWork.playgroundpage
          . . .
```



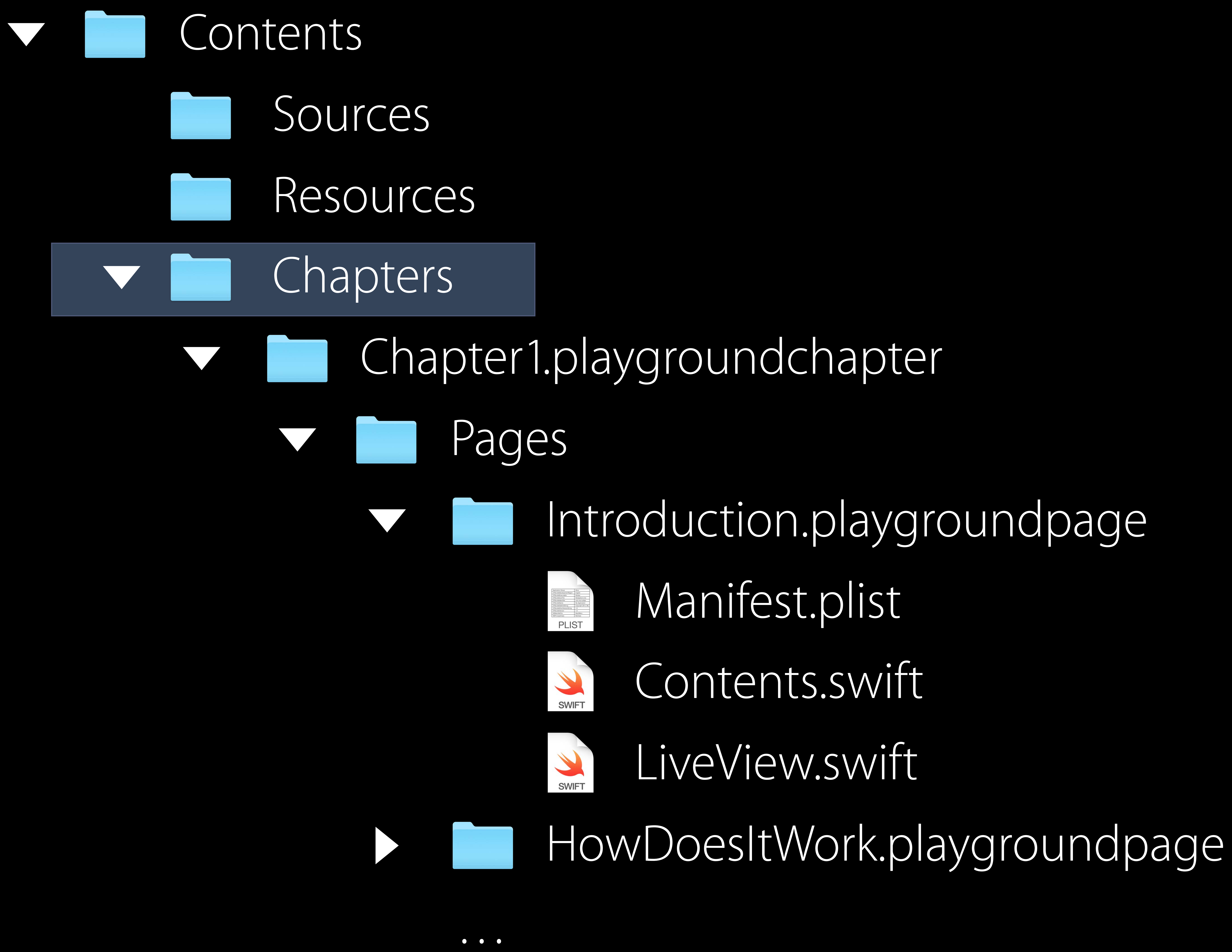
```
▼ Contents
      Sources
      Resources
    ▼ Chapters
        ▼ Chapter1.playgroundchapter
            ▼ Pages
                ▼ Introduction.playgroundpage
                      Manifest.plist
                      Contents.swift
                      LiveView.swift
                ▶ HowDoesItWork.playgroundpage
                  . . .
```



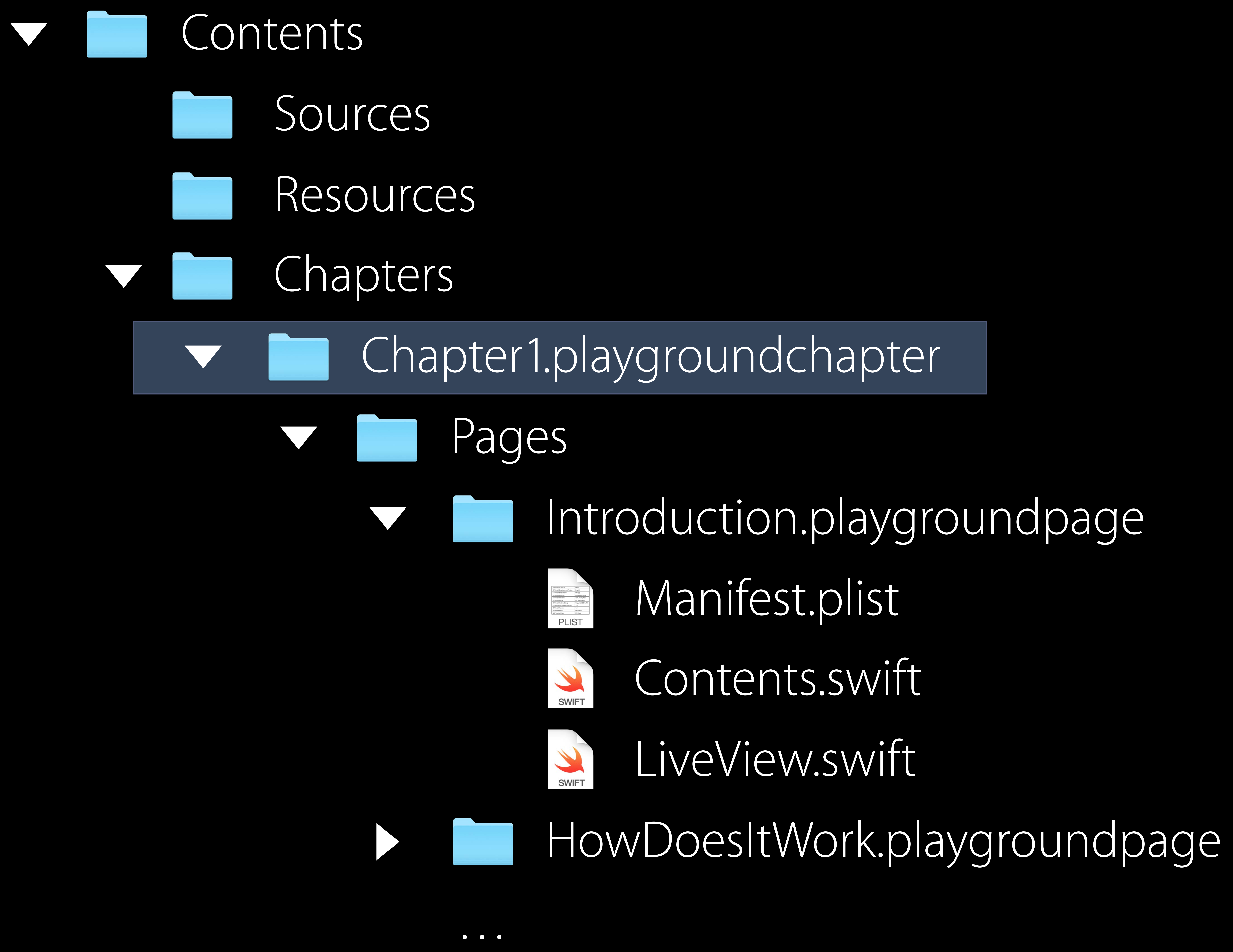
```
▼ Contents
      Sources
      Resources
  ▼ Chapters
    ▼ Chapter1.playgroundchapter
      ▼ Pages
        ▼ Introduction.playgroundpage
              Manifest.plist
              Contents.swift
              LiveView.swift
        ▶ HowDoesItWork.playgroundpage
          ...
```



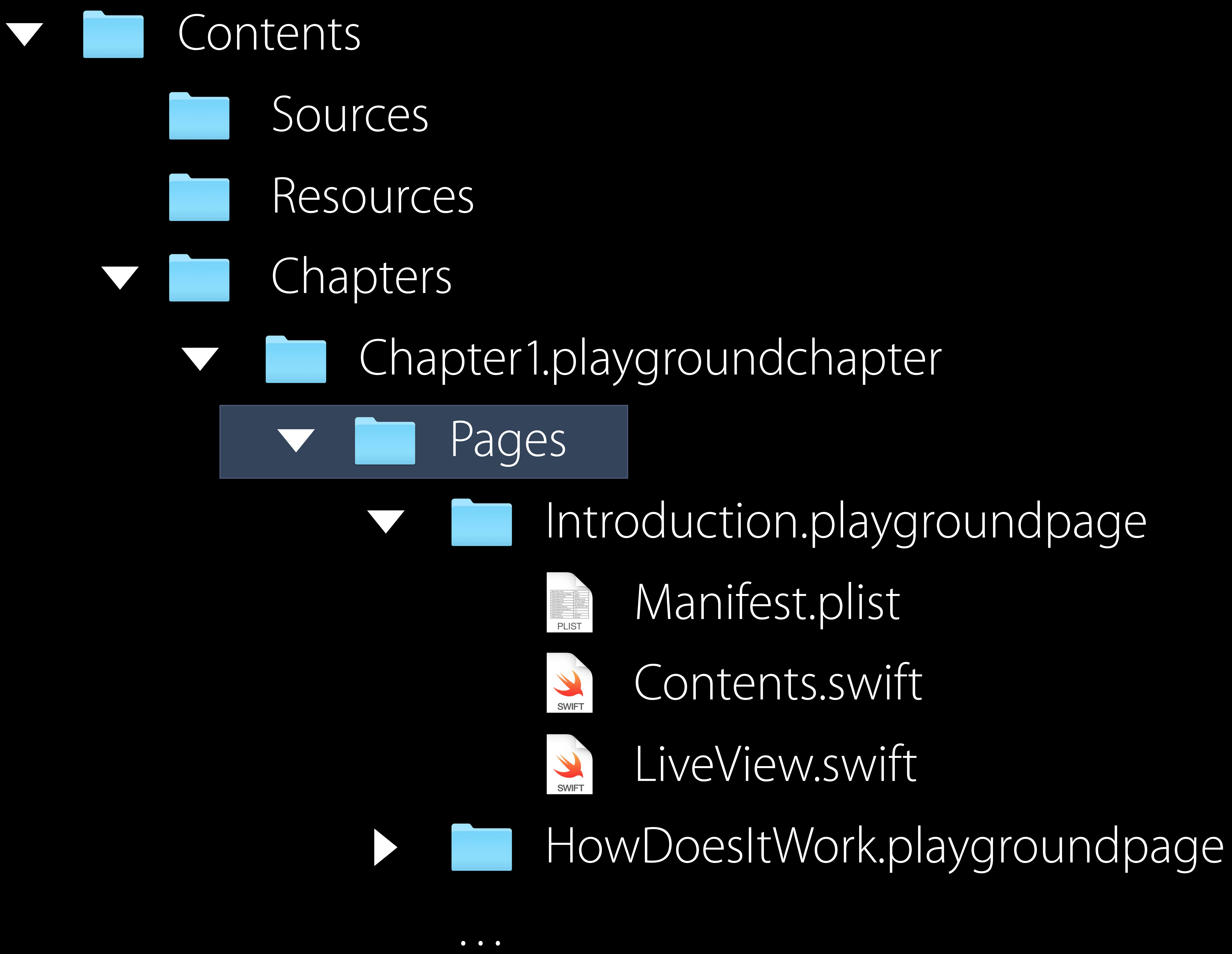
```
▼ Contents
      Sources
      Resources
  ▼ Chapters
    ▼ Chapter1.playgroundchapter
      ▼ Pages
        ▼ Introduction.playgroundpage
              Manifest.plist
              Contents.swift
              LiveView.swift
        ▶ HowDoesItWork.playgroundpage
          . . .
```



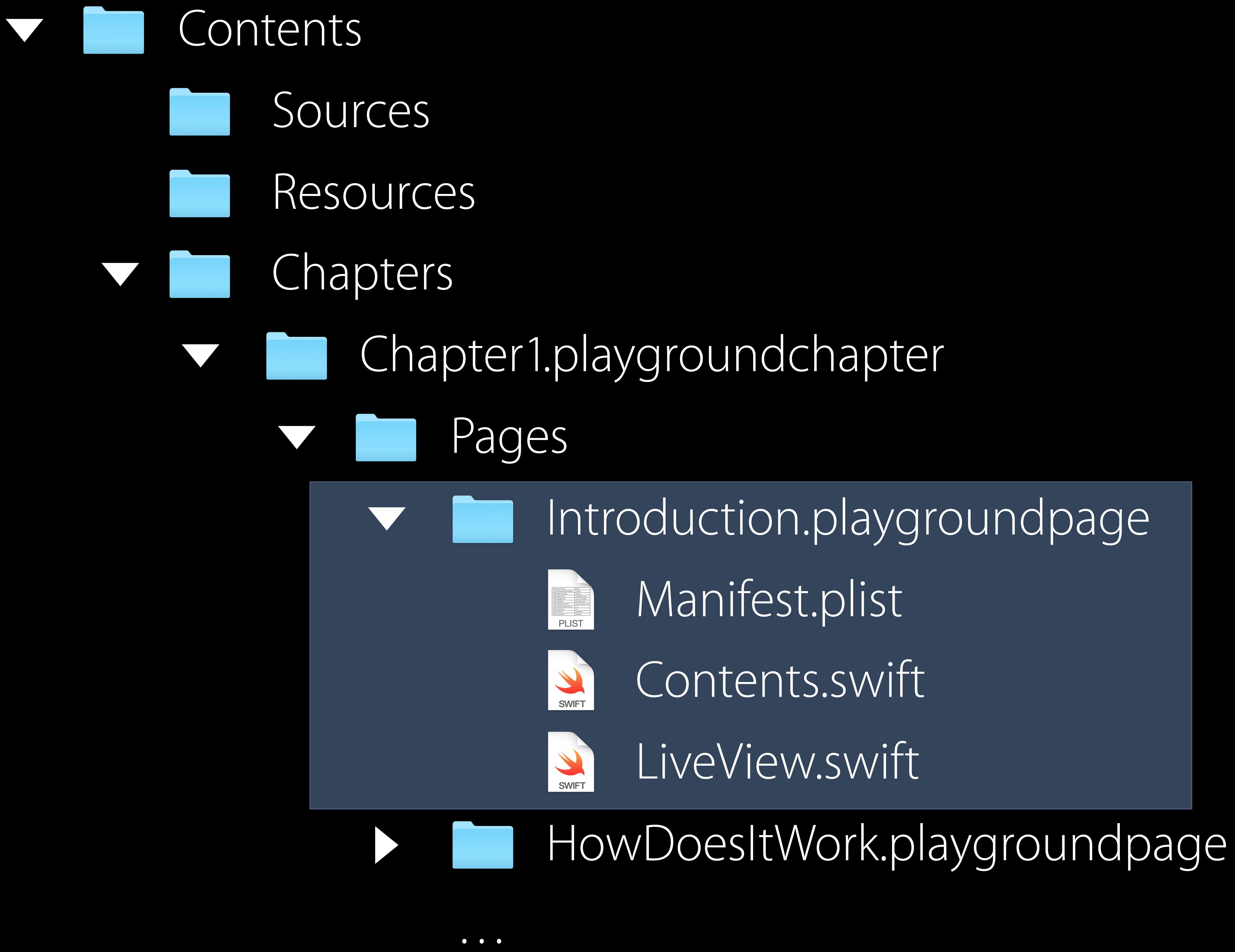
```
▼ Contents
      Sources
      Resources
  ▼ Chapters
      ▼ Chapter1.playgroundchapter
          ▼ Pages
              ▼ Introduction.playgroundpage
                    Manifest.plist
                    Contents.swift
                    LiveView.swift
              ▶ HowDoesItWork.playgroundpage
                ...
```



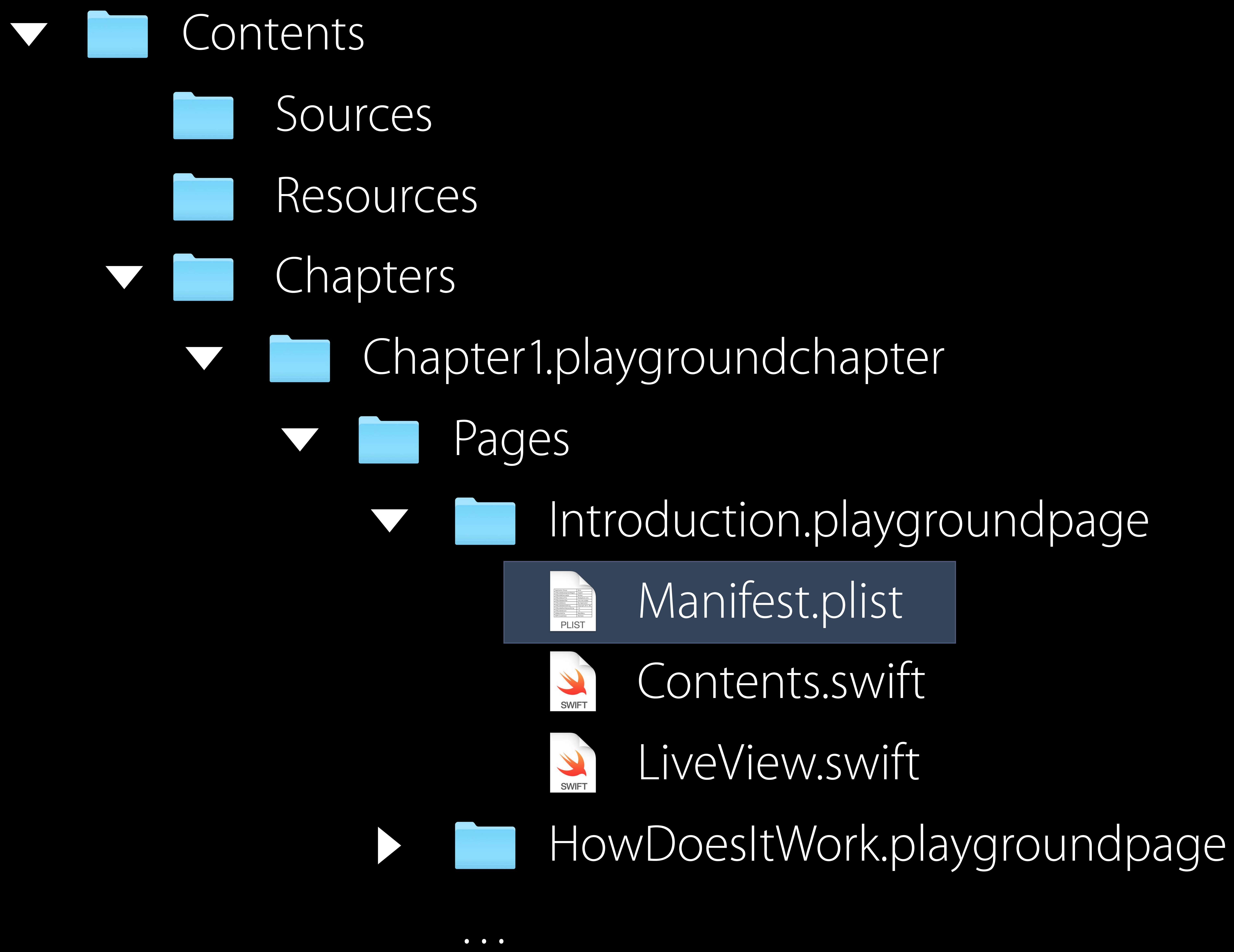
```
▼ Contents
      Sources
      Resources
  ▼ Chapters
    ▼ Chapter1.playgroundchapter
        ▼ Pages
            ▼ Introduction.playgroundpage
                  Manifest.plist
                  Contents.swift
                  LiveView.swift
            ▶ HowDoesItWork.playgroundpage
              . . .
```



```
▼ Contents
    Sources
    Resources
  ▼ Chapters
    ▼ Chapter1.playgroundchapter
      ▼ Pages
        ▼ Introduction.playgroundpage
            Manifest.plist
            Contents.swift
            LiveView.swift
        ▶ HowDoesItWork.playgroundpage
          . . .
```



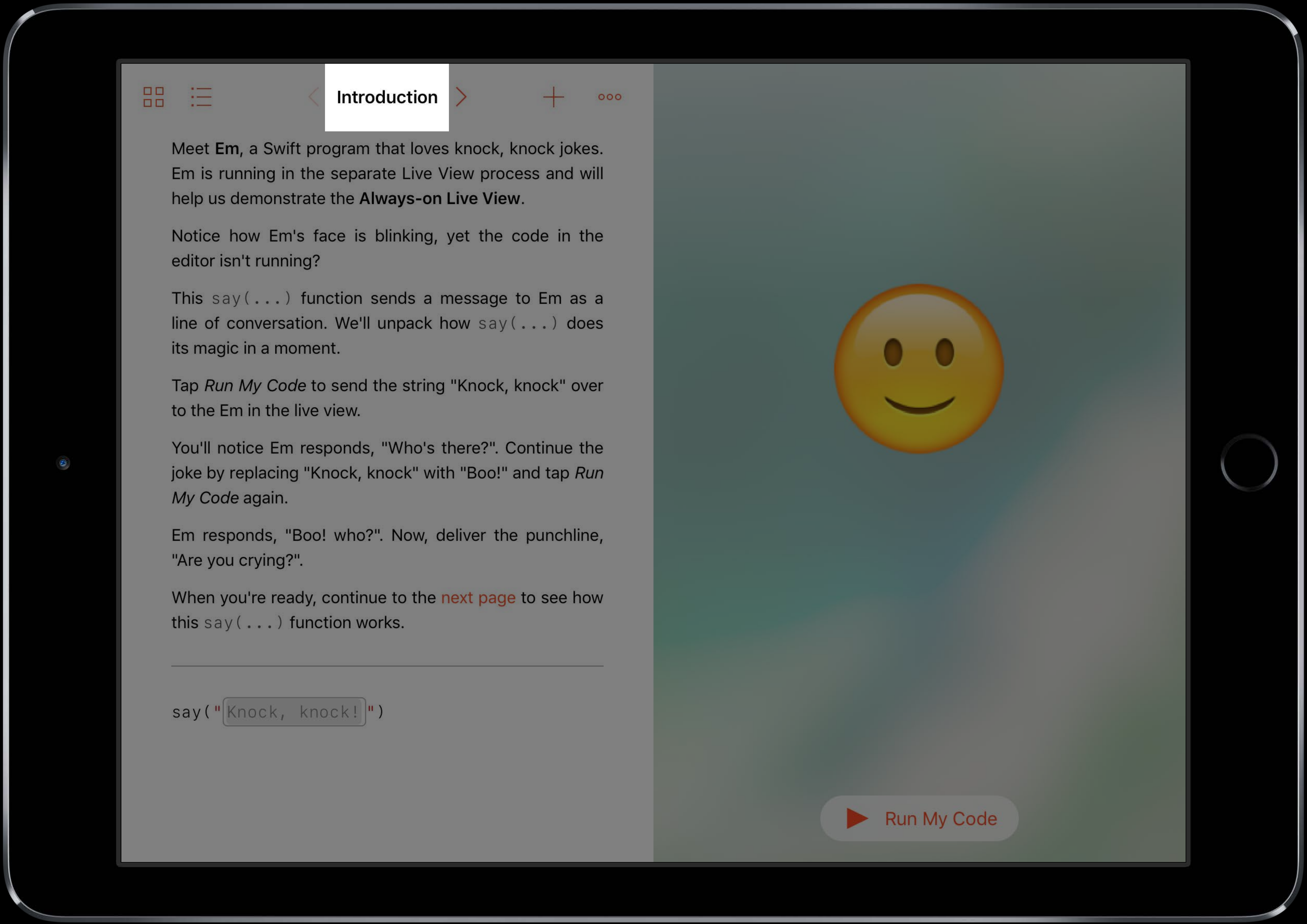
```
▼ Contents
    Sources
    Resources
  ▼ Chapters
    ▼ Chapter1.playgroundchapter
      ▼ Pages
        ▼ Introduction.playgroundpage
            Manifest.plist
            Contents.swift
            LiveView.swift
        ▶ HowDoesItWork.playgroundpage
          ...
```



```
▼ Contents
    Sources
    Resources
  ▼ Chapters
    ▼ Chapter1.playgroundchapter
      ▼ Pages
        ▼ Introduction.playgroundpage
            Manifest.plist
            Contents.swift
            LiveView.swift
        ▶ HowDoesItWork.playgroundpage
        ...
```

Page Manifest

Key	Type	Value
▼ Root	Dictionary	
Name	String	Introduction
LiveViewMode	String	VisibleByDefault
PosterReference	String	LiveViewPoster.png
LiveViewEdgeToEdge	Boolean	YES
PlaygroundLoggingMode	String	Off



Meet **Em**, a Swift program that loves knock, knock jokes. Em is running in the separate Live View process and will help us demonstrate the **Always-on Live View**.

Notice how Em's face is blinking, yet the code in the editor isn't running?

This `say(...)` function sends a message to Em as a line of conversation. We'll unpack how `say(...)` does its magic in a moment.

Tap *Run My Code* to send the string "Knock, knock" over to the Em in the live view.

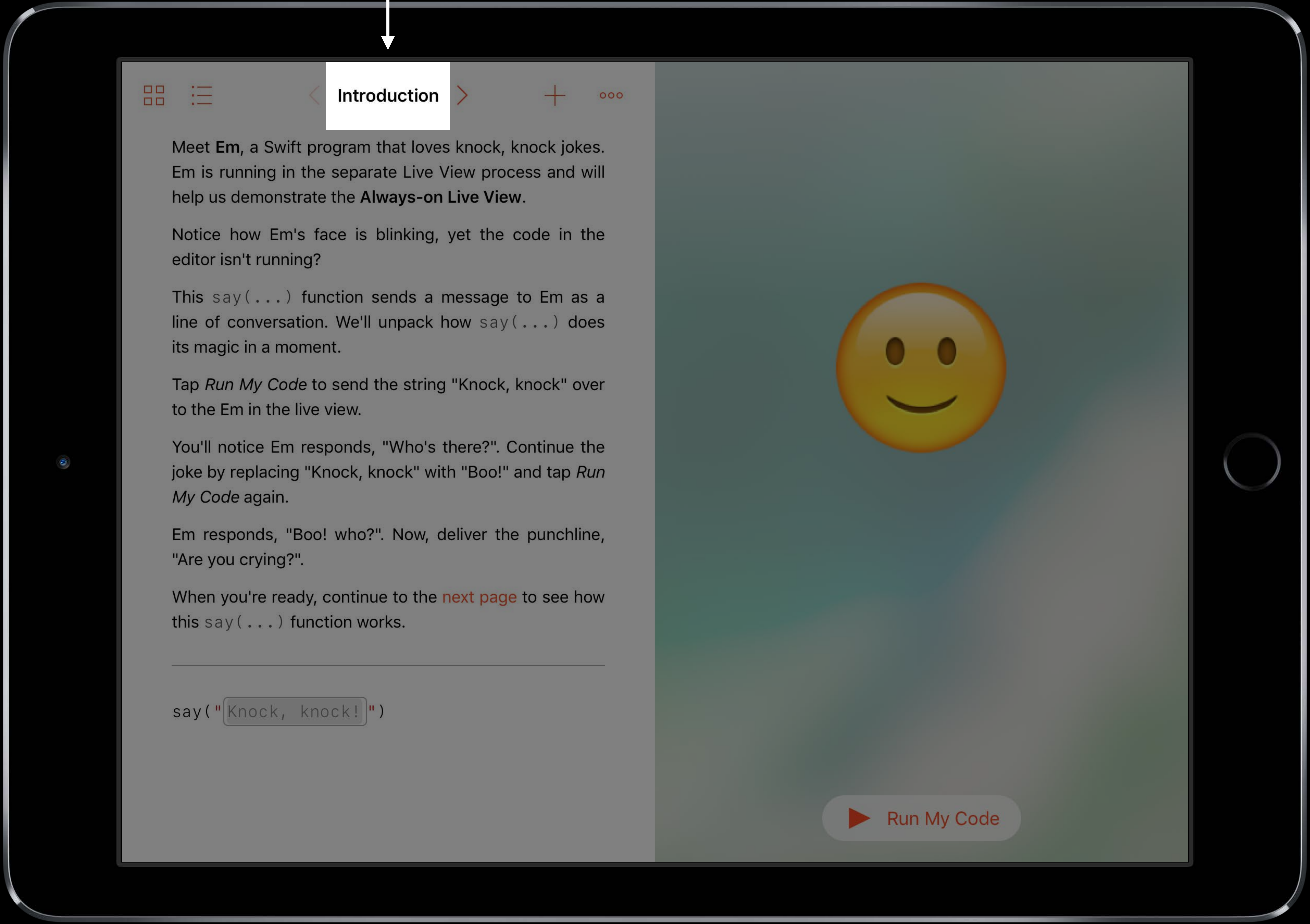
You'll notice Em responds, "Who's there?". Continue the joke by replacing "Knock, knock" with "Boo!" and tap *Run My Code* again.

Em responds, "Boo! who?". Now, deliver the punchline, "Are you crying?".

When you're ready, continue to the next page to see how this `say(...)` function works.

```
say("Knock, knock!")
```

Name = "Introduction"



Meet **Em**, a Swift program that loves knock, knock jokes. Em is running in the separate Live View process and will help us demonstrate the **Always-on Live View**.

Notice how Em's face is blinking, yet the code in the editor isn't running?

This `say(...)` function sends a message to Em as a line of conversation. We'll unpack how `say(...)` does its magic in a moment.

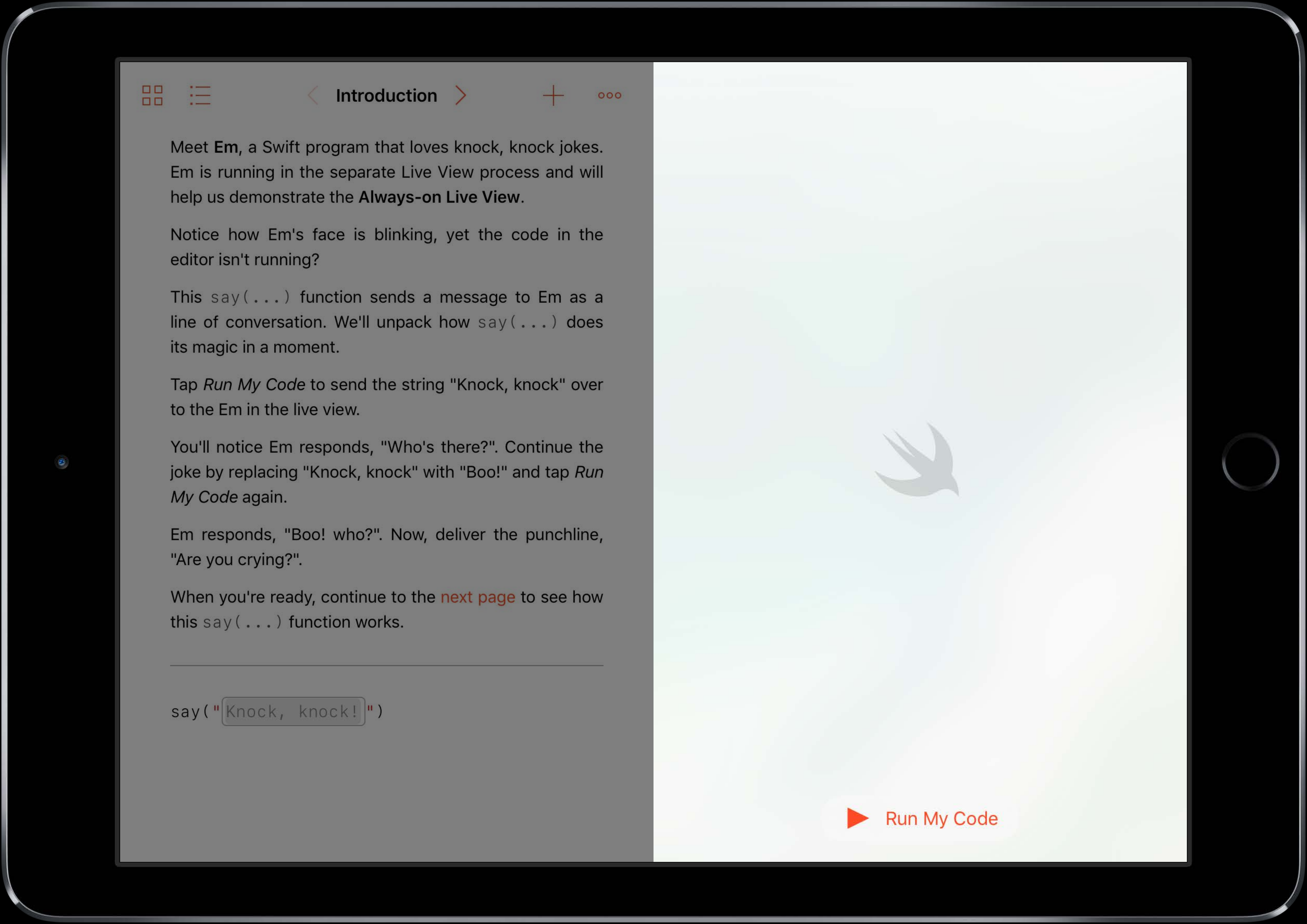
Tap *Run My Code* to send the string "Knock, knock" over to the Em in the live view.

You'll notice Em responds, "Who's there?". Continue the joke by replacing "Knock, knock" with "Boo!" and tap *Run My Code* again.

Em responds, "Boo! who?". Now, deliver the punchline, "Are you crying?".

When you're ready, continue to the next page to see how this `say(...)` function works.

```
say("Knock, knock!")
```



Meet **Em**, a Swift program that loves knock, knock jokes. Em is running in the separate Live View process and will help us demonstrate the **Always-on Live View**.

Notice how Em's face is blinking, yet the code in the editor isn't running?

This `say(...)` function sends a message to Em as a line of conversation. We'll unpack how `say(...)` does its magic in a moment.

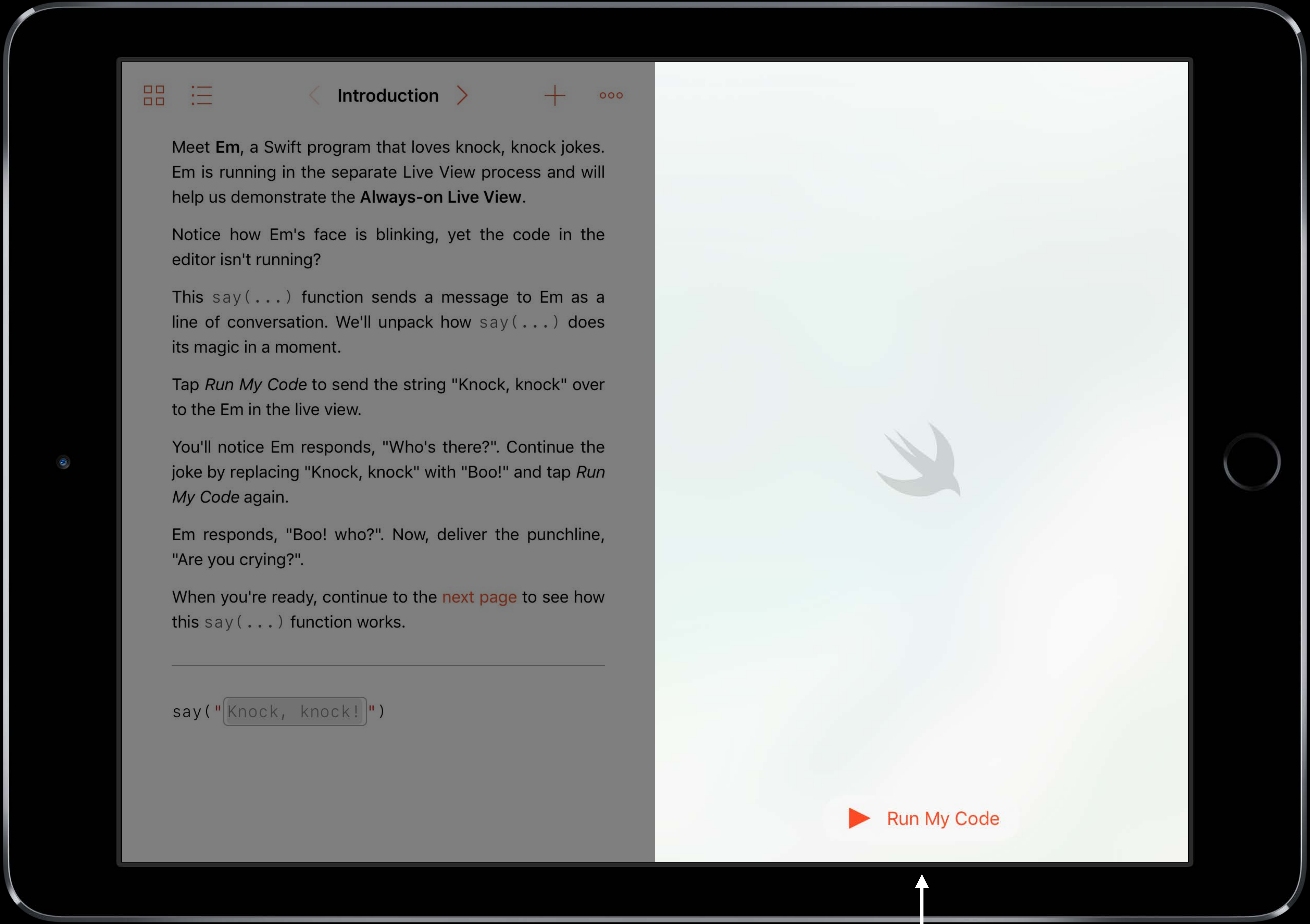
Tap *Run My Code* to send the string "Knock, knock" over to the Em in the live view.

You'll notice Em responds, "Who's there?". Continue the joke by replacing "Knock, knock" with "Boo!" and tap *Run My Code* again.

Em responds, "Boo! who?". Now, deliver the punchline, "Are you crying?".

When you're ready, continue to the next page to see how this `say(...)` function works.

```
say("Knock, knock!")
```



Meet **Em**, a Swift program that loves knock, knock jokes. Em is running in the separate Live View process and will help us demonstrate the **Always-on Live View**.

Notice how Em's face is blinking, yet the code in the editor isn't running?

This `say(...)` function sends a message to Em as a line of conversation. We'll unpack how `say(...)` does its magic in a moment.

Tap *Run My Code* to send the string "Knock, knock" over to the Em in the live view.

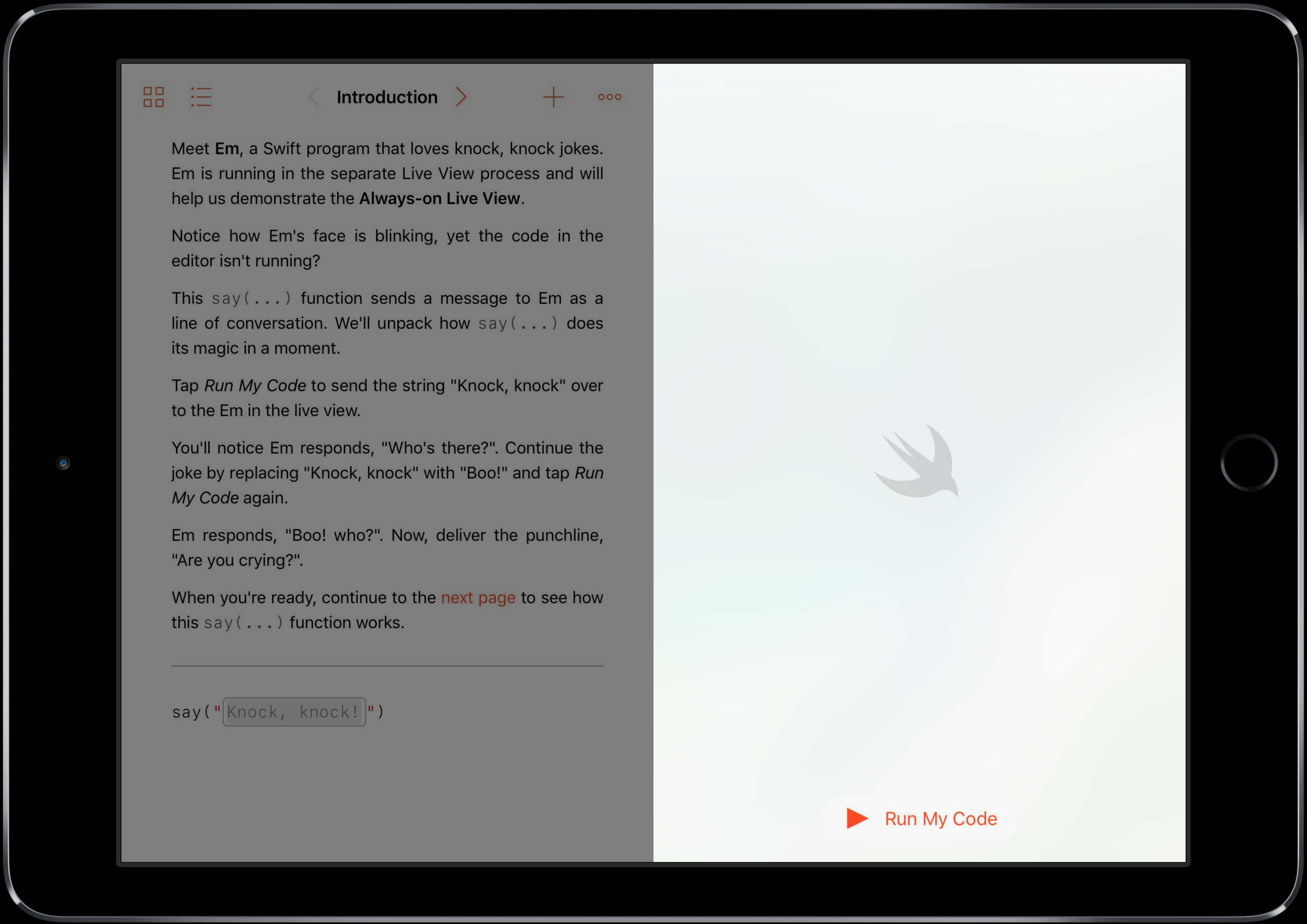
You'll notice Em responds, "Who's there?". Continue the joke by replacing "Knock, knock" with "Boo!" and tap *Run My Code* again.

Em responds, "Boo! who?". Now, deliver the punchline, "Are you crying?".

When you're ready, continue to the next page to see how this `say(...)` function works.

```
say("Knock, knock!")
```

LiveViewMode = "VisibleByDefault"



Meet **Em**, a Swift program that loves knock, knock jokes. Em is running in the separate Live View process and will help us demonstrate the **Always-on Live View**.

Notice how Em's face is blinking, yet the code in the editor isn't running?

This `say(...)` function sends a message to Em as a line of conversation. We'll unpack how `say(...)` does its magic in a moment.

Tap *Run My Code* to send the string "Knock, knock" over to the Em in the live view.

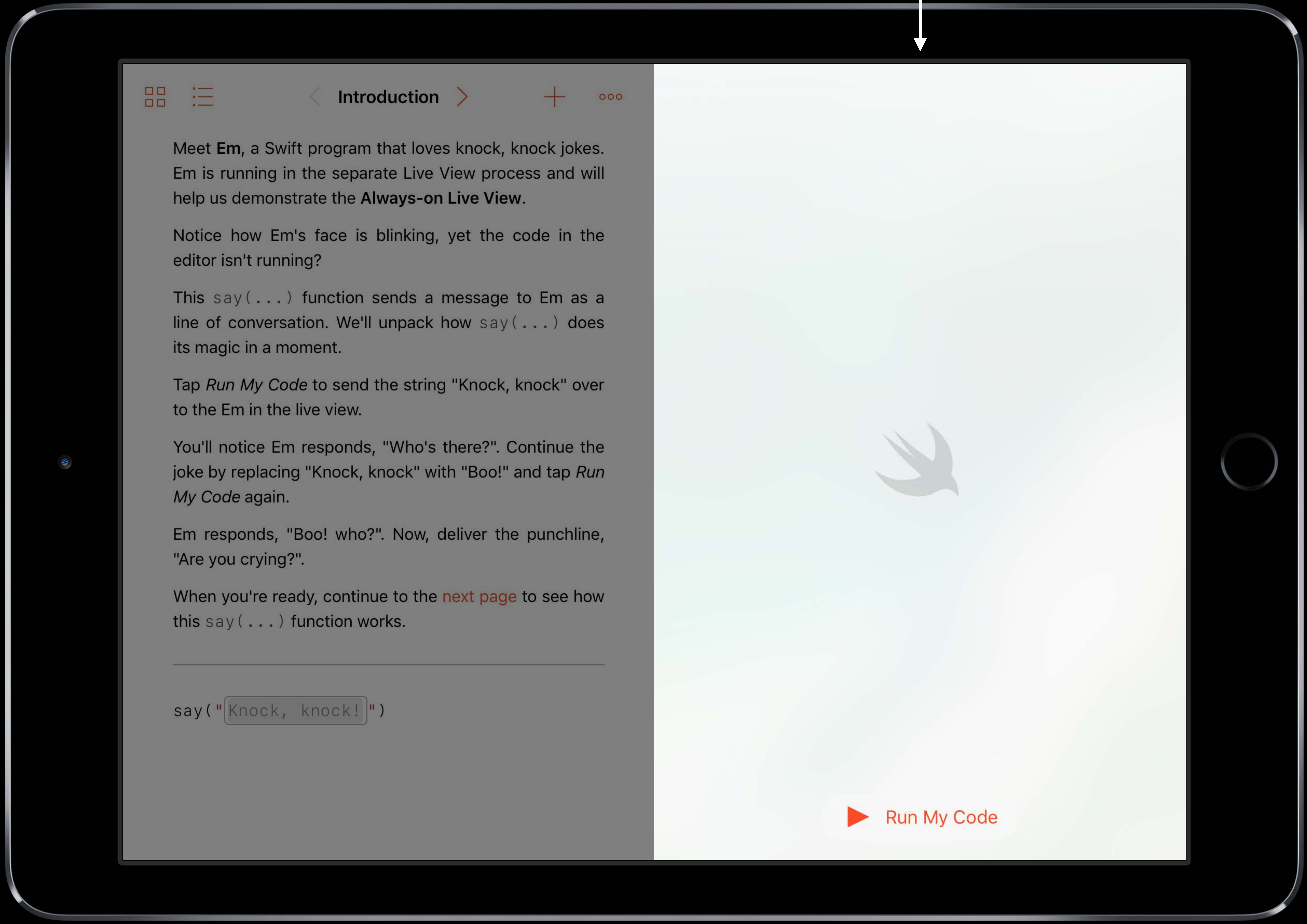
You'll notice Em responds, "Who's there?". Continue the joke by replacing "Knock, knock" with "Boo!" and tap *Run My Code* again.

Em responds, "Boo! who?". Now, deliver the punchline, "Are you crying?".

When you're ready, continue to the next page to see how this `say(...)` function works.

```
say("Knock, knock!")
```

PosterReference = “LiveViewPoster.png”



Meet **Em**, a Swift program that loves knock, knock jokes. Em is running in the separate Live View process and will help us demonstrate the **Always-on Live View**.

Notice how Em's face is blinking, yet the code in the editor isn't running?

This `say(...)` function sends a message to Em as a line of conversation. We'll unpack how `say(...)` does its magic in a moment.

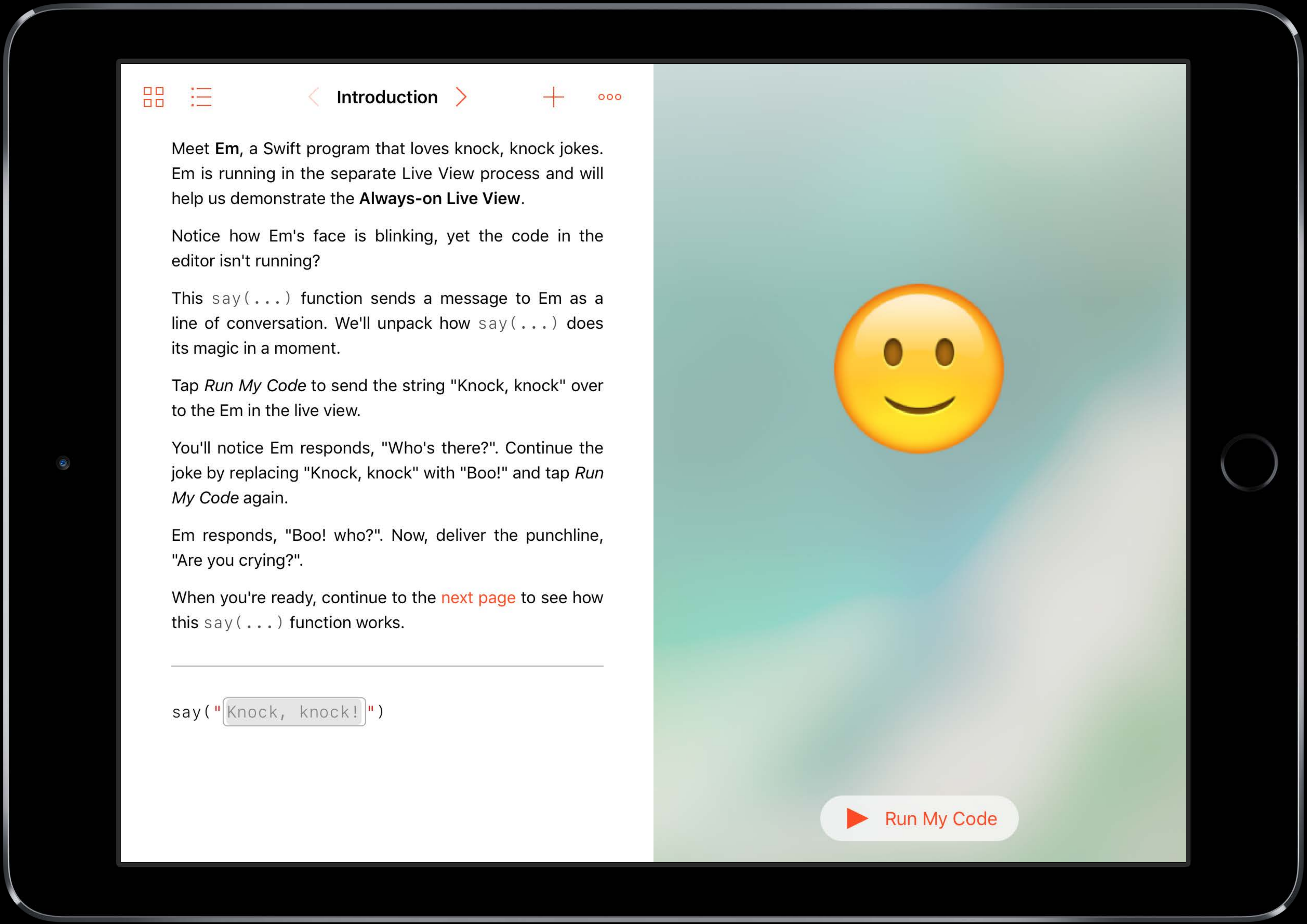
Tap *Run My Code* to send the string "Knock, knock" over to the Em in the live view.

You'll notice Em responds, "Who's there?". Continue the joke by replacing "Knock, knock" with "Boo!" and tap *Run My Code* again.

Em responds, "Boo! who?". Now, deliver the punchline, "Are you crying?".

When you're ready, continue to the next page to see how this `say(...)` function works.

```
say("Knock, knock!")
```



Meet **Em**, a Swift program that loves knock, knock jokes. Em is running in the separate Live View process and will help us demonstrate the **Always-on Live View**.

Notice how Em's face is blinking, yet the code in the editor isn't running?

This `say(...)` function sends a message to Em as a line of conversation. We'll unpack how `say(...)` does its magic in a moment.

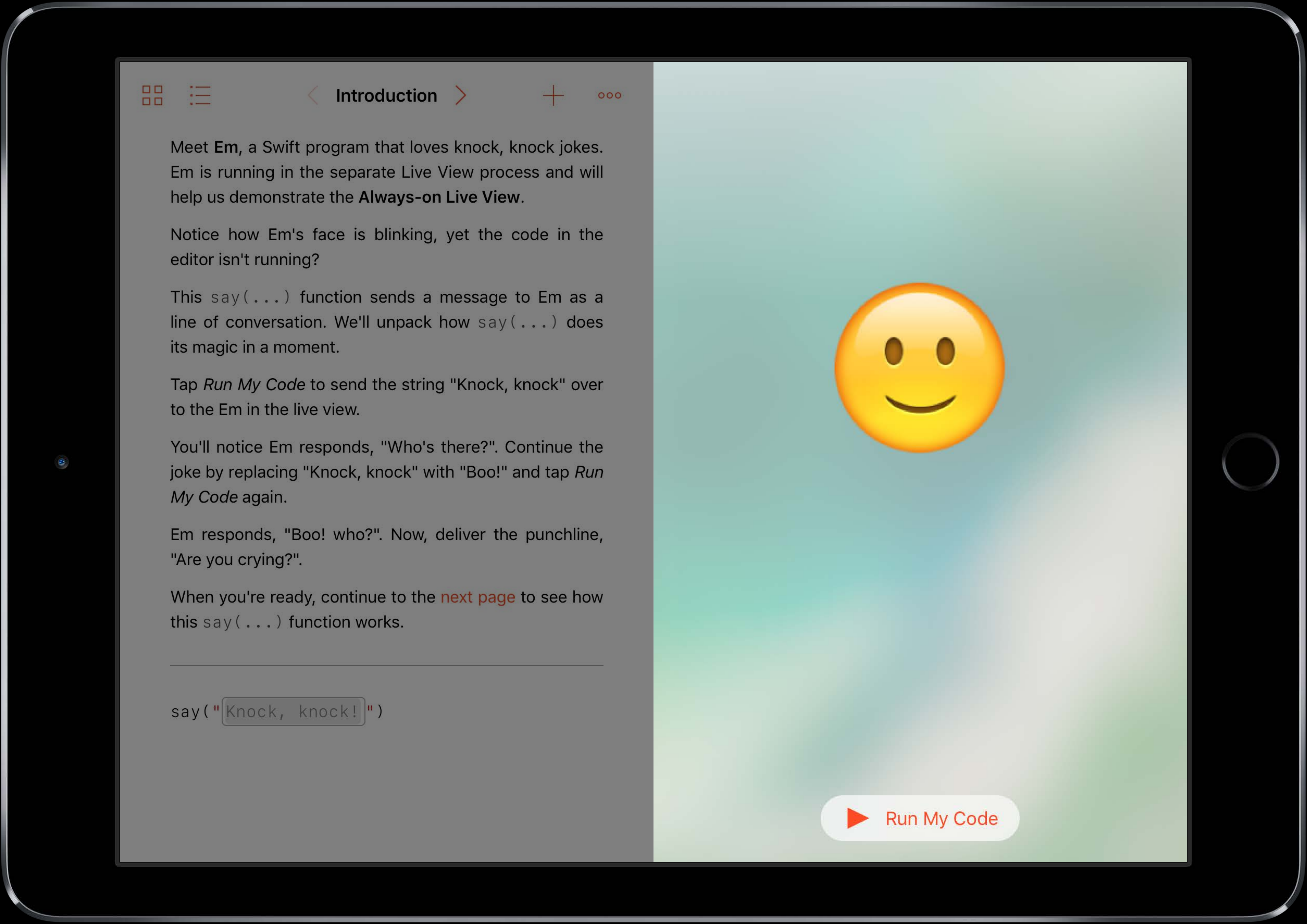
Tap *Run My Code* to send the string "Knock, knock" over to the Em in the live view.

You'll notice Em responds, "Who's there?". Continue the joke by replacing "Knock, knock" with "Boo!" and tap *Run My Code* again.

Em responds, "Boo! who?". Now, deliver the punchline, "Are you crying?".

When you're ready, continue to the next page to see how this `say(...)` function works.

```
say("Knock, knock!")
```



Meet **Em**, a Swift program that loves knock, knock jokes. Em is running in the separate Live View process and will help us demonstrate the **Always-on Live View**.

Notice how Em's face is blinking, yet the code in the editor isn't running?

This `say(...)` function sends a message to Em as a line of conversation. We'll unpack how `say(...)` does its magic in a moment.

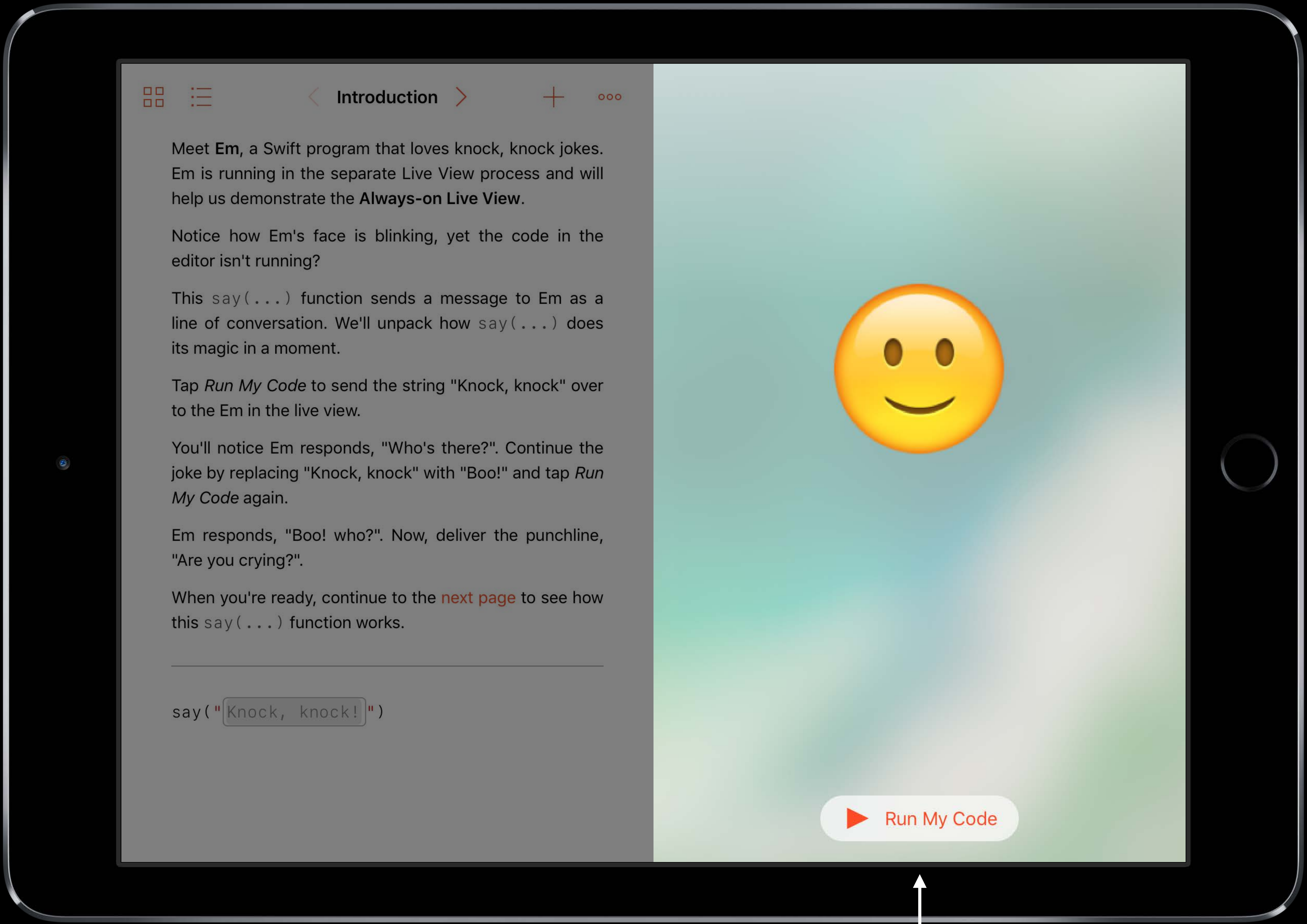
Tap *Run My Code* to send the string "Knock, knock" over to the Em in the live view.

You'll notice Em responds, "Who's there?". Continue the joke by replacing "Knock, knock" with "Boo!" and tap *Run My Code* again.

Em responds, "Boo! who?". Now, deliver the punchline, "Are you crying?".

When you're ready, continue to the next page to see how this `say(...)` function works.

```
say("Knock, knock!")
```



Meet **Em**, a Swift program that loves knock, knock jokes. Em is running in the separate Live View process and will help us demonstrate the **Always-on Live View**.

Notice how Em's face is blinking, yet the code in the editor isn't running?

This `say(...)` function sends a message to Em as a line of conversation. We'll unpack how `say(...)` does its magic in a moment.

Tap *Run My Code* to send the string "Knock, knock" over to the Em in the live view.

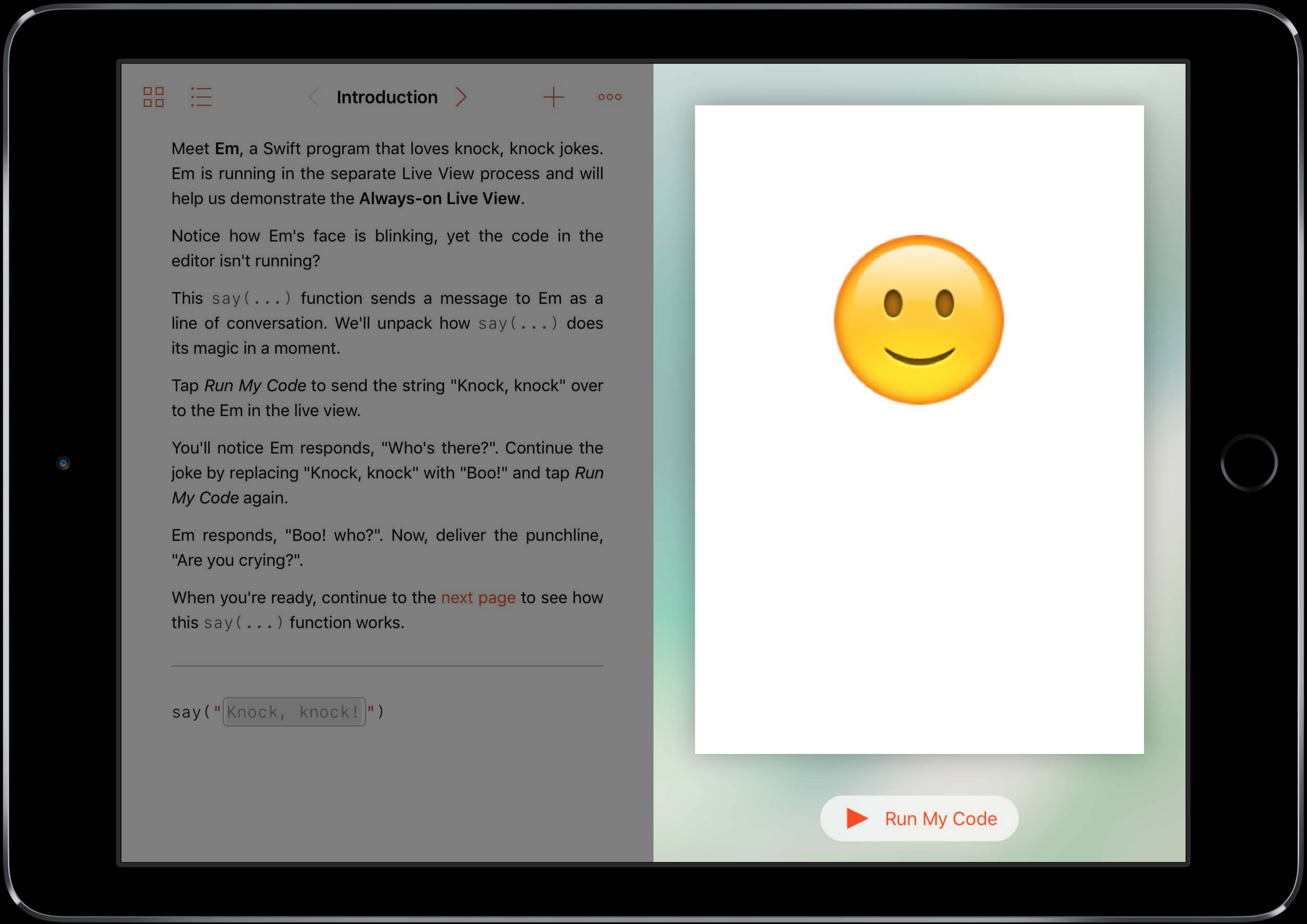
You'll notice Em responds, "Who's there?". Continue the joke by replacing "Knock, knock" with "Boo!" and tap *Run My Code* again.

Em responds, "Boo! who?". Now, deliver the punchline, "Are you crying?".

When you're ready, continue to the next page to see how this `say(...)` function works.

```
say("Knock, knock!")
```

LiveViewEdgeToEdge = YES



Meet **Em**, a Swift program that loves knock, knock jokes. Em is running in the separate Live View process and will help us demonstrate the **Always-on Live View**.

Notice how Em's face is blinking, yet the code in the editor isn't running?

This `say(...)` function sends a message to Em as a line of conversation. We'll unpack how `say(...)` does its magic in a moment.

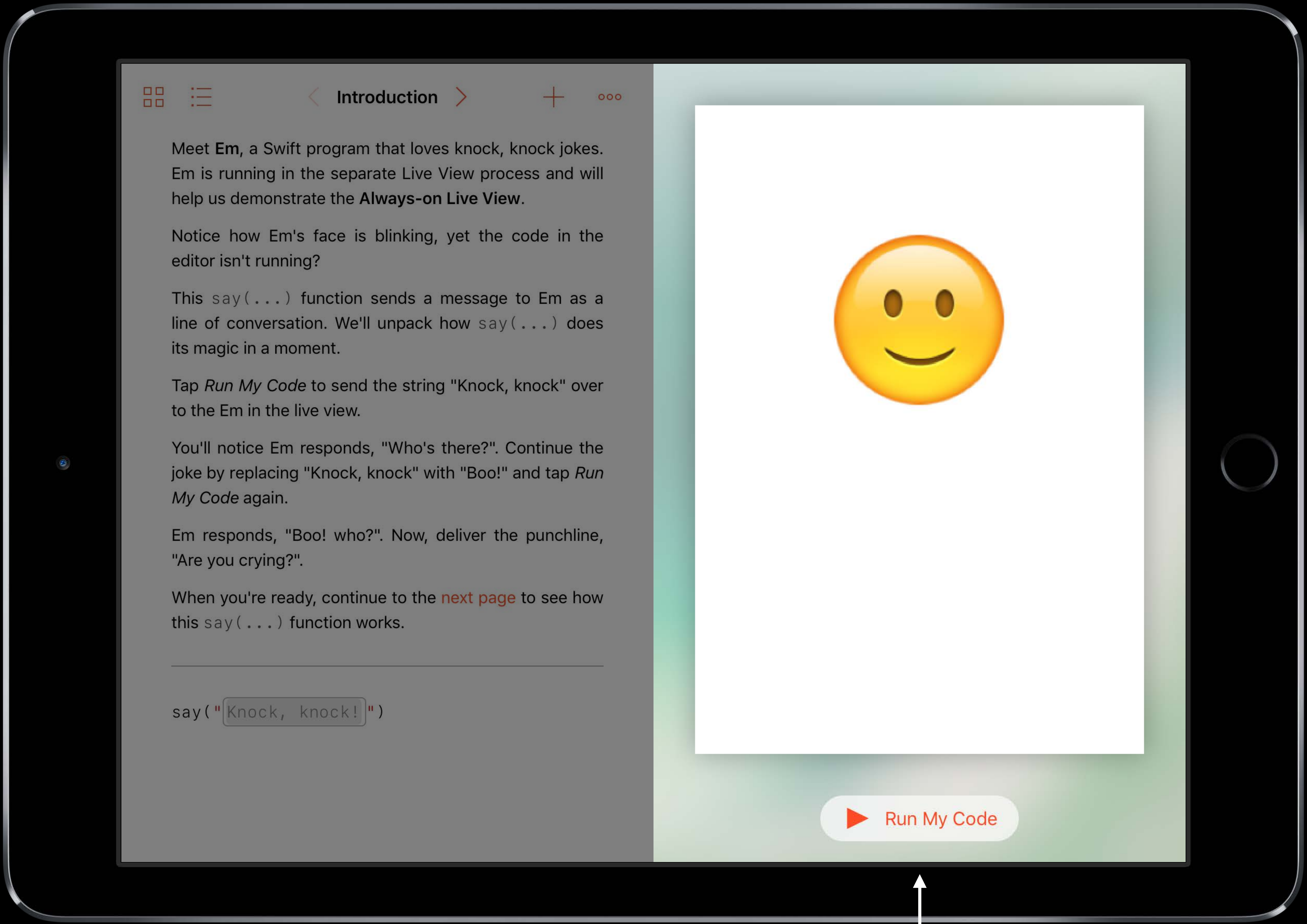
Tap *Run My Code* to send the string "Knock, knock" over to the Em in the live view.

You'll notice Em responds, "Who's there?". Continue the joke by replacing "Knock, knock" with "Boo!" and tap *Run My Code* again.

Em responds, "Boo! who?". Now, deliver the punchline, "Are you crying?".

When you're ready, continue to the next page to see how this `say(...)` function works.

```
say("Knock, knock!")
```



Introduction

Meet **Em**, a Swift program that loves knock, knock jokes. Em is running in the separate Live View process and will help us demonstrate the **Always-on Live View**.

Notice how Em's face is blinking, yet the code in the editor isn't running?

This `say(...)` function sends a message to Em as a line of conversation. We'll unpack how `say(...)` does its magic in a moment.

Tap *Run My Code* to send the string "Knock, knock" over to the Em in the live view.

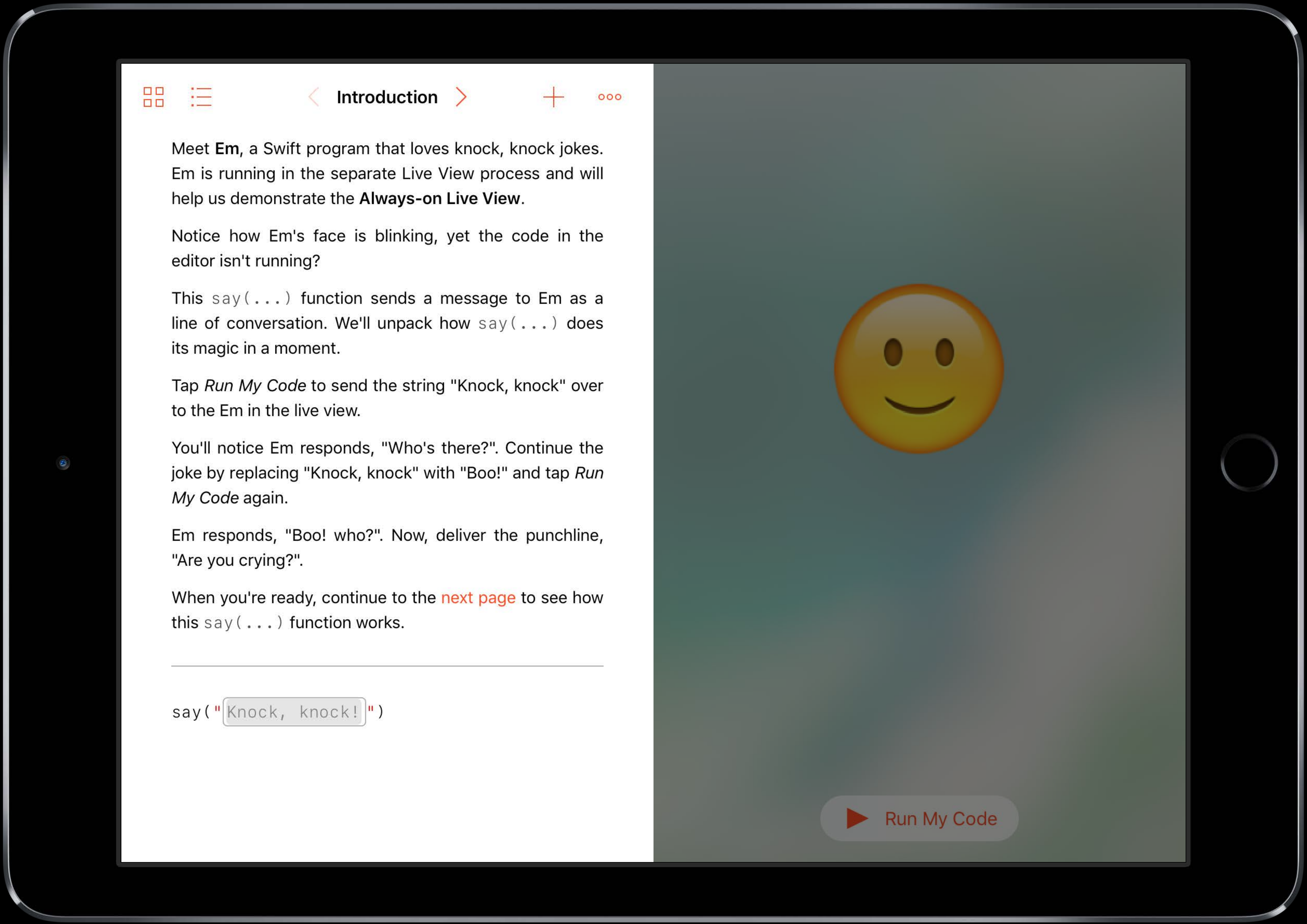
You'll notice Em responds, "Who's there?". Continue the joke by replacing "Knock, knock" with "Boo!" and tap *Run My Code* again.

Em responds, "Boo! who?". Now, deliver the punchline, "Are you crying?".

When you're ready, continue to the next page to see how this `say(...)` function works.

```
say("Knock, knock!")
```

LiveViewEdgeToEdge = NO



Meet **Em**, a Swift program that loves knock, knock jokes. Em is running in the separate Live View process and will help us demonstrate the **Always-on Live View**.

Notice how Em's face is blinking, yet the code in the editor isn't running?

This `say(...)` function sends a message to Em as a line of conversation. We'll unpack how `say(...)` does its magic in a moment.

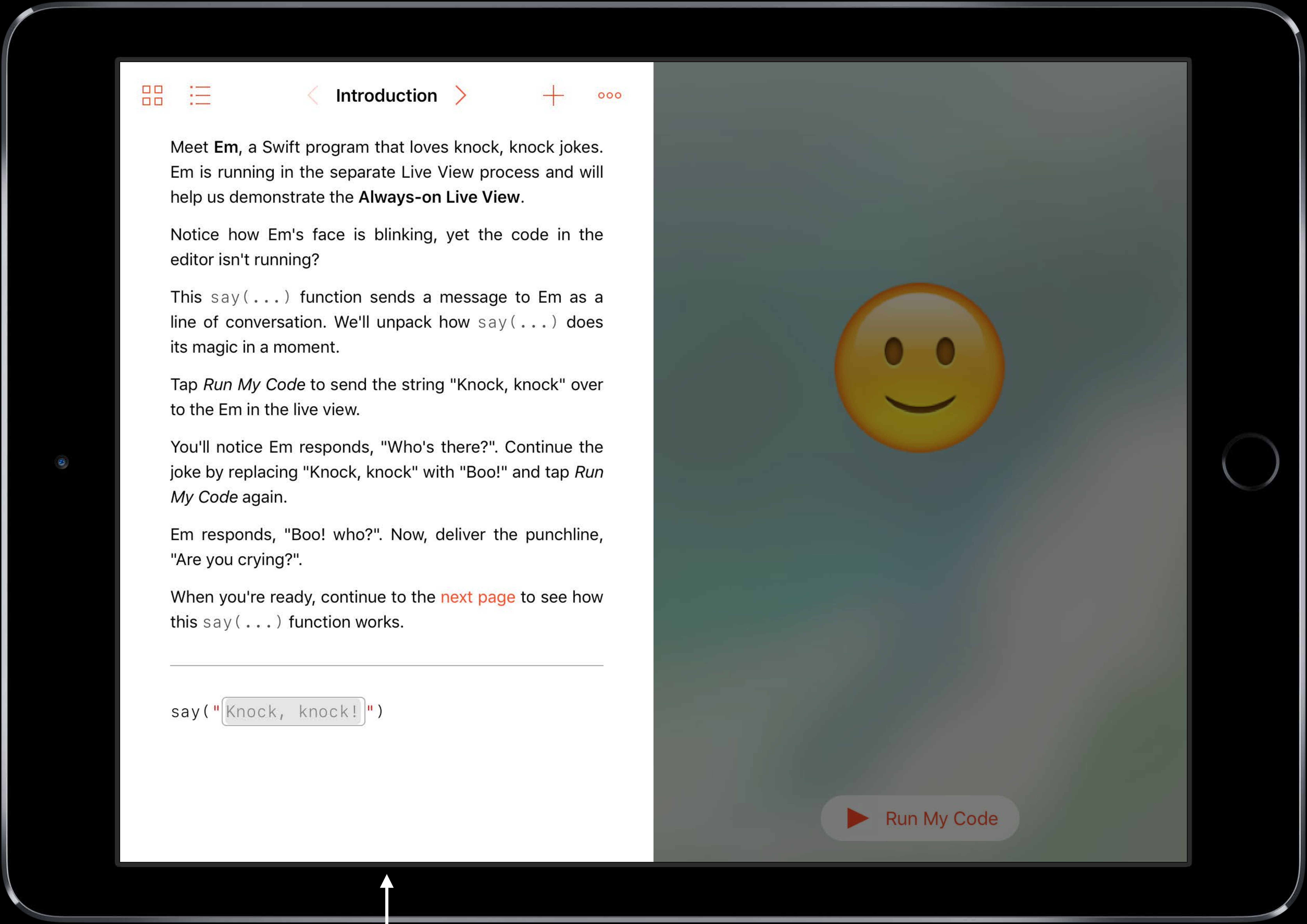
Tap *Run My Code* to send the string "Knock, knock" over to the Em in the live view.

You'll notice Em responds, "Who's there?". Continue the joke by replacing "Knock, knock" with "Boo!" and tap *Run My Code* again.

Em responds, "Boo! who?". Now, deliver the punchline, "Are you crying?".

When you're ready, continue to the next page to see how this `say(...)` function works.

```
say("Knock, knock!")
```



Meet **Em**, a Swift program that loves knock, knock jokes. Em is running in the separate Live View process and will help us demonstrate the **Always-on Live View**.

Notice how Em's face is blinking, yet the code in the editor isn't running?

This `say(...)` function sends a message to Em as a line of conversation. We'll unpack how `say(...)` does its magic in a moment.

Tap *Run My Code* to send the string "Knock, knock" over to the Em in the live view.

You'll notice Em responds, "Who's there?". Continue the joke by replacing "Knock, knock" with "Boo!" and tap *Run My Code* again.

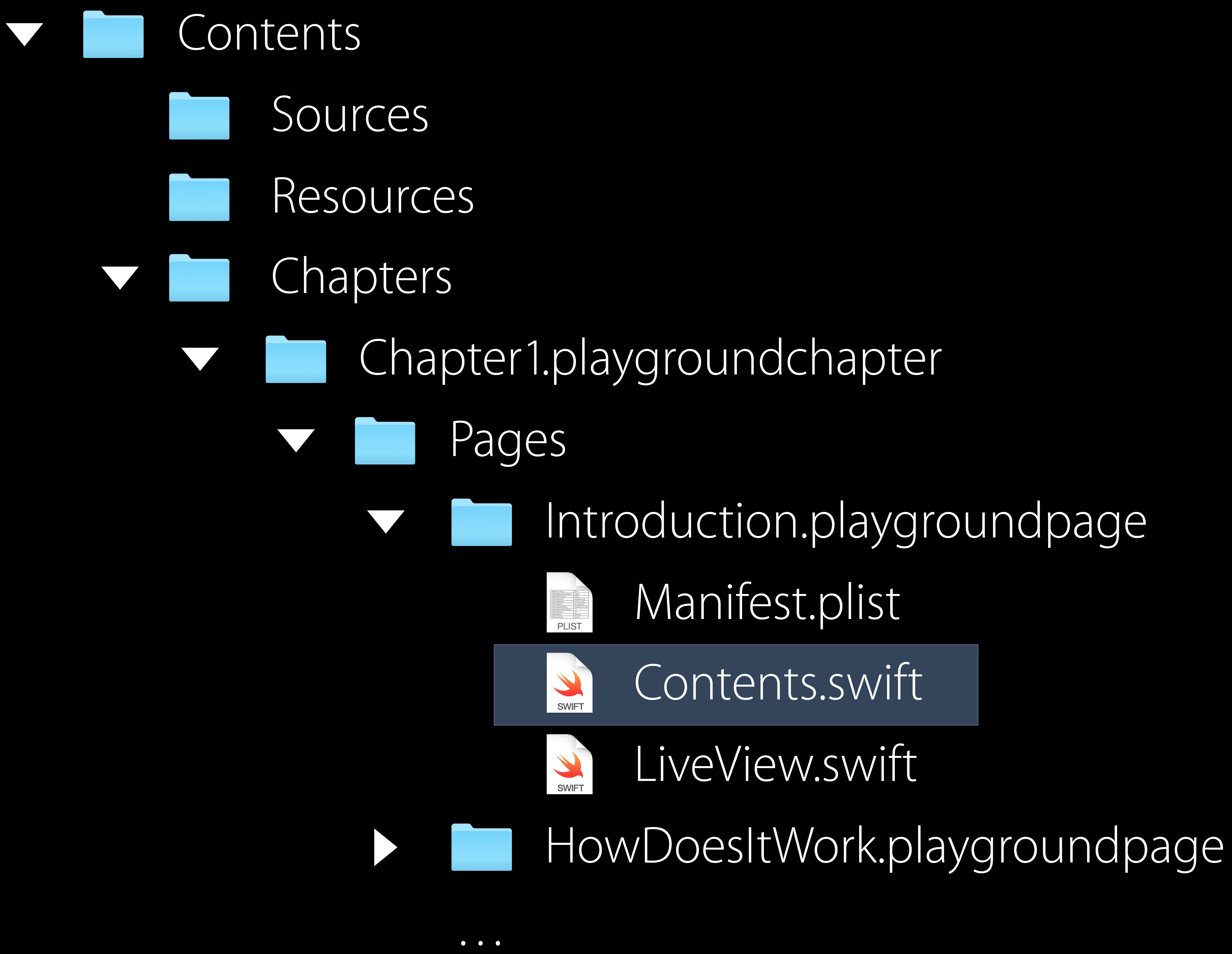
Em responds, "Boo! who?". Now, deliver the punchline, "Are you crying?".

When you're ready, continue to the next page to see how this `say(...)` function works.

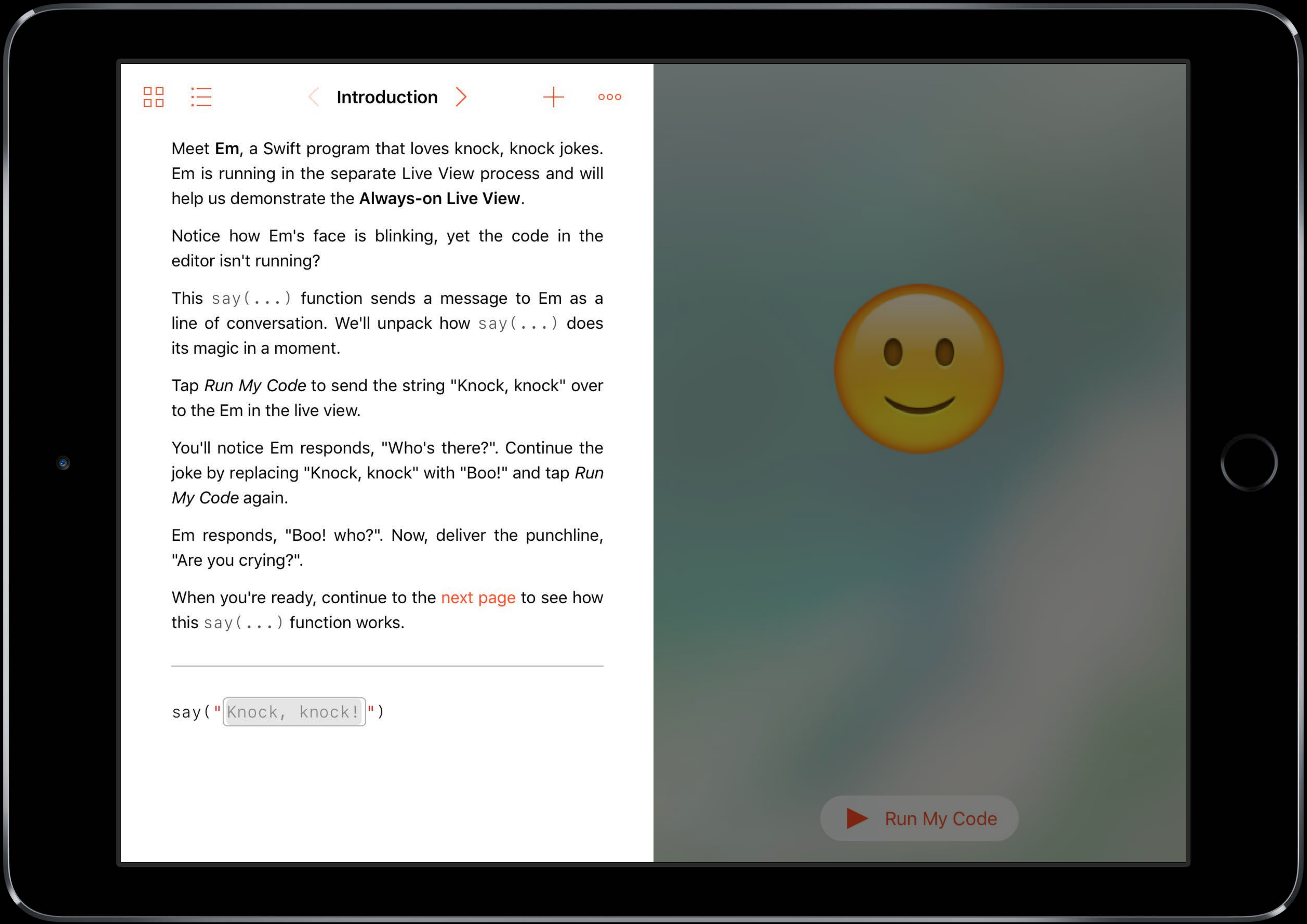
```
say("Knock, knock!")
```

PlaygroundLoggingMode = "Off"

```
▼ Contents
      Sources
      Resources
  ▼ Chapters
      ▼ Chapter1.playgroundchapter
          ▼ Pages
              ▼ Introduction.playgroundpage
                    Manifest.plist
                    Contents.swift
                    LiveView.swift
              ▶ HowDoesItWork.playgroundpage
                ...
```



```
▼ Contents
      Sources
      Resources
  ▼ Chapters
    ▼ Chapter1.playgroundchapter
      ▼ Pages
        ▼ Introduction.playgroundpage
              Manifest.plist
              Contents.swift
              LiveView.swift
        ▶ HowDoesItWork.playgroundpage
          ...
```



Meet **Em**, a Swift program that loves knock, knock jokes. Em is running in the separate Live View process and will help us demonstrate the **Always-on Live View**.

Notice how Em's face is blinking, yet the code in the editor isn't running?

This `say(...)` function sends a message to Em as a line of conversation. We'll unpack how `say(...)` does its magic in a moment.

Tap *Run My Code* to send the string "Knock, knock" over to the Em in the live view.

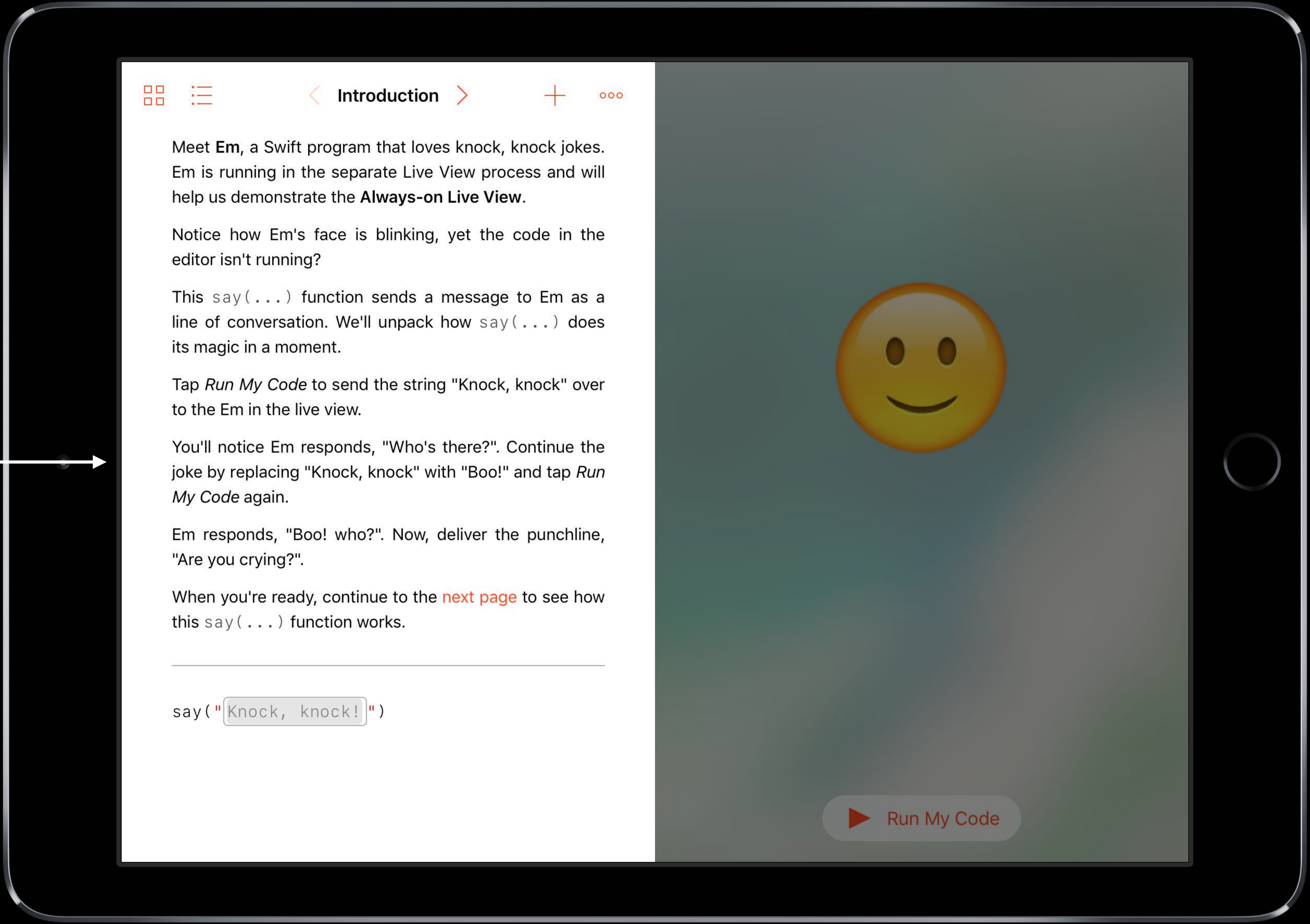
You'll notice Em responds, "Who's there?". Continue the joke by replacing "Knock, knock" with "Boo!" and tap *Run My Code* again.

Em responds, "Boo! who?". Now, deliver the punchline, "Are you crying?".

When you're ready, continue to the next page to see how this `say(...)` function works.

```
say("Knock, knock!")
```

Contents.swift



Meet **Em**, a Swift program that loves knock, knock jokes. Em is running in the separate Live View process and will help us demonstrate the **Always-on Live View**.

Notice how Em's face is blinking, yet the code in the editor isn't running?

This `say(...)` function sends a message to Em as a line of conversation. We'll unpack how `say(...)` does its magic in a moment.

Tap *Run My Code* to send the string "Knock, knock" over to the Em in the live view.

You'll notice Em responds, "Who's there?". Continue the joke by replacing "Knock, knock" with "Boo!" and tap *Run My Code* again.

Em responds, "Boo! who?". Now, deliver the punchline, "Are you crying?".

When you're ready, continue to the next page to see how this `say(...)` function works.

```
say("Knock, knock!")
```

```swift
// Contents.swift

/*:
 Instructions about the page...
 */
//#-hidden-code
import PlaygroundSupport
func say(_ message: String) {
    let page = PlaygroundPage.current
    if let proxy = page.liveView as? PlaygroundRemoteLiveViewProxy {
        proxy.send(.string(message))
    }
}
//#-end-hidden-code

say(/*#-editable-code*/"<#Knock, knock!#>"/*#-end-editable-code*/)
```

```swift
// Contents.swift

/*:
 Instructions about the page...
 */
//#-hidden-code
import PlaygroundSupport
func say(_ message: String) {
    let page = PlaygroundPage.current
    if let proxy = page.liveView as? PlaygroundRemoteLiveViewProxy {
        proxy.send(.string(message))
    }
}
//#-end-hidden-code

say(/*#-editable-code*/"<#Knock, knock!#>"/*#-end-editable-code*/)
```

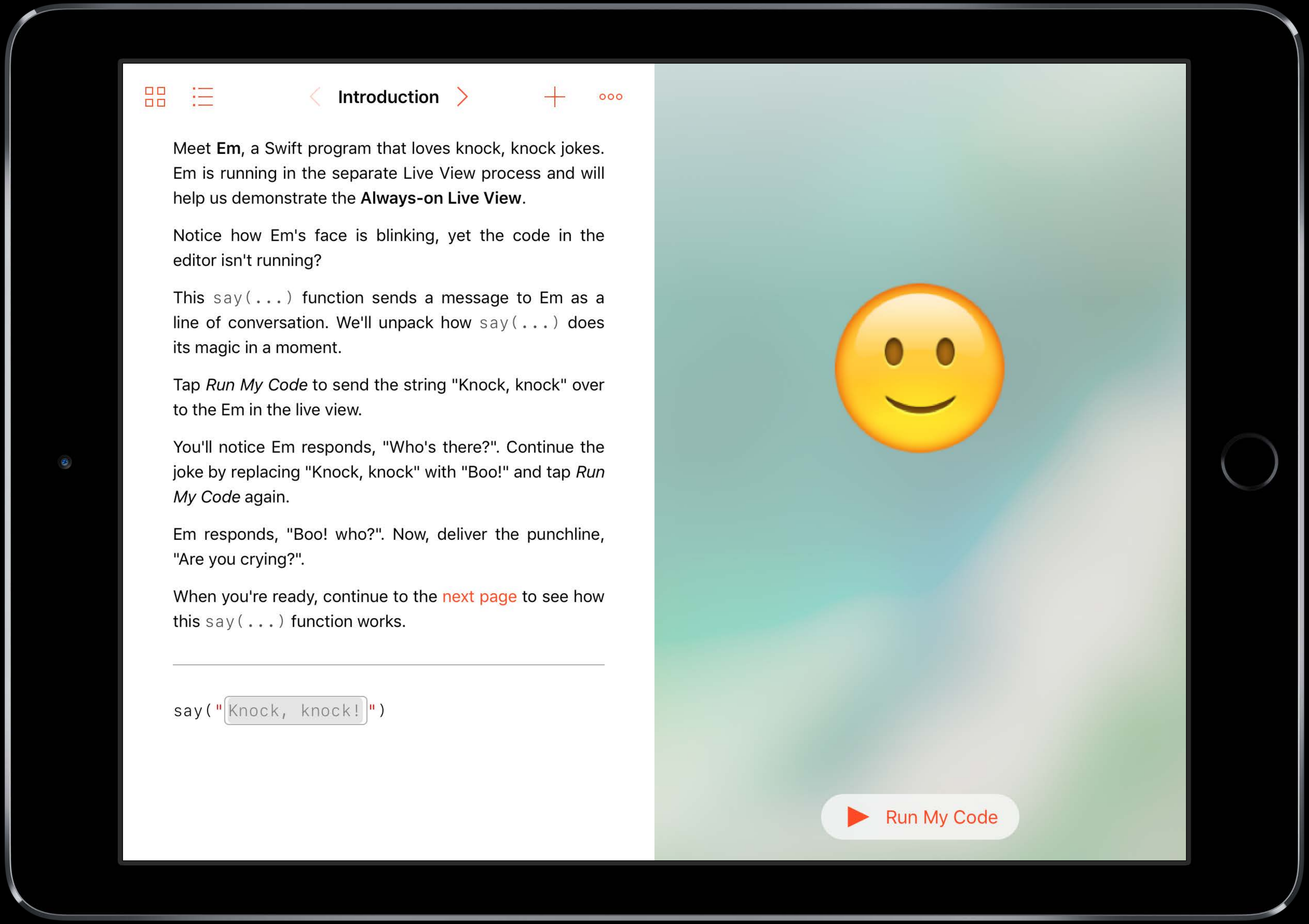
```swift
// Contents.swift

/*:
 Instructions about the page...
 */
//#-hidden-code
import PlaygroundSupport
func say(_ message: String) {
    let page = PlaygroundPage.current
    if let proxy = page.liveView as? PlaygroundRemoteLiveViewProxy {
        proxy.send(.string(message))
    }
}
//#-end-hidden-code

say(/*#-editable-code*/"<#Knock, knock!#>"/*#-end-editable-code*/)
```

```swift
// Contents.swift

/*:
 Instructions about the page...
 */
//#-hidden-code
import PlaygroundSupport
func say(_ message: String) {
    let page = PlaygroundPage.current
    if let proxy = page.liveView as? PlaygroundRemoteLiveViewProxy {
        proxy.send(.string(message))
    }
}
//#-end-hidden-code

say(/*#-editable-code*/"<#Knock, knock!#>"/*#-end-editable-code*/)
```

```swift
// Contents.swift

/*:
 Instructions about the page...
 */
//#-hidden-code
import PlaygroundSupport
func say(_ message: String) {
    let page = PlaygroundPage.current
    if let proxy = page.liveView as? PlaygroundRemoteLiveViewProxy {
        proxy.send(.string(message))
    }
}
//#-end-hidden-code

say(/*#-editable-code*/"<#Knock, knock!#>"/*#-end-editable-code*/)
```

```swift
// Contents.swift

/*:
 Instructions about the page...
 */
//#-hidden-code
import PlaygroundSupport
func say(_ message: String) {
    let page = PlaygroundPage.current
    if let proxy = page.liveView as? PlaygroundRemoteLiveViewProxy {
        proxy.send(.string(message))
    }
}
//#-end-hidden-code

say(/*#-editable-code*/"<#Knock, knock!#>"/*#-end-editable-code*/)
```

```swift
// Contents.swift

/*:
 Instructions about the page...
 */
//#-hidden-code
import PlaygroundSupport
func say(_ message: String) {
    let page = PlaygroundPage.current
    if let proxy = page.liveView as? PlaygroundRemoteLiveViewProxy {
        proxy.send(.string(message))
    }
}
//#-end-hidden-code

say(/*#-editable-code*/"<#Knock, knock!#>"/*#-end-editable-code*/)
```



Meet **Em**, a Swift program that loves knock, knock jokes. Em is running in the separate Live View process and will help us demonstrate the **Always-on Live View**.

Notice how Em's face is blinking, yet the code in the editor isn't running?

This `say(...)` function sends a message to Em as a line of conversation. We'll unpack how `say(...)` does its magic in a moment.

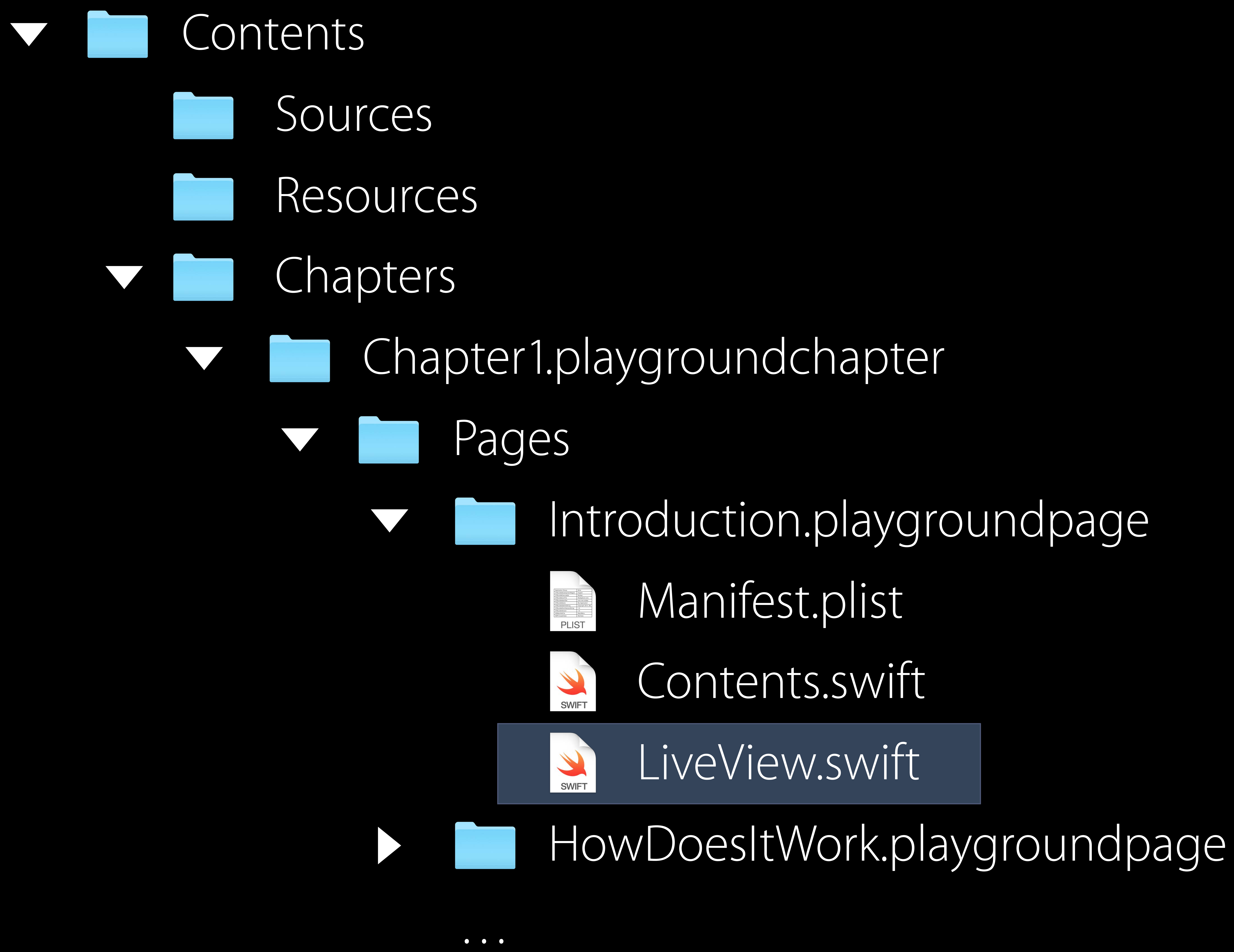
Tap *Run My Code* to send the string "Knock, knock" over to the Em in the live view.

You'll notice Em responds, "Who's there?". Continue the joke by replacing "Knock, knock" with "Boo!" and tap *Run My Code* again.

Em responds, "Boo! who?". Now, deliver the punchline, "Are you crying?".

When you're ready, continue to the next page to see how this `say(...)` function works.

```
say("Knock, knock!")
```



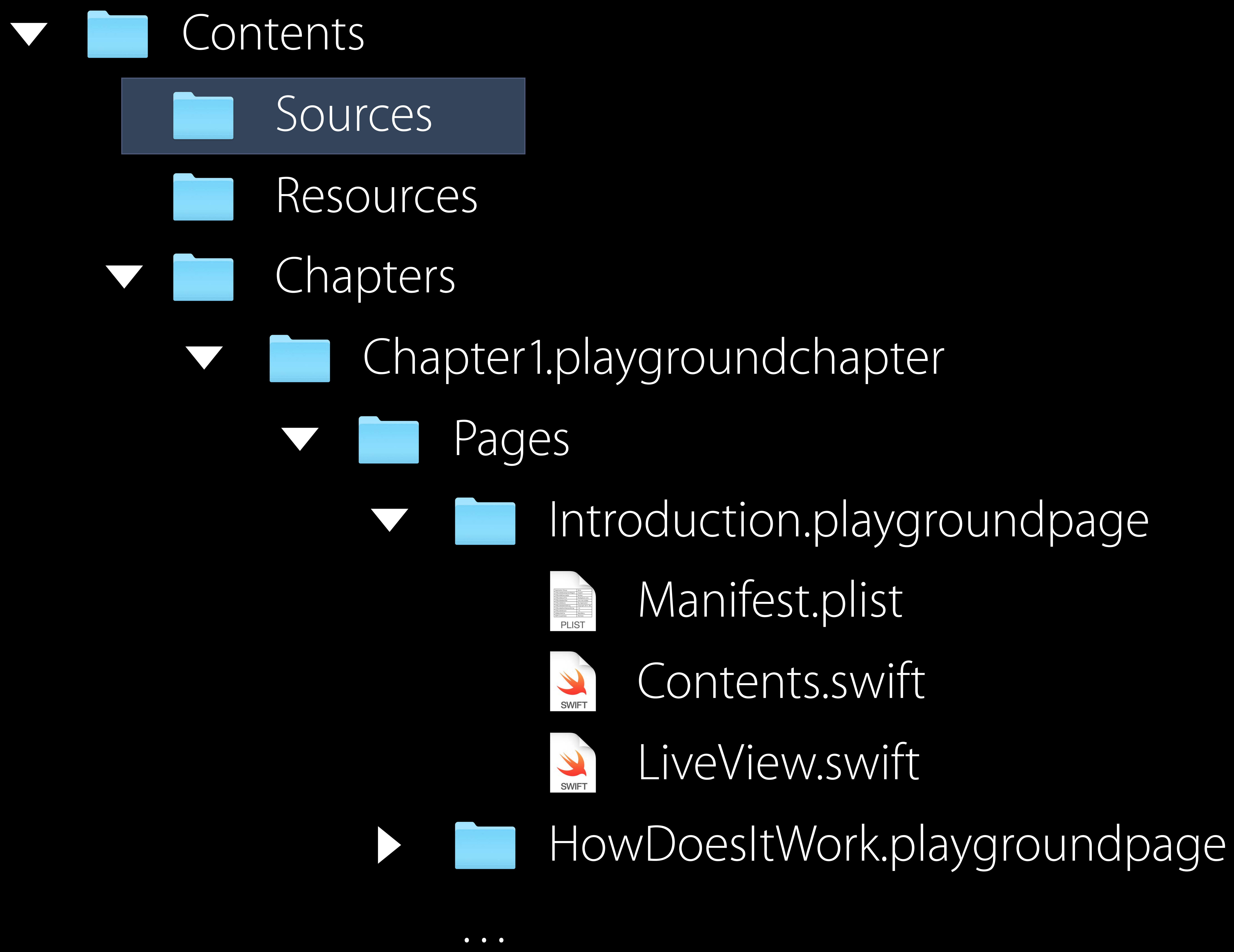
```
Contents
    Sources
    Resources
    Chapters
        Chapter1.playgroundchapter
            Pages
                Introduction.playgroundpage
                    Manifest.plist
                    Contents.swift
                    LiveView.swift
                HowDoesItWork.playgroundpage
                ...
```

```swift
// LiveView.swift

import PlaygroundSupport
let page = PlaygroundPage.current
page.liveView = FaceViewController()
```

```swift
// LiveView.swift

import PlaygroundSupport
let page = PlaygroundPage.current
page.liveView = FaceViewController()
```

```swift
// LiveView.swift

import PlaygroundSupport
let page = PlaygroundPage.current
page.liveView = FaceViewController()
```

```swift
// LiveView.swift

import PlaygroundSupport
let page = PlaygroundPage.current
page.liveView = FaceViewController()
```



```
▼ Contents
    Sources
    Resources
  ▼ Chapters
    ▼ Chapter1.playgroundchapter
      ▼ Pages
        ▼ Introduction.playgroundpage
            Manifest.plist
            Contents.swift
            LiveView.swift
        ▶ HowDoesItWork.playgroundpage
          ...
```

```swift
import PlaygroundSupport
let page = PlaygroundPage.current
page.liveView = FaceViewController()
```

Contents.swift

```swift
import PlaygroundSupport
let page = PlaygroundPage.current
page.liveView = FaceViewController()
```

Contents.swift

```swift
import PlaygroundSupport
let page = PlaygroundPage.current
page.liveView = FaceViewController()
```

In main process

Contents.swift

```swift
import PlaygroundSupport
let page = PlaygroundPage.current
page.liveView = FaceViewController()
```

In main process

Only active while running

Contents.swift

```swift
import PlaygroundSupport
let page = PlaygroundPage.current
page.liveView = FaceViewController()
```

In main process

Only active while running

Contents.swift

```swift
import PlaygroundSupport
let page = PlaygroundPage.current
page.liveView = FaceViewController()
```

LiveView.swift

```swift
import PlaygroundSupport
let page = PlaygroundPage.current
page.liveView = FaceViewController()
```

In main process

Only active while running

Contents.swift

```swift
import PlaygroundSupport
let page = PlaygroundPage.current
page.liveView = FaceViewController()
```

In main process

Only active while running

LiveView.swift

```swift
import PlaygroundSupport
let page = PlaygroundPage.current
page.liveView = FaceViewController()
```

Separate process

Contents.swift

```swift
import PlaygroundSupport
let page = PlaygroundPage.current
page.liveView = FaceViewController()
```

In main process

Only active while running

LiveView.swift

```swift
import PlaygroundSupport
let page = PlaygroundPage.current
page.liveView = FaceViewController()
```

Separate process

Running all the time

Main Process (Contents.swift)

```swift
import PlaygroundSupport
let page = PlaygroundPage.current
if let proxy = page.liveView as? PlaygroundRemoteLiveViewProxy {
    let message: PlaygroundValue = .string("Knock, knock!")
    proxy.send(message)
}
```

Main Process (Contents.swift)

```swift
import PlaygroundSupport
let page = PlaygroundPage.current
if let proxy = page.liveView as? PlaygroundRemoteLiveViewProxy {
    let message: PlaygroundValue = .string("Knock, knock!")
    proxy.send(message)
}
```

Main Process (Contents.swift)

```swift
import PlaygroundSupport
let page = PlaygroundPage.current
if let proxy = page.liveView as? PlaygroundRemoteLiveViewProxy {
    let message: PlaygroundValue = .string("Knock, knock!")
    proxy.send(message)
}
```

Main Process (Contents.swift)

```
import PlaygroundSupport
let page = PlaygroundPage.current
if let proxy = page.liveView as? PlaygroundRemoteLiveViewProxy {
    let message: PlaygroundValue = .string("Knock, knock!")
    proxy.send(message)
}
```

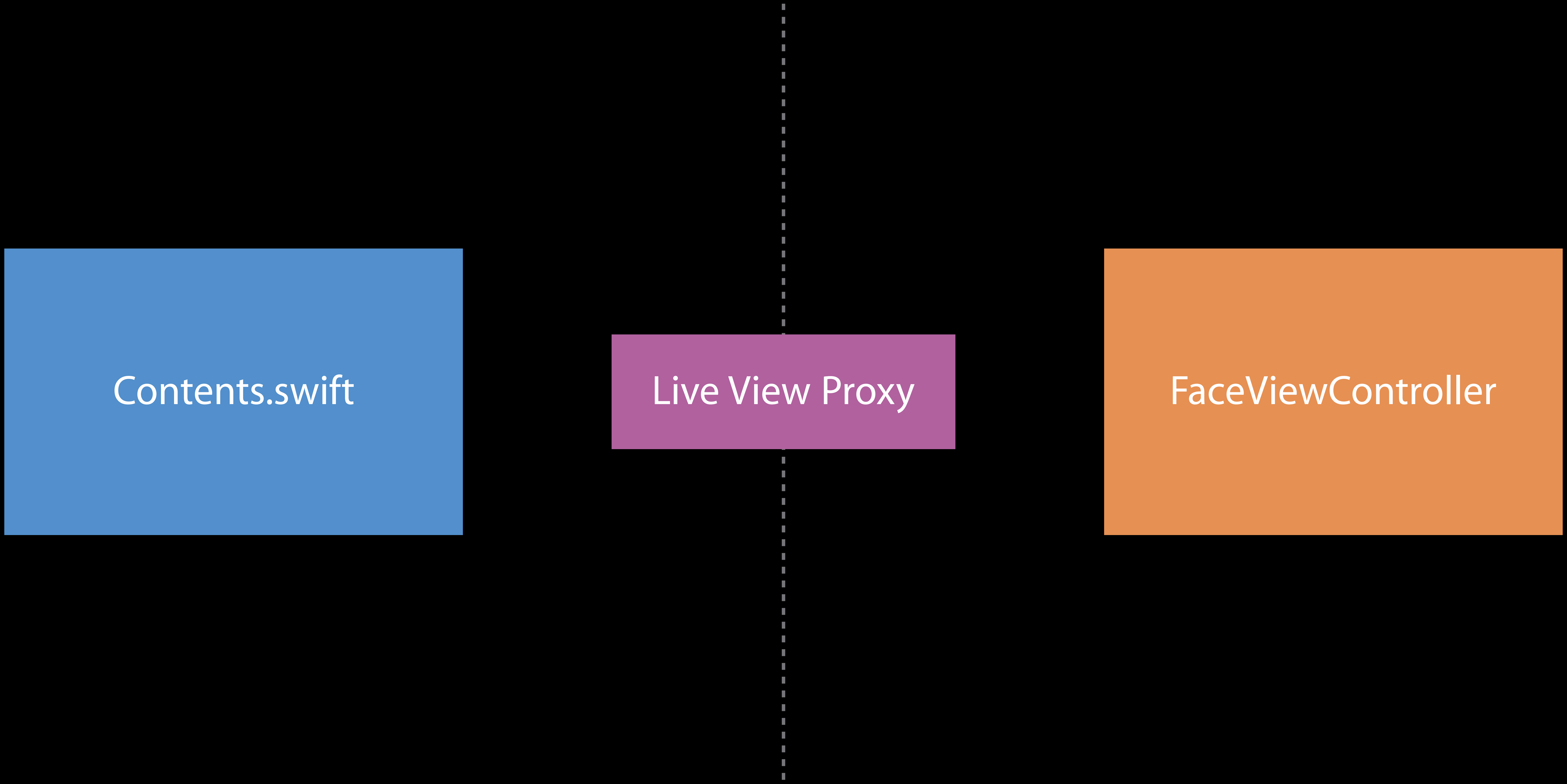
Main Process (Contents.swift)

```swift
import PlaygroundSupport
let page = PlaygroundPage.current
if let proxy = page.liveView as? PlaygroundRemoteLiveViewProxy {
    let message: PlaygroundValue = .string("Knock, knock!")
    proxy.send(message)
}
```

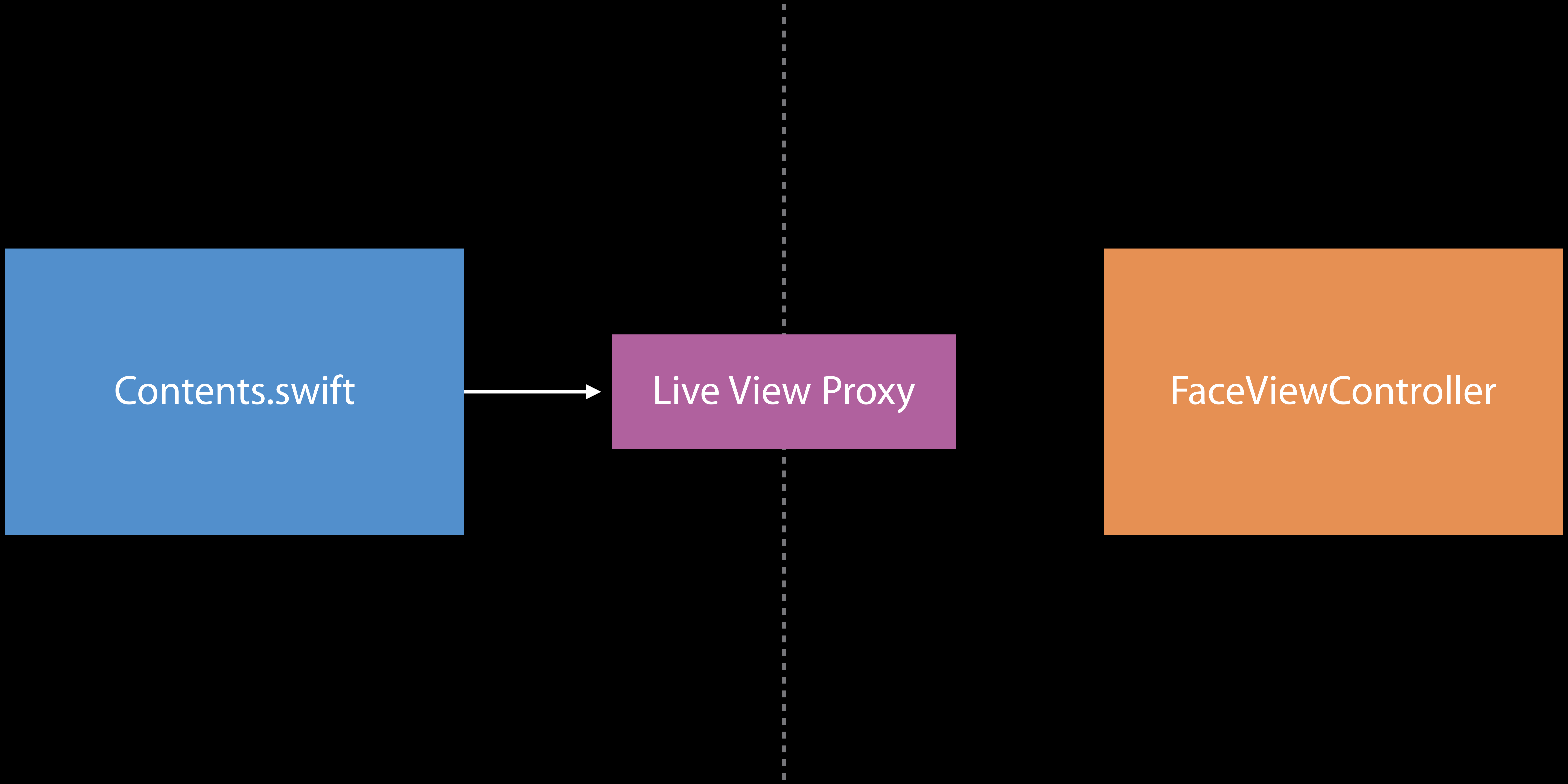
Main Process (Contents.swift)

```swift
import PlaygroundSupport
let page = PlaygroundPage.current
if let proxy = page.liveView as? PlaygroundRemoteLiveViewProxy {
    let message: PlaygroundValue = .string("Knock, knock!")
    proxy.send(message)
}
```

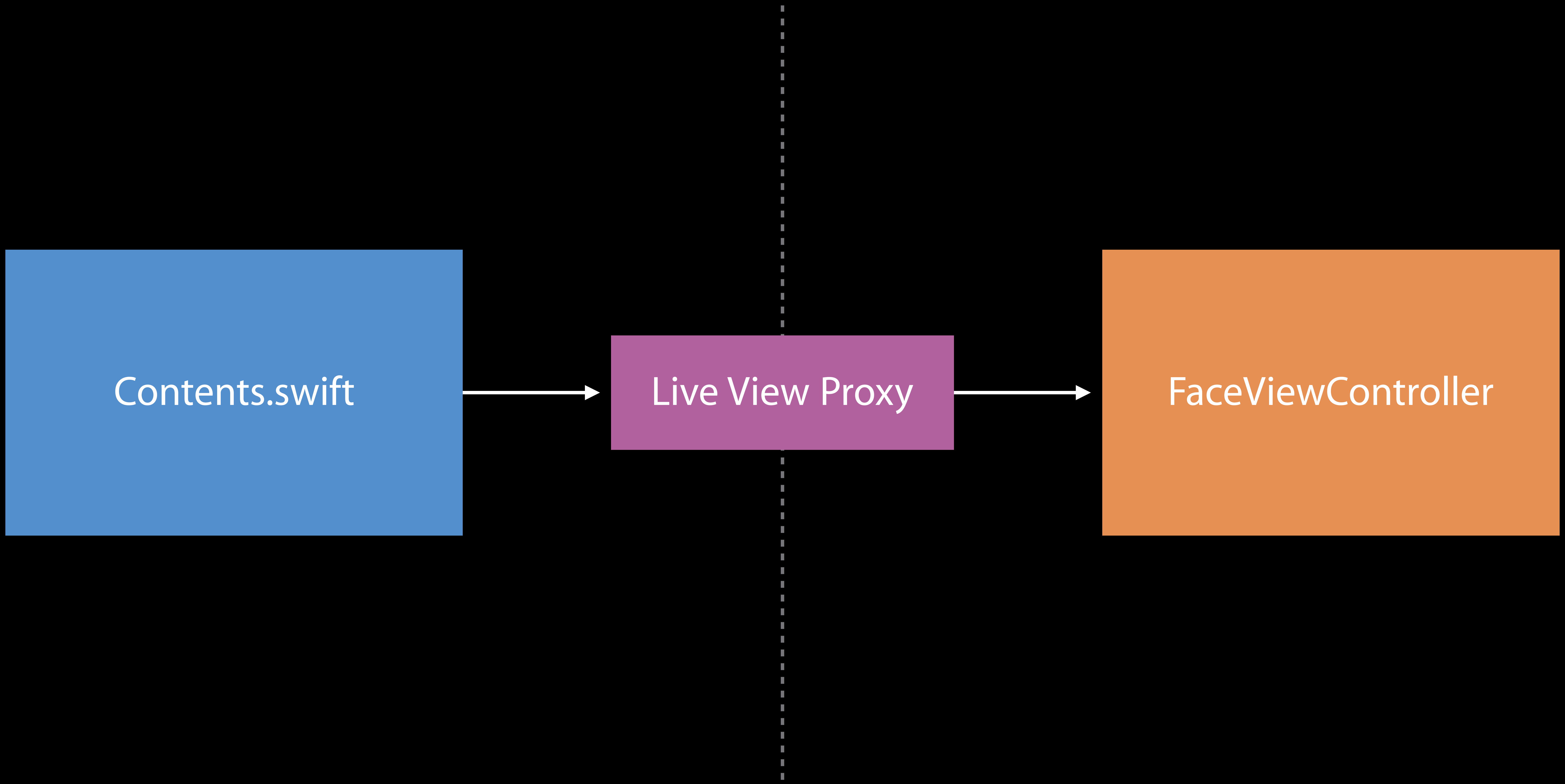
Sending to the Live View Process



Sending to the Live View Process



Sending to the Live View Process



Live View Process (LiveView.swift)

```swift
extension FaceViewController: PlaygroundLiveViewMessageHandler {
   public func receive(_ message: PlaygroundValue) {
        if case let .string(text) = message {
            processConversationLine(text)
        }
    }
}
```

Live View Process (LiveView.swift)

```swift
extension FaceViewController: PlaygroundLiveViewMessageHandler {
   public func receive(_ message: PlaygroundValue) {
        if case let .string(text) = message {
            processConversationLine(text)
        }
    }
}
```

Live View Process (LiveView.swift)

```swift
extension FaceViewController: PlaygroundLiveViewMessageHandler {
    public func receive(_ message: PlaygroundValue) {
        if case let .string(text) = message {
            processConversationLine(text)
        }
    }
}
```

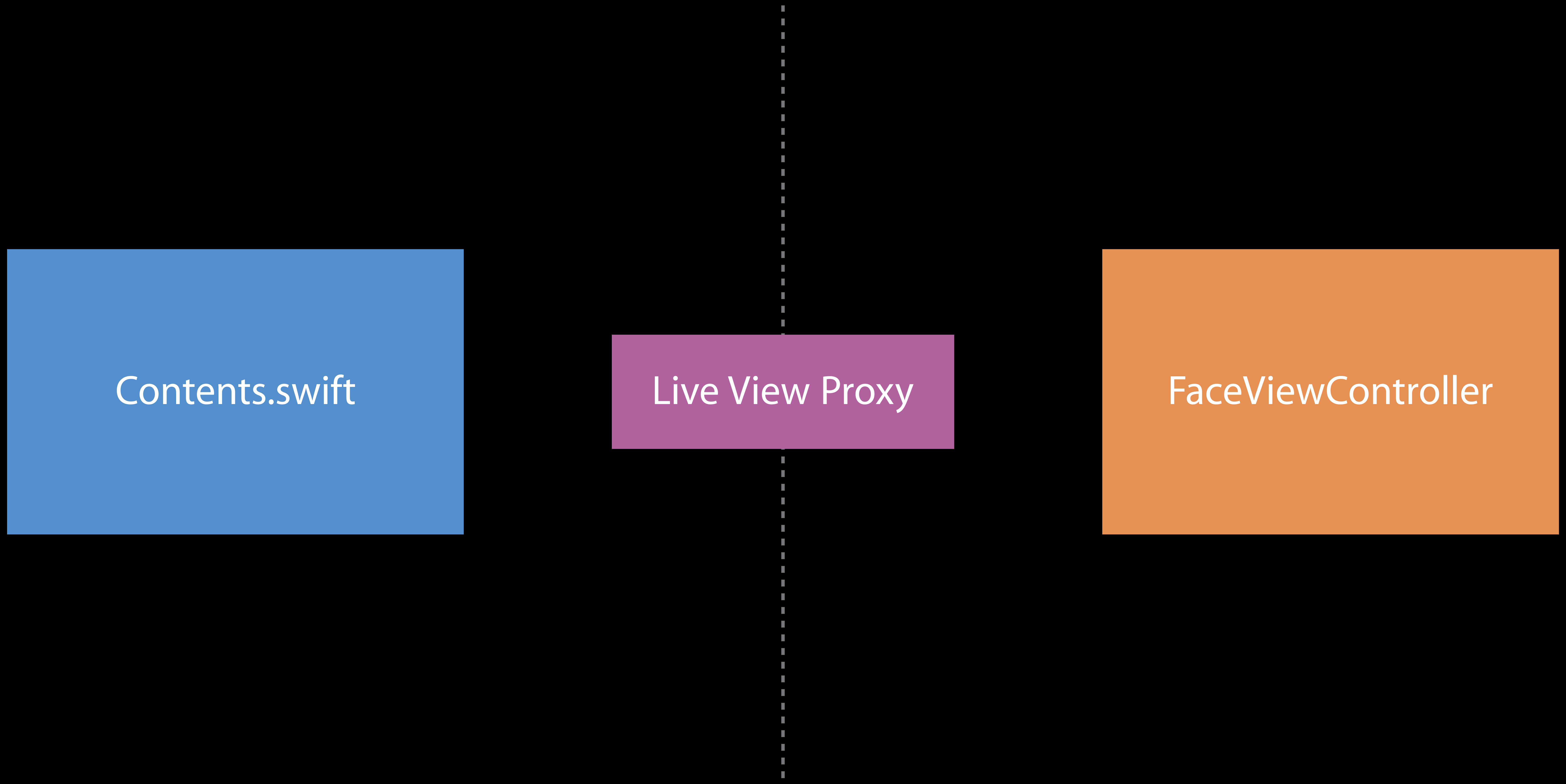
Live View Process (LiveView.swift)

```swift
extension FaceViewController: PlaygroundLiveViewMessageHandler {
   public func receive(_ message: PlaygroundValue) {
        if case let .string(text) = message {
            processConversationLine(text)
        }
    }
}
```

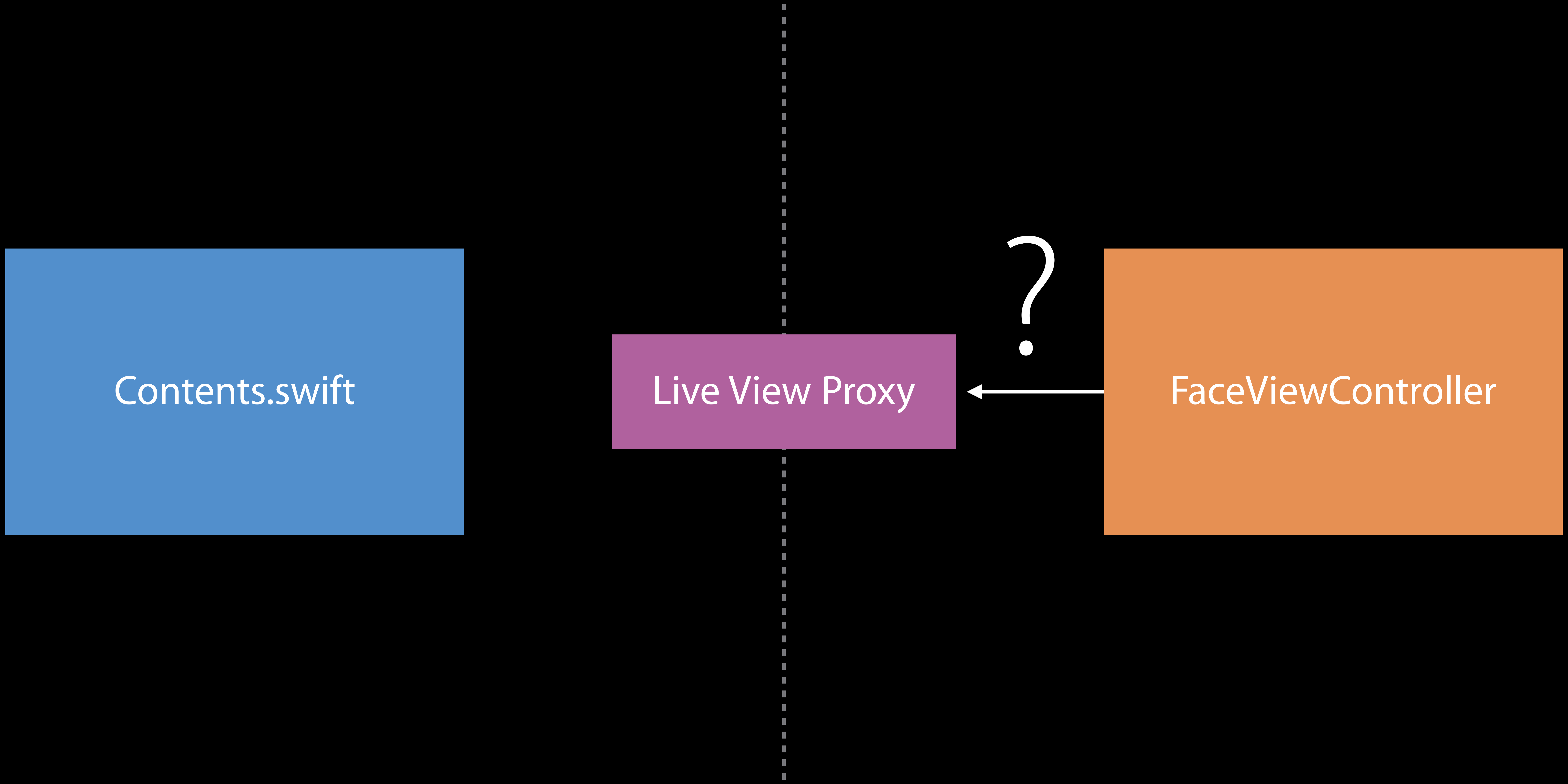
Live View Process (LiveView.swift)

```swift
extension FaceViewController: PlaygroundLiveViewMessageHandler {
   public func receive(_ message: PlaygroundValue) {
        if case let .string(text) = message {
            processConversationLine(text)
        }
    }
}
```

Sending from the Live View Process



Sending from the Live View Process



Live View Process (LiveView.swift)

```swift
extension FaceViewController: PlaygroundLiveViewMessageHandler {
   public func tapped() {
        let message: PlaygroundValue = .string("Hello!")
        send(message)
    }
}
```

Live View Process (LiveView.swift)

```swift
extension FaceViewController: PlaygroundLiveViewMessageHandler {
    public func tapped() {
        let message: PlaygroundValue = .string("Hello!")
        send(message)
    }
}
```

Live View Process (LiveView.swift)

```swift
extension FaceViewController: PlaygroundLiveViewMessageHandler {
    public func tapped() {
        let message: PlaygroundValue = .string("Hello!")
        send(message)
    }
}
```

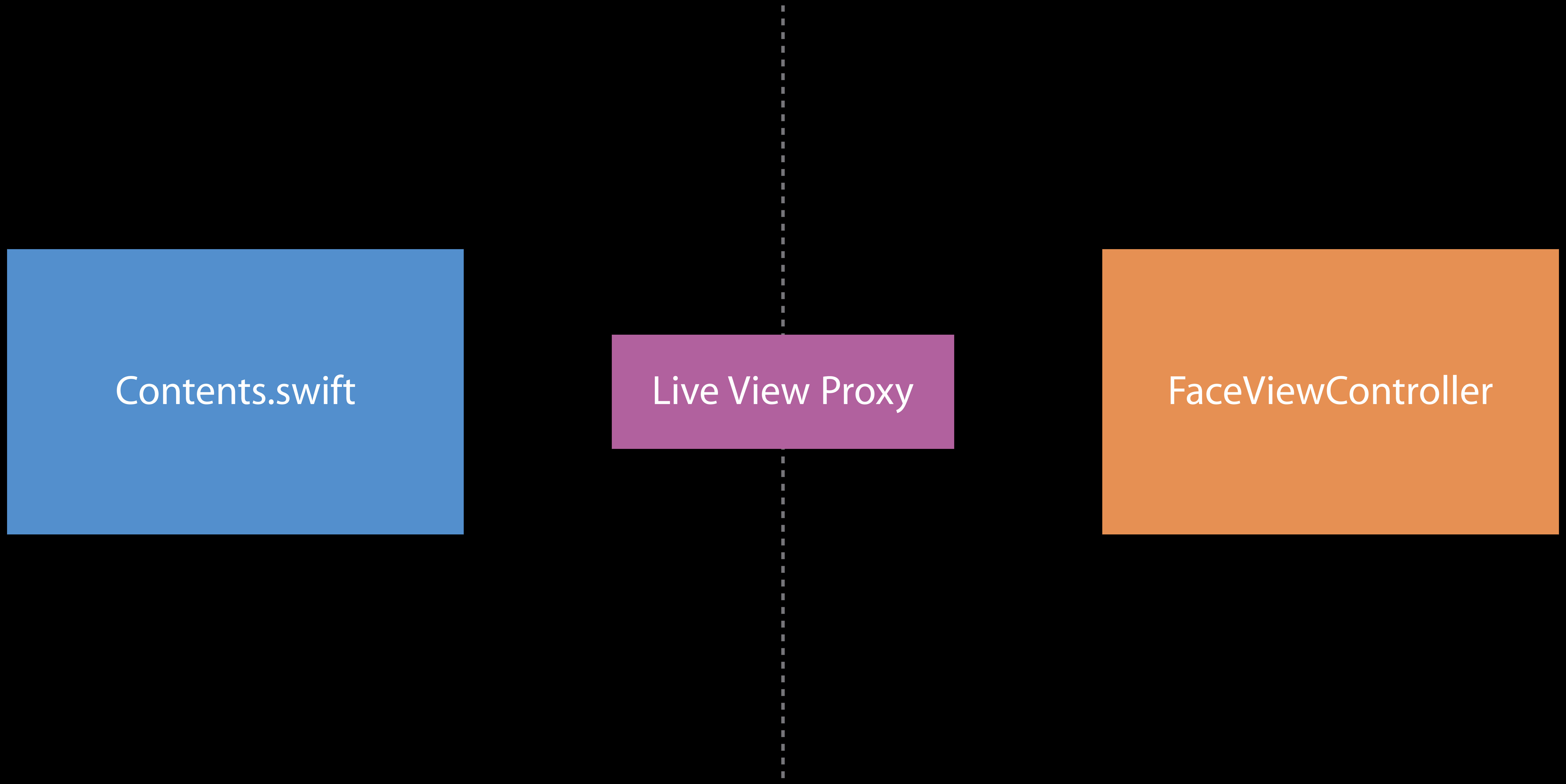
Live View Process (LiveView.swift)

```
extension FaceViewController: PlaygroundLiveViewMessageHandler {
   public func tapped() {
        let message: PlaygroundValue = .string("Hello!")
        send(message)
    }
}
```

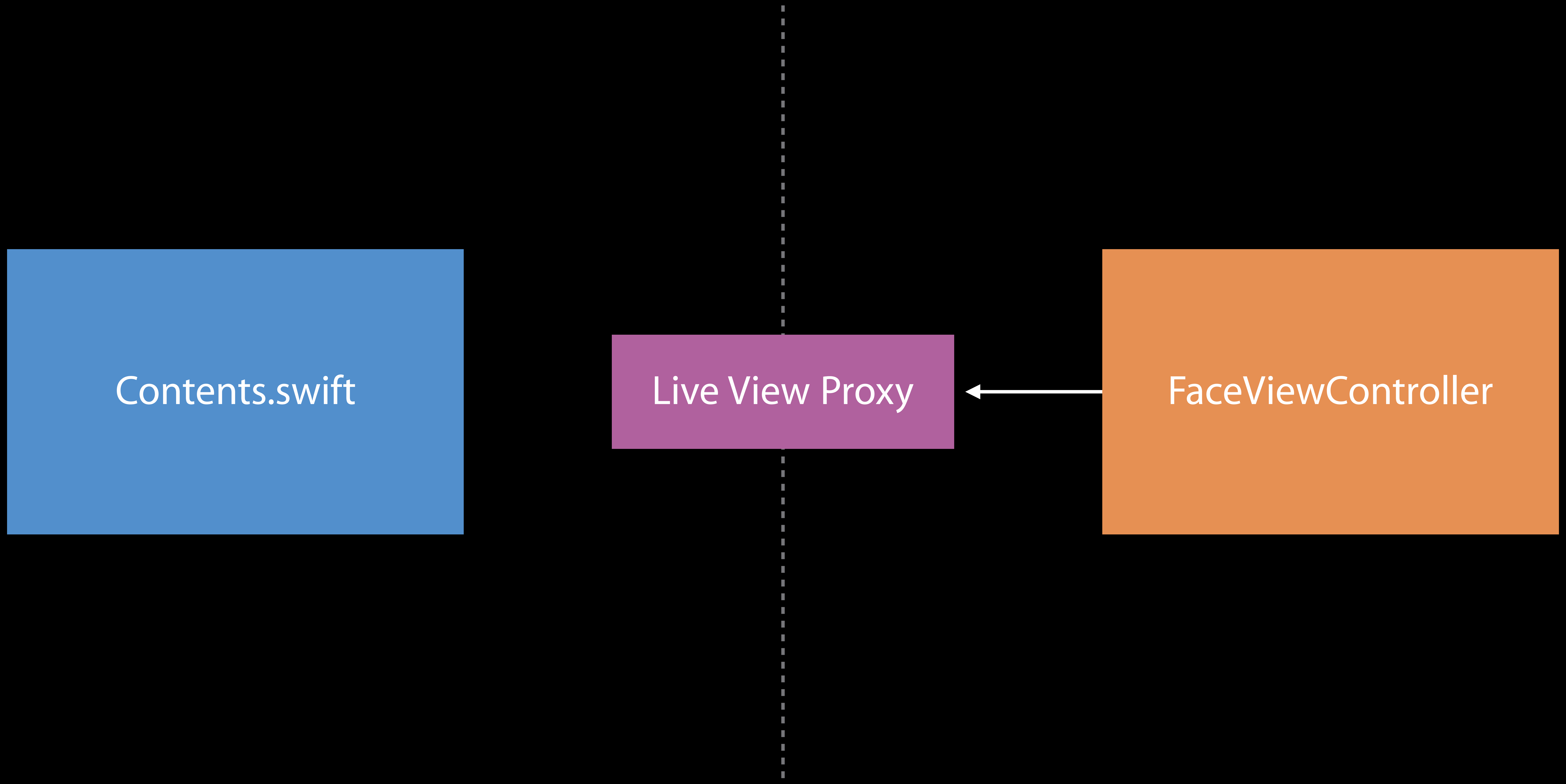
Live View Process (LiveView.swift)

```swift
extension FaceViewController: PlaygroundLiveViewMessageHandler {
   public func tapped() {
        let message: PlaygroundValue = .string("Hello!")
        send(message)
    }
}
```

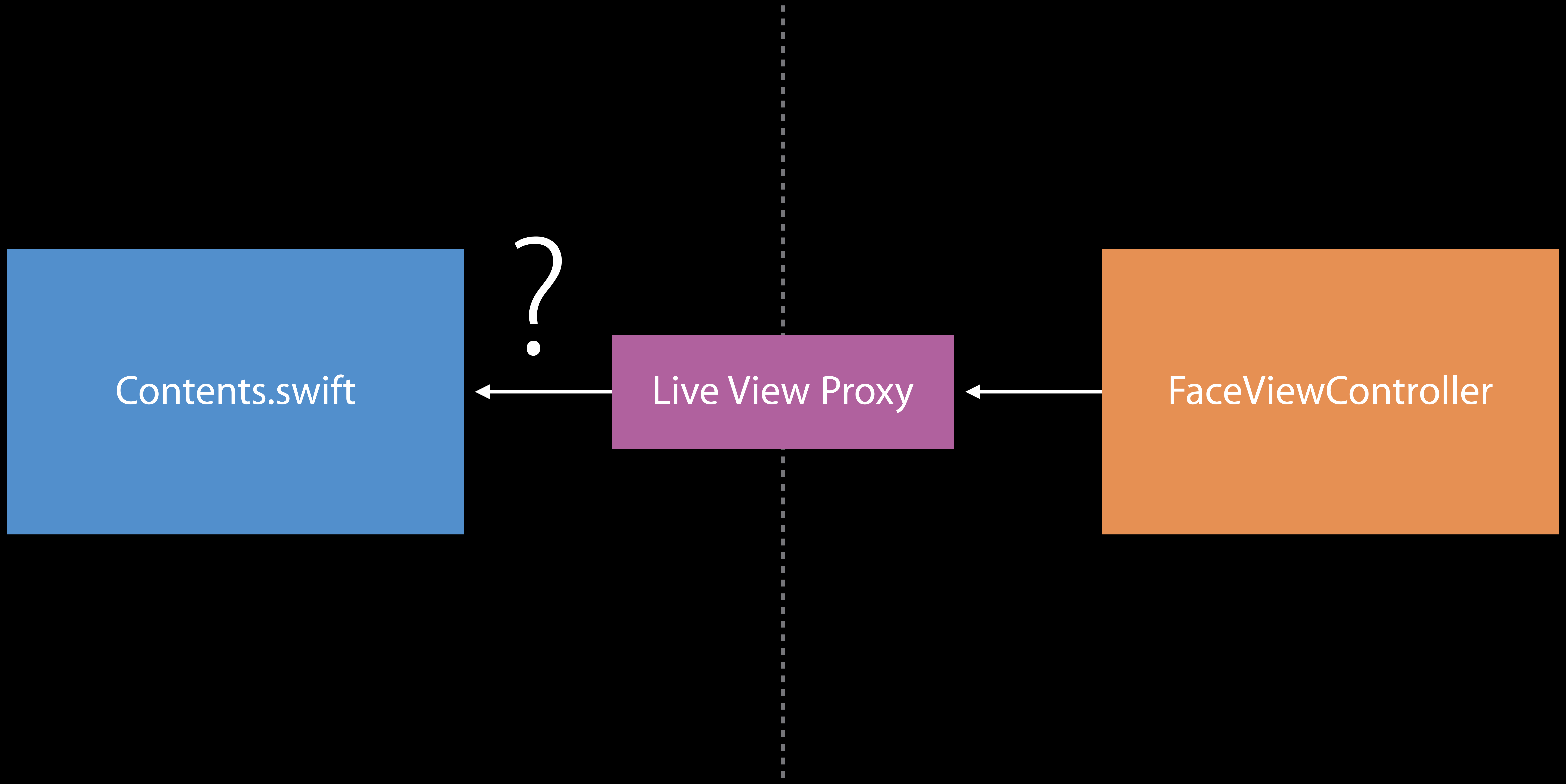
Sending from the Live View Process



Sending from the Live View Process



Sending from the Live View Process



Main Process (Contents.swift)

```swift
let page = PlaygroundPage.current
page.needsIndefiniteExecution = true
let proxy = page.liveView as? PlaygroundRemoteLiveViewProxy
class MyClassThatListens: PlaygroundRemoteLiveViewProxyDelegate {
   func remoteLiveViewProxy(_ remoteLiveViewProxy: PlaygroundRemoteLiveViewProxy,
                            received message: PlaygroundValue) {
        if case let .string(text) = message {
            doSomethingWithString(text)
        }
    }
}
let listener = MyClassThatListens()
proxy?.delegate = listener
```

Main Process (Contents.swift)

```
let page = PlaygroundPage.current
page.needsIndefiniteExecution = true
let proxy = page.liveView as? PlaygroundRemoteLiveViewProxy
class MyClassThatListens: PlaygroundRemoteLiveViewProxyDelegate {
   func remoteLiveViewProxy(_ remoteLiveViewProxy: PlaygroundRemoteLiveViewProxy,
                            received message: PlaygroundValue) {
        if case let .string(text) = message {
            doSomethingWithString(text)
        }
    }
}
let listener = MyClassThatListens()
proxy?.delegate = listener
```

Main Process (Contents.swift)

```
let page = PlaygroundPage.current
page.needsIndefiniteExecution = true
let proxy = page.liveView as? PlaygroundRemoteLiveViewProxy
class MyClassThatListens: PlaygroundRemoteLiveViewProxyDelegate {
   func remoteLiveViewProxy(_ remoteLiveViewProxy: PlaygroundRemoteLiveViewProxy,
                            received message: PlaygroundValue) {
        if case let .string(text) = message {
            doSomethingWithString(text)
        }
    }
}
let listener = MyClassThatListens()
proxy?.delegate = listener
```

Main Process (Contents.swift)

```swift
let page = PlaygroundPage.current
page.needsIndefiniteExecution = true
let proxy = page.liveView as? PlaygroundRemoteLiveViewProxy
class MyClassThatListens: PlaygroundRemoteLiveViewProxyDelegate {
   func remoteLiveViewProxy(_ remoteLiveViewProxy: PlaygroundRemoteLiveViewProxy,
                            received message: PlaygroundValue) {
        if case let .string(text) = message {
            doSomethingWithString(text)
        }
    }
}
let listener = MyClassThatListens()
proxy?.delegate = listener
```

Main Process (Contents.swift)

```
let page = PlaygroundPage.current
page.needsIndefiniteExecution = true
let proxy = page.liveView as? PlaygroundRemoteLiveViewProxy
class MyClassThatListens: PlaygroundRemoteLiveViewProxyDelegate {
   func remoteLiveViewProxy(_ remoteLiveViewProxy: PlaygroundRemoteLiveViewProxy,
                            received message: PlaygroundValue) {
        if case let .string(text) = message {
            doSomethingWithString(text)
        }
    }
}
let listener = MyClassThatListens()
proxy?.delegate = listener
```

Main Process (Contents.swift)

```swift
let page = PlaygroundPage.current
page.needsIndefiniteExecution = true
let proxy = page.liveView as? PlaygroundRemoteLiveViewProxy
class MyClassThatListens: PlaygroundRemoteLiveViewProxyDelegate {
   func remoteLiveViewProxy(_ remoteLiveViewProxy: PlaygroundRemoteLiveViewProxy,
                            received message: PlaygroundValue) {
        if case let .string(text) = message {
            doSomethingWithString(text)
        }
    }
}
let listener = MyClassThatListens()
proxy?.delegate = listener
```

Main Process (Contents.swift)

```swift
let page = PlaygroundPage.current
page.needsIndefiniteExecution = true
let proxy = page.liveView as? PlaygroundRemoteLiveViewProxy
class MyClassThatListens: PlaygroundRemoteLiveViewProxyDelegate {
   func remoteLiveViewProxy(_ remoteLiveViewProxy: PlaygroundRemoteLiveViewProxy,
                            received message: PlaygroundValue) {
        if case let .string(text) = message {
            doSomethingWithString(text)
        }
    }
}
let listener = MyClassThatListens()
proxy?.delegate = listener
```

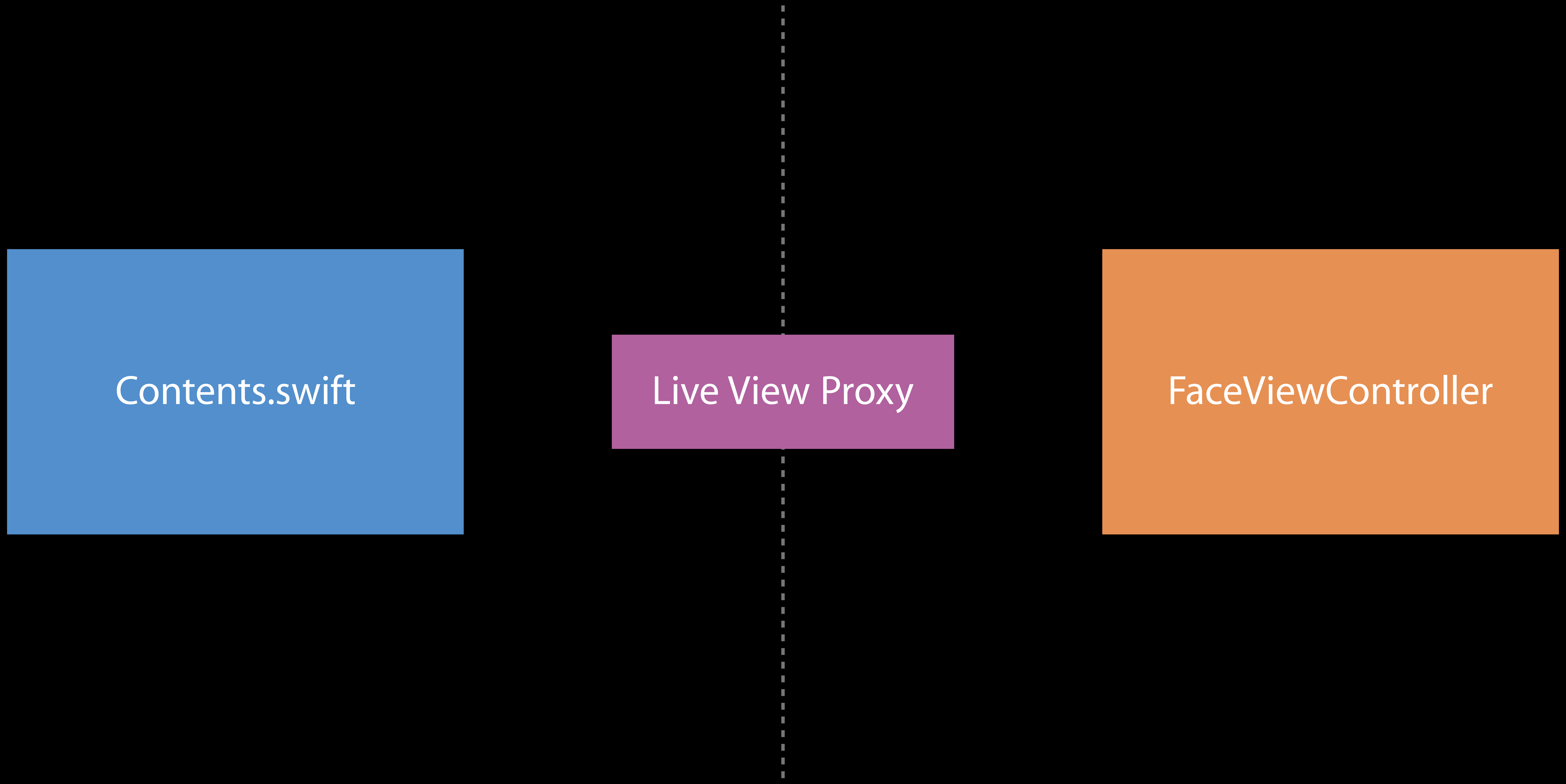
Main Process (Contents.swift)

```swift
let page = PlaygroundPage.current
page.needsIndefiniteExecution = true
let proxy = page.liveView as? PlaygroundRemoteLiveViewProxy
class MyClassThatListens: PlaygroundRemoteLiveViewProxyDelegate {
   func remoteLiveViewProxy(_ remoteLiveViewProxy: PlaygroundRemoteLiveViewProxy,
                            received message: PlaygroundValue) {
        if case let .string(text) = message {
            doSomethingWithString(text)
        }
    }
}
let listener = MyClassThatListens()
proxy?.delegate = listener
```

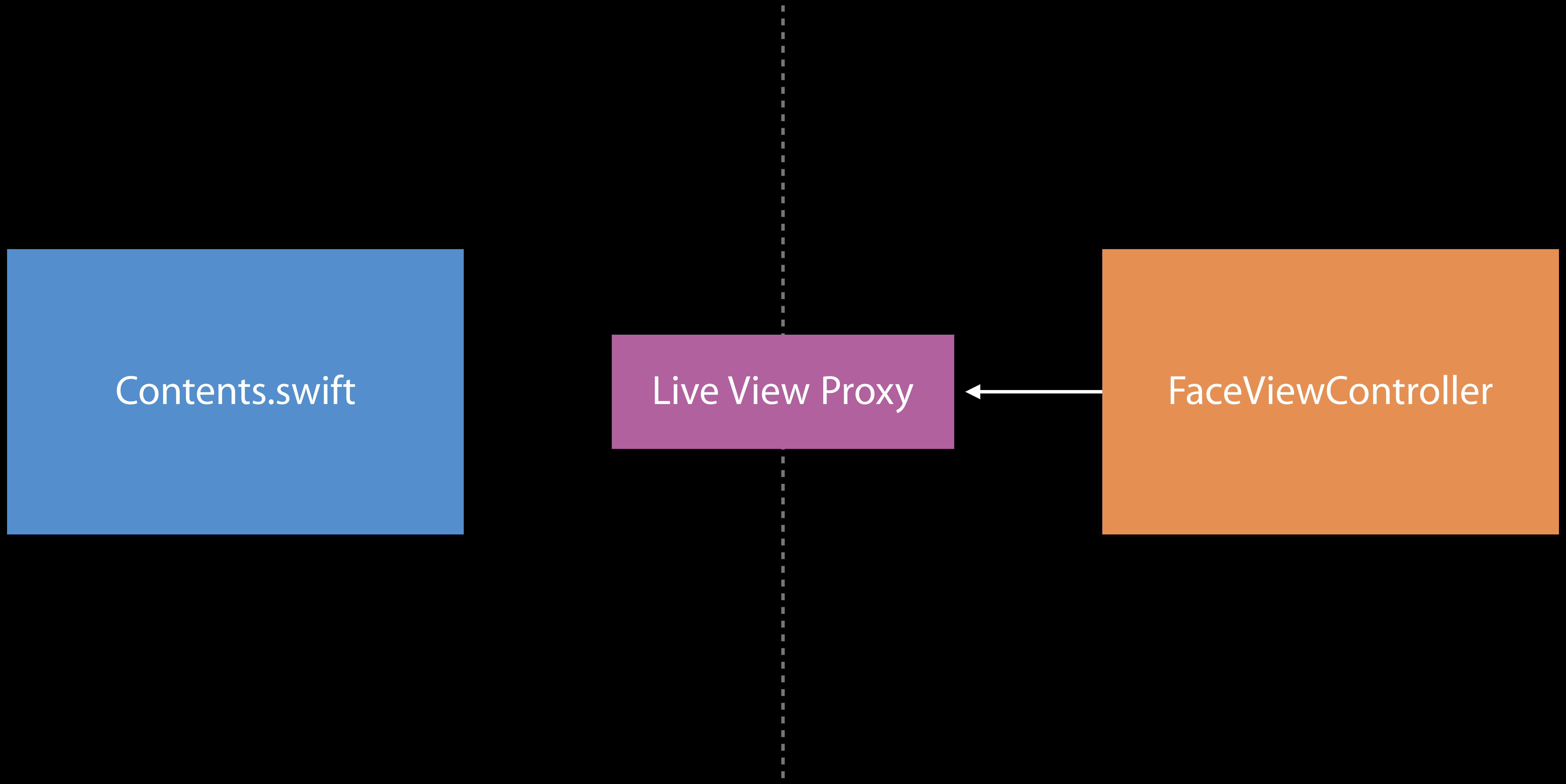
Main Process (Contents.swift)

```swift
let page = PlaygroundPage.current
page.needsIndefiniteExecution = true
let proxy = page.liveView as? PlaygroundRemoteLiveViewProxy
class MyClassThatListens: PlaygroundRemoteLiveViewProxyDelegate {
   func remoteLiveViewProxy(_ remoteLiveViewProxy: PlaygroundRemoteLiveViewProxy,
                            received message: PlaygroundValue) {
        if case let .string(text) = message {
            doSomethingWithString(text)
        }
    }
}
let listener = MyClassThatListens()
proxy?.delegate = listener
```

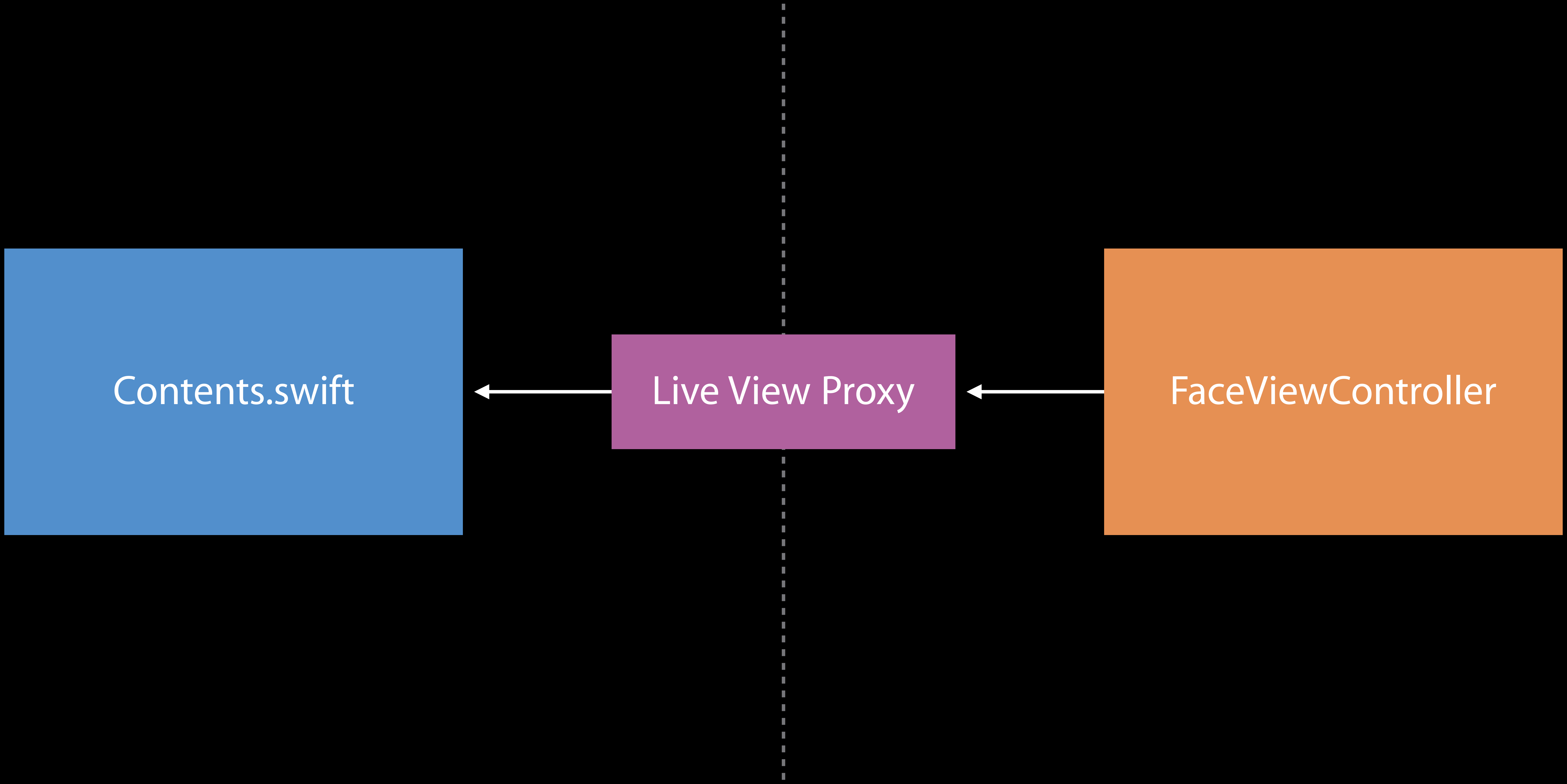
Sending from the Live View Process



Sending from the Live View Process



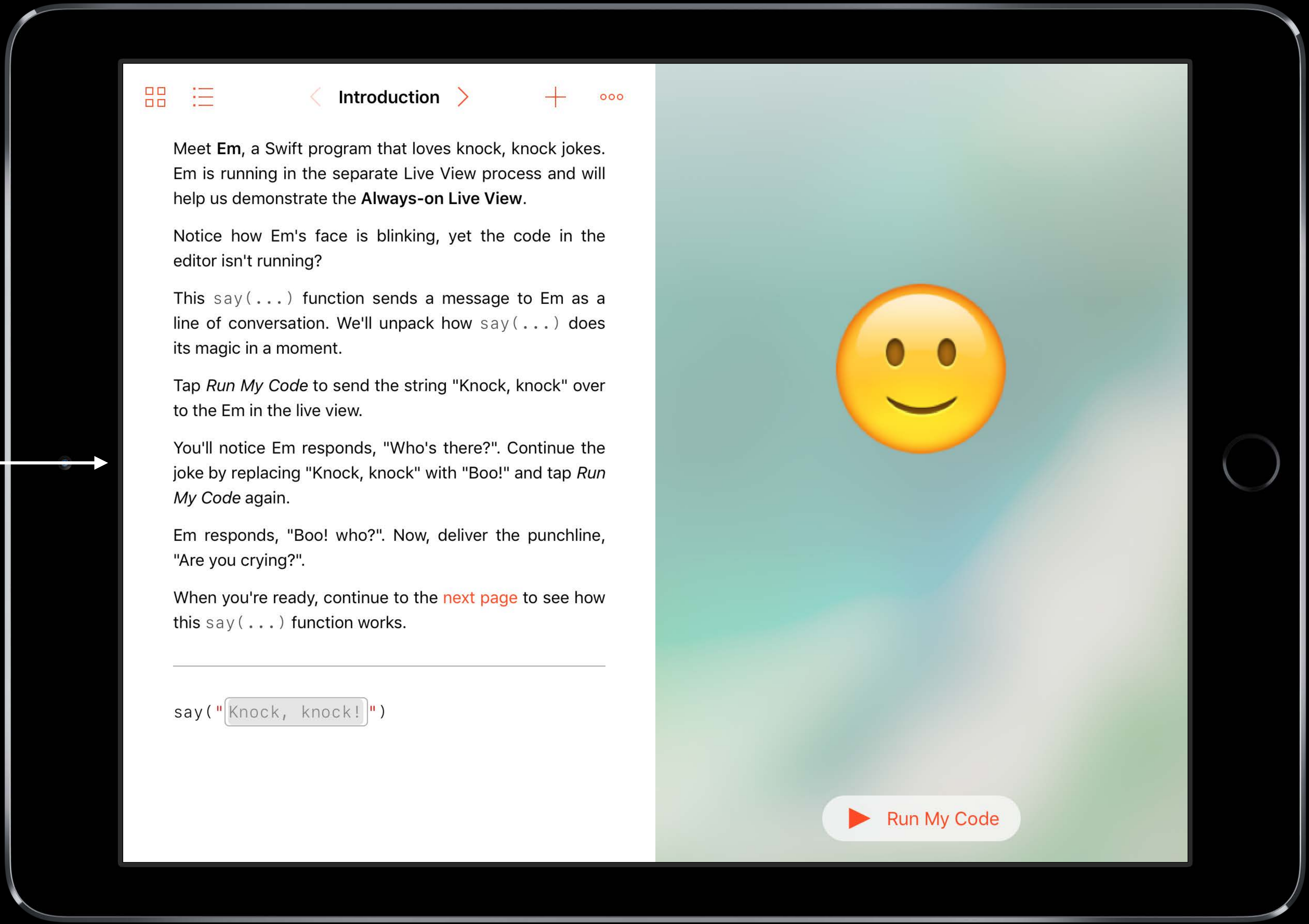
Sending from the Live View Process



```swift
// Playground Values

public enum PlaygroundValue {
    case array([PlaygroundValue])
    case dictionary([String: PlaygroundValue])
    case string(String)
    case data(Data)
    case date(Date)
    case integer(Int)
    case floatingPoint(Double)
    case boolean(Bool)
}
```

```swift
// Key/Value Store

import PlaygroundSupport

let store = PlaygroundPage.current.keyValueStore

store["Greeting"] = .string("Hello, WWDC!")

if case let .string(greeting)? = store["Greeting"] {
    print(greeting)     // "Hello, WWDC!"
}
```

Contents.swift



Meet **Em**, a Swift program that loves knock, knock jokes. Em is running in the separate Live View process and will help us demonstrate the **Always-on Live View**.

Notice how Em's face is blinking, yet the code in the editor isn't running?

This `say(...)` function sends a message to Em as a line of conversation. We'll unpack how `say(...)` does its magic in a moment.

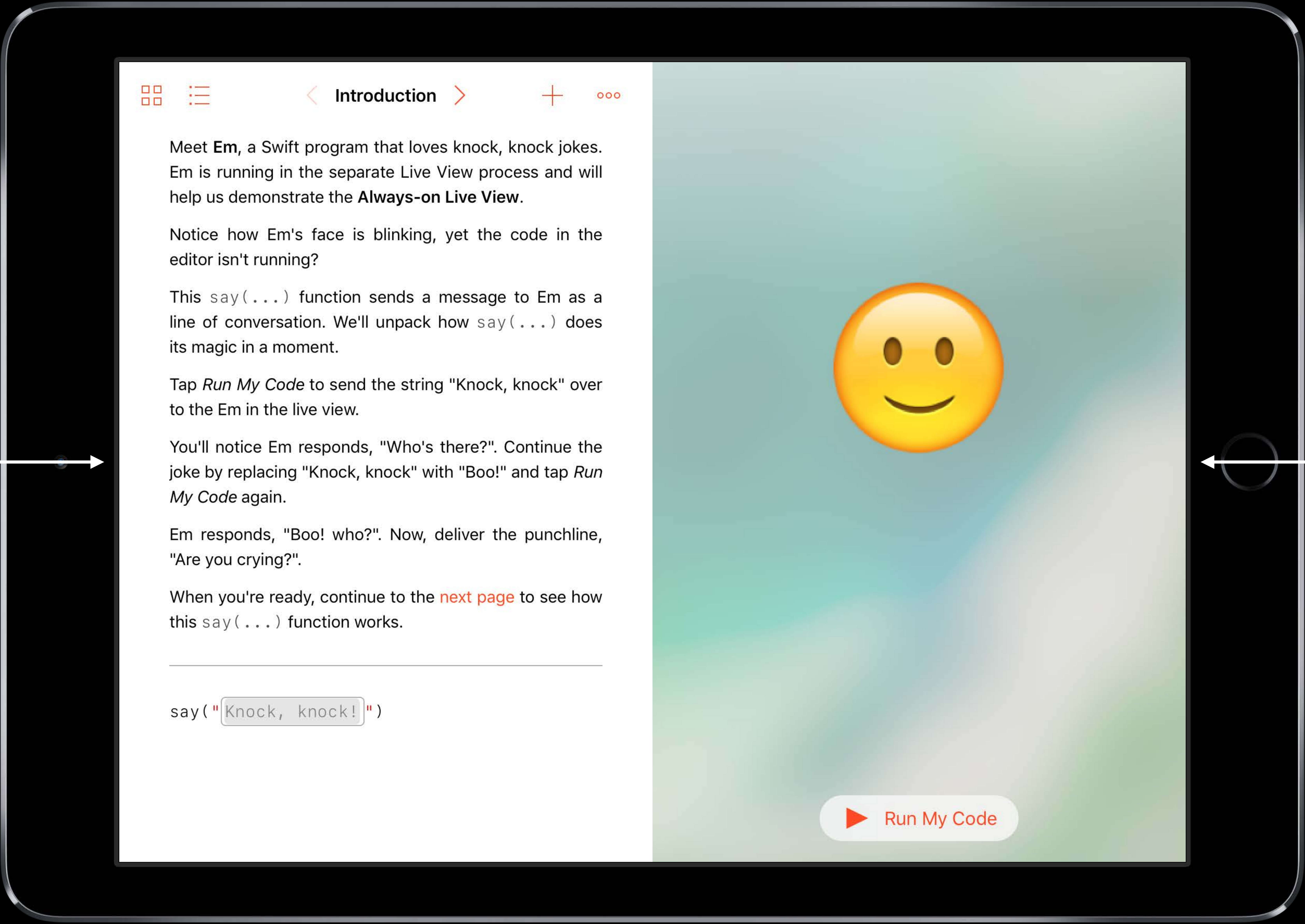
Tap *Run My Code* to send the string "Knock, knock" over to the Em in the live view.

You'll notice Em responds, "Who's there?". Continue the joke by replacing "Knock, knock" with "Boo!" and tap *Run My Code* again.

Em responds, "Boo! who?". Now, deliver the punchline, "Are you crying?".

When you're ready, continue to the next page to see how this `say(...)` function works.

```
say("Knock, knock!")
```



Meet **Em**, a Swift program that loves knock, knock jokes. Em is running in the separate Live View process and will help us demonstrate the **Always-on Live View**.

Notice how Em's face is blinking, yet the code in the editor isn't running?

This `say(...)` function sends a message to Em as a line of conversation. We'll unpack how `say(...)` does its magic in a moment.

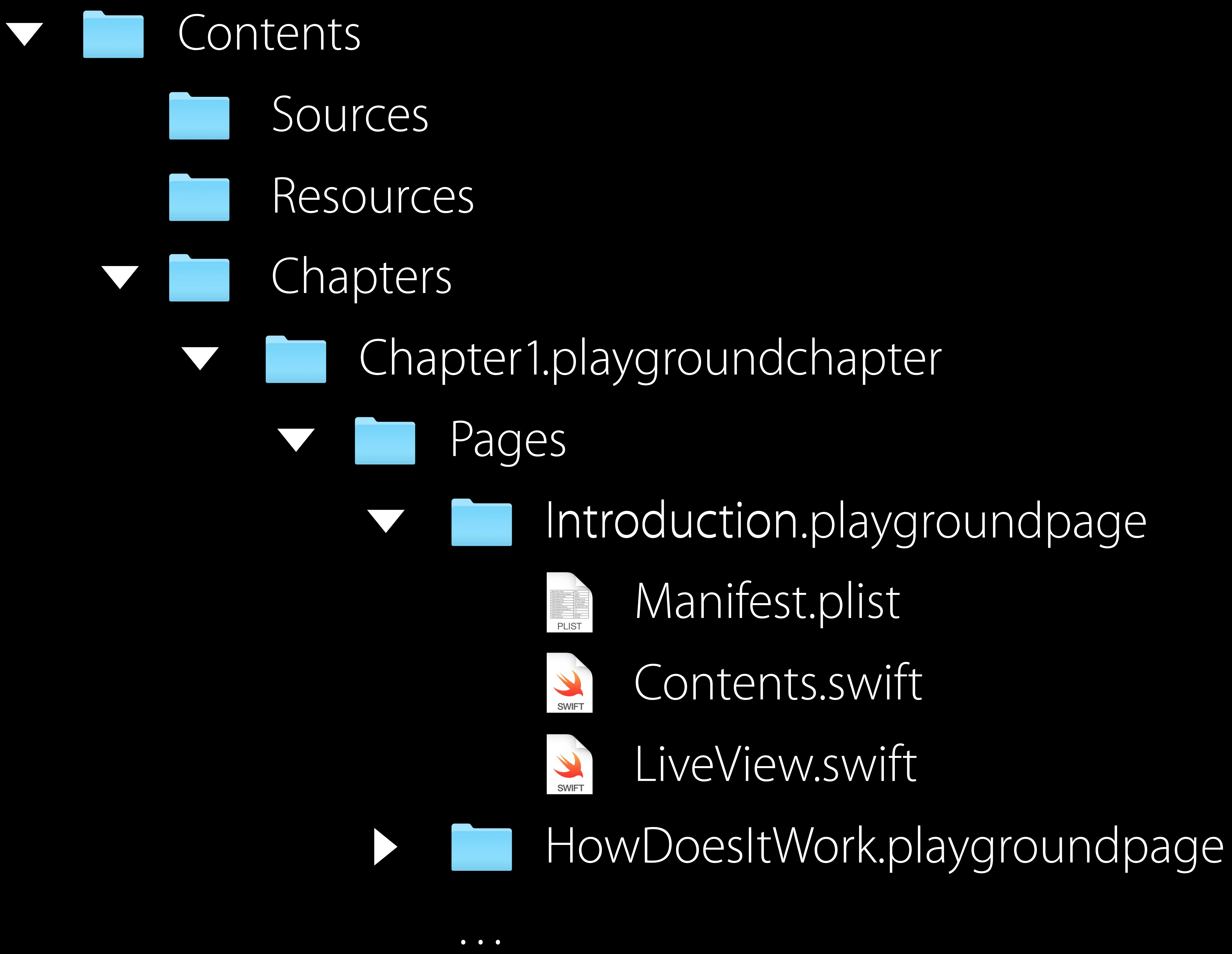
Tap *Run My Code* to send the string "Knock, knock" over to the Em in the live view.

You'll notice Em responds, "Who's there?". Continue the joke by replacing "Knock, knock" with "Boo!" and tap *Run My Code* again.

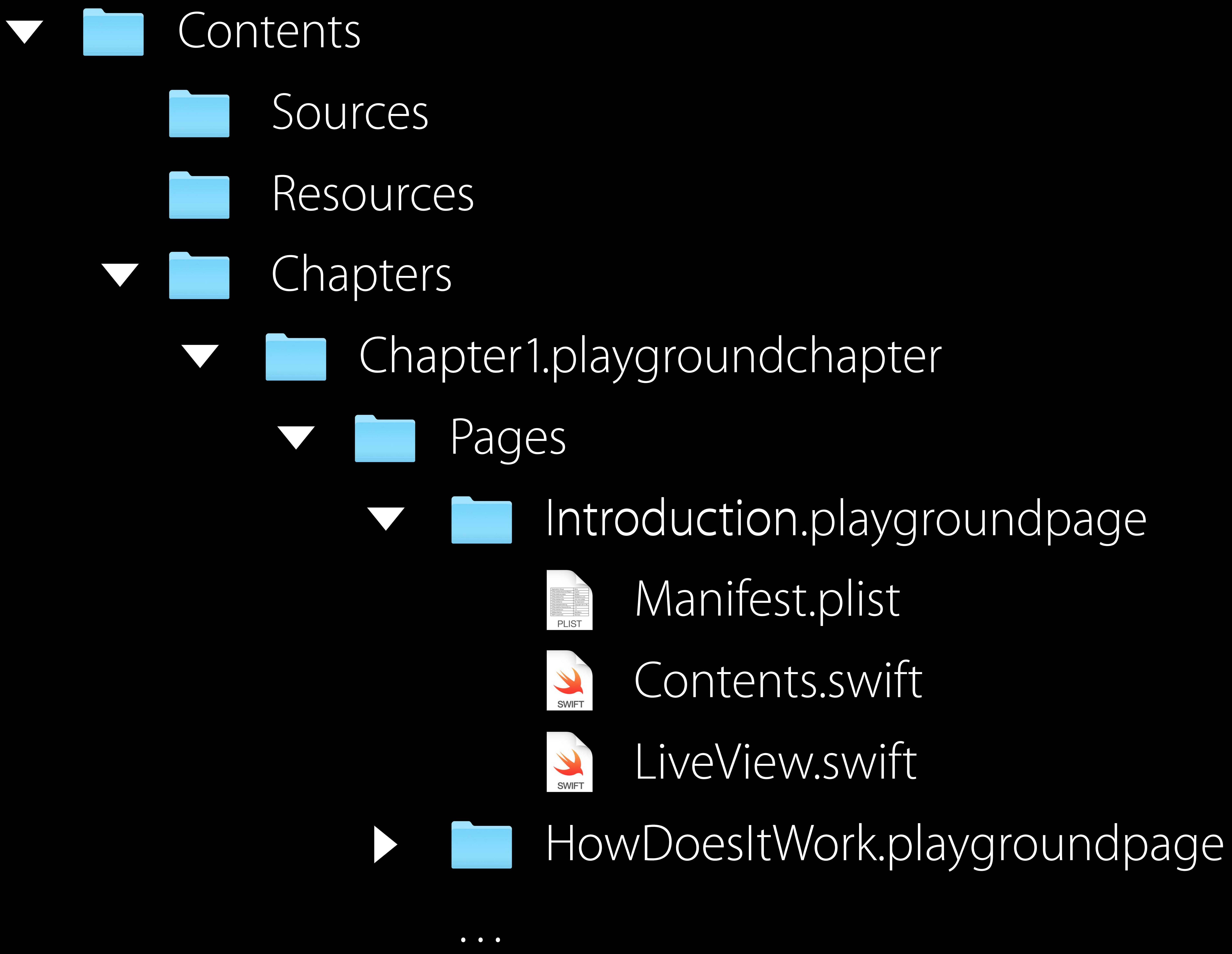
Em responds, "Boo! who?". Now, deliver the punchline, "Are you crying?".

When you're ready, continue to the next page to see how this `say(...)` function works.

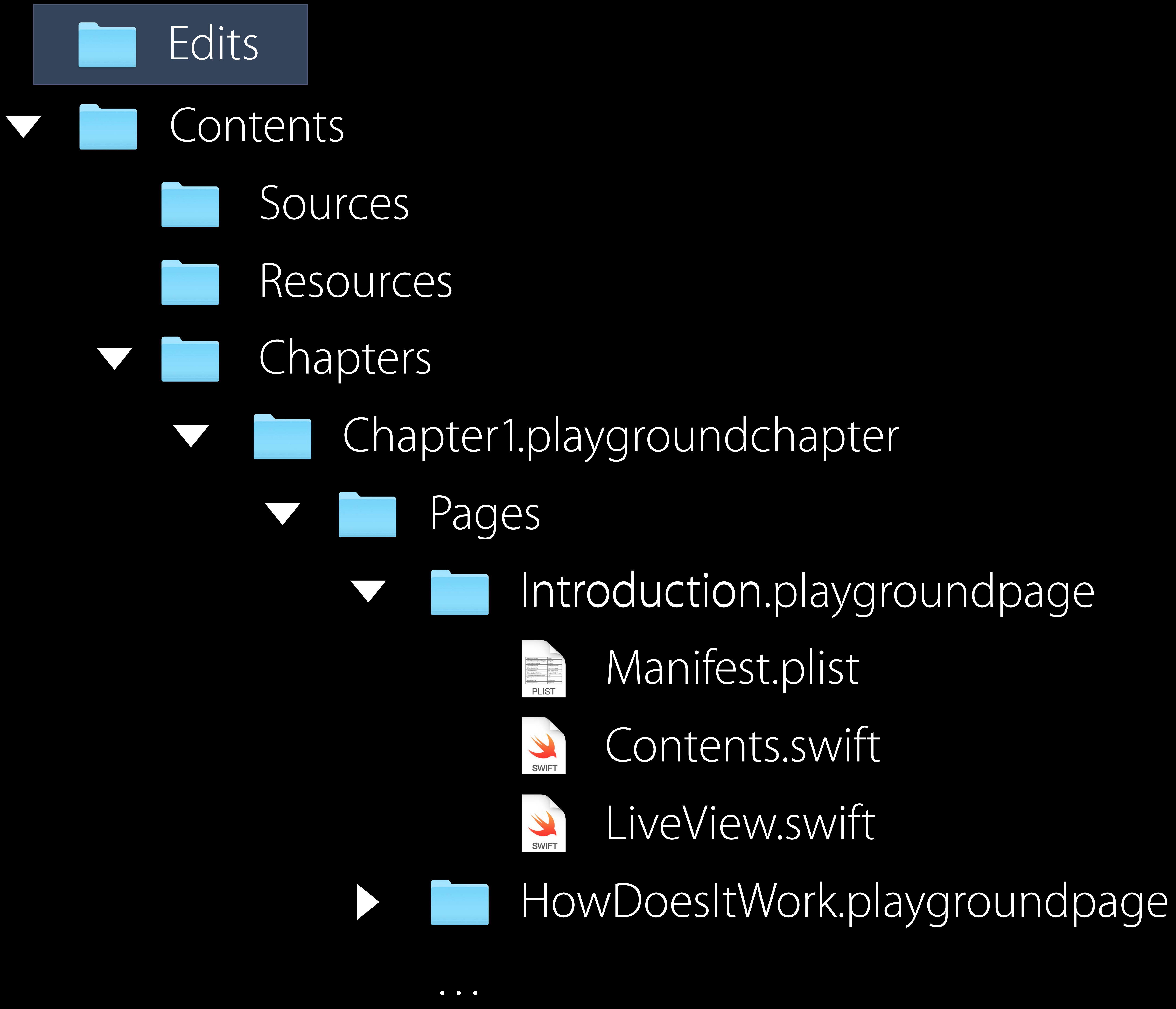
```
say("Knock, knock!")
```



```
Contents
    Sources
    Resources
    Chapters
        Chapter1.playgroundchapter
            Pages
                Introduction.playgroundpage
                    Manifest.plist
                    Contents.swift
                    LiveView.swift
                HowDoesItWork.playgroundpage
                ...
```



```
▼ Contents
    Sources
    Resources
  ▼ Chapters
    ▼ Chapter1.playgroundchapter
      ▼ Pages
        ▼ Introduction.playgroundpage
            Manifest.plist
            Contents.swift
            LiveView.swift
        ▶ HowDoesItWork.playgroundpage
          ...
```

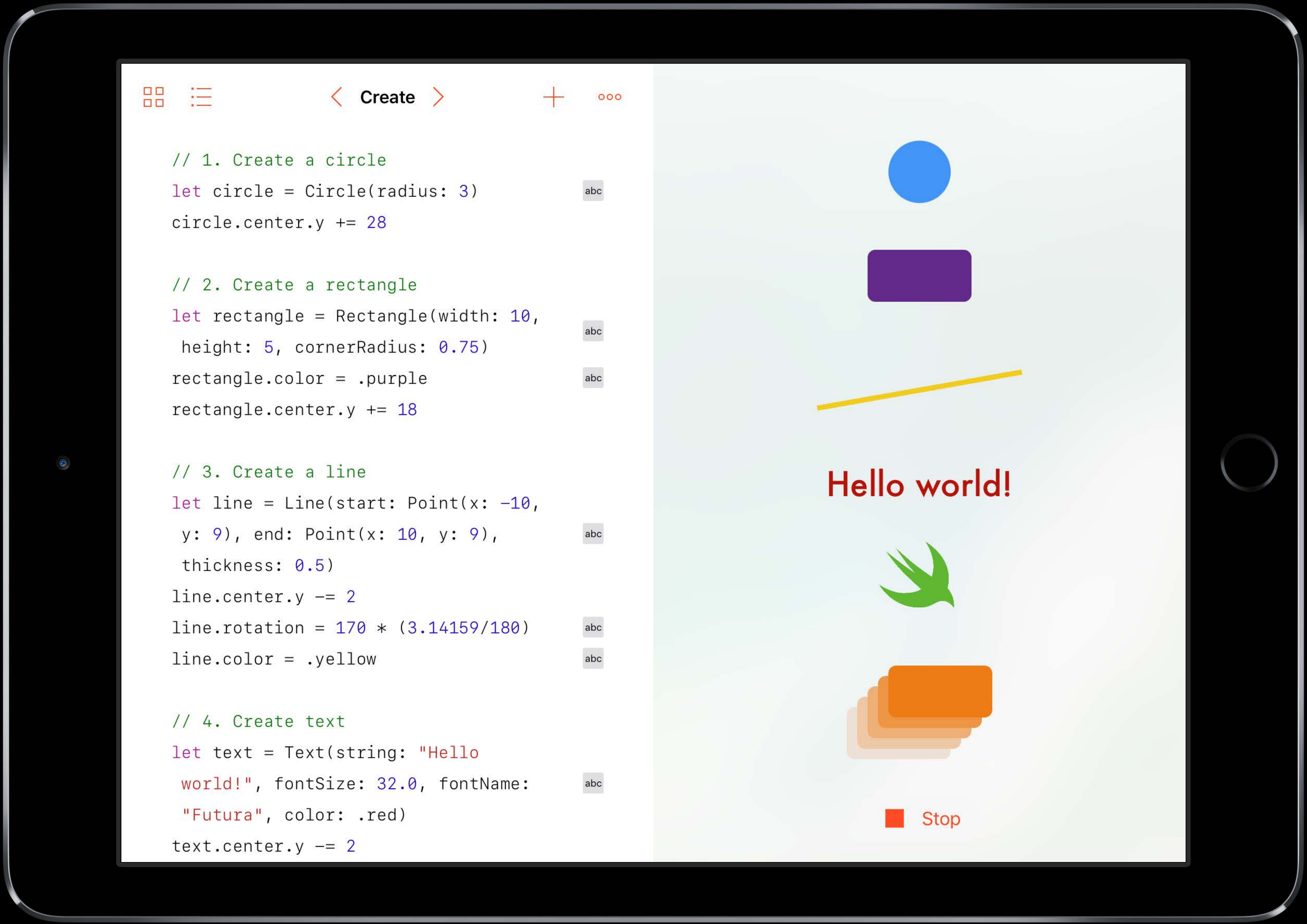


```
Edits
▼ Contents
    Sources
    Resources
  ▼ Chapters
    ▼ Chapter1.playgroundchapter
      ▼ Pages
        ▼ Introduction.playgroundpage
            Manifest.plist
            Contents.swift
            LiveView.swift
        ▶ HowDoesItWork.playgroundpage
          . . .
```

BOOK

developer.apple.com

BOOK



```
// 1. Create a circle
let circle = Circle(radius: 3)
circle.center.y += 28

// 2. Create a rectangle
let rectangle = Rectangle(width: 10,
 height: 5, cornerRadius: 0.75)
rectangle.color = .purple
rectangle.center.y += 18

// 3. Create a line
let line = Line(start: Point(x: -10,
 y: 9), end: Point(x: 10, y: 9),
 thickness: 0.5)
line.center.y -= 2
line.rotation = 170 * (3.14159/180)
line.color = .yellow

// 4. Create text
let text = Text(string: "Hello
 world!", fontSize: 32.0, fontName:
 "Futura", color: .red)
text.center.y -= 2
```

Hello world!

Demo

Growing and Exploring

Izzy Fraimow

Playgrounds Engineer

Swift Playgrounds

Summary

Swift Playgrounds

Summary

Touch-focused experience for experimenting with Swift

Swift Playgrounds

Summary

Touch-focused experience for experimenting with Swift

Rich new document format for creating engaging content

Swift Playgrounds

Summary

Touch-focused experience for experimenting with Swift

Rich new document format for creating engaging content

Powerful access to iOS SDK

Playgrounds

More Information

https://developer.apple.com/wwdc16/408

Related Sessions

Keynote	Bill Graham	Monday 10:00AM
Platforms State of the Union	Bill Graham	Monday 2:30PM
What’s New in Swift	Presidio	Tuesday 9:00AM

Labs

Swift Open Hours	Developer Tools Lab A	Tuesday 12:00PM
Swift Open Hours	Developer Tools Lab A	Wednesday–Friday 9:00AM
Creating Content for Swift Playgrounds	Dev Tools Lab C	Wednesday 12:00PM
Xcode Open Hours	Developer Tools Labs B	Wednesday 3:00PM
Xcode Open Hours	Developer Tools Labs C	Thursday 9:00AM
Creating Content for Swift Playgrounds	Dev Tools Lab C	Friday 12:00PM

WWDC16