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
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 Cards
Issuing Commands

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.

```swift
moveForward()
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.

```swift
moveForward()
movesForward()
movesForward()
movesForward()
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)
- \( r \) = the second circle's radius
- \( d \) = distance from center of second circle, where to place the pen
- \( s \) = scale = zoom factor

Try changing each of these values to see what happens.

```javascript
var c = Spiral({
  R: 5,
  radius: 3,
  d: 5
});
```

c.circleColor = 0x0000FF

c.lineColor = 0xFF0000

There are many options you can set to modify how the spiral is drawn. A few are listed below.
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

```javascript
var c = new Spiral(R: 5,
                     radius: 3,
                     d: 5)
c.circleColor = #ff0000
c.lineColor = #00ff00
```

There are many options you can set to modify how the circle is drawn. 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
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.

```swift
moveForward()
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.

```swift
moveForward()
movingForward() 
movingForward() 
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.

```swift
moveForward()
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.

```swift
moveForward()
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.

```swift
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()
movesForward()
movesForward()
movesForward()
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.

```swift
moveForward()
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()
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.

```swift
moveForward()
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.

```swift
moveForward()
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()
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.

```swift
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.

```swift
moveForward()
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.

```swift
moveForward()
mov```
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.

```swift
moveForward()
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.

```swift
moveForward()
movesForward()
movesForward()
movesForward()
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()
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.

```swift
moveForward()
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.

```swift
moveForward()
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.

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

Run My Code
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()
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.

```swift
moveForward()
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()
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()
Issuing Commands

Your character, Byte, loves to collect gems. In this first part, Byte needs your help to collect a gem. Use the commands to move Byte to the gem.

1. Look for the gem in the sky.
2. Enter the correct commands:
   - `moveForward()` and `collectGem()`.
3. Tap Run My Code.

Example commands:
- `moveForward()`
- `moveForward()`
- `moveForward()`
- `collectGem()`

Cheat Sheet:

- `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` creates a code path based on value.
- `func` creates a function.
Your character, Byte, loves gem collecting. In this first part of the quest, you must use commands to move Byte to a gem.

1. Look for the gem in the grass.
2. Enter the correct commands:
   moveForward() and:
3. Tap Run My Code.

- **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**
  - Creates a code path based on value.
Snippets

Your character, Byte, loves it alone. In this first part, you will need to move Byte using various commands.

1. Look for the gem in
2. Enter the correct command
3. Tap Run My Code.

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

- **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

Run My Code Hint
Your character, Byte, loves to explore. In this first part, let's use some commands to move Byte to the gem.

1. Look for the gem in the image.
2. Enter the correct commands:
   - moveForward()
   - moveForward()
   - moveForward()
   - collectGem()
3. Tap Run My Code.

Here are some commands you can use:

- **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**: Creates a code path based on value.
Your character, Byte, loves to collect gems. In this first part, you will learn how to use commands to move Byte and collect a gem.

1. Look for the gem in the sea.
2. Enter the correct code:
   - `moveForward()` and `collectGem()`
3. Tap `Run My Code` to see the results.

Here are some basic commands:

- **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

Use these commands to guide Byte and collect the gem.
Your character, Byte, loves to move around by himself. In this first part, you need to issue commands to move Byte and collect gems.

1. Look for the gem in the grass.
2. Enter the correct command: `moveForward()` and press Run My Code.
3. Tap Run My Code.

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

### Keywords

- **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**: Creates a code path based on value.
Your character, Byte, loves the game. In this first quest, Byte needs to use commands to move Byte to a gem.

1. Look for the gem in the terrain.
2. Enter the correct command: moveForward() and collectGem()反复使用。
3. Tap Run My Code.

- `let` creates a variable, which can 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` creates a code path based on value.
Your character, Byte, loves to collect gems. In this first puzzle, you need to use

**commands** to move Byte across the gem.

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

```
moveForward()
movingForward()
movingForward()
collectGem()
```
Your character, Byte, loves to collect gems. In this first puzzle, you need to use commands to move Byte across the puzzle.

1. Look for the gem in the puzzle.
2. Enter the correct commands to move Byte.
3. Tap Run My Code.

```javascript
moveForward();
moveForward();
moveForward();
collectGem();
```
Demo

Using Swift Playgrounds
Authoring for Swift Playgrounds

Jonathan Penn
Playgrounds Engineer
Chapters / Pages

Chapter: Commands
- Introduction
- Issuing Commands
- ...

Chapter: Functions
- Introduction
- Composing a New Behavior
- ...
Cutscenes

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

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

```javascript
moveForward()
```
Glossary

Finding and Fixing Bugs

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**

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

**Goal:** Find the bugs and fix them.

When you write code, a mistake that keeps you from seeing how it's supposed to work is called a bug, and debugging is the process of finding and fixing bugs.

The code below contains some bugs. 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()
<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
<th>Introduced In</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td>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.</td>
<td>Algorithms</td>
</tr>
<tr>
<td>argument</td>
<td>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.</td>
<td>Parameters</td>
</tr>
<tr>
<td>arithmetic operator</td>
<td>A symbol, such as +, -, * or /, that performs a basic mathematical operation on one or more numbers. For example, 42 / 7 and 17 - 9 use arithmetic operators.</td>
<td>Variables</td>
</tr>
<tr>
<td>assignment</td>
<td>An action that sets the value of a variable or constant.</td>
<td>Variables</td>
</tr>
<tr>
<td>assignment operator</td>
<td>The <code>=</code> symbol used to set the value of a variable. For example, greeting = &quot;hello&quot; sets the value of greeting to &quot;hello&quot;.</td>
<td>Variables</td>
</tr>
<tr>
<td>Boolean</td>
<td>A type that has a value of either true or false. For example, 9 &lt; 7 returns a Boolean value of false.</td>
<td>Conditional Code</td>
</tr>
<tr>
<td>bug</td>
<td>An error in code that prevents a program from running as expected.</td>
<td>Commands</td>
</tr>
<tr>
<td>call</td>
<td>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.</td>
<td>Functions</td>
</tr>
<tr>
<td>coding</td>
<td>The act of composing commands, code structures, and algorithms to create a computer program.</td>
<td>Commands</td>
</tr>
</tbody>
</table>
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.

```javascript
func FuncName() {
    // Add commands to your function
}
```

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

```plaintext
func FunctionName() {
    Add commands to your function
}
```

Tap to enter 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 can add a new function to handle each gem-
and you can name that function anything you’d like. You can use the same name as a switch, set its name, just like all the other functions you’ve defined so far.

```javascript
// Define your function
function yourSetupFunction() {
  // Add commands to your function
}

// Include your function here
```

```javascript
// #hidden-code
yourSetupFunction();
// #end-hidden-code
```
Configurable Code Completion

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

```cpp
//#-code-completion(everything, hide)
//#-code-completion(currentmodule, show)
//#-code-completion(identifier, show, moveForward(), turnLeft())
```
"Always-On" Live View

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

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.

Tap to enter code
Hints

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.

```swift
moveForward()
movwFoward()
movwFoward()
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.

Next Page
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.

```swift
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

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
// 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.

```swift
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()
movewForward()
movewForward()
collectGem()
```
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.

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.

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

Cancel  Reset
Resettable
Resettable
Resettable
Playground Book Package Format

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 commands to make Byte move across the puzzle world and collect a gem.
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.

```swift
say("Knock, knock!")
```
MyFirst.playgroundbook
Contents
- Sources
- Resources
- Chapters
  - Chapter1.playgroundchapter
  - Pages
    - Introduction.playgroundpage
      - Manifest.plist
      - Contents.swift
      - LiveView.swift
    - HowDoesItWork.playgroundpage
      ...

Resources

Sources

Chapter1.playgroundchapter

Pages

Introduction.playgroundpage

Manifest.plist

Contents.swift

LiveView.swift

HowDoesItWork.playgroundpage

...

Manifest.plist

Contents.swift

LiveView.swift

HowDoesItWork.playgroundpage

...

Resources

Sources
## Page Manifest

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>▼ Root</td>
<td>Dictionary</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>String</td>
<td>Introduction</td>
</tr>
<tr>
<td>LiveViewMode</td>
<td>String</td>
<td>VisibleByDefault</td>
</tr>
<tr>
<td>PosterReference</td>
<td>String</td>
<td>LiveViewPoster.png</td>
</tr>
<tr>
<td>LiveViewEdgeToEdge</td>
<td>Boolean</td>
<td>YES</td>
</tr>
<tr>
<td>PlaygroundLoggingMode</td>
<td>String</td>
<td>Off</td>
</tr>
</tbody>
</table>
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.

```swift
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.

```swift
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.

```swift
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.

```swift
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.

```swift
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.

```swift
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.

```swift
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!")
```
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.

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

  //#-end-editable-code

say("#Knock, knock!")
import PlaygroundSupport

func say(_ message: String) {
    let page = PlaygroundPage.current
    if let proxy = page.liveView as? PlaygroundRemoteLiveViewProxy {
        proxy.send(.string(message))
    }
}

say("<#Knock, knock!#>")
import PlaygroundSupport

func say(_ message: String) {
    let page = PlaygroundPage.current
    if let proxy = page.liveView as? PlaygroundRemoteLiveViewProxy {
        proxy.send(.string(message))
    }
}

say("<#Knock, knock#!>")
import PlaygroundSupport

func say(_ message: String) {
    let page = PlaygroundPage.current
    if let proxy = page.liveView as? PlaygroundRemoteLiveViewProxy {
        proxy.send(.string(message))
    }
}

say("<Knock, knock!>")
// 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

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

//#!-end-editable-code
// Contents.swift

/*:
Instructions about the page...
*/

import PlaygroundSupport

func say(_ message: String) {
    let page = PlaygroundPage.current
    if let proxy = page.liveView as? PlaygroundRemoteLiveViewProxy {
        proxy.send(.string(message))
    }
}

say("<Knock, knock!>")
// 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

//#-end-editable-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!")
```
// LiveView.swift

import PlaygroundSupport

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

let page = PlaygroundPage.current
page.liveView = FaceViewController()
// LiveView.swift

import PlaygroundSupport

let page = PlaygroundPage.current
page.liveView = FaceViewController()
// LiveView.swift

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

let page = PlaygroundPage.current

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

let page = PlaygroundPage.current
page.liveView = FaceViewController()
import PlaygroundSupport
let page = PlaygroundPage.current
page.liveView = FaceViewController()
import PlaygroundSupport
let page = PlaygroundPage.current
page.liveView = FaceViewController()
In main process
Only active while running

import PlaygroundSupport
let page = PlaygroundPage.current
page.liveView = FaceViewController()
In main process
Only active while running

Separate process
In main process
Only active while running

Separate process
Running all the time
import PlaygroundSupport

let page = PlaygroundPage.current

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

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

let page = PlaygroundPage.current

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

let page = PlaygroundPage.current

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

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

Contents.swift  Live View Proxy  FaceViewController
Sending to the Live View Process

- Contents.swift
- Live View Proxy
- FaceViewController
Sending to the Live View Process

Contents.swift → Live View Proxy → FaceViewController
extension FaceViewController: PlaygroundLiveViewMessageHandler {
    public func receive(_ message: PlaygroundValue) {
        if case let .string(text) = message {
            processConversationLine(text)
        }
    }
}
extension FaceViewController: PlaygroundLiveViewMessageHandler {
    public func receive(_ message: PlaygroundValue) {
        if case let .string(text) = message {
            processConversationLine(text)
        }
    }
}
extension FaceViewController: PlaygroundLiveViewMessageHandler {

    public func receive(_ message: PlaygroundValue) {
        if case let .string(text) = message {
            processConversationLine(text)
        }
    }
}
extension FaceViewController: PlaygroundLiveViewMessageHandler {
    public func receive(_ message: PlaygroundValue) {
        if case let .string(text) = message {
            processConversationLine(text)
        }
    }
}
extension FaceViewController: PlaygroundLiveViewMessageHandler {
    public func receive(_: message: PlaygroundValue) {
        if case let .string(text) = message {
            processConversationLine(text)
        }
    }
}
Sending from the Live View Process

- Contents.swift
- Live View Proxy
- FaceViewController
Sending from the Live View Process

Contents.swift → Live View Proxy → FaceViewController

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

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

    public func tapped() {
        let message: PlaygroundValue = .string("Hello!")
        send(message)
    }
}
Sending from the Live View Process

Contents.swift  Live View Proxy  FaceViewController
Sending from the Live View Process

Contents.swift

Live View Proxy

FaceViewController
Sending from the Live View Process

Contents.swift

Live View Proxy

FaceViewController
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
```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
```
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
```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
```
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
```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
```
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
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
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

Contents.swift

Live View Proxy

FaceViewController
Sending from the Live View Process

- Contents.swift
- Live View Proxy
- FaceViewController
Sending from the Live View Process

Contents.swift  Live View Proxy  FaceViewController
// 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)
}
// 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!"
}
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.

```swift
say("Knock, knock!")
```
// 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.x = 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
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

<table>
<thead>
<tr>
<th>Session</th>
<th>Speaker</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keynote</td>
<td>Bill Graham</td>
<td>Monday 10:00AM</td>
</tr>
<tr>
<td>Platforms State of the Union</td>
<td>Bill Graham</td>
<td>Monday 2:30PM</td>
</tr>
<tr>
<td>What’s New in Swift</td>
<td>Presidio</td>
<td>Tuesday 9:00AM</td>
</tr>
<tr>
<td>Event</td>
<td>Location</td>
<td>Time</td>
</tr>
<tr>
<td>------------------------------</td>
<td>------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>Swift Open Hours</td>
<td>Lab A</td>
<td>Tuesday 12:00PM</td>
</tr>
<tr>
<td>Swift Open Hours</td>
<td>Lab A</td>
<td>Wednesday–Friday 9:00AM</td>
</tr>
<tr>
<td>Creating Content for Swift Playgrounds</td>
<td>Lab C</td>
<td>Wednesday 12:00PM</td>
</tr>
<tr>
<td>Xcode Open Hours</td>
<td>Labs B</td>
<td>Wednesday 3:00PM</td>
</tr>
<tr>
<td>Xcode Open Hours</td>
<td>Labs C</td>
<td>Thursday 9:00AM</td>
</tr>
<tr>
<td>Creating Content for Swift Playgrounds</td>
<td>Lab C</td>
<td>Friday 12:00PM</td>
</tr>
</tbody>
</table>