Teaching with Swift Playgrounds
Session 416

Bill Dudney, Table Inverter
Elizabeth Salazar, Storyteller
Teaching
Goal
Goal
Mathematics
Mathematics

$$ax^2 + bx + c = 0$$

Polynomials

$$\sin(x)$$

Trigonometry

$$e^x$$

Exponentials

$$\log_{10}(x)$$

Logarithms
When will I use this?!
Start with a question.
Audience
Typical Teaching—Syntax
Goal: Use a for loop to repeat a sequence of commands.

In this puzzle, you must collect four gems that are located in the same relative locations around a square. You’ll create a loop that repeats the code below for each of the sides to solve the entire puzzle.

1. Drag a for loop from the code library, then drop it above the existing code.
2. Tap the bottom curly brace to select the loop.
3. Tap and hold that curly brace, then drag it downward to pull the existing code into the loop.

```python
for i in 1 ... 4 {
    moveForward()
    collectGem()
    moveForward()
    moveForward()
    moveForward()
    turnRight()
}
```
Typical Teaching—
Algorithms and Data Structure
Fermat Primality Test  Biconnectivity  Dice’s Coefficient  Shadow Volume
Interval Halving  Laplacian Smoothing  Simplex Noise  Binary Search
Double Metaphone  Soundex  Heap Sort  Axis Aligned Bounding Boxes
Quick Sort  Gradient Descent  Hashing  Recursion  Depth First Search
Gibbs Sampling  Rasterisation  Spatial Partitioning  Gauss-Legendre
Gouraud Shading  Hamming distance  Newton’s Method
Goal: Use a for loop to repeat a sequence of commands.

In this puzzle, you must collect four gems that are located in the same relative locations around a square. You’ll create a loop that repeats the code below for each of the sides to solve the entire puzzle.

1. Drag a for loop from the code library, then drop it above the existing code.
2. Tap the bottom curly brace to select the loop.
3. Tap and hold that curly brace, then drag it downward to pull the existing code into the loop.

```plaintext
for i in 1 ... 4 {
    moveForward()
    collectGem()
    moveForward()
    moveForward()
    moveForward()
    turnRight()
}
```
Challenge: Implement the most efficient algorithm to collect the gems and activate the switches.

For the last challenge of Learn to Code 1, you'll test your algorithm design skills. There are many different algorithms you could use to solve the puzzle, and many different ways to structure your code.

If you're not able to find a solution right away, that's okay! Coding often requires trying different solutions to a problem until you find the one that works best. When you're ready, you can move on to Learn to Code 2.

Tap to enter code
The easy parts.
Session Table View Controller

Session Table View Cell

Session
Open GL Pipeline

1. Primitive Processing
2. Vertex Shader
3. Primitive Assembly
4. Rasterizer
5. Fragment Shader

6. Depth Stencil
7. Color Buffer Blend
8. Dither
9. Frame Buffer
Fun
Fun
Passion
Passion
Teaching with Swift Playgrounds
Teaching with Swift Playgrounds
Teaching
Designing

conditional conditional
#selector return #selector return
class #line false class #line false
operators static operators static
protocol do continue protocol do continue
default #if subscript default #if subscript
#sourceLocation self #sourceLocation self
continue #function continue #function
strings nil didSet strings nil didSet
#available try #available try
static as
types true
self else
true case
nil loop
class types
didSet public
Designing Swift Playgrounds

Elizabeth Salazar, Storyteller
What is a Swift Playground?
Goal: Use a for loop to repeat a sequence of commands.

To break down coding tasks, you wrote functions for repeated patterns. Now you’ll call one function multiple times using a loop. With a loop, you write your code once and enter the number of times to repeat it.

In this puzzle, there’s a gem in the same position in each row. You will collect the gems by following the same pattern multiple times. This is the perfect place for a loop!

1. Enter the solution for one row inside the curly braces.
2. Decide how many times to repeat the loop.
3. Tap the number placeholder and specify the number of repetitions.

```python
for i in 1 ... number {
  Tap to enter code
}
```
Goal: Use a for loop to repeat a sequence of commands.

To break down coding tasks, you wrote functions for repeated patterns. Now you’ll call one function multiple times using a loop. With a loop, you write your code once and enter the number of times to repeat it.

In this puzzle, there’s a gem in the same position in each row. You will collect the gems by following the same pattern multiple times. This is the perfect place for a loop!

1. Enter the solution for one row inside the curly braces.
2. Decide how many times to repeat the loop.
3. Tap the number placeholder and specify the number of repetitions.

```python
for i in 1 ... number
  Tap to enter code
```


Goal: Use a for loop to repeat a sequence of commands.

To break down coding tasks, you wrote functions for repeated patterns. Now you’ll call one function multiple times using a loop. With a loop, you write your code once and enter the number of times to repeat it.

In this puzzle, there’s a gem in the same position in each row. You will collect the gems by following the same pattern multiple times. This is the perfect place for a loop!

1. Enter the solution for one row inside the curly braces.
2. Decide how many times to repeat the loop.
3. Tap the number placeholder and specify the number of repetitions.

```python
for i in 1 ... number:
    Tap to enter code
```
Goal: Use a for loop to repeat a sequence of commands.

To break down coding tasks, you wrote functions for repeated patterns. Now you'll call one function multiple times using a loop. With a loop, you write your code once and enter the number of times to repeat it.

In this puzzle, there's a gem in the same position in each row. You will collect the gems by following the same pattern multiple times. This is the perfect place for a loop!

1. Enter the solution for one row inside the curly braces.
2. Decide how many times to repeat the loop.
3. Tap the number placeholder and specify the number of repetitions.

```swift
for i in 1 ... number {
    // Tap to enter code
}
```
Goal: Use a for loop to repeat a sequence of commands.

To break down coding tasks, you wrote functions for repeated patterns. Now you’ll call one function multiple times using a loop. With a loop, you write your code once and enter the number of times to repeat it.

In this puzzle, there’s a gem in the same position in each row. You will collect the gems by following the same pattern multiple times. This is the perfect place for a loop!

1. Enter the solution for one row inside the curly braces.
2. Decide how many times to repeat the loop.
3. Tap the number placeholder and specify the number of repetitions.

```python
for i in 1 ... number {
    Tap to enter code
}
```
Goal: Use a for loop to repeat a sequence of commands.

To break down coding tasks, you wrote functions for repeated patterns. Now you’ll call one function multiple times using a loop. With a loop, you write your code once and enter the number of times to repeat it.

In this puzzle, there’s a gem in the same position in each row. You will collect the gems by following the same pattern multiple times. This is the perfect place for a loop!

1. Enter the solution for one row inside the curly braces.
2. Decide how many times to repeat the loop.
3. Tap the number placeholder and specify the number of repetitions.

```python
for i in 1 ... number:
    Tap to enter code
```
A few other features...
In Learn to Code 3, you’ll use code to fill my universe with animated images and text.
Approaching the reference desk, you clear your throat, and Mr. Nefarian eyes you suspiciously.

“Yes? Can I help you?”

“Please, do you have the answer?” you ask.

Mr. Nefarian looks at you blankly. “The answer to what?” he replies.

You're not sure what to do now—he must be "N," but he doesn't seem to know anything!

Maybe you're forgetting something from the message...

```javascript
// Give the password
let password: String = "password"
giveThePassword(password: password)
```
Approaching the reference desk, you clear your throat, and Mr. Nefarian eyes you suspiciously.

“Yes? Can I help you?”

“Please, do you have the answer?” you ask.

Mr. Nefarian looks at you blankly. “The answer to what?” he replies.

You’re not sure what to do now—he must be “N,” but he doesn’t seem to know anything!

Maybe you’re forgetting something from the message...

```swift
// Give the password
let password: String = “password”
giveThePassword(password: password)
```
Solutions

Goal: Use a for loop to repeat a sequence of commands.

In this puzzle, you must collect four gems that are located in the same relative locations around a square. You'll create a loop that repeats the code below for each of the sides to solve the entire puzzle.

1. Drag a for loop from the code library, then drop it above the existing code.
2. Tap the bottom curly brace to select the loop.
3. Tap and hold that curly brace, then drag it downward to pull the existing code into the loop.

```javascript
moveForward()
collectGem()
moveForward()
moveForward()
moveForward()
moveForward()
turnRight()
```
Solutions

Goal: Use a for loop to repeat a sequence of commands.

In this puzzle, you must collect four gems that are located in the same relative locations around a square. You'll create a loop that repeats the code below for each of the sides to solve the entire puzzle.

1. Drag a for loop from the code library, then drop it above the existing code.
2. Tap the bottom curly brace to select the loop.
3. Tap and hold that curly brace, then drag it downward to pull the existing code into the loop.

moveForward()
collectGem()
moveForward()
moveForward()
moveForward()
turnRight()
Goal: Create an ellipse.

You now have all the ingredients you need to create trochoids and cycloids, of both the hypo- and epi-varieties. Your final challenge is to figure out how to draw a very special shape—an ellipse.

Although they don't look as exciting as the spirals from the previous pages, ellipses are pretty amazing. All the planets in our solar system move around the sun in elliptical orbits! In fact, the moon, the sun itself, and all the stars you can see in the sky move along their own elliptical paths through the universe.

Are you ready to try making an ellipse of your own? Here's the recipe:

1. The track radius is exactly twice as long as the wheel radius.
2. An ellipse is a special type of hypotrochoid.
3. The spoke length is either greater than or less than the wheel radius.

Tip:

The ellipse you're seeing in the live view is just one possibility. Yours might be long and thin, or short and fat, or even vertical! Good luck!

// This needs to be \*twice as long\* as the wheelRadius.
Goal: Create an ellipse.

You now have all the ingredients you need to create trochoids and cycloids, of both the hypo- and epi-varieties. Your final challenge is to figure out how to draw a very special shape—an ellipse.

Although they don’t look as exciting as the spirals from the previous pages, ellipses are pretty amazing. All the planets in our solar system move around the sun in elliptical orbit! In fact, the moon, the sun itself, and all the stars you can see in the sky move along their own elliptical paths through the universe.

Are you ready to try making an ellipse of your own? Here’s the recipe:

1. The track radius is exactly twice as long as the wheel radius.
2. An ellipse is a special type of hypotrochoid.
3. The spoke length is either greater than or less than the wheel radius.

Tip:
The ellipse you’re seeing in the live view is just one possibility. Yours might be long and thin, or short and fat, or even vertical! Good luck!

// This needs to be *twice as long* as the wheelRadius.
Coding time?
To the drawing board! 🎨
Step 2
continue chain?

Step 4
animate()

Teach them what a spine is? maybe what 3D/2D graphics mean?

Vocab: model, shaders, geometry, bunch of other stuff...
What is the learner's goal?
What is the learner's goal?
Learner Goals
Learner Goals

Complete a task

"Collect 6 gems and flip all the switches."
Learner Goals

Complete a task
Experiment

"Try changing the radius to a negative number—see what happens!"
Learner Goals

Complete a task
Experiment
Practice a new skill

"Remember for loops?
Use one now for the fastest solution!"
Learner Goals

Complete a task
Experiment
Practice a new skill
Think about a concept

"Cryptography is the science of studying hidden writing."
Learner Goals

Complete a task
Experiment
Practice a new skill
Think about a concept
Create

"Now you have all the skills you need: create your own work of art in the LiveView!"
Prose Tools
Prose Tools—Glossary Entries
You'll need to write an algorithm that lets you keep your character moving efficiently around the puzzle world, picking up gems that appear.
You'll need to write an algorithm that lets you keep your character moving efficiently through the puzzle world, picking up gems that appear.
"A Substitution Cipher is one in which each letter of the message is substituted (or exchanged) for a different letter."
"A Substitution Cipher is one in which each letter of the message is substituted (or exchanged) for a different letter."
Here's a lot of prose, lots and lots of it oh yes I could go on like this forever. First I'll start making a point and giving you a bit of color and explanation.

Then I'll start the second point and etc etc etc.

Hello There

Here's the important bit!

newAwesomeFunction()

And now for the wrap-up and a bit more prose, yes, on we go. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
Here's a lot of prose, lots and lots of it oh yes I could go on like this forever. First I'll start making a point and giving you a bit of color and explanation.

Then I'll start the second point and etc etc etc.

Hello There

Hello's the important bit!

newAwesomeFunction()
Prose Tools—Shortcut Bar

Only include relevant choices
Prose Tools—Shortcut Bar

Only include relevant choices

Use concise method names
Prose Tools—Shortcut Bar

Only include relevant choices

Use concise method names

Provide common keywords
Who is your audience?
Who is your audience?
Goal: Set a background image for the scene.

Think of the scene as a bit like a scene in a movie, with graphics as actors. And just like in a movie, a scene can have a background. You can set a background image for the scene.

```javascript
// Setting the background image
scene.backgroundImage = image;
```

1. Write `scene.backgroundImage = image;` in the code, and then choose ![image] from the shortcut bar.

2. Choose an image for your background.

```
// Set the background image.
```

Tap to enter code
Complete a Task—Beginner

Goal: Set a background image for the scene.

Think of the scene as a bit like a scene in a movie, with graphics as actors. And just like in a movie, a scene can have a background. You can set a background image for the scene.

Setting the background image

```javascript
scene.backgroundImage = image; // image is the path to your image.
```

1. Write `scene.backgroundImage = image;` in the code, and then choose 📷 from the shortcut bar.
2. Choose an image for your background.

```javascript
// Set the background image.
```

Tap to enter code
Complete a Task—Beginner

**Goal:** Set a background image for the scene.

Think of the scene as a bit like a scene in a movie, with graphics as actors. And just like in a movie, a scene can have a background. You can set a background image for the scene.

```
Setting the background image

scene.backgroundColor = 
```

1. Write `scene.backgroundColor =` in the code, and then choose from the shortcut bar.

2. Choose an image for your background.

```javascript
// Set the background image.
```

Tap to enter code
Complete a Task—Beginner

**Goal:** Set a background image for the scene.

Think of the scene as a bit like a scene in a movie, with graphics as actors. And just like in a movie, a scene can have a background. You can set a background image for the scene.

```
scene.backgroundImage = Ⓟ
```

1. **Write** `scene.backgroundImage` in the code, and then choose 🖼️ from the shortcut bar.
2. Choose an image for your background.

// Set the background image.
Complete a Task—Beginner

**Goal:** Set a background image for the scene.

Think of the scene as a bit like a scene in a movie, with graphics as actors. And just like in a movie, a scene can have a background. You can set a background image for the scene.

```javascript
setting the background image
scene.backgroundImage = image;
```

1. Write `scene.backgroundImage = image;` in the code, and then choose the shortcut bar.

2. Choose an image for your background.

```javascript
// Set the background image.
```

Tap to enter code
Complete a Task—Advanced

Goal: Use a breadth-first search algorithm to escape the maze.

*Breadth-first search* is a type of pathfinding algorithm. In the name of this algorithm, *breadth* refers to the number of things it checks at each step in the search. When the algorithm looks at a tile, it also checks all four of its neighbors, searching a broad area to find a path.

1. Start by adding all neighbors of the starting tile to a queue.
2. Pop tiles from the queue to see what kind of tiles they are.
3. Each time you pop a tile from the queue, add all of that tile’s neighbors to the queue.

```swift
func findPath(in maze: Maze) {
    // Use the types, methods, and properties from the
    // introduction page to write your own maze-solving code.
}
```
Complete a Task—Advanced

**Goal:** Use a breadth-first search algorithm to escape the maze.

Breadth-first search is a type of pathfinding algorithm. In the name of this algorithm, breadth refers to the number of things it checks at each step in the search. When the algorithm looks at a tile, it also checks all four of its neighbors, searching a broad area to find a path.

1. Start by adding all neighbors of the starting tile to a queue.
2. Pop tiles from the queue to see what kind of tiles they are.
3. Each time you pop a tile from the queue, add all of that tile's neighbors to the queue.

```swift
func findPath(in maze: Maze) {
    // Use the types, methods, and properties from the introduction page to write your own maze-solving code.
}
```
**Complete a Task—Advanced**

**Goal:** Use a breadth-first search algorithm to escape the maze.

*Breadth-first search* is a type of pathfinding algorithm. In the name of this algorithm, *breadth* refers to the number of things it checks at each step in the search. When the algorithm looks at a tile, it also checks all four of its neighbors, searching a broad area to find a path.

1. Start by adding all neighbors of the starting tile to a queue.
2. Pop tiles from the queue to see what kind of tiles they are.
3. Each time you pop a tile from the queue, add *all* of that tile's neighbors to the queue.

```func findPath(in maze: Maze) {
    // Use the types, methods, and properties from the introduction page to write your own maze-solving code.
}
```
Complete a Task—Advanced

**Goal:** Use a breadth-first search algorithm to escape the maze.

*Breadth-first search* is a type of pathfinding algorithm. In the name of this algorithm, *breadth* refers to the number of things it checks at each step in the search. When the algorithm looks at a tile, it also checks all four of its neighbors, searching a broad area to find a path.

1. Start by adding all neighbors of the starting tile to a queue.
2. Pop tiles from the queue to see what kind of tiles they are.
3. Each time you pop a tile from the queue, add *all* of that tile's neighbors to the queue.

```swift
func findPath(in maze: Maze) {
    // Use the types, methods, and properties from the introduction page to write your own maze-solving code.
}
```
Think About a Concept
Blink: A Cell Simulator

Blink is a simulation that explores how a living cell reproduces or dies given a certain set of rules. Your goal is to understand the algorithms that run the simulation so that you can create your own version, with your own rules.

This playground is running a modified version of Conway's Game of Life, which presents cells reproducing and dying based upon the status of the 8 neighboring cells. You will see this simulation in the live view when you run the code.

The rules for this simulation are:

- Any living cell with fewer than two living neighbors dies.
- Any living cell with two or three living neighbors lives on.
- Any living cell with more than three living neighbors dies.
- Any dead cell with exactly three living neighbors becomes a living cell.

The cell simulator uses a loop to evaluate all cells on the grid. For each iteration of the loop, the rules are applied and a new generation of cells is created. Experiment with stepping through the simulation to watch this happen. On the next page, you'll explore modifying this algorithm.
Blink: A Cell Simulator

Blink is a simulation that explores how a living cell reproduces or dies given a certain set of rules. Your goal is to understand the algorithms that run the simulation so that you can create your own version, with your own rules.

This playground is running a modified version of Conway's Game of Life, which presents cells reproducing and dying based upon the status of the 8 neighboring cells. You will see this simulation in the live view when you run the code.

The rules for this simulation are:

- Any living cell with fewer than two living neighbors dies.
- Any living cell with two or three living neighbors lives on.
- Any living cell with more than three living neighbors dies.
- Any dead cell with exactly three living neighbors becomes a living cell.

The cell simulator uses a loop to evaluate all cells on the grid. For each iteration of the loop, the rules are applied and a new generation of cells is created.
Think About a Concept

Blink: A Cell Simulator

Blink is a simulation that explores how a living cell reproduces or dies given a certain set of rules. Your goal is to understand the algorithms that run the simulation so that you can create your own version, with your own rules.

This playground is running a modified version of Conway's Game of Life, which presents cells reproducing and dying based upon the status of the 8 neighboring cells. You will see this simulation in the live view when you run the code.

The rules for this simulation are:

- Any living cell with fewer than two living neighbors dies.
- Any living cell with two or three living neighbors lives on.
- Any living cell with more than three living neighbors dies.
- Any dead cell with exactly three living neighbors becomes a living cell.

The cell simulator uses a loop to evaluate all cells on the grid. For each iteration of the loop, the rules are applied and a new generation of cells is created.
Think About a Concept

Blink: A Cell Simulator

Blink is a simulation that explores how a living cell reproduces or dies given a certain set of rules. Your goal is to understand the algorithms that run the simulation so that you can create your own version, with your own rules.

This playground is running a modified version of Conway’s Game of Life, which presents cells reproducing and dying based upon the status of the 8 neighboring cells. You will see this simulation in the live view when you run the code.

The rules for this simulation are:

- Any living cell with fewer than two living neighbors dies.
- Any living cell with two or three living neighbors lives on.
- Any living cell with more than three living neighbors dies.
- Any dead cell with exactly three living neighbors becomes a living cell.

The cell simulator uses a loop to evaluate all cells on the grid. For each iteration of the loop, the rules are applied and a new generation of cells is created.
Think About a Concept

Blink: A Cell Simulator

Blink is a simulation that explores how a living cell reproduces or dies given a certain set of rules. Your goal is to understand the algorithms that run the simulation so that you can create your own version, with your own rules.

This playground is running a modified version of Conway's Game of Life, which presents cells reproducing and dying based upon the status of the 8 neighboring cells. You will see this simulation in the live view when you run the code.

The rules for this simulation are:

- Any living cell with fewer than two living neighbors dies.
- Any living cell with two or three living neighbors lives on.
- Any living cell with more than three living neighbors dies.
- Any dead cell with exactly three living neighbors becomes a living cell.

The cell simulator uses a loop to evaluate all cells on the grid. For each iteration of the loop, the rules are applied and a new generation of cells is created.
Think About a Concept

This playground is running a modified version of Conway's Game of Life, which presents cells reproducing and dying based upon the status of the 8 neighboring cells. You will see this simulation in the live view when you run the code.

The rules for this simulation are:

- Any living cell with fewer than two living neighbors dies.
- Any living cell with two or three living neighbors lives on.
- Any living cell with more than three living neighbors dies.
- Any dead cell with exactly three living neighbors becomes a living cell.

The cell simulator uses a loop to evaluate all cells on the grid. For each iteration of the loop, the rules are applied and a new generation of cells is created. Experiment with stepping through the simulation to watch this happen. On the next page, you'll explore modifying this algorithm.
Think About a Concept

This playground is running a modified version of Conway’s Game of Life, which presents cells reproducing and dying based upon the status of the 8 neighboring cells. You will see this simulation in the live view when you run the code.

The rules for this simulation are:

- Any living cell with fewer than two living neighbors dies.
- Any living cell with two or three living neighbors lives on.
- Any living cell with more than three living neighbors dies.
- Any dead cell with exactly three living neighbors becomes a living cell.

The cell simulator uses a loop to evaluate all cells on the grid. For each iteration of the loop, the rules are applied and a new generation of cells is created.

Experiment with stepping through the simulation to watch this happen. On the next page, you’ll explore modifying this algorithm.
Is it coding time yet?
Bring your passion.
Bring your passion.
conditional conditional
#selector return #selector return
class #line false class #line false
operators static operators static
protocol do continue protocol do continue
default #if subscript default #if subscript
#sourceLocation self #sourceLocation self
continue #function continue #function
strings nil didSet strings nil didSet
#available try #available try
static as
  types true
  self else
  true case
  nil loop
  class types
didSet public
Designing

Teaching

conditional
#selector return
class #line false
operators static
protocol do continue
default #if subscript
#sourceLocation self
continue #function
strings nil didSet
#available try
as
true
else
case
loop
types
public
Finally, time to code.
Goal
Goal

Audience
Goal  Passion  Audience
Now go forth, and teach!
More Information

### Related Sessions

<table>
<thead>
<tr>
<th>Session</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>SceneKit in Swift Playgrounds</td>
<td>WWDC 2017</td>
</tr>
<tr>
<td>What’s New in Swift Playgrounds</td>
<td>WWDC 2017</td>
</tr>
<tr>
<td>Localizing Content for Swift Playgrounds</td>
<td>WWDC 2017</td>
</tr>
</tbody>
</table>