

놀아요

# Swift Playgrounds

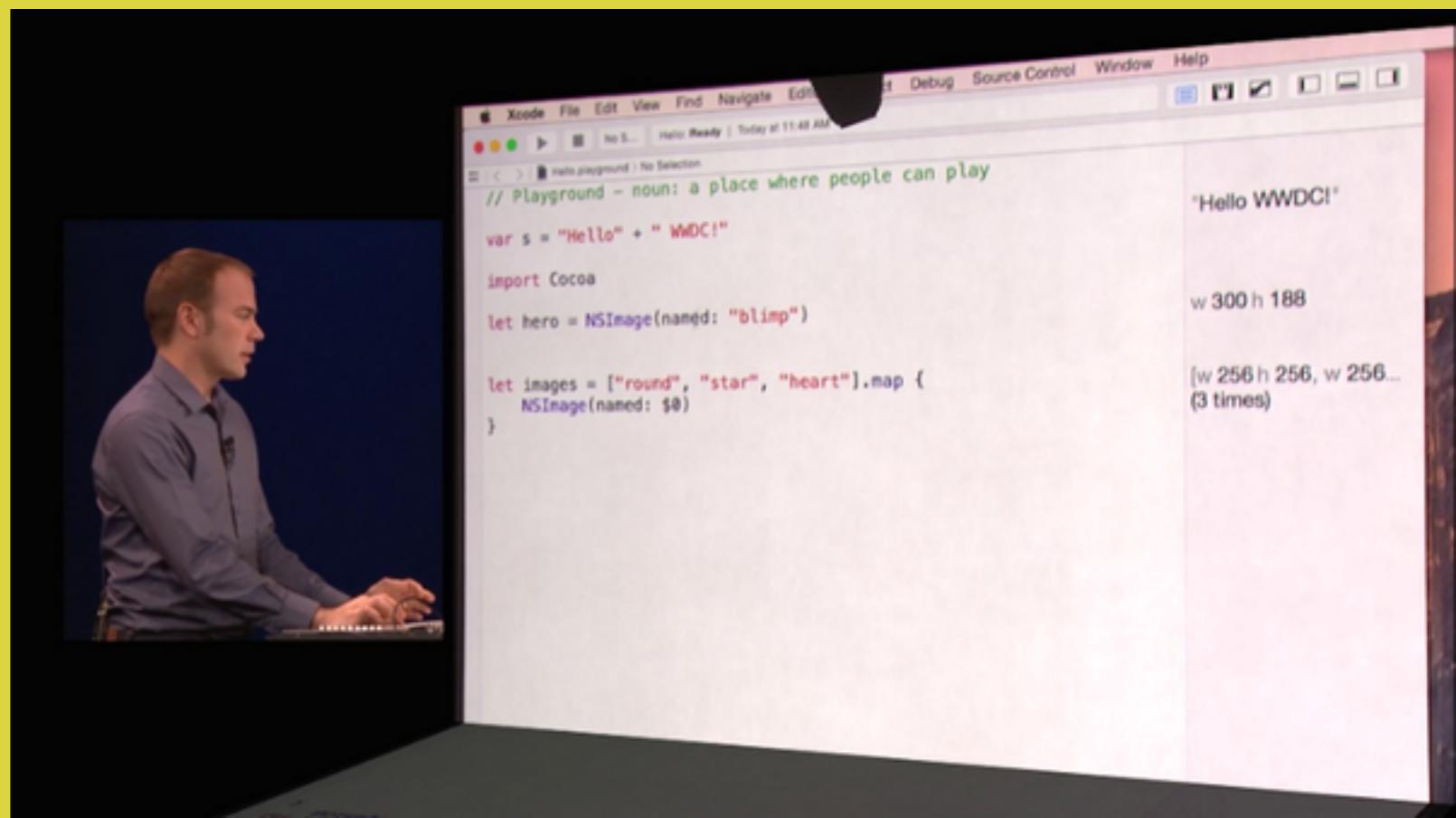
발표자 : [github.com/wookay](https://github.com/wookay)

일시 : 2014년 7월 5일 토요일 1시~6시

장소 : 판교 H스퀘어 N동 4층 - NHN NEXT 홀

# Chris Lattner - Playgrounds Demo

Apple SWIFT programming language demo at WWDC 2014  
[http://youtu.be/I62x80q\\_QP4?t=3m5s](http://youtu.be/I62x80q_QP4?t=3m5s)



# 인물과 배경

- Chris Lattner - LLVM
- Bret Victor
- Chris Granger - Light Table

# Chris Lattner

Chris Lattner's Homepage

<http://nondot.org/sabre/>

- The Xcode Playgrounds feature and REPL were a personal passion of mine, to make programming more interactive and approachable. The Xcode and LLDB teams have done a phenomenal job turning crazy ideas into something truly great. Playgrounds were heavily influenced by **Bret Victor**'s ideas, by **Light Table** and by many other interactive systems. I hope that by making programming more approachable and fun, we'll appeal to the next generation of programmers and to help redefine how Computer Science is taught.



# Bret Victor - Inventing on Principle

Friday, January 20th, 2012

<http://vimeo.com/36579366>

**CUSEC (Canadian University Software Engineering Conference)**

<http://2012.cusec.net>



# Bret Victor - Inventing on Principle (2)



```
        this.yVelocity = -25;
    }

    this.yVelocity += 100 * dt;
    this.y += this.yVelocity * dt;

    var hitInfo = this.hitTest();

    this.jumping = (hitInfo.bumpedY >= this.y);
    if (hitInfo.bumpedY != this.y) {
        this.yVelocity = 0;
    }

    this.x = hitInfo.bumpedX;
    this.y = hitInfo.bumpedY;

    this.running = (this.x != oldX) && !this.jumping;
    this.facingLeft = (this.x < oldX) || (this.x == oldX &&
        this.y > oldY);

    if (hitInfo.hitObject) {
        this.yVelocity = -45;
        hitInfo.hitObject.stomp();
    }

    if (this.y > 45 || this.y < -40) {
        this.initialize();
    }
};

//=====
// GameEnemy
var GameEnemy = new Class({
    Extends: GameObject,
```

# Bret Victor - Inventing on Principle (3)

```
function binarySearch (key, array) {  
    var low = 0;  
    var high = array.length - 1;  
  
    while (1) {  
  
        var mid = floor((low + high)/2);  
        var value = array[mid];  
  
        if (value < key) {  
            low = mid + 1;  
        }  
        else if (value > key) {  
            high = mid - 1;  
        }  
        else {  
            return mid;  
        }  
    }  
}  
  
key = 'e'  
array = ['a', 'b', 'c', 'd', 'e', 'f']  
low = 0  
high = 5  
  
low = 0 | 3  
high = 5 | 5  
mid = 2 | 4  
value = 'c' | 'e'  
  
low = 3 |  
  
return | 4
```

# Bret Victor - Alesis

2003 ~ 2005

<http://worrydream.com/#!/Alesis>

- \* ALESIS ION
- \* ALESIS MICRON
- \* ALASIS FUSION

Alesis Micron - Quick demo

<http://www.youtube.com/watch?v=nqoN1dpwA30>



# Bret Victor - Learnable Programming

September 2012

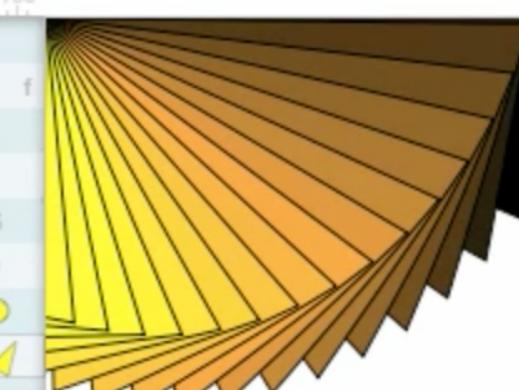
Learnable Programming

<http://worrydream.com/LearnableProgramming/>

step 143

```
var i = 0;
while (i < 20) {
    var scaleFactor = 1 + (20 - i)/20;
    resetMatrix();
    scale(scaleFactor);
    rotate(i * 6);
    fill(i * 30, i * 18, 0);
    triangle(0,0, 100,-20, 95,40); Draw a triangle.
    i += 1;
}
```

20	110	120	130	140	150	160	f
t	t	t	t	t	t	t	
1.35	1.3	1.25	1.2	1.15	1.1	1.05	
•	•	•	•	•	•	•	
1.4	1.35	1.3	1.25	1.2	1.15	1.1	1.05
Q	Q	Q	Q	Q	Q	Q	
○	○	○	○	○	○	○	
13	14	15	16	17	18	19	20



```
fill(0,0,0);
shape("rect", 80,80, 120,105);
shape("triangle", 80,20, 140,60, 80,60);
```

- "line"
- "triangle" 
- "rect" 
- "ellipse" 
- "bezier" 

# Chris Granger - Light Table

12 Apr 2012

Light Table - a new IDE concept

<http://www.chris-granger.com/2012/04/12/light-table---a-new-ide-concept/>

- Bret Victor hinted at the idea that we can do much better than we are now - we can provide instant feedback, we can show you how your changes affect a system. And I discovered he was right.

## Instant feedback

- In *Inventing on Principle*, Bret showed us that we could live-edit games and write binary search in an editor that is constantly evaluating and showing you what's going on. The lispers among us are used to using the REPL to have an environment where we can try things out. But we can do better - we can do it in place and instantaneously. For example here I type the code `(+ 3 4)` and we immediately see that it evaluates to 7 - no ctrl-enter or anything else.

# Chris Granger - Light Table : Playground

24 Jun 2012

It's playtime - Light Table Playground released

<http://www.chris-granger.com/2012/06/24/its-playtime/>

## Instarepl

- **real-time evaluation**

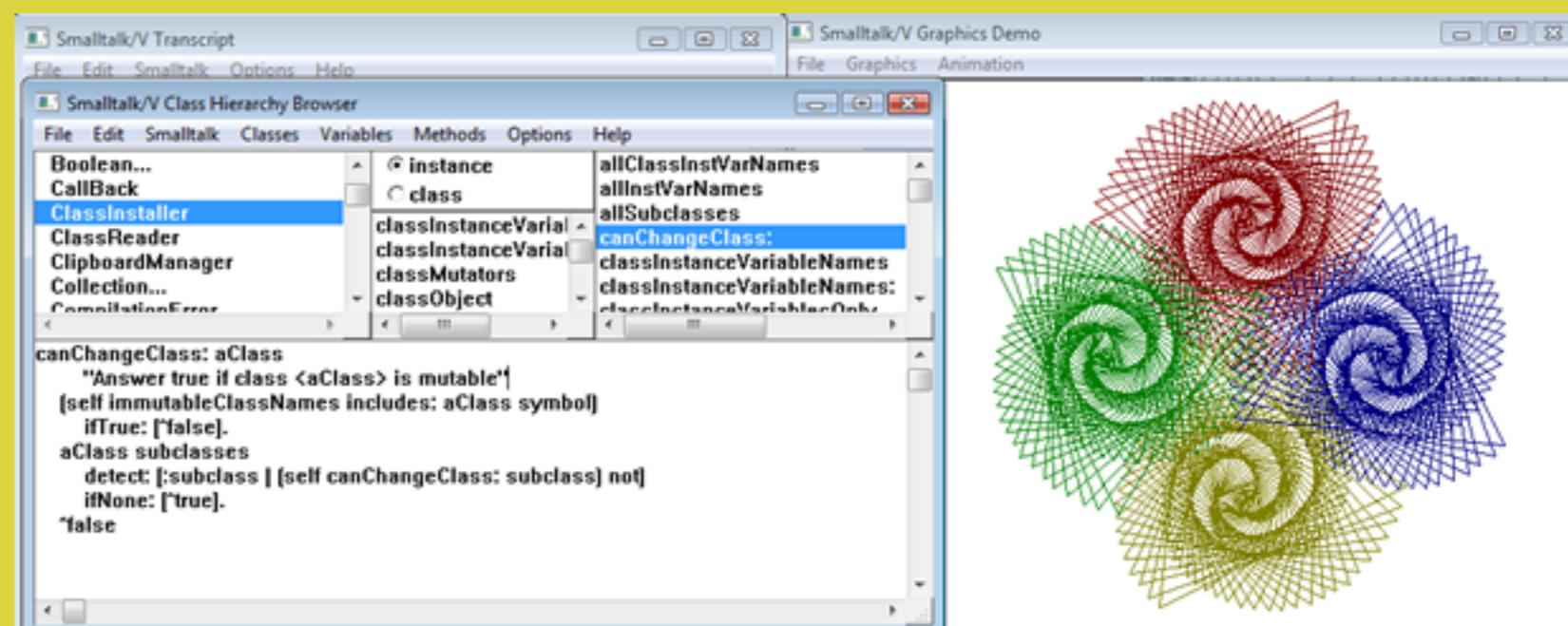
```
; ;Welcome to the instarepl.  
; ;Anything typed into this box will immediately be evaluated  
; ;with the result being shown on the right hand side.  
; ;For example:  
  
(+ 3 4) => 7  
(- 10 20) => -10  
  
; ;This allows you to very quickly see changes to all the  
; ;things you're working with. You'll notice that the code  
; ;on the right also has the variables' values filled in in  
; ;green, making it easy to see how information is flowing  
; ;through the program.  
  
(defn my-add [3 45]  
  (+ 3 45))  
  
(my-add 3 45) => 48  
  
; ;If you end up stuck, you can press Ctrl+D to cancel  
; ;everything that is currently executing. Ctrl+R will also  
; ;reset the state of the GUI if you want to start over fresh.
```

# Live Programming

## A History of Live Programming

<http://liveprogramming.github.io/liveblog/2013/01/a-history-of-live-A-History-of-Live-Programmingprogramming/>

- Live programming is an idea espoused by programming environments from the earliest days of computing, such as Lisp machines, Logo, Hypercard, and Smalltalk. In common with all these systems was liveness: Feedback is nearly instantaneous and evaluation is always accessible. For example, any part of the SmallTalk environment could be modified at any time and reflected instantly. Likewise, in HyperCard, the state of any object was considered to be live and editable at any time.

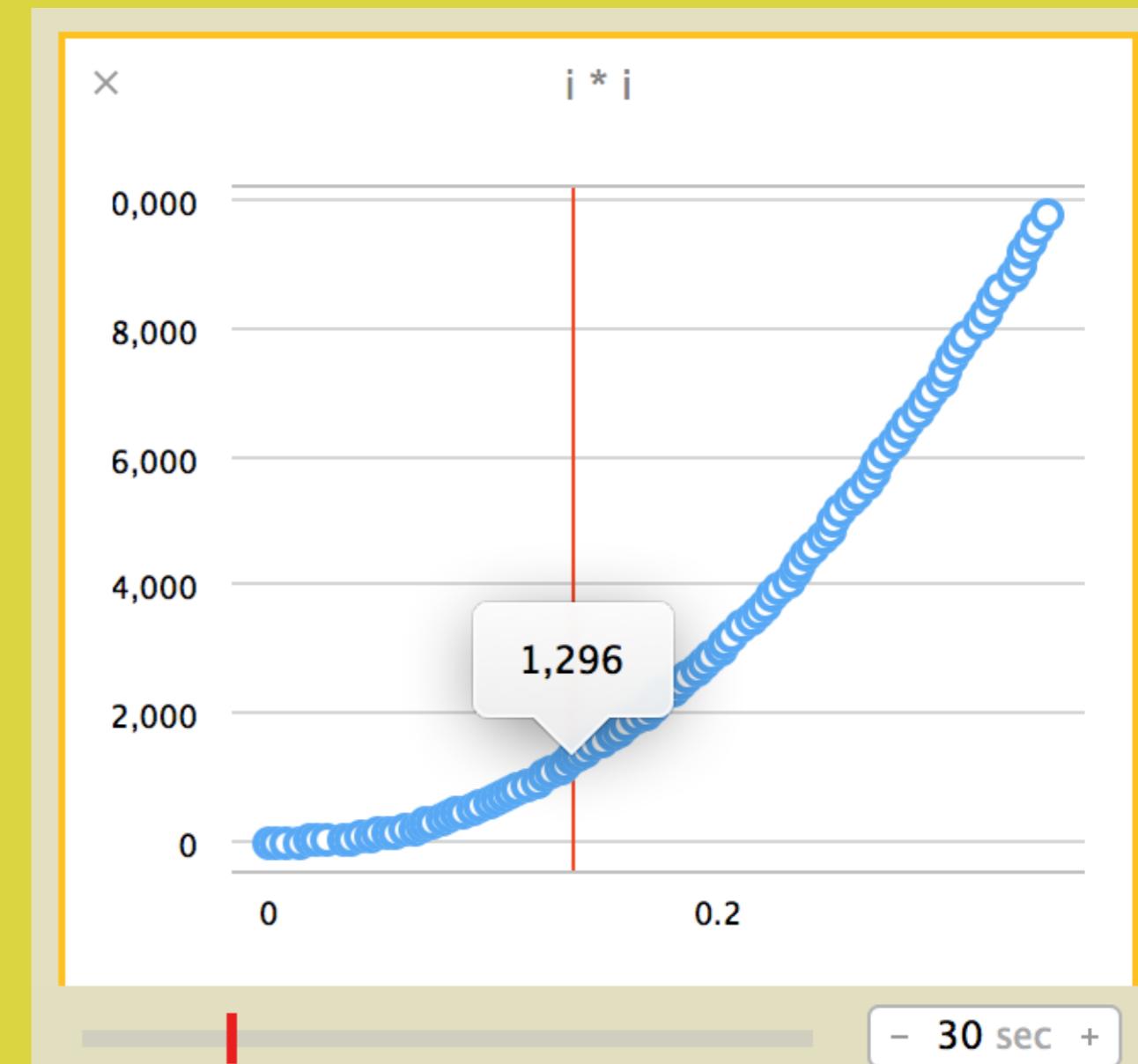


# Playgrounds

- \* Timeline
- \* Quick Looks
- \* Settings
- \* XCPlayground

# Timeline

- 타임라인
- 시간 흐름에 대한 값의 변화를 파악

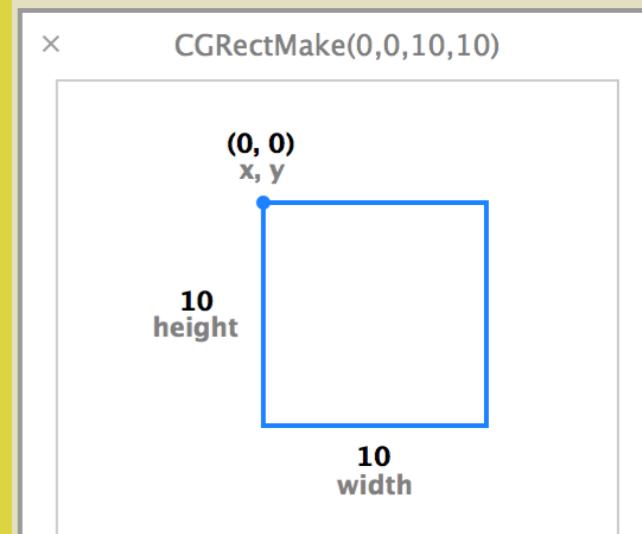
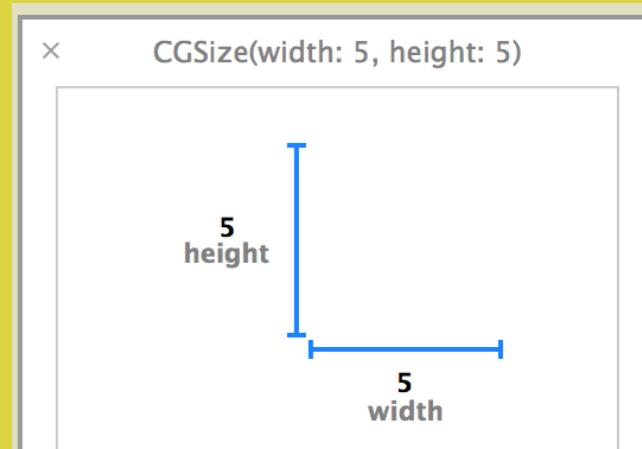


# Quick Looks

- 빠른 보기

## 지원하는 타입

- Colors
- Strings (plain and attributed)
- Images
- Views
- Arrays and dictionaries
- Points, rects, sizes
- Bezier paths
- URLs
- Classes and structs



# Settings

파일 인스펙터에서 플랫폼, 리소스 경로 설정

## Platform

- OS X
- iOS

## Resource Path

- Absolute Path
- Relative to Playground
- Inside Playground

## 사용

- `NSImage(named:)`
- `NSBundle.pathForResource(_:_ofType:)`

# XCPPlayground

## OS X에서만 되는 기능

### 캡쳐할 값 지정

- `func XCPCaptureValue<T>(identifier: String, value: T)`

### 타임라인에 라이브 뷰 보이기

- `func XCPShowView(identifier: String, view: NSView)`

### 비동기 지원

- `func XCPSetExecutionShouldContinueIndefinitely(continueIndefinitely: Bool = true)`

# 예제 모음

## Session 408

- GuidedTour.playground
- MyPlayground.playground
- InsertionSort.playground
- TestTableViewCell.playground
- PlaygroundIconView.playground
- Async.playground

# Session 408 - Swift Playgrounds

Xcode Engineers : Rick Ballard, Connor Wakamo

**WWDC 2014 Session Videos - Apple Developer**

<https://developer.apple.com/videos/wwdc/2014/>

- 408\_swift\_playgrounds.pdf
- 408\_hd\_swift\_playgrounds.mov

**ASCIIwwdc - Swift Playgrounds**

<http://asciiwwdc.com/2014/sessions/408>

**WWDC-2014-Subtitle / 408 Swift Playgrounds [English].srt**

<https://github.com/walkingway/WWDC-2014-Subtitle/blob/master/408%20Swift%20Playgrounds%20%5BEnglish%5D.srt>

# GuidedTour.playground

```
GuidedTour.playground $ cat contents.xcplayground
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<playground version='3.0' sdk='macosx' allows-reset='YES'>
  <sections>
    <documentation relative-path='section-1.html'/>
    <documentation relative-path='section-2.html'/>
    <code source-file-name='section-3.swift'/>
    <documentation relative-path='section-4.html'/>
    <code source-file-name='section-5.swift'/>
    <documentation relative-path='section-6.html'/>
    <code source-file-name='section-7.swift'/>
    ...
    <documentation relative-path='section-74.html'/>
    <code source-file-name='section-75.swift'/>
    <documentation relative-path='section-76.html'/>
    <code source-file-name='section-77.swift'/>
    <documentation relative-path='section-78.html'/>
    <code source-file-name='section-79.swift'/>
    <documentation relative-path='section-80.html'/>
    <code source-file-name='section-81.swift'/>
    <documentation relative-path='section-82.html'/>
    <code source-file-name='section-83.swift'/>
    <documentation relative-path='section-84.html'/>
    <code source-file-name='section-85.swift'/>
    <documentation relative-path='section-86.html'/>
    <code source-file-name='section-87.swift'/>
    <documentation relative-path='section-88.html'/>
    <documentation relative-path='section-89.html'/>
  </sections>
  <timeline fileName='timeline.xctimeline' />
</playground>
```



Tradition suggests that the first program in a new language should print the words “Hello, world” on the screen. In Swift, this can be done in a single line:

```
1 println("Hello, world")
```

If you have written code in C or Objective-C, this syntax looks familiar to you—in Swift, this line of code is a complete program. You don’t need to import a separate library for functionality like input/output or string handling. Code written at global scope is used as the entry point for the program, so you don’t need a `main` function. You also don’t need to write semicolons at the end of every statement.

This tour gives you enough information to start writing code in Swift by showing you how to accomplish a variety of programming tasks. Don’t worry if you don’t understand something—everything introduced in this tour is explained in detail in the rest of this book.

## Simple Values

Use `let` to make a constant and `var` to make a variable. The value of a constant doesn’t need to be known at compile time, but you must assign it a value exactly once. This means you can use constants to name a value that you determine once but use in many places.

2	<code>var myVariable = 42</code>	42
3	<code>myVariable = 50</code>	50
4	<code>let myConstant = 42</code>	42

# MyPlayground.playground

## Cocoa

- **NSAttributedString**
  - **NSColor**
  - **NSFont**
- **NSImageView**
  - **NSImage**

```
x let attrStr = NSAttributedString(...(name: "BM JUA_TTF", size: 50))
```

안녕 세미

```
x imageView.image = myImage
```

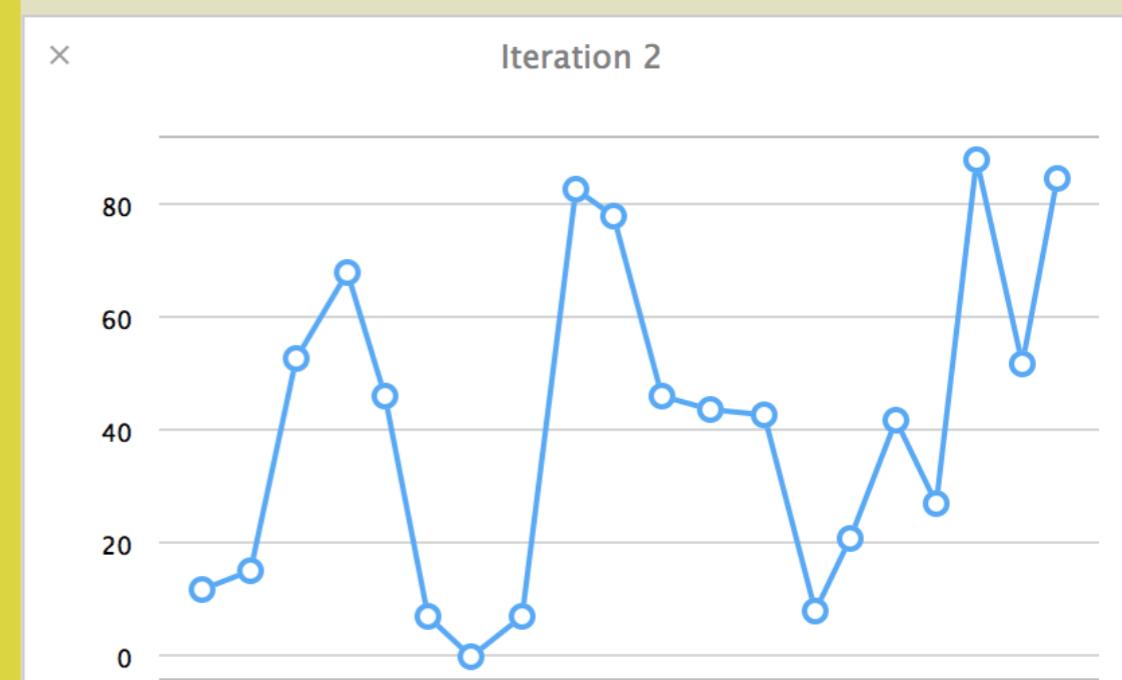
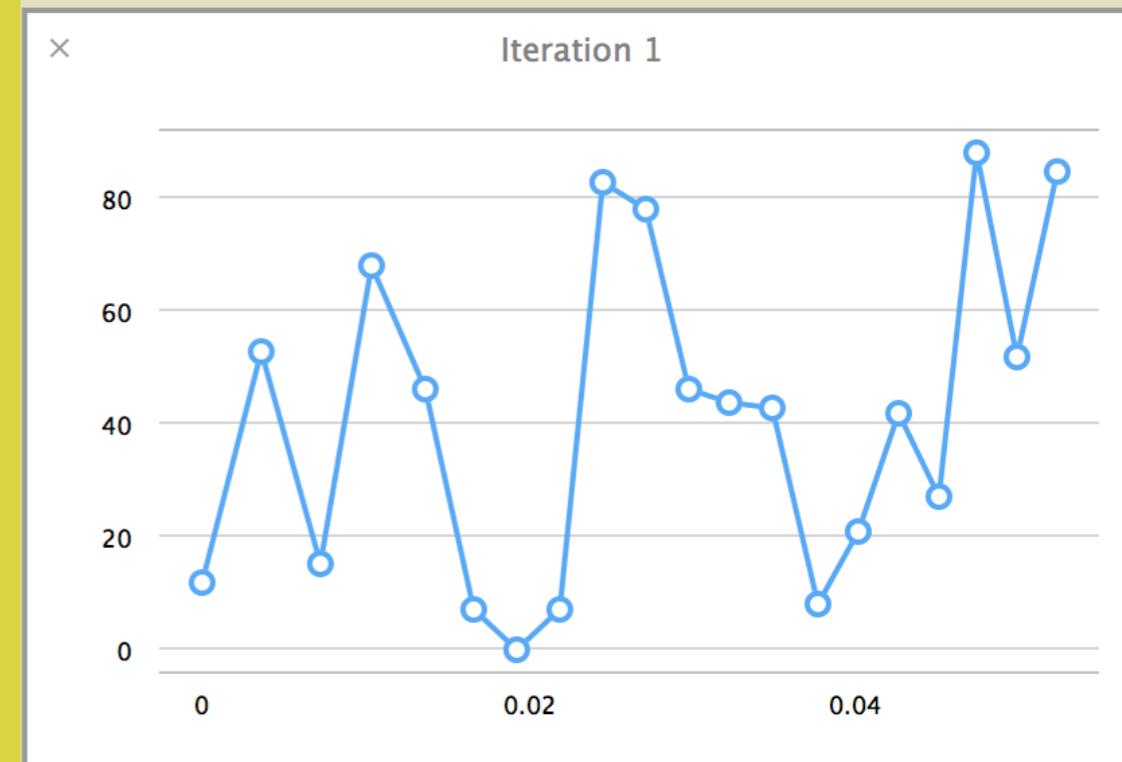


# InsertionSort.playground

## XCPlayground

- **XCPCaptureValue**

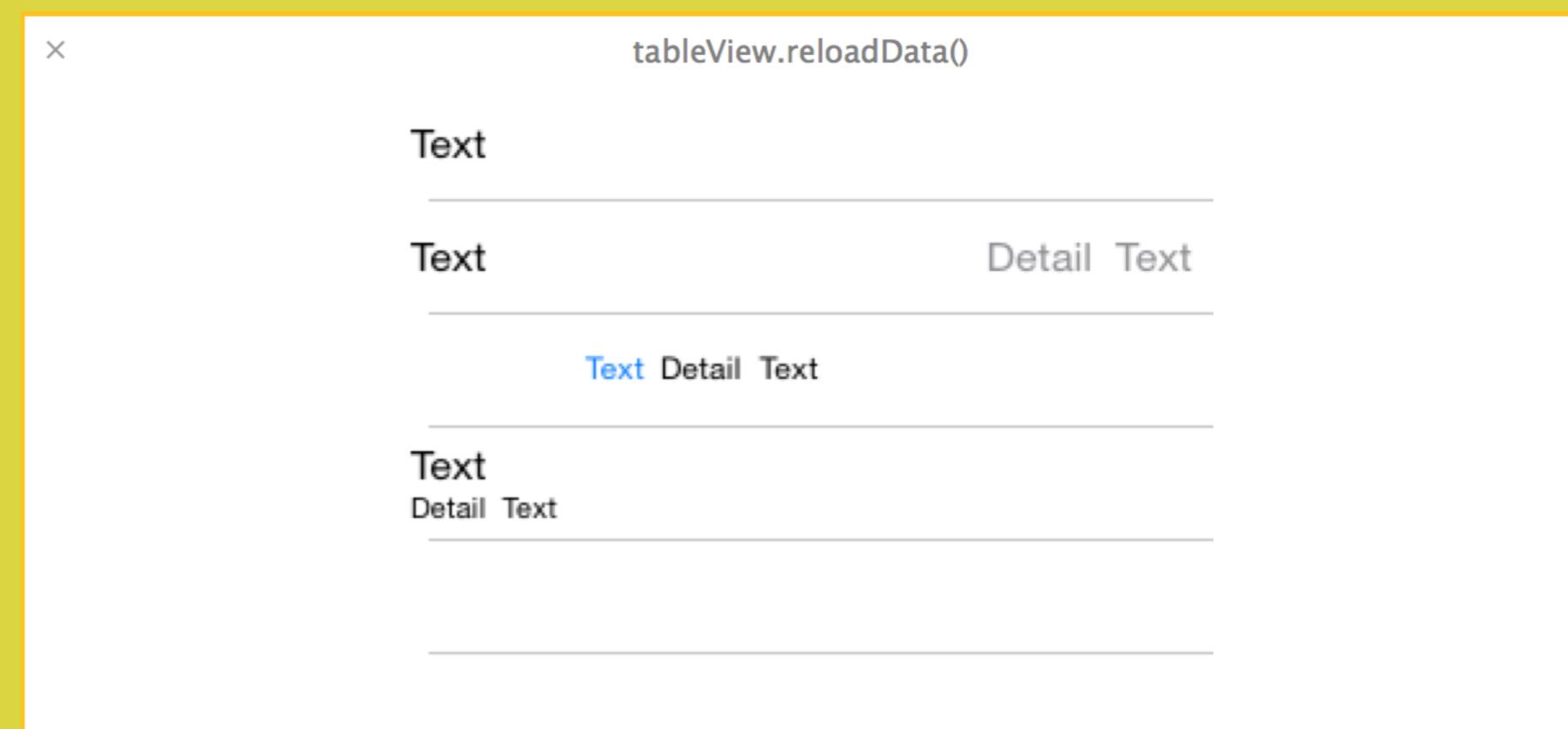
x data  
[0] 0  
[1] 7  
[2] 7  
[3] 8  
[4] 12



# TestTableViewCell.playground

## UIKit

- **UITableView**
- **UITableViewCell**



# PlaygroundIconView.playground

## Cocoa

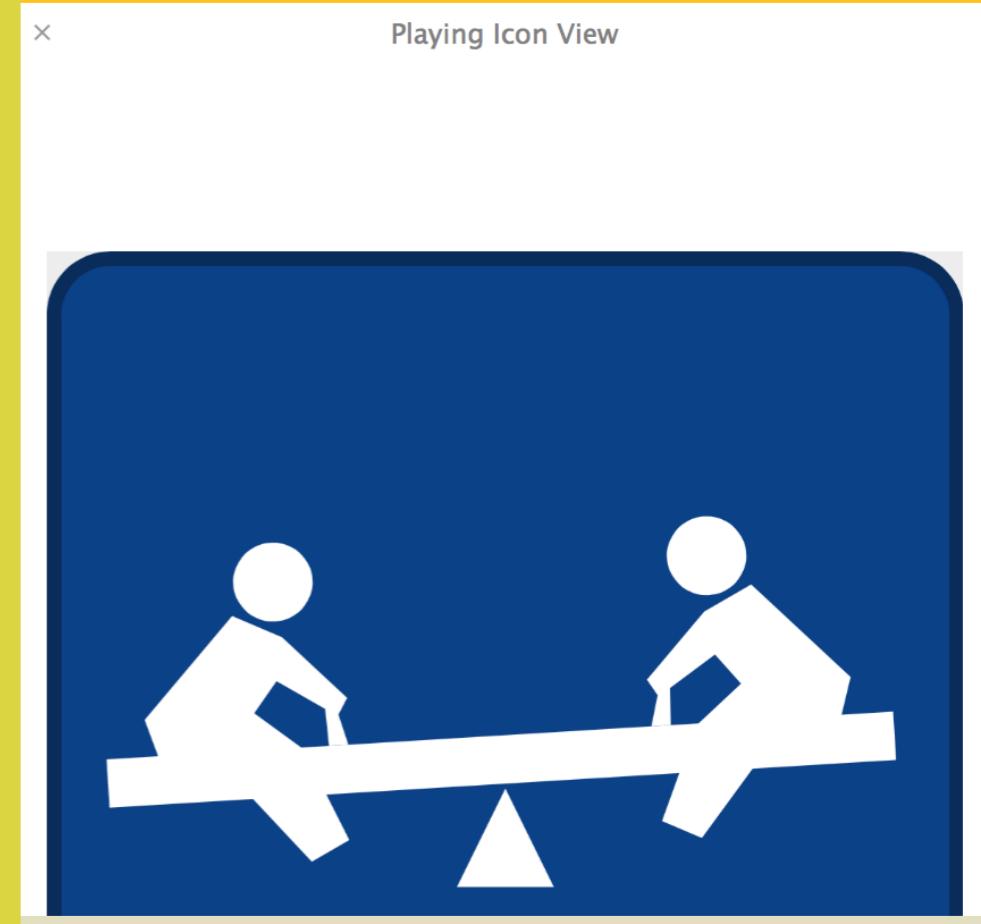
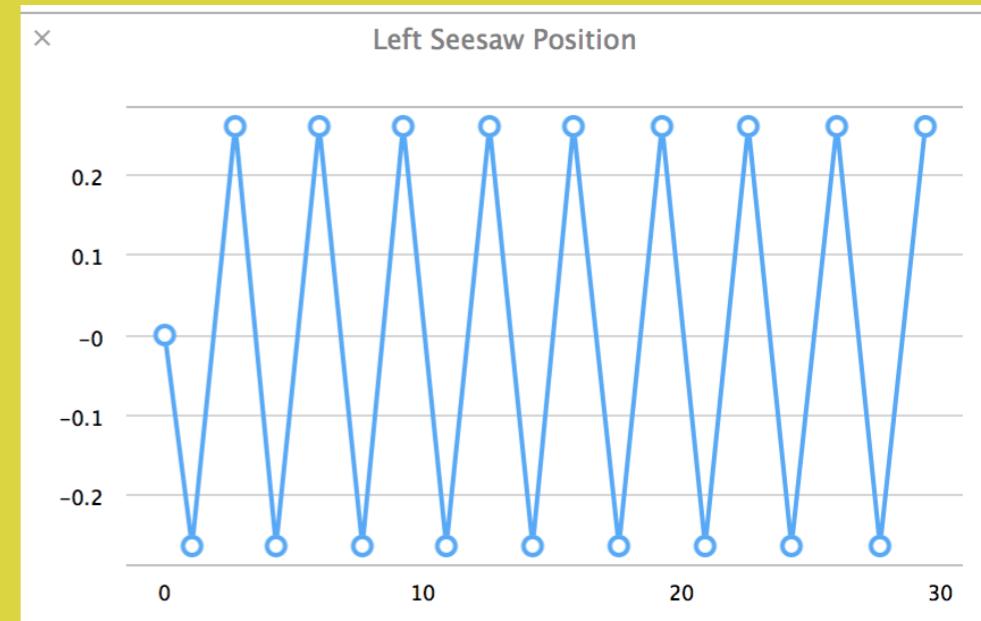
- **CAShapeLayer**
- **CAAnimation**
- **NSBezierPath**

## XCPlayground

- **XCPShowView**

paradin / swift\_tutorial

[https://github.com/paradin/swift\\_tutorial](https://github.com/paradin/swift_tutorial)



# Async.playground

## Cocoa

- `NSURL`
- `NSURLSession`
- `NSURLSessionDelegate`

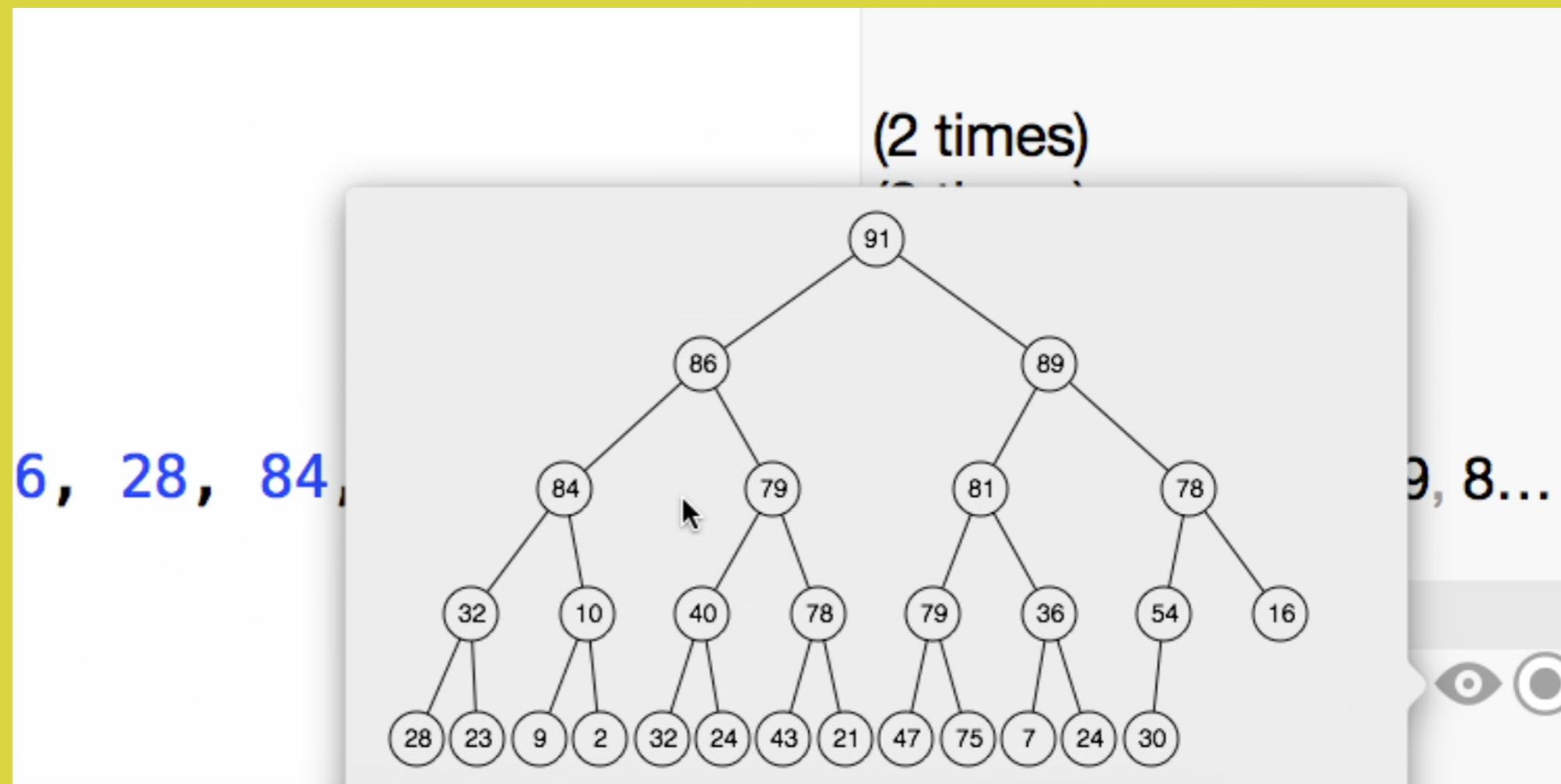
## XCPlayground

- `XCPSetExecutionShouldContinueIndefinitely`

# Custom Quick Look Support

- `debugQuickLookObject()` 메서드 추가

QuickLook.xcodeproj



# SpriteKit

The image shows a Xcode interface with two main panes. The left pane displays Swift code for a `setupHero(_:_:)` function. The right pane shows the resulting game scene titled "Balloons".

**Code (Balloons.playground):**

```
func didMoveToView(scene : SKScene, delegate : SKPhysicsContactDelegate) {  
    // ===== Blimp Control =====  
    yOffsetForTime = { i in  
        return 80 * sin(i / 10.0)  
    }  
    // ===== Scene Configuration =====  
    // Set up balloon lighting and per-pixel collisions.  
    balloonConfigurator = { b in  
        b.physicsBody.categoryBitMask = CONTACT_CATEGORY  
        b.physicsBody.fieldBitMask = WIND_FIELD_CATEGORY  
        b.lightingBitMask = BALLOON_LIGHTING_CATEGORY  
    }  
    // Load images for balloon explosion.  
    balloonPop = (1...4).map {  
        SKTexture(imageNamed: "explode_0\($0)")  
    }  
    // Install turbulent field forces.  
    var turbulence = SKFieldNode.noiseFieldWithSmoothness(0.7, animationSpeed:0.8)  
    turbulence.categoryBitMask = WIND_FIELD_CATEGORY  
    turbulence.strength = 0.21  
    scene.addChild(turbulence)  
    cannonStrength = 210.0  
    // ===== Scene Initialization =====  
    // Do the rest of the setup and start the scene.  
    setupHero(scene, delegate)  
    setupFan(scene, delegate)  
    setupCannons(scene, delegate)  
}  
  
func handleContact(bodyA : SKSpriteNode, bodyB : SKSpriteNode){  
    if (bodyA == hero) {  
        bodyB.normalTexture = nil  
        bodyB.runAction(removeBalloonAction)  
    } else if (bodyB == hero) {  
        bodyA.normalTexture = nil  
        bodyA.runAction(removeBalloonAction)  
    }  
}
```

**Game Scene (Balloons):**

The game scene features a blue sky with various colored balloons (pink, yellow, blue, red). A large orange blimp flies in the upper right. In the foreground, there's a green grassy area with a red and white striped tent, a white satellite dish, and a large Ferris wheel with colorful seats. The scene is set against a blue background.

**Timeline (Balloons.playground Timeline):**

The timeline shows a graph of a sine wave with the equation  $y = 80 \sin(x)$ . The vertical axis ranges from -50 to 50, and the horizontal axis ranges from -30 sec to +30 sec. The wave oscillates between approximately -80 and 80.

# Playgrounds Limitations

- Playground cannot be used for performance testing.
- Does not support User Interaction.
- Does not support On-device execution.
- Cannot use your app or framework code.
- Does not support custom entitlements.

# Playgrounds Limitations (2)

추가 단점

- Xcode 자주 사망

# 질문 답변

# 쌤유

- 듣느라 수고하셨습니다
- 끝