From Zero to Hero:

Making your iOS App Accessible to VoiceOver and Beyond

By Sommer Panage
@sommer

Accessibility means "Using technology to overcome challenges."

— Apple

Visual

- Blind
- Low-vision
- Color-blindness
- Using device outdoors
- Using device while driving

Motor

- Cerebral palsy
- Muscular dystrophy
- Multiple sclerosis
- Broken hand
- Using device while driving

Auditory

- Deaf
- Hard-of-hearing
- Mono-audio
- Using device in noisy area
- Using device with no audio

Learning / Cognitive

- Autism spectrum disorders
- Dyslexia
- ADHD
- Young child
- Older adult

Accessibility is for everyone.

CLEARING A PATH
FOR PEOPLE WITH SPECIAL NEEDS
CLEARS THE PATH FOR EVERYONE!

VoiceOver

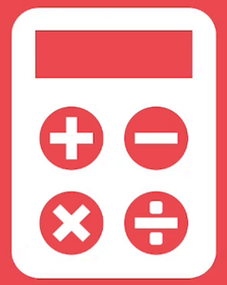
Supporting VoiceOver

1. Audit - know what you need
2. Code - write what you need

Audit

Let's Learn!

Learn Numbers



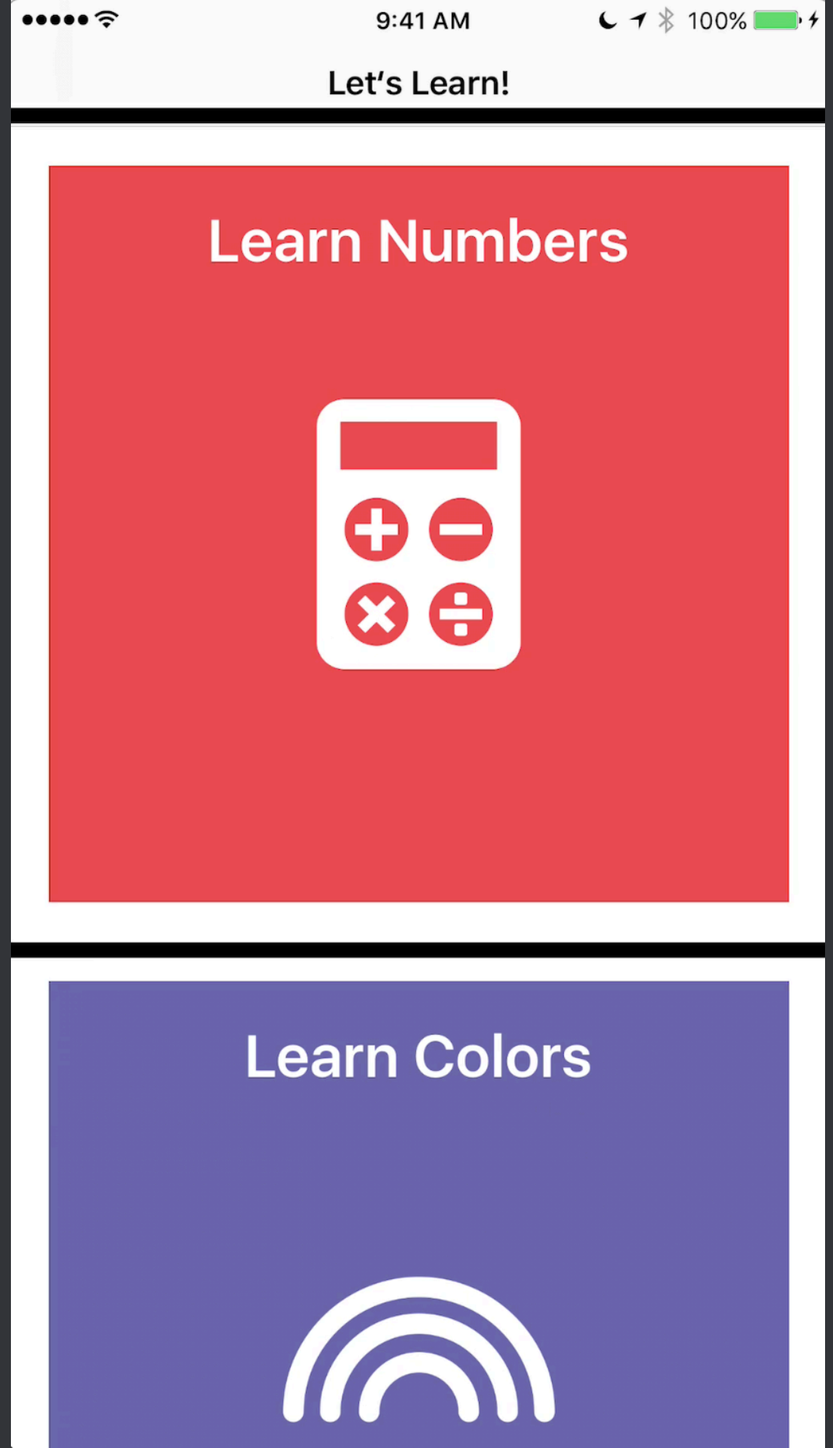
Learn Colors

Audit

- Turn on VoiceOver
- Navigate thru app
- Look for items that VO skip
- Look for items that do not explain:
  - What it does
  - Who it is

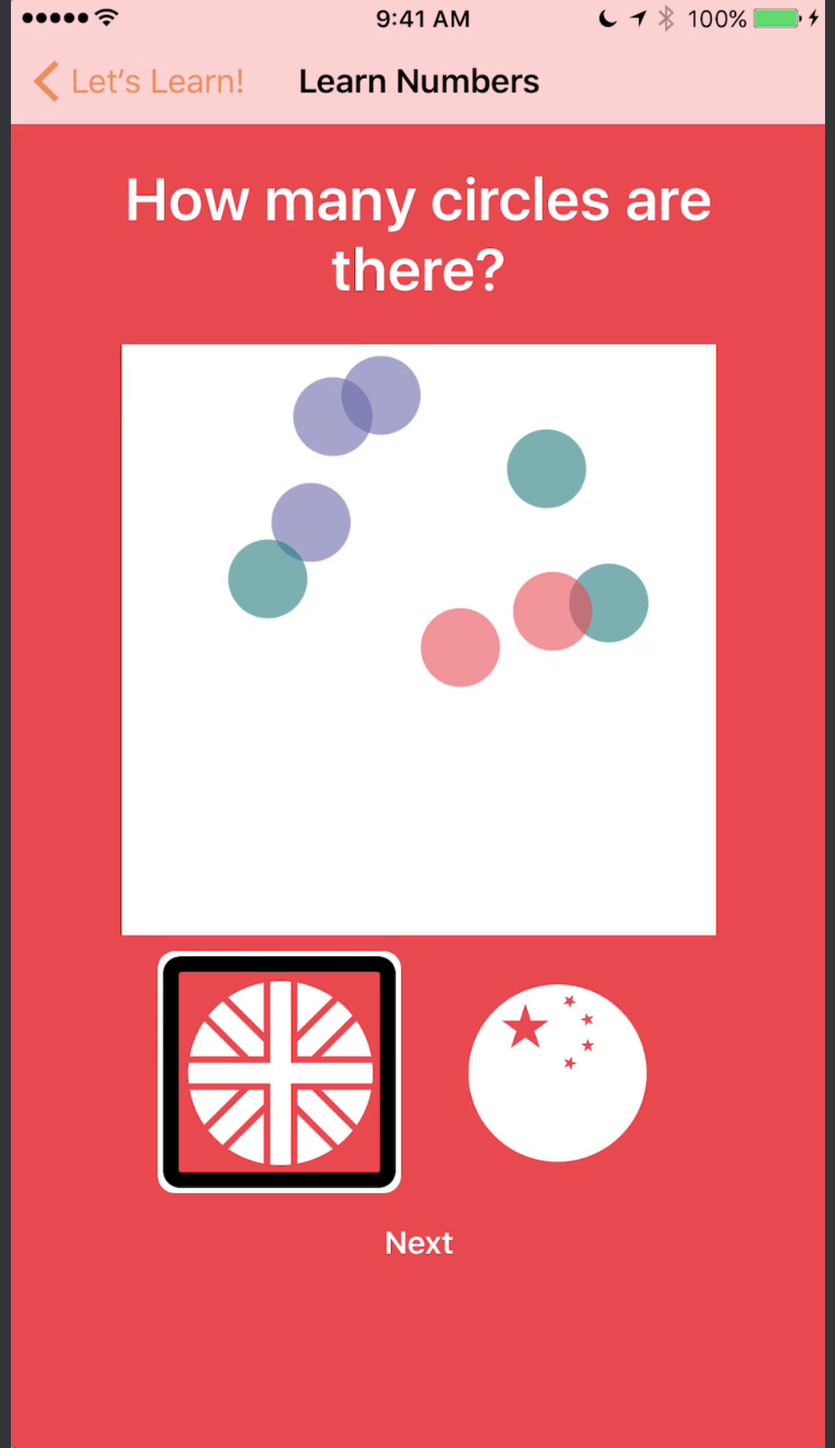
Issues in our app

- "Learn" items don't indicate they're tappable



Issues in our app

- Answer buttons need names
- Can't see our items in the white box!

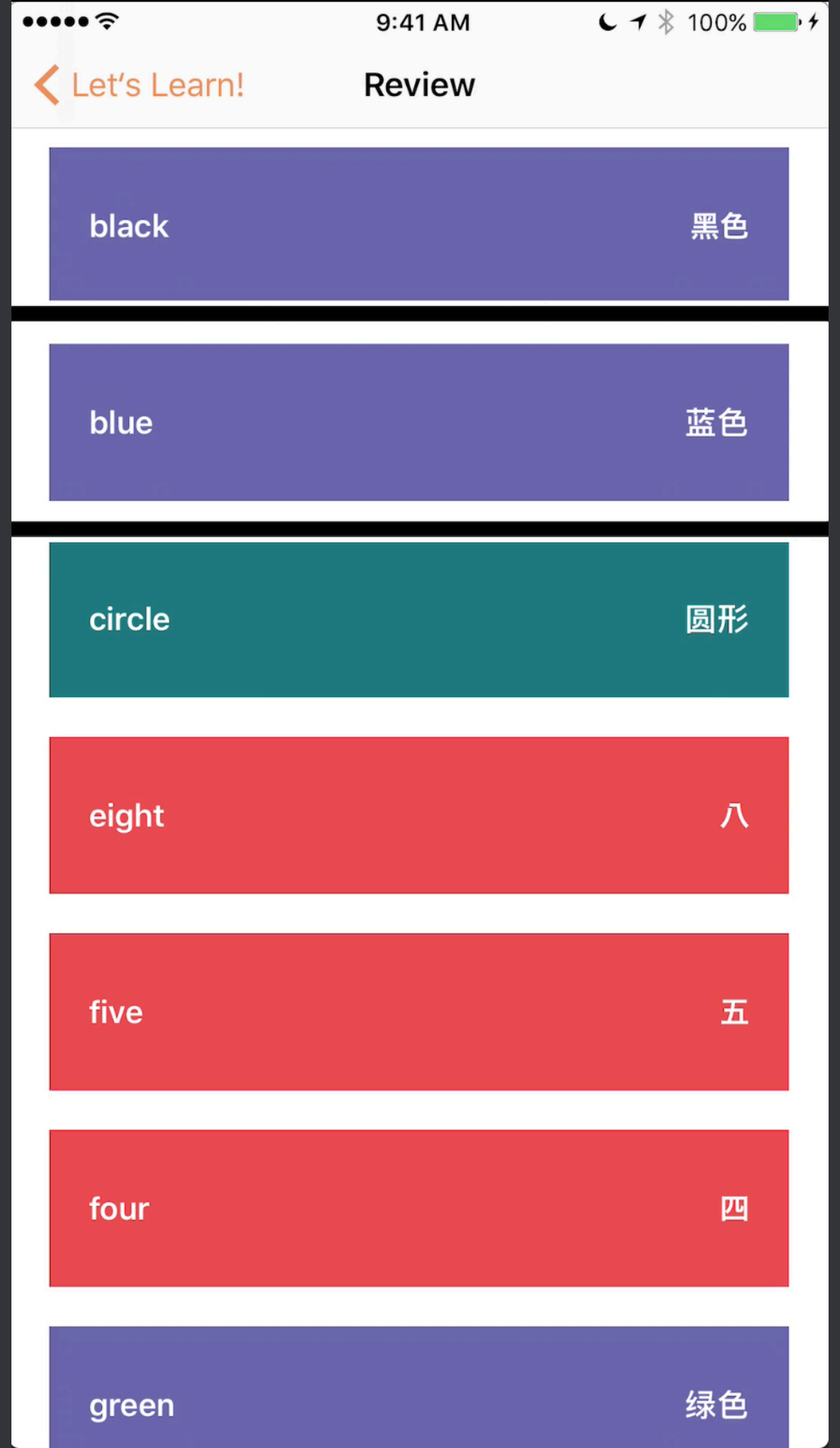


Issues in our app

- Not clear that "Got it!" goes back

Issues in our app

- No audio differentiation of review items



Code

To follow along in the code go to

http://bit.ly/2qctKyb

or

https://github.com/spanage/HelloA11y-Swift

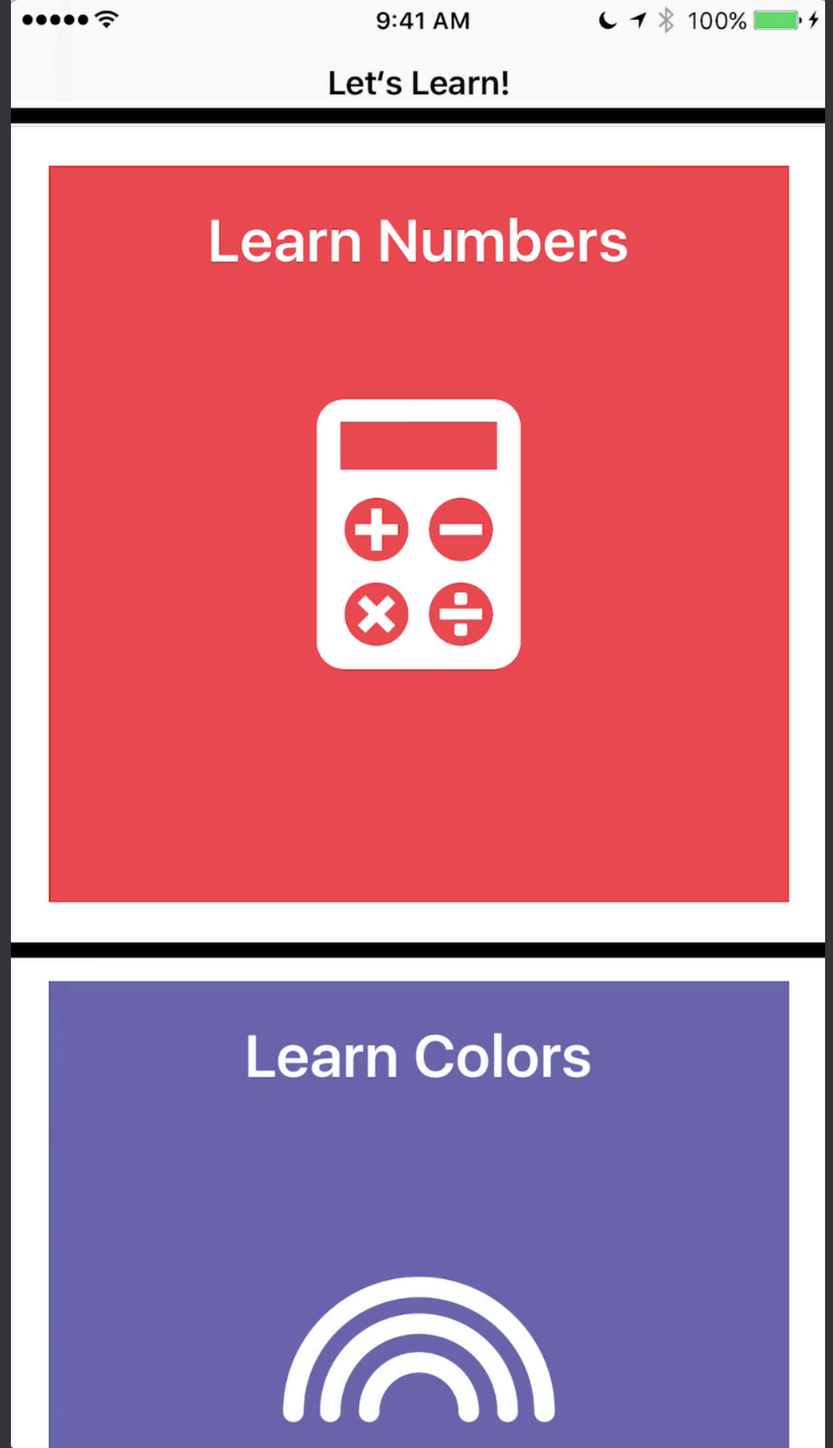
.....

- **master** branch for original code without accessibility
- **a11y** branch for accessible code

Issues in our app

- "Learn" items don't indicate they're tappable

We need these items to tell us what they are!



What? Accessibility Traits

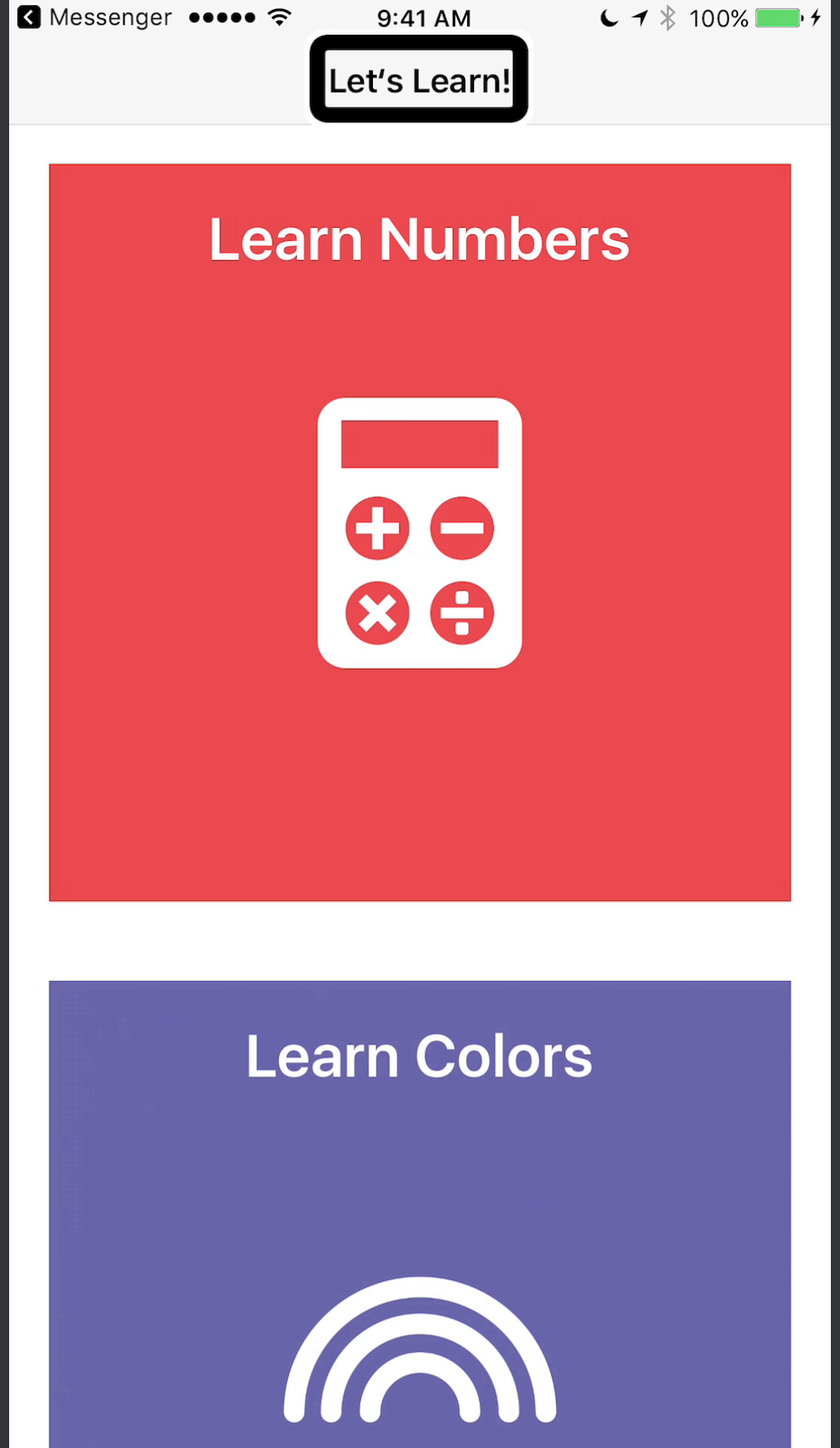
- `myItem.accessibilityTraits`
- Indicate what an items is
- Automatically supplied for UIControls
- Should use bitwise or (`|`) to preserve original traits

Common Accessibility Traits

- UIAccessibilityTraitButton --> *'Tap me!'*
- UIAccessibilityTraitHeader --> *Makes nav easier*
- UIAccessibilityTraitLink --> *'I jump around or leave the app'*
- UIAccessibilityTraitImage --> *'I'm an image'*
- UIAccessibilityTraitAdjustable --> *'Swipe up/down to adjust me'*

Cell code before

```
func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
    let cell: UITableViewCell
    switch Section(rawValue: indexPath.section)! {
    case .items:
        let mainCell: MainTableViewCell = tableView.dequeueReusableCell(
            withIdentifier: MainTableViewCell.reuseID,
            for: indexPath) as! MainTableViewCell
        mainCell.item = items[indexPath.row]
        cell = mainCell
    case .review:
        let reviewCell: ReviewTableViewCell = tableView.dequeueReusableCell(
            withIdentifier: ReviewTableViewCell.reuseID,
            for: indexPath) as! ReviewTableViewCell
        cell = reviewCell
    }
    return cell
}
```



Accessibility Traits are easy!

We just need to tell our cells to behave to Accessibility like buttons!

```
cell.accessibilityTraits |= UIAccessibilityTraitButton
```

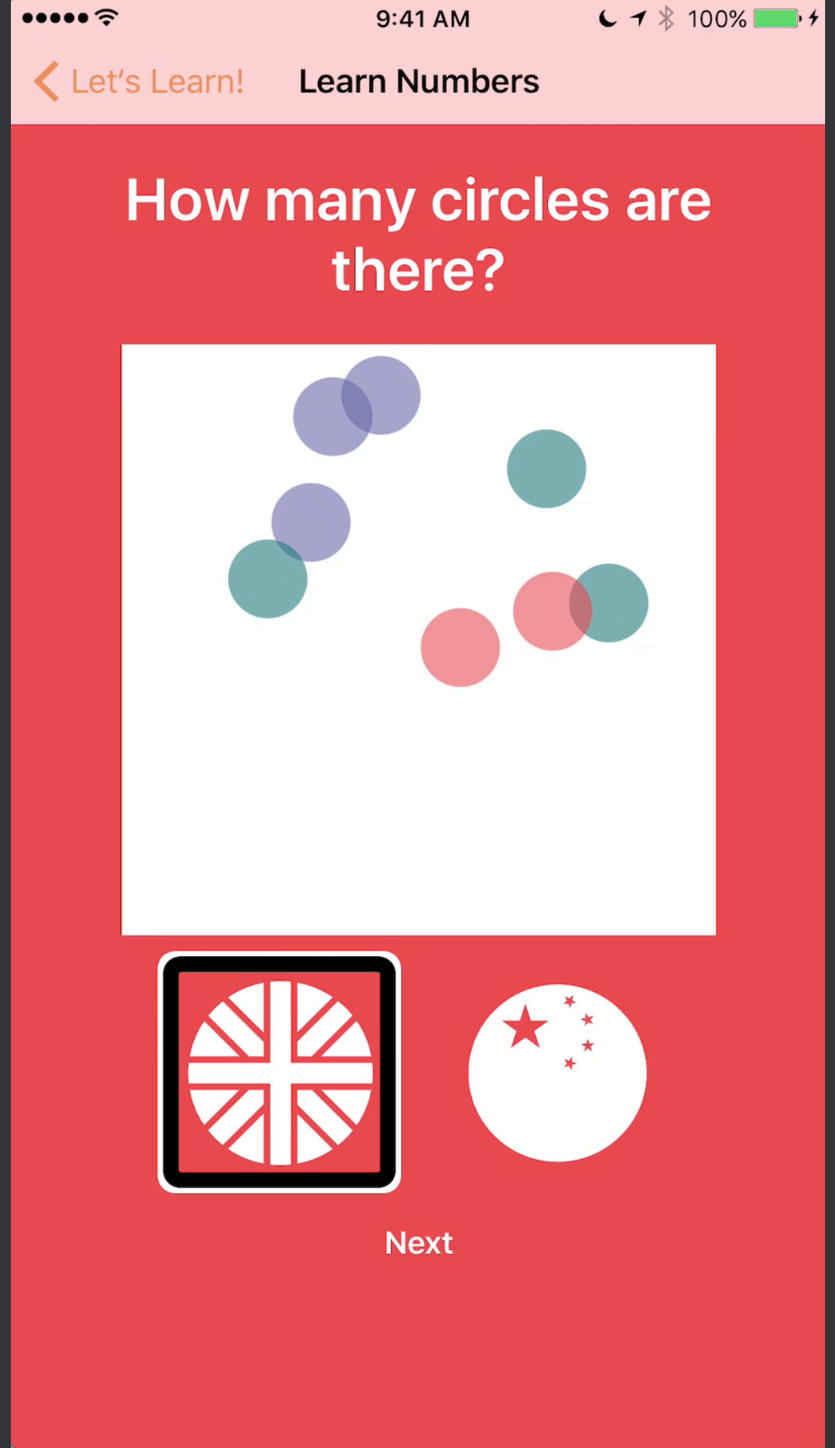
Cell code after

```
func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
    let cell: UITableViewCell
    switch Section(rawValue: indexPath.section)! {
    case .items:
        let mainCell: MainTableViewCell = tableView.dequeueReusableCell(
            withIdentifier: MainTableViewCell.reuseID,
            for: indexPath) as! MainTableViewCell
        mainCell.item = items[indexPath.row]
        cell = mainCell
    case .review:
        let reviewCell: ReviewTableViewCell = tableView.dequeueReusableCell(
            withIdentifier: ReviewTableViewCell.reuseID,
            for: indexPath) as! ReviewTableViewCell
        cell = reviewCell
    }

    // Our cells behave like buttons
    cell.accessibilityTraits |= UIAccessibilityTraitButton
    return cell
}
```

Issues in our app

- Answer buttons need names

We need to provide labels for these buttons!

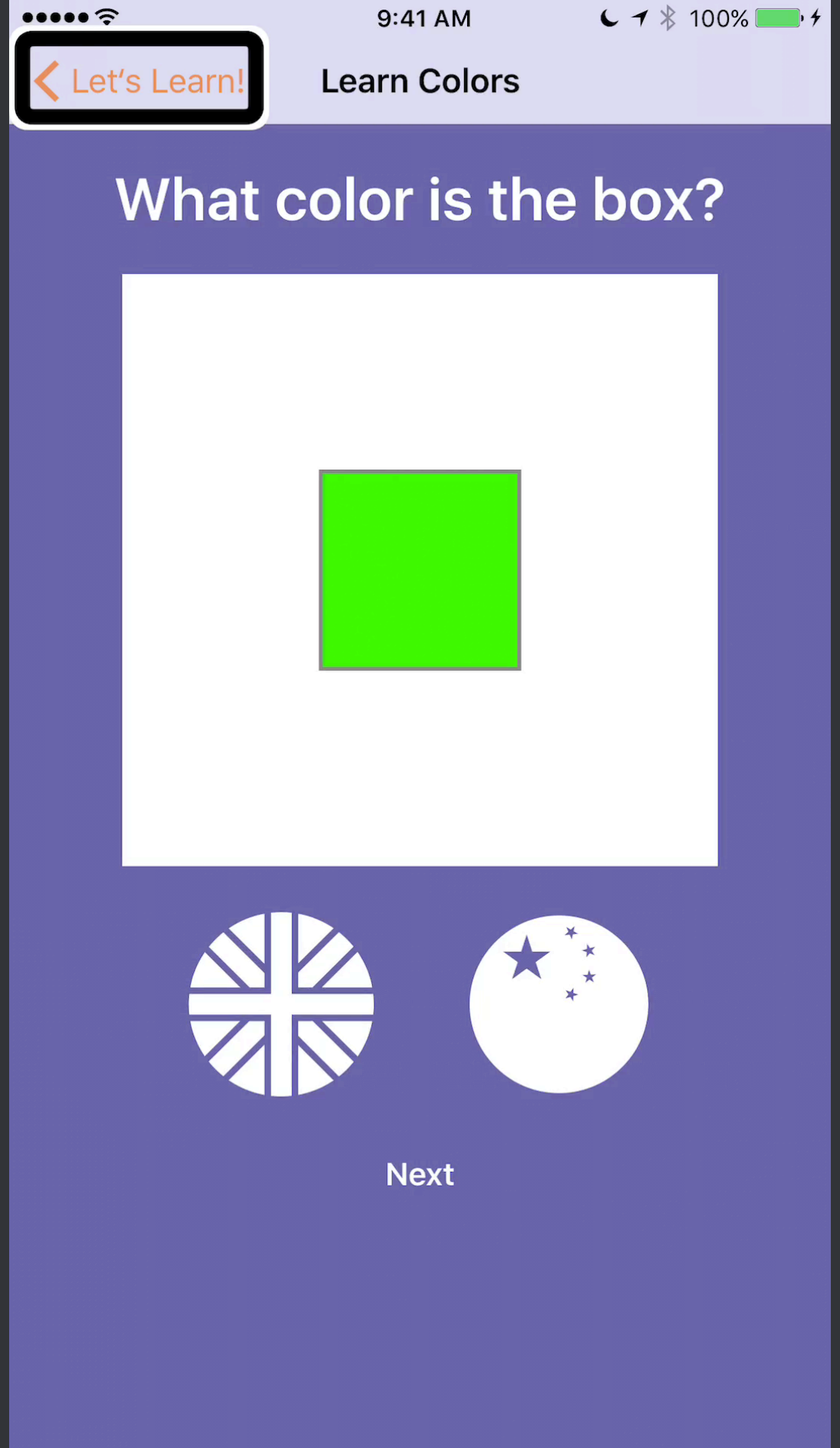


Who? Accessibility Label

- `myItem.accessibilityLabel`
- Indicates the name of an item / what it does
- Automatically supplied for text and cells (generally)
- Should be short and clear
- Should *not* contain trait info, i.e. "Answer button"

Button code before

```
let englishButton: UIButton = {
    let button = UIButton(type: .system)
    button.setBackgroundImage(#imageLiteral(resourceName: "union_jack")
        .withRenderingMode(.alwaysTemplate), for: .normal)
    return button
}()
```



Accessibility Label (and Hint)

```
button.accessibilityLabel = "English Answer"
button.accessibilityHint = "Shows the answer in English"
```

accessibilityLabel

Who am I?

accessibilityHint

More (optional) info on what I do!

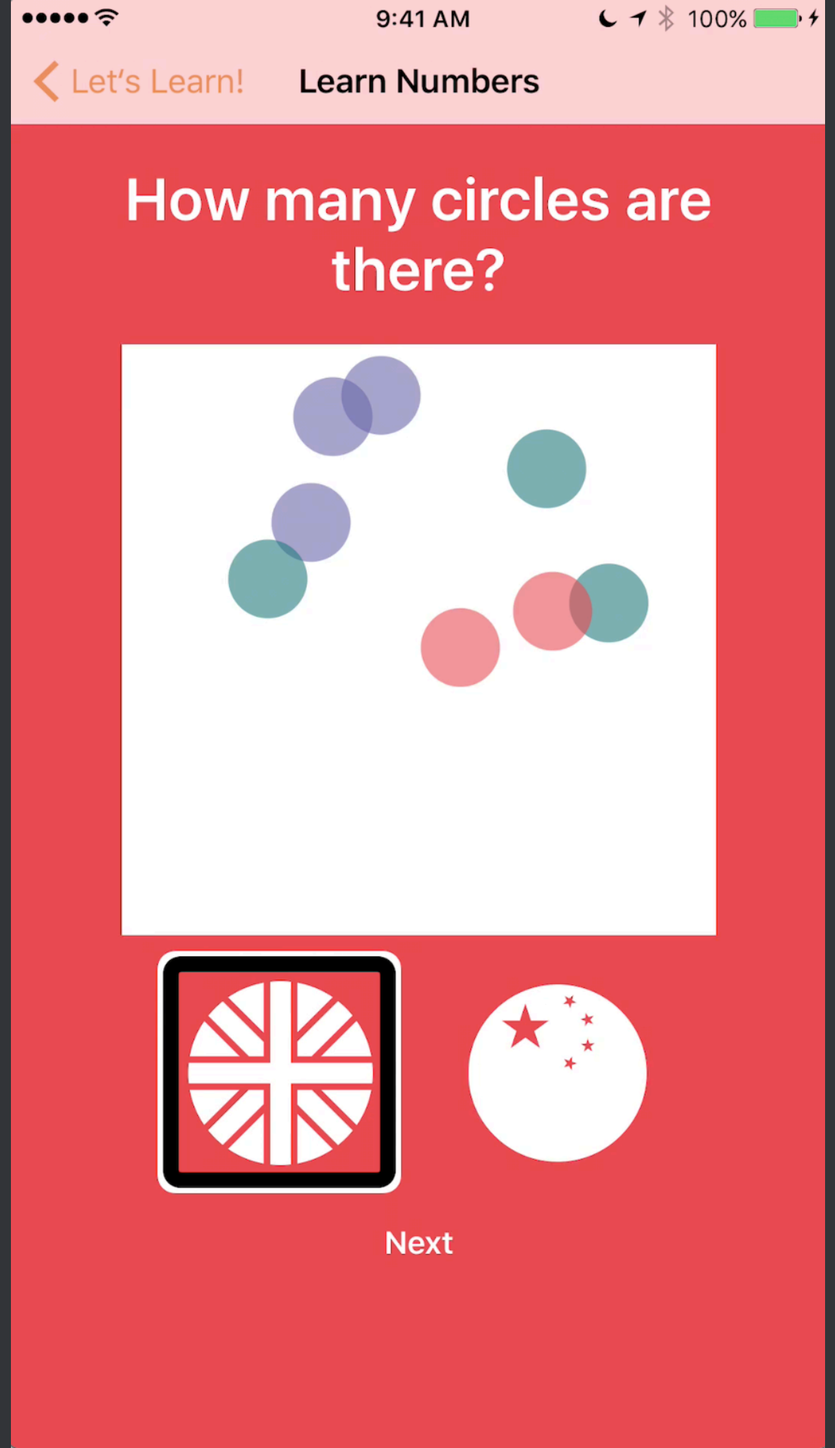
Button code after

```
let englishButton: UIButton = {
    let button = UIButton(type: .system)
    button.setBackgroundImage(#imageLiteral(resourceName: "union_jack")
        .withRenderingMode(.alwaysTemplate), for: .normal)

    button.accessibilityLabel = "English Answer"
    button.accessibilityHint = "Shows the answer in English"

    return button
}()
```

Issues in our app

- Can't see our items in the white box!

We need to provide custom Accessibility Elements for these lesson drawings!



Custom Accessibility Elements

- Sometimes we don't use UIViews (i.e. custom drawing)
- Create a custom **`UIAccessibilityElement`** in a container UIView for each object we want VO to see
- Our view must:
  - have **`isAccessibilityElement = false`**
  - set **`accessibilityElements`** to our array of **`UIAccessibilityElement`** objects

Custom Accessibility Elements

Each **`UIAccessibilityElement` must** have an **`accessibiliyLabel`** and some form of frame:

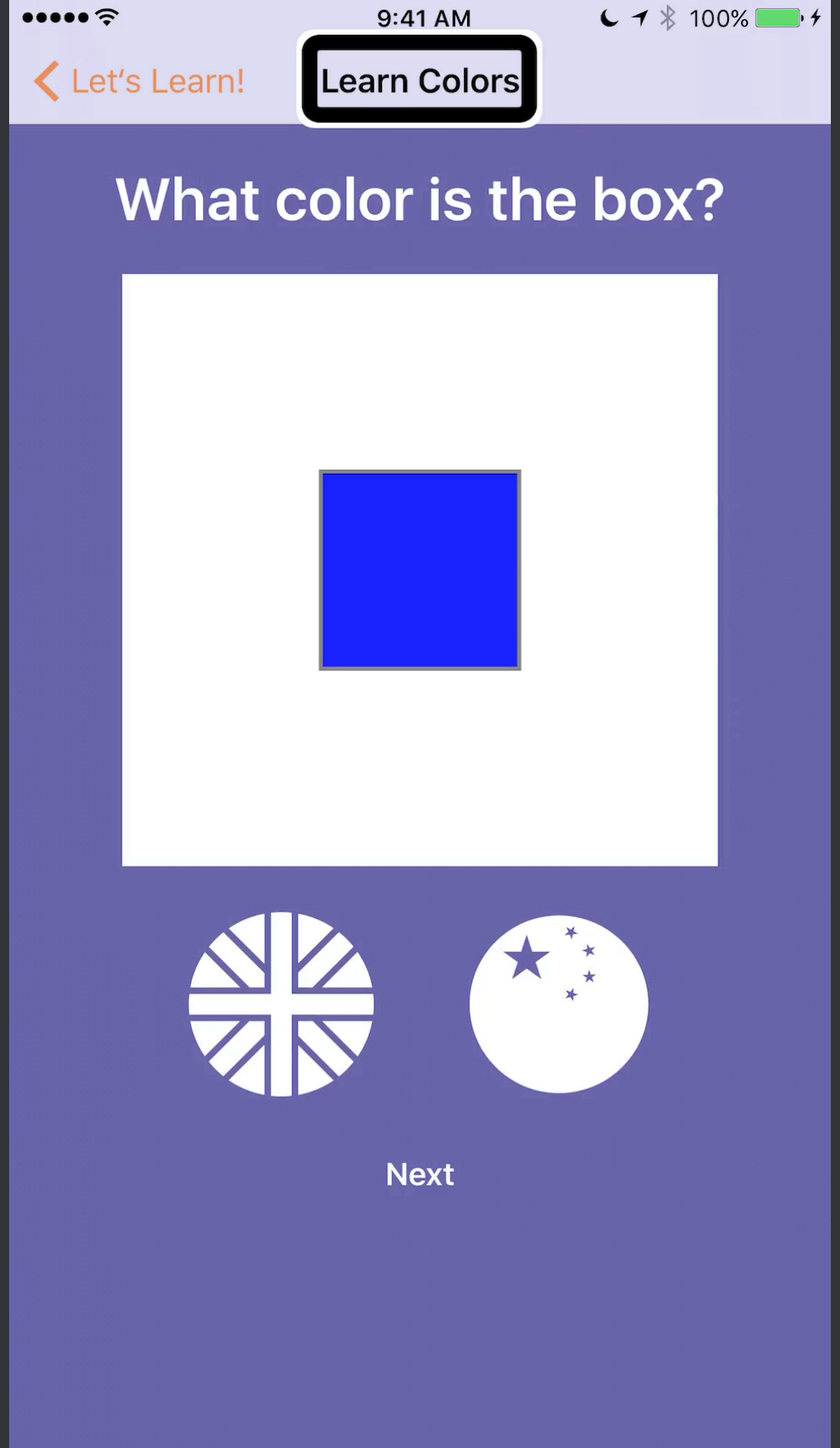
- **`accessibilityFrame`**: in screen coordinates
- **`accessibilityFrameInContainerSpace`**: in container coordinates
- **`accessibilityPath`**: in screen coordinates

Drawing before

```
protocol Lesson {
    var english: String { get }
    var chinese: String { get }
    func draw(in view: UIView)
}
```

Drawing before

```
// Draw a square filled with a given color
private static let squareDimension: CGFloat = 100
func draw(in view: UIView) {
    let rect = view.bounds
    let d = ColorLesson.squareDimension
    let context = UIGraphicsGetCurrentContext()
    let color = self.uiColor
    context?.setFillColor(color.cgColor)
    context?.setStrokeColor(UIColor.gray.cgColor)
    context?.setLineWidth(2.0)
    let x = (rect.width - d) / 2.0 + rect.minX
    let y = (rect.height - d) / 2.0 + rect.minY
    let colorRect = CGRect(x: x, y: y, width: d, height: d)
    context?.fill(colorRect)
    context?.stroke(colorRect)
}
```



Creating a custom UIAccessibilityElement

```
let element = UIAccessibilityElement(
    accessibilityContainer: parentView)
element.accessibilityFrameInContainerSpace = CGRect(
    x: 0, y: 0, width: 100, height: 100)
element.accessibilityLabel = "Box, the color of the sky & ocean"
parentView.accessibilityElements = [element]
```

Drawing after

```
protocol Lesson {
    var english: String { get }
    var chinese: String { get }
    func drawAccessibly(in view: UIView) -> [UIAccessibilityElement]
}
```

Drawing after

```
private static let squareDimension: CGFloat = 100
func drawAccessibly(in view: UIView) -> [UIAccessibilityElement] {
    // drawing code here!

    let element = UIAccessibilityElement(accessibilityContainer: view)
    element.accessibilityFrameInContainerSpace = colorRect
    element.accessibilityLabel = accessibilityDescription
    return [element]
}
```

Drawing after

```
private class LessonView: UIView {
    // ivars, etc.

    init(lesson: Lesson, drawAccessiblyForLesson: @escaping (Lesson, UIView) -> [UIAccessibilityElement]) {
        self.lesson = lesson
        self.drawAccessiblyForLesson = drawAccessiblyForLesson
        super.init(frame: .zero)

        isAccessibilityElement = false

        backgroundColor = .white
    }

    override func draw(_ rect: CGRect) {
        let a11yElements = drawAccessiblyForLesson(lesson, self)
        accessibilityElements = a11yElements
    }
}
```

Issues in our app

- Not clear that "Got it!" goes back

We need to (a) add hint to the button and (b) support the default back gesture for VoiceOver

Special accessibility gestures

APIs to cusomize

- Scrolling announcements (3-finger scroll)
- Magic Tap (2-finger double tap)
- Escape gesture (2-finger Z-shape)

A closer look at Escape

- Two-finger Z shape
- Supported automatically by `UINavigationController`
- Must be manuallys supported for modals via `accessibilityPerformEscape()`

A closer look at Escape

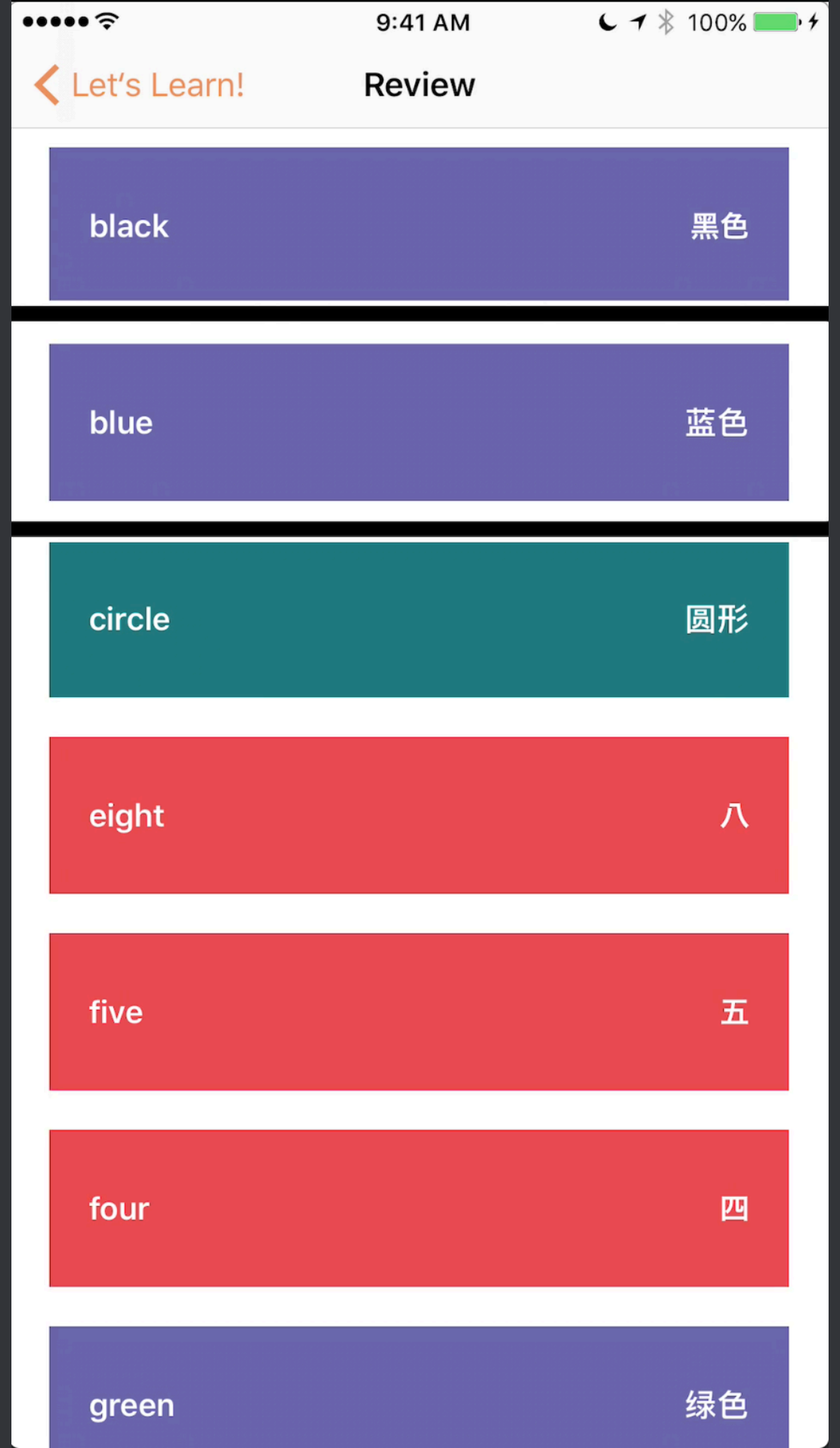
```
final class AnswerViewController: UIViewController {
    // ivars, init, etc...

    override func viewDidLoad() {
        super.viewDidLoad()

        doneButton.addTarget(self, action: #selector(didSelectDone), for: .touchUpInside)

        // setup views here...
    }

    @objc private func didSelectDone() {
        dismiss(animated: true, completion: nil)
    }

    override func accessibilityPerformEscape() -> Bool {
        didSelectDone()
        return true
    }
}
```

Escape in action!

Issues in our app

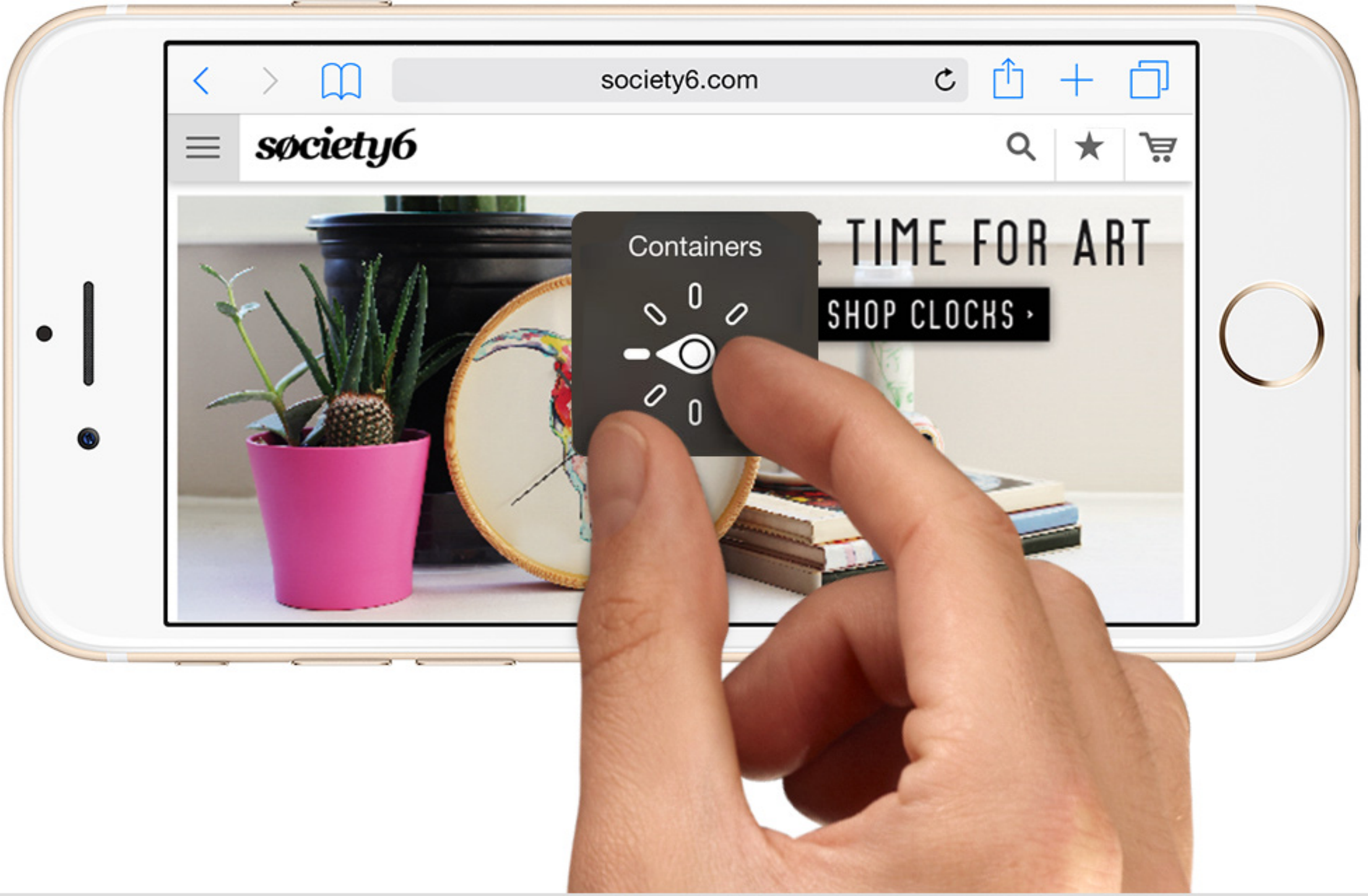
- No audio differentiation of review items

We need to provide a way to navigate by shape, color, or number!



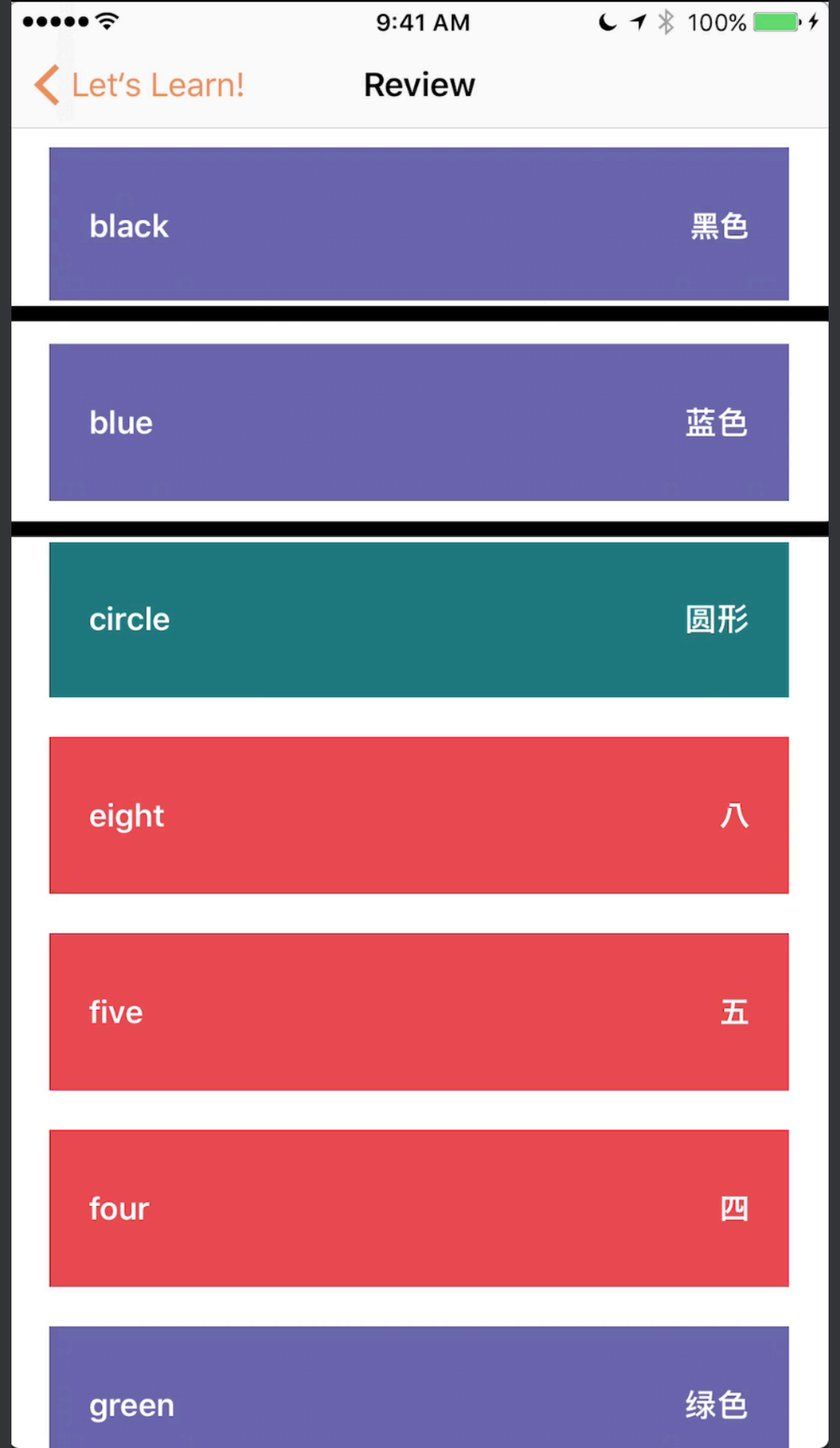
The Rotor: new power in iOS 10

- Primarily for navigation
- Navigate by headings, words, letters, etc.
- iOS 10 lets us add our own navigation keys


society6.com
society6
TIME FOR ART
SHOP CLOCKS ›
Containers

Navigation by Rotor

- Select item, e.g. "colors"
- Swipe up, down to go to preverious/next item
- Matches the expereince of sighted users, who can use color to visually scan for different cell types



Rotor code

```
final class ReviewViewController: UIViewController {
    // ivars, etc.

    init(items: [ReviewItem]) {
        // sort items alphabetically in Engish
        self.items = items.sorted { itemA, itemB in
            itemA.englishText < itemB.englishText
        }
        super.init(nibName: nil, bundle: nil)
        // other setup...

        setupRotors()
    }
}
```

Rotor code

```
private func setupRotors() {
    let categories = Set(items.map { $0.rotorCategory })
    let rotors = categories.map { category in
        UIAccessibilityCustomRotor(name: category, itemSearch: { (predicate) -> UIAccessibilityCustomRotorItemResult? in
            guard !self.items.isEmpty else { return nil }

            let forward = predicate.searchDirection == .next

            // figure out starting point
            var currentIndex = forward ? -1 : self.items.count
            if let cell = predicate.currentItem.targetElement as? UITableViewCell {
                currentIndex = self.tableView.indexPath(for: cell)?.row ?? currentIndex
            }

            // helper for search
            func next(index: Int) -> Int { return forward ? index + 1 : index - 1 }

            var index = next(index: currentIndex)
            while index >= 0 && index < self.items.count {
                if self.items[index].rotorCategory == category {
                    let indexPath = IndexPath(row: index, section: 0)
                    self.tableView.scrollToRow(at: indexPath, at: .none, animated: false)
                    let cell = self.tableView.cellForRow(at: indexPath)!
                    return UIAccessibilityCustomRotorItemResult(targetElement: cell, targetRange: nil)
                }
                index = next(index: index)
            }
            return nil
        })
    }
    accessibilityCustomRotors = rotors
}
```

Rotor code

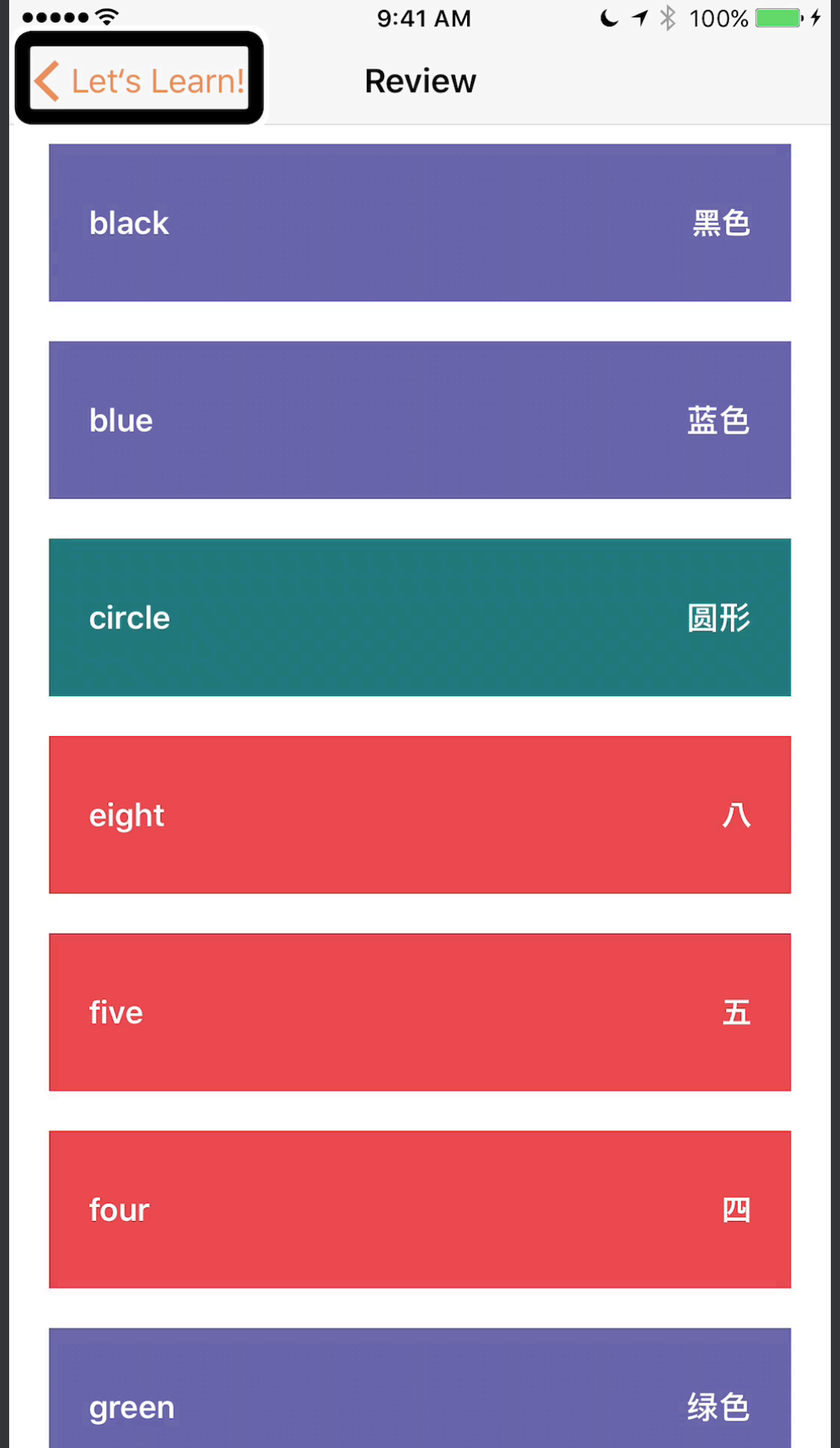
```
private func setupRotors() {
    let categories = Set(items.map { $0.rotorCategory })
    let rotors = categories.map { category in
        // create a rotor for each category
    }
    accessibilityCustomRotors = rotors
}
```

Rotor code

```
UIAccessibilityCustomRotor(name: category, itemSearch: { (predicate) -> UIAccessibilityCustomRotorItemResult? in
    guard !self.items.isEmpty else { return nil }

    let forward = predicate.searchDirection == .next

    // figure out starting point
    var currentIndex = forward ? -1 : self.items.count
    if let cell = predicate.currentItem.targetElement as? UITableViewCell {
        currentIndex = self.tableView.indexPath(for: cell)?.row ?? currentIndex
    }

    // ... and more ...
})
```

Rotor code

```
// helper for search
func next(index: Int) -> Int { return forward ? index + 1 : index - 1 }

var index = next(index: currentIndex)
while index >= 0 && index < self.items.count {
    if self.items[index].rotorCategory == category {
        let indexPath = IndexPath(row: index, section: 0)
        self.tableView.scrollToRow(at: indexPath, at: .none, animated: false)
        let cell = self.tableView.cellForRow(at: indexPath)!
        return UIAccessibilityCustomRotorItemResult(targetElement: cell, targetRange: nil)
    }
    index = next(index: index)
}
return nil
```



Rotor code, explained

1. Determine if you're going forwards or backwards
2. Figure out your starting point
3. Search from your starting point to find the next item that matches the rotor selection.
4. Return the item wrapped as a `UIAccessibilityCustomRotorItemResult` or `nil` if there's no such item

Summary

1. Accessibility helps **all** users overcome challeneges
2. Audit your app!
3. Basic accessibility - labels and traits!
4. Advanced accessibility - gestures and rotors!

Beyond VoiceOver

- Switch Systems
- Braille
- Captioning
- Accessible Design
- And much more...

In conclusion...

谢谢

Thank you!

Contact Sommer Panage

- sommer@panage.org
- **@sommer** on Twitter