HOME WORK #1

Task description:

You shall create an app in line with Google's style.

Screen 1:

- One button to change colors.
- 6 to 10 views in variety of shapes and forms.

Once the button is tapped each view change its color to a different unique one.

During this task:

- You are not allowed to use React, SwiftUI, Objc-C or anything except vanilla Swift.
- You are allowed to use any architecture or not to use one.
- You can only use StoryBoard to layout the app.

Grading:

- **0 Points:** The task was not submitted or it contains nothing.
- **1 Point:** The task has nothing to do with the assignment.
- **2 Points:** The project has the button and all the views.
- **3 Points:** When the button is pressed, views change colors.
- **4 Points:** Each view has its own unique color.
- **5 Points:** The button can be used more than once.
- **6 Points:** Views have different shapes and rounded corners.
- **7 Points:** The color change is done using animation.
- **8 Points:** The button is disabled during the color changing animation.
- **9 Points:** The app looks similar on devices with different screen size.
- **10 Points:** The random color is generated in HEX and then converted to SRGB using UIColor extension.

If your project contains **interesting solutions**, exceptional code style, good presentation, easter eggs, jokes, unique references or well designed clean code (patterns, paradigms) up to two bonus points might be added. This means the maximum grade a student can receive per homework is 12. If interesting solution repeats in different projects bonus points won't be added. It is pointless to try and prove to us that your solution is interesting. There won't be any appeals regarding bonus points.

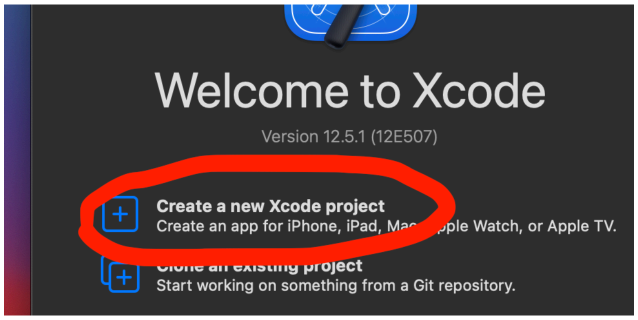
If the task is overdue, for each day **after deadline** your maximum grade goes down by 1. Bonus points might not be added for projects submitted late. The maximum penalty is 5 points. Outdated submission can get you up to 5 points.

The grade will be decreased for code style violation and bad code starting HW#3.

If you stumble across any questions feel free to contact teachers or assistants! Our goal is to help you develop skills required to create iOS Apps.

HW#1 Tutorial:

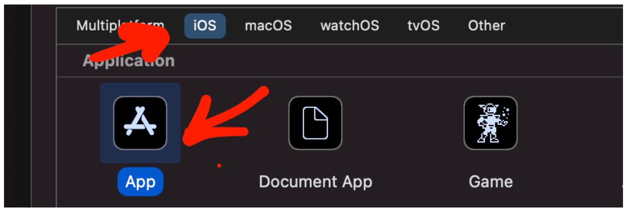
Open Xcode and create a new project on the starting screen (img. 1).



Welcome to Xcode
Version 12.5.1 (12E507)
Create a new Xcode project
Create an app for iPhone, iPad, Mac, Apple Watch, or Apple TV.
Clone an existing project
Start working on something from a Git repository.

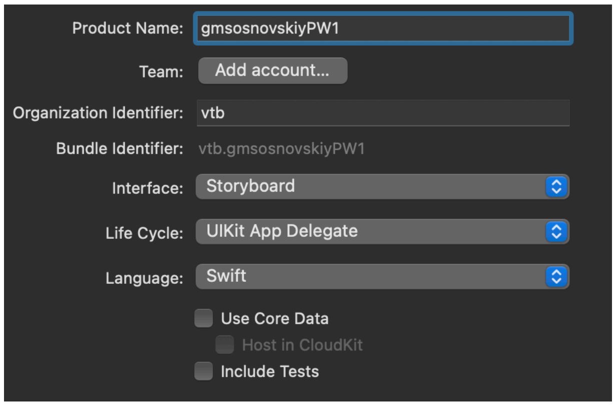
Img. 1

On the second screen please check that you are creating an app for iOS. Proceed with creating an app (Img. 2)



Img. 2

Name the app *Your_study_email_before_@*PW*homework_number* (Img. 3) and press continue.



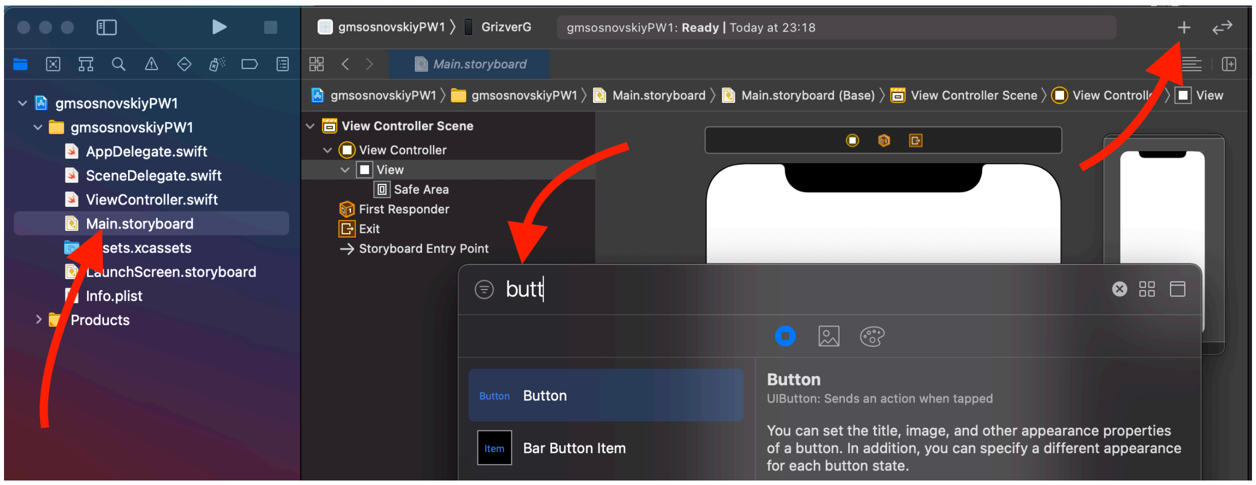
Product Name:	gmsosnovskiyPW1
Team:	Add account...
Organization Identifier:	vtb
Bundle Identifier:	vtb.gmsosnovskiyPW1
Interface:	Storyboard
Life Cycle:	UIKit App Delegate
Language:	Swift

- [ ] Use Core Data
  - [ ] Host in CloudKit
- [ ] Include Tests

Img. 3

Create the app.

Open Main.storyboard file and tap the plus button in the top right corner (Img. 4) and search for "Button".



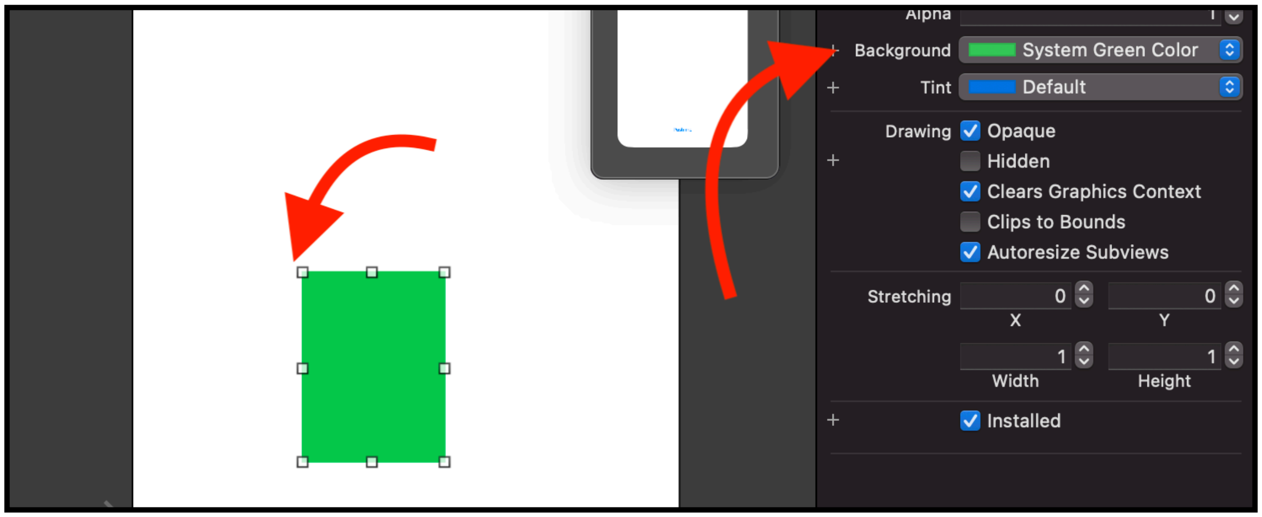
Button
UIButton: Sends an action when tapped

You can set the title, image, and other appearance properties of a button. In addition, you can specify a different appearance for each button state.

Img. 4

Drag&Drop the button to an iPhone screen (white screen on Img. 4, ViewController representation on storyboard)

Search for "View" the same way you found button. Change its size and position using corners. Change the background color to any color you like. (Img. 5)

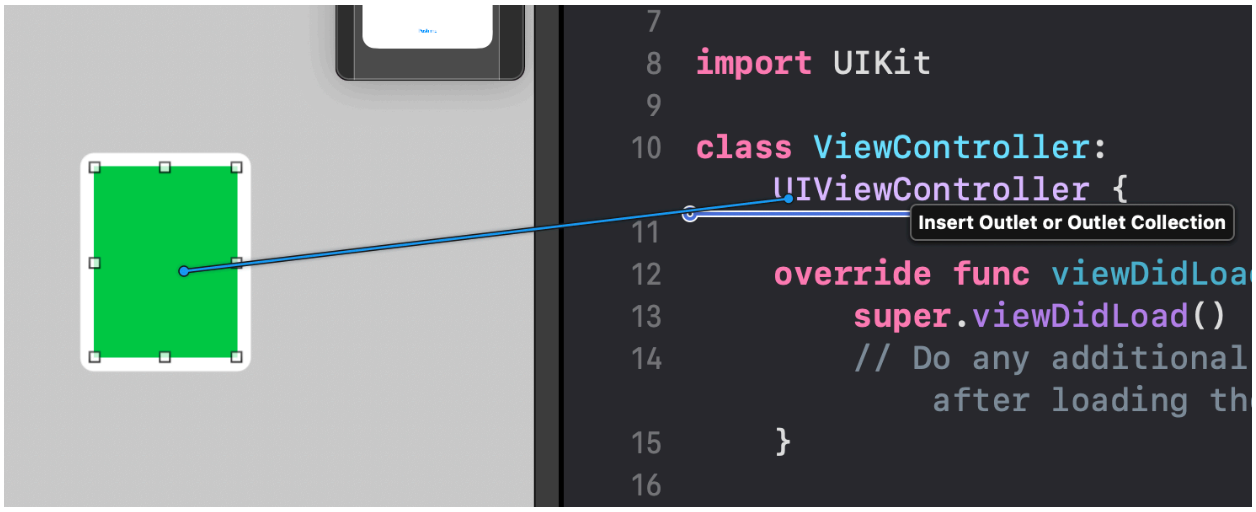


Background: System Green Color
Tint: Default
Drawing: ✓ Opaque, Hidden, ✓ Clears Graphics Context, Clips to Bounds, ✓ Autoresize Subviews
Stretching: X 0, Y 0, Width 1, Height 1
✓ Installed

Img. 5

Press down option (alt) and open ViewController file (above storyboard on Img. 4). It alows you to open two file on one screen at once.

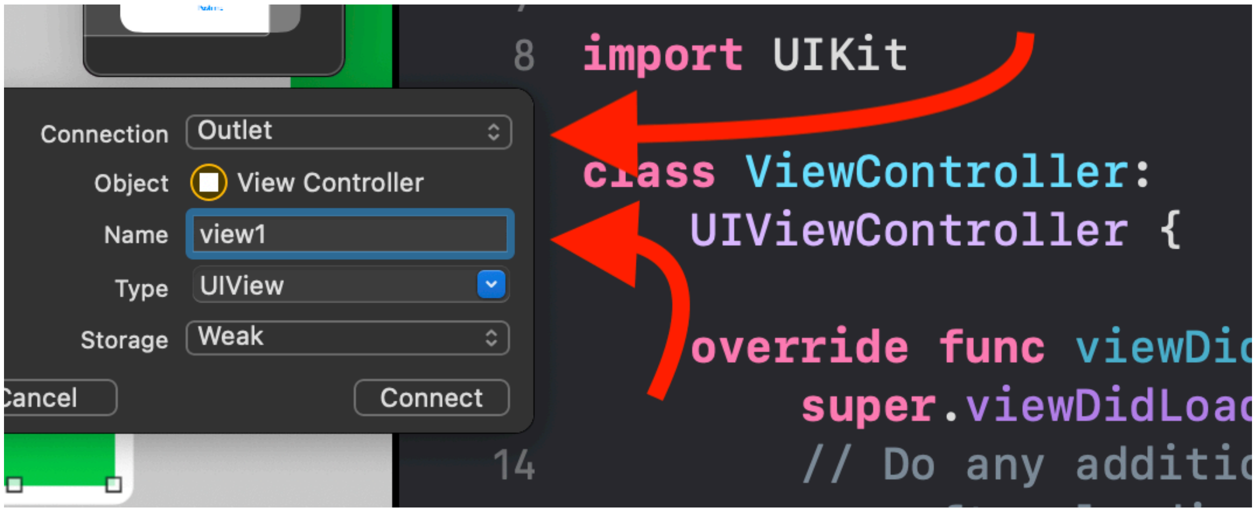
Press down control (do not mix up with command) "drag" our rectangle to the ViewController's, top (right after class definition) (Img. 6)



```
import UIKit

class ViewController:
    UIViewController {

    override func viewDidLoa
        super.viewDidLoad()
        // Do any additional
            after loading th
    }
```

Img. 6

This way we linked our view to class ViewController, and we can now work with it like it is class property. Let's call it view1 (Img. 7).

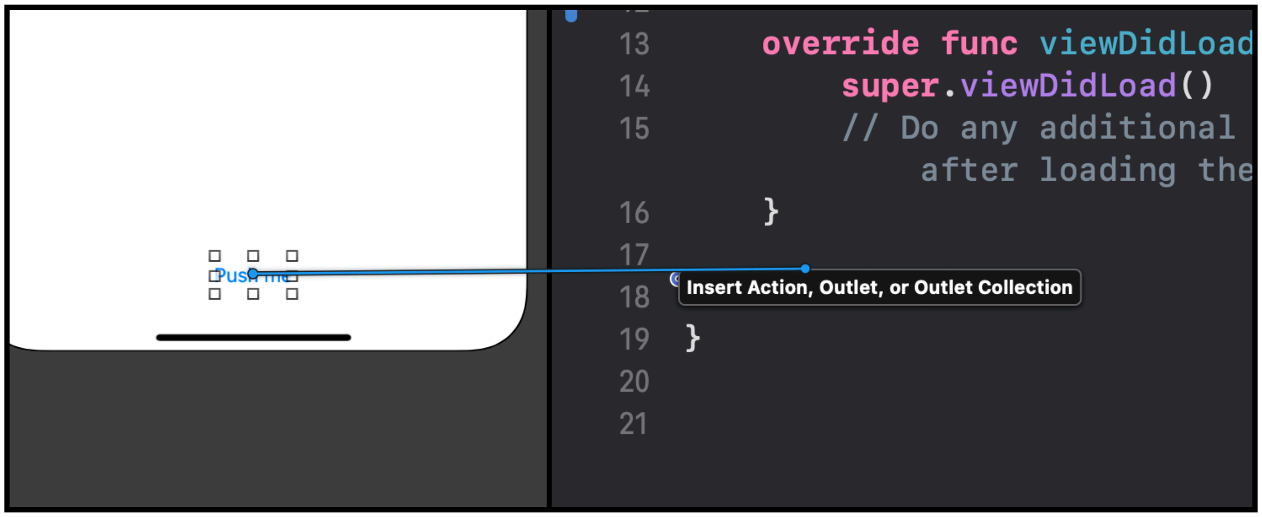


Connection	Outlet
Object	View Controller
Name	view1
Type	UIView
Storage	Weak

```
import UIKit

class ViewController:
    UIViewController {

    override func viewDi
        super.viewDidLoa
        // Do any additi
```

Img. 7

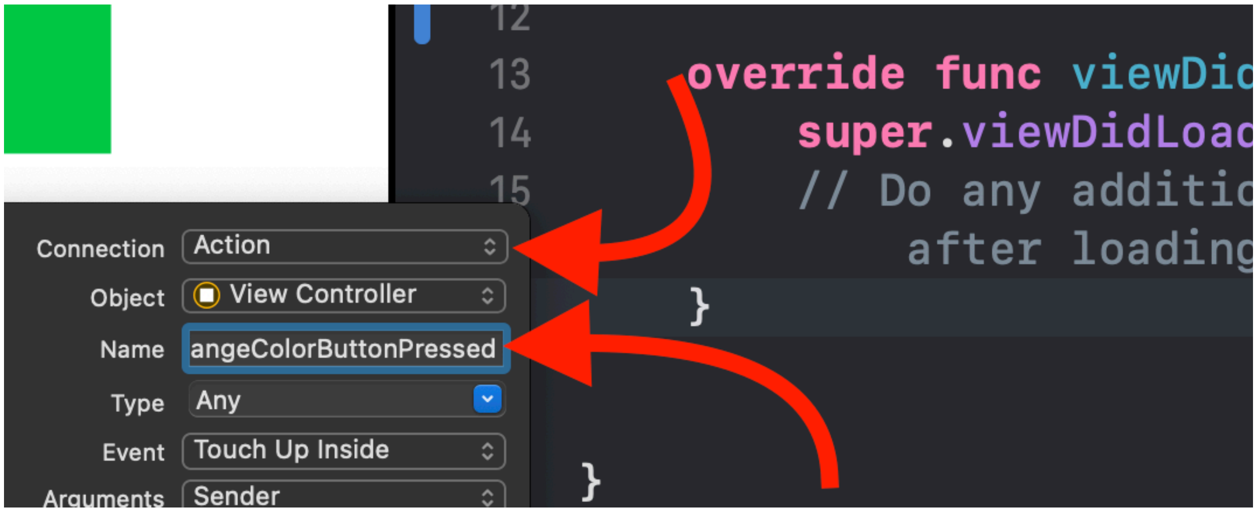
Drag our button like you did with view to ViewController’s bottom after viewDidLoad method. (Img. 8)



```
    override func viewDidLoad
        super.viewDidLoad()
        // Do any additional
           after loading the
    }

}
```

Img. 8

We will be able to create an Action instead of an Outlet (check if it is true, it is important). We will call it changeColorButtonPressed (Img. 9).



```
    override func viewDic
        super.viewDidLoac
        // Do any additic
           after loading
    }

}
```

Connection	Action
Object	View Controller
Name	angeColorButtonPressed
Type	Any
Event	Touch Up Inside
Arguments	Sender

Img. 9

This Action – method, that will be called once the button is tapped. Since we are supposed to change views color lets write some code:

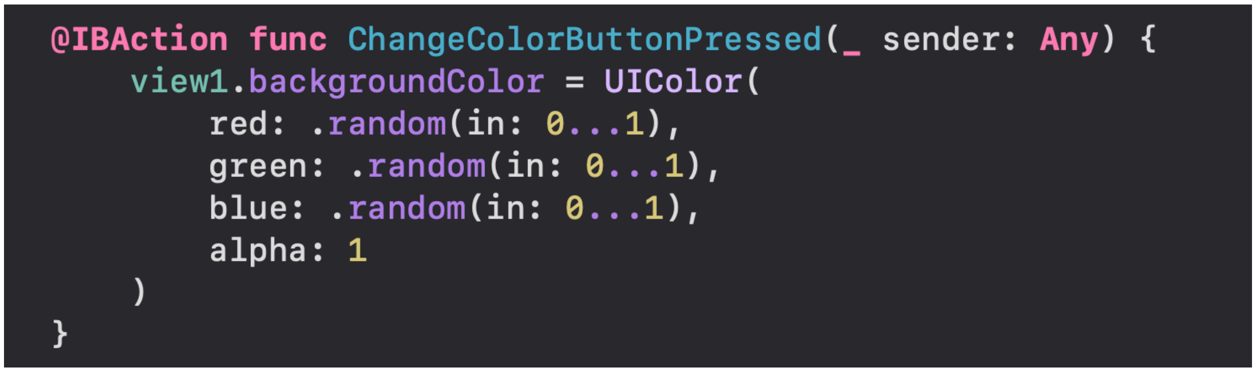
```
        view1.backgroundColor = UIColor(
            red: .random(in: 0...1),
            green: .random(in: 0...1),
            blue: .random(in: 0...1),
            alpha: 1
        )
```

Let's analyze this code. We change field backgroundColor that is UIColor to a new object of the same Type. We initialize color using constructor with four arguments: red, green, blue and alpha. First three are simply RGB parameters of type CGFloat. They are supposed to store value between 0 and 1. That is because it resembles any value between 0 and 255 divided by 255 (all possible RGB color values). If you would pass a greater value than 1 it will be the same as just passing 1. The fourth argument is alpha and it shows the transparency of an object (or color). It is also a value between 0 and 1. UIColor.clear is just color white with alpha equal to 0. Since we are after deep, rich, saturated colors let's keep it equal to 1.

CGFloat starting with Swift 4.2 has method random, that accepts an interval. New random number will be generated that belongs to the passed interval. In swift intervals are created using binary operands ..< ..> …, the last one needs only one custom number, either the lower bound or higher bound.

Swift is aware, that red, green etc. are CGFloat so we can write simply .random instead of CGFloat.random. All number types have random method.
That is how our view will have new background each time we press the button.

Now our method should look something like this: (Img. 10)



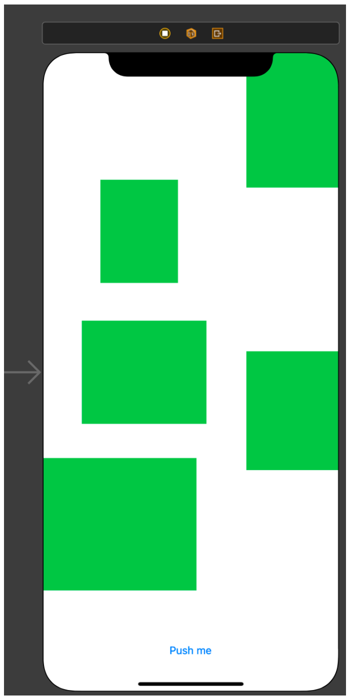
```swift
@IBAction func ChangeColorButtonPressed(_ sender: Any) {
    view1.backgroundColor = UIColor(
        red: .random(in: 0...1),
        green: .random(in: 0...1),
        blue: .random(in: 0...1),
        alpha: 1
    )
}
```

Img. 10

Pressing Command + R we can run our app and see how the view changes its color when the button is pressed.

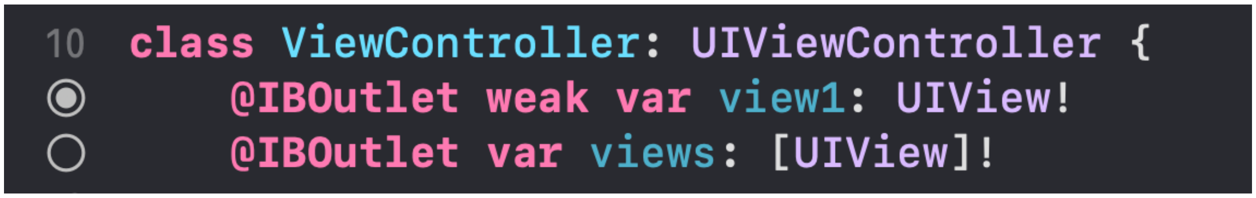
But we require many views. Let's copy our first view and place new views evenly on the screen (Img. 11).

Attention: your app must have 6 to 10 views and in our example you can see only 5. If you won't have 6 to 10 views you'll receive only 1 point.



Img.11

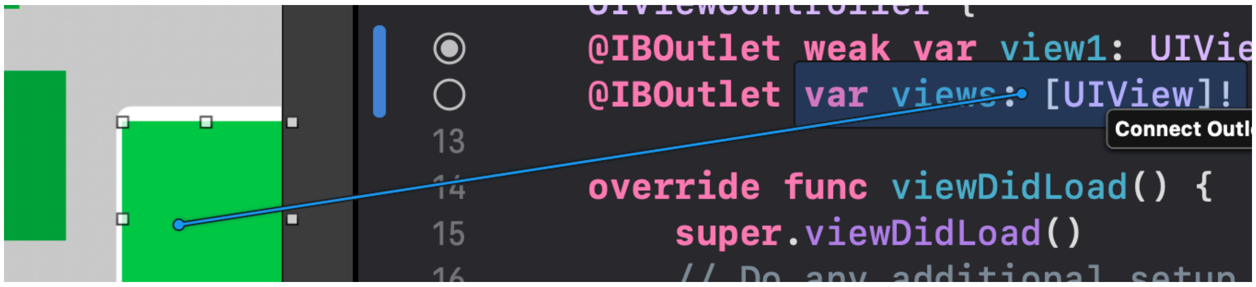
Creating an outlet for every view (Img. 7) is inconvenient. We would have 6 identical properties, view1…view6. That is hard to understand and to work with. It would be much better to allocate an array of UIViews (Img. 12)



```
class ViewController: UIViewController {
    @IBOutlet weak var view1: UIView!
    @IBOutlet var views: [UIView]!
```

Img. 12

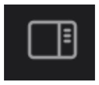
We can notice, that view1 has its circle full, while our array's circle is empty. This shows, that view1 has objects from storyboard linked to it, and that array does not. To add links we have to press down control again and drag&drop the line into our array. (Img. 13)



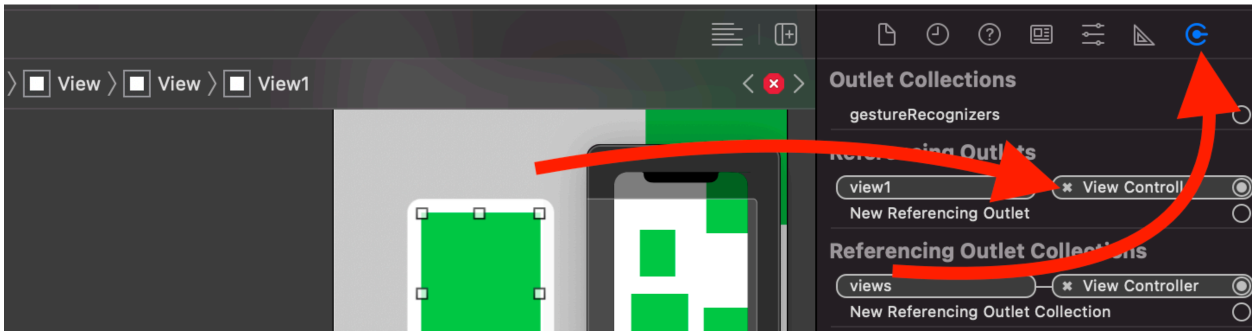
```
    @IBOutlet weak var view1: UIVie
    @IBOutlet var views: [UIView]!

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup
```

Img. 13

We need to do this with every view from storyboard, including the first one. Storing those views in an array allows us to create simple ways to change them all at once. We can now remove view1 from the code. Delete **@IBOutlet weak var** view1 UIView and remove the link it has with the view from storyboard. Ignoring the link leads to an error occurring. To remove link you shall find the first view, and in the right menu to remove the link in link section (img. 14) If you do not have the right or the left menu, they are toggled using top right and left buttons in the corners of Xcode. (Img. 13.5)



Img. 13.5



Img. 14

Now we need to change the code that changes colors. We want it to utilise the array we created. To gain 4 point anв above we need every view to have it's own color. Simple code like the one listed below:

```
        for view in views {
            view.backgroundColor = UIColor(
                red: .random(in: 0...1),
                green: .random(in: 0...1),
                blue: .random(in: 0...1),
                alpha: 1
            )
        }
```

Won't cut it. There is a small chance you might end up with two or more views being same color. To solve this problem we can use Set. In Swift to create set you can simply write Set<Type>, were type is a protocol saying what objects can be stored in the set. Since set can store unique objects only, it won't allow two identical colors. The resulting code would be:

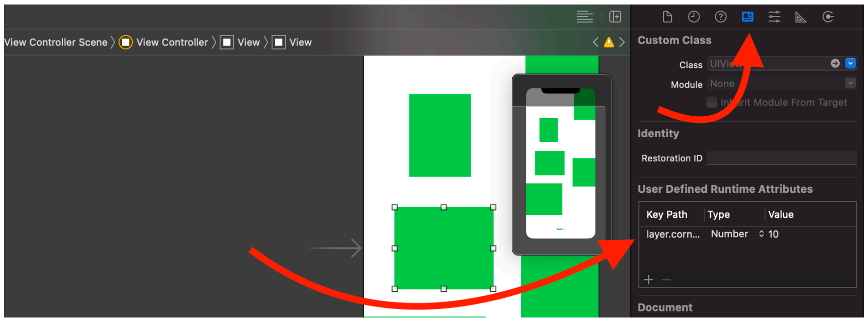
```
        var set = Set<UIColor>()
        while set.count < views.count {
            set.insert(
                UIColor(
                    red: .random(in: 0...1),
                    green: .random(in: 0...1),
                    blue: .random(in: 0...1),
                    alpha: 1
                )
            )
        }
```

```
        for view in views {
            view.backgroundColor = set.popFirst()
        }
```

This code can be optimised in many ways, but it shows how set, for and while work. Here you can start being creative for bonus points.

Let's run our app and press the button to see that all views change their colors to unique new ones.

We have earned 5 points! It is not "Great" though. Let's create rounded corners to achieve 6 points. You can go ahead and do it for every view manually (Img. 15)



Key Path	Type	Value
layer.corn...	Number	10

Img.15

To do this you can open “User Defined Runtime Attributes” for every view and set layer.cornerRadius to type number and change the value (for example 10). To add a new line to the attributes press the plus button in the bottom left corner (Img. 15). Since I have as many as five views it is easier for me to set it from code:

```
        for view in views {
            view.layer.cornerRadius = 10
            view.backgroundColor = set.popFirst()
        }
```

To earn 7 points we’d have to animate the color change. Animation is a method belonging to UIView class. It is **important** to know, that any change to app’s UI, including animation, must happen on the main thread. Threads will be studied later. UIView has method UIView.animate(withDuration: TimeInterval, animations: {}). It allows to animate UI changes. The fist parameter is time to animate in seconds (type TimeInterval that is typealias to double). The second parameter is the animation itself. We will talk about parameters storing code aka delegate later. Our color change animation would look something like this:

```
            UIView.animate(withDuration: 1, animations: {
                view.backgroundColor = set.popFirst()
            })
```

This code makes color change animated, but pressing the button rapidly will make colors flash without an animation. To stop it let’s disable the button until the animation is over. In the start of our button action lets insert “changeColorButton.isEnabled = false” and add “changeColorButton.isEnabled = true” in the end. To do so we need to create button outlet, like we did with view1 (Img. 6).
Or we can look closely to action signature. It has parameter sender of type any. We know the action sender is UIButton, so we can downcast (look it up) sender to UIButton from any. It can be done like this:

```
        let button = sender as? UIButton
        button?.isEnabled = false
```

Here you can see an “as?”. This is a way to cast types in swift. We are trying to create a button immutable variable of type UIButton from sender. We will talk about ‘?’ aka optional later. If the sender is not a UIButton button will be nil (null in c#):

```
        button?.isEnabled = true
```

Run the app and rapidly press the button. Sadly the button becomes disabled only for a split second, and then becomes enabled again. This happens because animation is asynchronous, for finished calling all animations and enabled = true was called before animations have finished. To avoid it we have to wait for animation completion. Let’s put for inside of animation and use UIView.animate(withDuration: TimeInterval, animations: {}, completion: {}) instead of the previous animation method. The code will resemble this:

```
        UIView.animate(withDuration: 2, animations: {
            for view in self.views {
                view.layer.cornerRadius = 10
                view.backgroundColor = set.popFirst()
            }
        }) { completion in
            button?.isEnabled = true
        }
```

We will talk about completion in and why we need self before views later.

Conclusion:

If we run the app right now it would start without colors or rounded corners. To avoid that we can decompose our action to a different method and **call it** from viewDidLoad after super.viewDidLoad.

If you successfully finish this tutorial you can get up to **8 points**. This is an excellent grade, but to earn more you need to do leftover tasks on your own. You would have to learn about constraints, autoresizing and autolayout. You would have to try extensions, show some creativity and show off your programming skills. **Astonish** homework inspectors and gain bonus points!

Lots of steps of this tutorial is purely educational and can be skipped, but it won't be possible without reading the task first, before doing it head on. Next time you might want to **read the task prior to doing it**.

Students who already are able to animate and create UI can call the teacher for the **harder** home task.