UIView

In iOS, a **view** (UIView) is a rectangular area on the screen that displays content.

UIView is the foundational class for creating visual elements in iOS apps.

In UIKit, everything we see on the screen (buttons, labels, images) is a type of **UIView**. It is the base class for all UI elements in UIKit.

Hierarchy:

When you create a UIViewController, it comes with a default **main view** (view property). This view is automatically created by the system. We just customize or add subviews to it.

mainView (superview) – created by system

└── view2 (subview of mainView, superview of view3)

└── view3 (subview of view2)

Syntax:

let myView = UIView(frame: CGRect(x: 20, y: 50, width: 200, height: 100))

// constraints

self.view.addSubview(myView)

// Adding and Removing views

let myView = UIView()

view.addSubview(myView) // Add a subview

myView.removeFromSuperview() // Remove the subview

**Properties and Options of UIView**

|  |  |  |
| --- | --- | --- |
| **Property** | **Description** | **Example** |
| frame | The view’s size and position in its parent’s coordinate system. | view.frame = CGRect(x: 10, y: 20, width: 100, height: 50) |
| bounds | The view’s size and position in its own coordinate system. | view.bounds = CGRect(x: 0, y: 0, width: 100, height: 50) |
| center | The view’s center point relative to its parent. | view.center = CGPoint(x: 100, y: 100) |
| backgroundColor | The background color of the view. | view.backgroundColor = .red |
| alpha | The opacity level of the view (0.0 to 1.0). | view.alpha = 0.5 |
| isHidden | Hides the view if true. | view.isHidden = true |
| clipsToBounds | Clips subviews to the view’s bounds if true. | view.clipsToBounds = true |
| layer.cornerRadius | Rounds the corners of the view. | view.layer.cornerRadius = 10 |
| layer.borderWidth | Adds a border around the view. | view.layer.borderWidth = 2 |
| layer.borderColor | Sets the border color (requires CGColor). | view.layer.borderColor = UIColor.black.cgColor |
| layer.shadowColor | Adds a shadow to the view (requires CGColor). | view.layer.shadowColor = UIColor.black.cgColor |
| layer.shadowOpacity | Controls shadow opacity (0.0 to 1.0). | view.layer.shadowOpacity = 0.5 |
| layer.shadowRadius | Controls the blur radius of the shadow. | view.layer.shadowRadius = 5 |
| transform | Applies 2D or 3D transformations to the view (rotation, scaling, translation). | view.transform = CGAffineTransform(scaleX: 2.0, y: 2.0) |
| contentMode | Defines how content is positioned/scaled within the view’s bounds. | view.contentMode = .scaleAspectFit |
| safeAreaInsets | Insets defining the area of the view that avoids system bars (like the notch or home indicator). | let insets = view.safeAreaInsets |

**Z-Order**

* The **Z-order** of a view determines its stacking order relative to other sibling views.
* Views added later using addSubview are rendered on top of earlier ones.
* You can reorder views using methods like bringSubviewToFront and sendSubviewToBack.

self.view.bringSubviewToFront(myView)

self.view.sendSubviewToBack(anotherView)

2. UILabel

UILabel is used to display a **static or dynamic text**.

Syntax:

let label = UILabel()

label.text = "Hello, UILabel!"

self.view.addSubview(label)

**Properties and Options of UILabel**

|  |  |  |
| --- | --- | --- |
| **Property** | **Description** | **Example** |
| text | The string displayed in the label. | label.text = "Hello, World!" |
| textColor | The color of the text. | label.textColor = .blue |
| font | The font of the text. | label.font = UIFont.boldSystemFont(ofSize: 20) |
| textAlignment | The alignment of the text within the label. | label.textAlignment = .center |
| numberOfLines | The maximum number of lines the label can display. 0 means unlimited. | label.numberOfLines = 2 |
| lineBreakMode | How the label truncates or wraps text. | label.lineBreakMode = .byTruncatingTail |
| attributedText | An attributed string allowing custom styles (colors, fonts) for portions of text. | label.attributedText = NSAttributedString(...) |
| adjustsFontSizeToFitWidth | Automatically adjusts the font size to fit the label’s width. | label.adjustsFontSizeToFitWidth = true |
| minimumScaleFactor | The smallest scale factor for text when using adjustsFontSizeToFitWidth. | label.minimumScaleFactor = 0.5 |
| baselineAdjustment | Adjusts the text's baseline relative to the label's bounds. | label.baselineAdjustment = .alignCenters |
| highlightedTextColor | The color of the text when the label is highlighted. | label.highlightedTextColor = .red |
| shadowColor | The color of the text shadow. | label.shadowColor = .gray |
| shadowOffset | The offset of the shadow relative to the text. | label.shadowOffset = CGSize(width: 2, height: 2) |
| isHighlighted | A Boolean indicating whether the label is highlighted. | label.isHighlighted = true |
| preferredMaxLayoutWidth | The maximum width the label can occupy, used with Auto Layout. | label.preferredMaxLayoutWidth = 200 |

UIButton

UIButton is a UIControl subclass that allows users to interact with your app by tapping it. It displays text, an image, or a combination of both, and supports different styles and states (normal, highlighted, disabled, etc.).

Syntax:

let button = UIButton(type: .system) // or .custom, .roundedRect, etc.

button.setTitle("Click Me", for: .normal)

button.backgroundColor = .systemBlue

button.addTarget(self, action: #selector(buttonTapped), for: .touchUpInside)

self.view.addSubview(button)

@objc func buttonTapped(){

performSome()

}

**Properties and Options of UIButton**

|  |  |  |
| --- | --- | --- |
| **Property** | **Description** | **Example** |
| setTitle(\_:for:) | Sets the button's title for a specific state. | button.setTitle("Submit", for: .normal) |
| setTitleColor(\_:for:) | Sets the title color for a specific state. | button.setTitleColor(.gray, for: .disabled) |
| setImage(\_:for:) | Sets an image for a specific state. | button.setImage(UIImage(named: "icon"), for: .normal) |
| contentEdgeInsets | Adds padding around the button’s content (title or image). | button.contentEdgeInsets = UIEdgeInsets(top: 10, left: 20, bottom: 10, right: 20) |
| titleEdgeInsets | Adds padding around the button’s title. | button.titleEdgeInsets = UIEdgeInsets(top: 0, left: 10, bottom: 0, right: 0) |
| imageEdgeInsets | Adds padding around the button’s image. | button.imageEdgeInsets = UIEdgeInsets(top: 0, left: 0, bottom: 0, right: 10) |
| backgroundColor | Sets the background color of the button. | button.backgroundColor = .systemRed |
| layer | Used to customize the button’s border, corner radius, or shadow. | button.layer.cornerRadius = 10 |
| isEnabled | Enables or disables the button. | button.isEnabled = false |
| isHighlighted | Indicates if the button is currently being tapped. | button.isHighlighted (read-only) |

Examples:

**Example**: A button with an image (or, icon) instead of text.

let imageButton = UIButton(type: .custom)

let buttonImage = UIImage(named: "icon.png")

imageButton.setImage(buttonImage, for: .normal)

imageButton.frame = CGRect(x: 100, y: 300, width: 50, height: 50)

imageButton.addTarget(self, action: #selector(imageButtonAction), for: .touchUpInside)

self.view.addSubview(imageButton)

@objc func imageButtonAction() {

print("Image button tapped!")

}

**Example**: A button with both text and an image aligned horizontally.

let combinedButton = UIButton(type: .custom)

combinedButton.setTitle("Play", for: .normal)

combinedButton.setImage(UIImage(named: "play\_icon.png"), for: .normal)

combinedButton.frame = CGRect(x: 50, y: 400, width: 200, height: 50)

combinedButton.imageView?.contentMode = .scaleAspectFit

combinedButton.titleEdgeInsets = UIEdgeInsets(top: 0, left: 10, bottom: 0, right: 0)

combinedButton.addTarget(self, action: #selector(combinedButtonAction), for: .touchUpInside)

self.view.addSubview(combinedButton)

@objc func combinedButtonAction() {

print("Combined button tapped!")

}

**Example**: A button that shows an action sheet when tapped.

let alertButton = UIButton(type: .system)

alertButton.setTitle("Show Action", for: .normal)

alertButton.frame = CGRect(x: 50, y: 800, width: 250, height: 50)

alertButton.addTarget(self, action: #selector(showActionSheet), for: .touchUpInside)

self.view.addSubview(alertButton)

@objc func showActionSheet() {

let actionSheet = UIAlertController(title: "Options", message: "Choose an option", preferredStyle: .actionSheet)

actionSheet.addAction(UIAlertAction(title: "Option 1", style: .default, handler: nil))

actionSheet.addAction(UIAlertAction(title: "Cancel", style: .cancel, handler: nil))

self.present(actionSheet, animated: true, completion: nil)

}

UITextField

UITextField is a UI component used to accept a single line of input from the user, such as text, passwords, or email addresses. It supports placeholders, text formatting, keyboard input, and various styles.

Syntax:

let textField = UITextField()

textField.frame = CGRect(x: 50, y: 100, width: 250, height: 40)

textField.placeholder = "Enter your name"

textField.delegate = self // for handling events

self.view.addSubview(textField)

**Properties and Options of UITextField**

|  |  |  |
| --- | --- | --- |
| **Property** | **Description** | **Example** |
| text | The text displayed in the field. | textField.text = "Hello" |
| placeholder | Placeholder text displayed when the field is empty. | textField.placeholder = "Enter text" |
| textColor | The color of the text. | textField.textColor = .black |
| font | The font of the text. | textField.font = UIFont.systemFont(ofSize: 16) |
| textAlignment | The alignment of the text. | textField.textAlignment = .center |
| borderStyle | Defines the border type (none, line, bezel, roundedRect). | textField.borderStyle = .roundedRect |
| isSecureTextEntry | Hides the text for password-like input. | textField.isSecureTextEntry = true |
| keyboardType | Specifies the keyboard type (default, numberPad, emailAddress, etc.). | textField.keyboardType = .emailAddress |
| returnKeyType | Specifies the return key type (done, next, search, etc.). | textField.returnKeyType = .done |
| clearButtonMode | Determines when the clear button appears (never, while editing, unless editing, always). | textField.clearButtonMode = .whileEditing |
| leftView / rightView | Adds custom views to the left or right of the text field. | textField.leftView = UIImageView(image: ...) |
| delegate | Handles interactions like text changes, beginning or ending editing. | textField.delegate = self |

UITextView

UITextView is used to display and edit multi-line, scrollable text. It supports text formatting, links, text selection and editing.

Syntax:

let textView = UITextView()

textView.frame = CGRect(x: 20, y: 100, width: 300, height: 200)

textView.text = "This is a UITextView"

textView.font = UIFont.systemFont(ofSize: 16)

self.view.addSubview(textView)

**Properties and Options of UITextView**

|  |  |  |
| --- | --- | --- |
| **Property** | **Description** | **Example** |
| text | The text displayed in the view. | textView.text = "Hello, World!" |
| font | The font of the displayed text. | textView.font = UIFont.boldSystemFont(ofSize: 18) |
| textColor | The color of the text. | textView.textColor = .blue |
| isEditable | Determines if the text can be edited. | textView.isEditable = false |
| isSelectable | Allows text selection. | textView.isSelectable = true |
| dataDetectorTypes | Automatically detects links, phone numbers, etc. | textView.dataDetectorTypes = .all |
| textAlignment | Alignment of the text (left, center, right, justified). | textView.textAlignment = .center |
| attributedText | Allows rich-text formatting using NSAttributedString. | textView.attributedText = NSAttributedString(string: "Styled Text") |
| backgroundColor | Sets the background color of the text view. | textView.backgroundColor = .lightGray |
| delegate | Handles events like text editing, scrolling, and interactions. | textView.delegate = self |
| contentInset | The padding inside the text view. | textView.contentInset = UIEdgeInsets(top: 10, left: 10, bottom: 10, right: 10) |