**Apple Vocabulary**

**Unit 1**

* Console: Programmers use the console as a message center to show details about the way a program is running.
* Open Source: An open source language is developed in the open, with community input and support. Everyone is welcome to contribute or simply to follow along.
* Playground: A playground is an environment in Xcode for experimenting with code and seeing instant results.
* Results sidebar: The results sidebar in a playground shows information about each line of code that produces an output value. As you add or change code, the playground runs the code again and updates the results in the sidebar.
* Bool: A Boolean type has only two possible values: true or false. Booleans are named after George Boole, a 19th century mathematician who realized how important is to ask clear questions with simple answers. In Swift, the Boolean type is called *Bool*.
* Constant: A constant is an immutable identifier. Once a constant is assigned a value, the value can never change; trying to assign a new value to an existing constant will generate an error. In Swift, a constant is declared with *let*.
* Comment: A readable explanation meant to help explain surrounding code Comments are not executed as part of the program. In Swift, comments are created by adding `//` in front of the comment text.
* Double: A Double is a number with a decimal point, like 7.78 or 1000.0. If you initialize a constant or variable in Swift to a literal number with a decimal point (like let distance = 26.2), the type is inferred to be a Double.
* Function: A function combines lines of code into a single block that can be used again. Some functions have inputs and change their behavior based on arguments from their caller, some functions have outputs and give a result, and some functions have neither.
* Immutable: An immutable object cannot be updated or changed. A constant is immutable; a variable is mutable.
* Int: An Int stores an integer, or whole number, such as 10 or 42.
* Let: let allows you to store a value in a named constant. The value of a constant doesn’t need to be known at compile time, but you must assign it a value once (and only once).
* Mutable: A mutable object can be updated or changed. A variable is mutable; a constant is not.
* Property: A property is a piece of data held by a structure, class, or enumeration. For example, each array instance has a count property that differs depending on the characteristics of the array.
* Type Inference: Swift uses type inference to work out an object’s type from the available information, even when the name of the type isn’t explicitly put into words in code. For example, if a function is declared to return a String, like func answer () -> String, and a constant is initialized with the answer function’s result, like let nextIdea = answer (), the type of nextIdea will be inferred to be a String.
* Type Safety: Swift has type safety to prevent writing code that uses types incorrectly or unexpectedly. For example, if a function expects a type Fruit and you try to call it with a Vegetable, your app or playground won’t run until you fix the error.
* Variable: A variable represents a value that can change after it’s been initialized, indicated in Swift by the var keyword.
* Var: var allows you to store a value in a named variable. The value of a variable can be mutated or changed.
* Compound Assignment: You use compound assignment to update the value of a variable in a way that depends on its current value, without typing the variable’s name twice. For example, += is shorthand for a compound assignment with addition (+); it checks the current value of the variable on the left-hand side, calculates the sum of that current value plus the value on the right-hand side, and then assigns the calculated sum as the variable’s new value. If you don’t want to use compound assignment, you can write.
* Operator: An operator is a symbol – such as +, -, or && - that represents an action on one or more values. For example, in 1 + 2, the addition operator (+) adds the number 1 and 2.
* Conditional: A conditional runs multiple checks and runs code based on the results. These conditional statements are part of a broader concept called control flow. As a developer, you have control flow tools that check for certain conditions and execute different blocks of code based on those conditions.
* If Statement: An if statement is a code structure for executing code based on the value of one or more conditions. The first block of code in an if statement is the if block. An if statement might also contain other blocks that provide additional checks, such as an else block and an else if block.
* If- else statement: An if-else statement is a code for executing code if a specified condition is true and for executing another block of code if its not true.
* Interval matching: When working with numbers, you can use interval matching to check for inclusion in a range.
* Logical Operator: A logical operator is a symbol – such as &&, | |, or ! – that modifies or combines the Boolean logic values true and false.
* Switch: The Switch operator keyword chooses between different courses of action based on a value’s characteristics. The options for the values are written in case statements. Once the switch statement finds the first match between the value and a case, the block of code next to the case statement is executed. Once one of the blocks has been executed, the program continues running the next code after the end of the entire swift statement.
* Ternary: A ternary is a control flow operator that can be used to assign or return a value depending on a Boolean comparison.
* Active Scheme: A single Xcode project can create multiple apps, for example, an iOS application and a watchOS application. The activity scheme tells Xcode which target to build.
* Assistant Editor: The Assistant editor is a second window in Xcode that displays a side-by-side editing interface.
* Compiler: A compiler processes statements in code into an executable program.
* Console pane or console area: Is a command-line environment in Xcode where you can debug or otherwise interact with an app.
* Debug area: The debug area allows you to inspect the current values of constants or variables in scope.
* Executable: An executable file contains an app’s compiled code. The name of your app’s executable file is the same as your app name minus the .app extension.
* Git: Git is a version control system used for keeping track of different branches of code and all modifications to the code.
* IDE or integrated development environment: An IDE is a tightly integrated software suite that consolidates developer tools for writing and testing apps.
* Project: Xcode opens project files that include all the code and assets used to compile a Swift or app.
* Project navigator: on the leftside of Xcode user interface, lists all files associated with your project, making it easy to move around your project files.
* Project template: Xcode includes a number of project templates or starting points for beginning a new app.
* Push notifications: A push notification is a message sent from the Apple Push Notification service (APNs) to an app that isn’t running in the foreground. The notification may be displayed as a user notification or it may be silently delivered, allowing the receiving app to wake up temporarily to process new data.
* Standard editor: The standard editor displays the currently selected file project navigator.
* Storyboard: A storyboard is a user interface file that tells the app where to display information on the screen. Xcode opens storyboards in interface builder and you edit storyboard by clicking, dragging, and choosing options in the utilities area.
* Target: A target represents the instructions for building a single finished product from a set of files, for example, a framework, a library, an application, or a command-line tool.
* Utilities area: The utilities area in Xcode displays the inspector pane and the library pane.
* Variables view: The variables view lists the value of each variable during active debugging.
* Version editor: A version editor displays side-by-side views of the current version and previous, when using a source control system (such as Git) to manage Xcode files.
* Breakpoint: A breakpoint is an international stopping or pausing place in a program. Programmers use breakpoints to check values or perform other debugging operations.
* Bug: A bug is an error in code that prevents an application from running as expected.
* Deprecated code: Refers to code that’s no longer supported or won’t be supported in future versions of iOS.
* Developer Mode: In Xcode, developer mode provides access to certain debugging features without requiring you to enter your password each time.
* Compiler error or error: An Xcode compiler error usually stems from incorrect syntax that prevents an app from compiling and running. Xcode displays a red error icon next to the offending code.
* Exception: An exception is also known as an error.
* Landscape: Landscape refers to an horizontal screen orientation, as in a landscape painting or photograph.
* Portrait: Portrait refers to a vertical screen orientation, as in a portrait painting or photograph.
* Step control buttons: You can use the step control buttons at the top of the debug area to step through code execution slowly.
* Warning: A compiler warning – represented visually by an error in the code – does not prevent the app from being compiled but should be addressed in order to compile and run the program safely.
* Documentation Browser: Is an Xcode tool for looking up and reading developer documentation.
* Quick Help: Feature in Xcode that provides a fast answer by displaying a popover with a brief description of the function and the OS version that support it. It also shows which framework the function belongs to.
* Symbol: Symbols are methods or properties on a given type.
* Action: An action, or IBAction, connects a method in source code and a control in Interface Builder, allowing particular code to run when a user interacts with the app’s controls. For example: a certain method may be associated with an action, such as a button tap or a switch change.
* Canvas: The canvas refers to the backdrop of a storyboard, where you add and arrange scenes and user interface elements.
* Document Outline: Is a feature in Interface Builder that displays a list of all scenes and UI elements in a given storyboard.
* Initial View Controller: The IVC is the first scene that will be viewed by the user when the app launches. In storyboards, the initial view controller is marked with a gray arrow, but you can also set the initial view controller in the Attributes inspector for an individual scene.
* Outlet: An outlet, or an IBOutlet, connects a variable in source code to and object in the storyboard, allowing the code to get information from the object and to make changes as the app is running.
* Scene: A scene is a storyboard representation of a screen of content in your app.
* View controller: A VC manages a set of views and coordinates the flow of information between the app’s data model and the views displaying the data.
* XIB: An XIB file represents a single scene or view in iOS, and can be opened and modified in Interface Builder. XIB files were more popular before storyboarding was introduced in iOS 5.

**Unit 2**

* Case sensitivity: Something is case-sensitive if it matters whether characters are uppercase or lowercase.
* Concatenate or Concatenation: Refers to the operation of joining two strings together.
* Equality: Use the == operator to check equality if two values are equal to each other. You can allow custom types to check for equality by adopting the Equatable protocol.
* Escape character: Use this character (\) in a string to tell Swift that It should treat what comes next as special. This symbol is called an escape character because it avoids, or escapes from the normal behavior of a string.
* Index: The index is the numbered position of an item in an ordered collection. The index of the first item is always 0, and the index of the last item will equal the total number of items minus one.
* Literal: A literal valie is typed or inserted directly into code with no initializers. A literal string would be “hello”, and a literal away would be [1,2,3]. A non-literal string might be initialized with String(), and a non-literal integer might be initialized with homeTeamPoints + awayTeamPoints.
* Range: A range represents a group of numbers from a minimum value to a maximum value.
* String Interpolation: Use string interpolation to put the value of a constant or a variable in the middle of a string literal. You add placeholders for constants or variables with an escape sequence that puts parentheses around the code to be replaced. For example, if friendName has been set as “Lee”, the expression “Have a good one, \(friendName)” will automatically be read as “Have a good one, Lee”.
* Substring: Is a portion of a string.
* Unicode: The Unicode standard defines a way to represent almost any character from any language in a consistent way. Member so the Unicode committee decide which new characters including emoji, to add to the Unicode standard and how to support them.
* Argument Label: An argument label introduces the argument passed into the function, just like you might write your name (“Maya”) next to a name label (“Name”) on a form. So if a function is declared as func drawLine(from fromPoint: Point, to toPoint: Point), the wrods “from” and “to” are argument labels because they’ll label the argument values when the function is called, as in drawLine(from: startPoint, to: endpoint).
* Parameter: A parameter refers to the input value, or values, inside the function. For example, if a function is declared as func drawLine(from fromPoint: Point, to toPoint: Point), the words fromPoint and toPoint are parameter names, because those are the names to be used in the function’s implementation. (Informally, developers may use parameter interchangeably with an argument).
* Return type: A return type indicates the type of value that will be return by a function or method.
* Return value: The return value is the value that’s returned, or given back to the caller, from a function or method.
* Computed property: Swift has a feature called computed property that allows a property to perform logic that returns a calculated value.
* Function: A function combines lines of code into a single block that can be used again. Some functions have inputs and change their behavior based on arguments from their caller, some functions have outputs and give a result, and some functions have neither.
* Initialization: To use a type, you must create an instance of it, a process called initialization.
* Initializer: An initializer sets up an instance so it’s ready to be used. After you declare a name for a constant or variable, you initialize the constant or variable by assigning the first value.
* Instance method: An instance method is a function declared inside a type such as a struct, a class, or an enum whose behavior depends on the current state of an individual entity.
* Memberwise initializer: When you create a new instance of a structure, Swift automatically creates a memberwise initializer- which you can then use to set each property on the new instance by name.
* Method: A method is a function defined inside a type. Methods can use the date stored in the type’s properties to do work.
* Property: A property is a piece of data held by a structure, class, or enumeration. For example, each array, instance has a count property that differs depending on the characteristics of the array.
* Self: The self keyword refers to the specific instance of an object, and is used when calling a method, or getting or setting a property within the definition of a type.
* Structure: A structure is a data type that’s similar to a class but doesn’t support inheritance and is passed by value instead of by reference.
* Type: A type is a named grouping of properties (the features) and methods (the behaviors) of a kind of data. A type name in Swift is always written in capital camel case, like Double, String, or BicycleRoute.
* Base Class: A base class is at the root of its class hierarchy (and therefore doesn’t have a superclass).
* Class: A class defines properties to store values, defines methods to provide functionality, and defines initializers to set up their initial state.
* Inheritance: When a class is a subclass of another class, it gets all of its behavior (methods, properties, and other characteristics) from its superclass. This is referred to as inheritance.
* State: State refers to the stored information of a variable, program, or system at a given time.
* Subclass: A subclass is a child of another class (which is known as its superclass).
* Superclass: A superclass is a parent of another class (which is known as its subclass).
* Array: You use an array to hold a list of items of the same type, keeping them in order. Items in an array don’t have to be unique.
* Dictionary: Like a real-world dictionary that contains a list of words and their definitions, a Swift dictionary is a list of keys, each with an associated value. For optimization purposes, the values in a dictionary do not have a reliable or consistent order (unlike a real-world dictionary) but are looked up using the key they’re assigned to.
* For loop: A for loop is a block of code that runs a certain number of times, offering a quick and easy way to do the same thing repeatedly.
* For-in-loop: A for in loop executes on each item in a collection, looping over the collection. The block of code inside the loop will be executed once for each property.
* While loop: A while loop is a block of code that runs for as long as a given condition remains true. When the condition changes to false, the loop stops running.
* Button: A button intercepts touch events and sends an action message to a target object when its tapped.
* Control: A control, or a UIControl instance, is a specialized type of view that responds to user input.
* Control Event: Controls are simple, straightforward, and familiar to users because they appear throughout most iOS apps. A control event is triggered when a button is tapped or when the value of a control changes.
* Date picker: the date picker is a control that allows the user to select a specific data and/or time.
* Image view: An image view displays an image file in a user interface.
* Label: A label is a view that displays one or more lines of read-only-text.
* Navigation Bar: A navigation bar controls navigation or hierarchical content, usually displayed at the top of the screen and used in conjunction with a navigation controller.
* Scroll View: A scroll view allows the user to see content that runs beyond the boundaries of the view. Use a scroll view when the information you want to display is larger than the device’s screen.
* Segmented control: A segmented control narrows the search results to a particular media type.
* Slider: A slider control is an interface feature that allows the user to select a single value between a minimum and a maximum number.
* Tab bar: A tab bar provides easy access via multiple tab bar item, to different views in an app.
* Table View: A table view user interface displays several items or records as a list.
* Text field: A text field, or UITextField, displays an editable text area in the app interface. The iOS Keyboard is configurable for many different types of input.
* Toolbar: The toolbar along the top of the Xcode window shows the current project’s status, along with options to hide/show other window areas, choose devices for running the app, and build/run the app.
* UIKit: This framework provides the crucial infrastructure needed to construct and manage iOS and tvOS apps. This framework provides the window and view architecture needed to manage an app’s user interface; the event-handling infrastructure needed to respond to user input and the app model needed to drive the main run loop and to interact with the system.
* Aspect Ratio: Aspect ratio refers to the relationship of the width to the height of an image or screen.
* Clipping: You may encounter a warning that clipping has occurred in your layout. In most cases, you can fix clipping issues using Auto Layout in Interface Builder.
* Content Mode: Choose the content mode of a view to adjust the way it fits inside the space you’ve allowed it. If an image file is larger than its image view’s frame, for example, the content mode. scaleToFill will squash and skew the image so it’s exactly the size of the frame, while the content mode .top will show as much of the top of the image as can fit.
* Dynamic data: Dynamic data is data that changes.
* Frame: A view’s frame describes its location as a rectangle a certain distance from the top and a certain distance from the left of tis parent view.
* Static Data: Data that does not change is static data.
* Gesture recognizer: A gesture recognizer is an object that will listen for the user to input a common gesture and will call an action when the particular gesture is recognized.
* Auto Layout: Use Auto layout in Interface Builder to build adaptive interfaces, so your user interface elements maintain the same relative positions, no matter the screen’s size or orientation. For example, you can add one rule that a button must always be a certain distance above an image view and another rule that the image view must always be centered at the bottom of the screen. By defining the rules in Auto Layout, the two elements will follow those rules – whether the screen is large or small, in portrait or landscape mode.
* Constraint: A constraint is a rule in Auto Layout that defines how views should be laid out or sized.
* Sibling: In a scenario with parent objects and child objects, two child objects have a sibling relationship. When discussing views, sibling views share a parent view or exist at the same level of a view hierarchy.
* Size class: Size classes represent the different sizes you can use to adapt your interface where height and wight are either small or regular size.
* Stack view: Use a stack view to set up elements in the user interface in a column from top to bottom or in a row from left to right.

**UNIT 3**

* Failable initializer: A failable initializer refers to an initializer that may fail and will return nil as a result.
* Force-unwrap: Use the force-unwrap operator (!) after an optional value to access its underlying value. Force-unwrapping an optional value will result in a crash if the value is nil.
* Implicitly unwrapped optional: This optional is an optional that can also be used like a nonoptional value—without the need to unwrap the optional value each time it is accessed—because it’s assumed to always have a value after that value is initially set. Your app will crash if you attempt to access an implicitly unwrapped optional tha’ts currently nil.
* Nested optional: An optional value can have optional properties (like a box within a box), referred to as nested optionals.
* Nil: nil indicates that a value is missing.
* Optional: An optional is a value that contains either an underlying value or nil (to indicate that the value is missing).
* Optional binding: Is the process of attempting to assign an optional value to a constant in a conditional statement to see if the optional contain an underlying value.
* Optional chaining: Allows you to work with nested optionals that are conditionally unwrapped. If any value along the way is nil, the action will fail.
* Any: Any, as the name implies, can represent an instance of any type: string, double, function, or whatever.
* AnyObject: AnyObject can represent any class within Swift, but not a structure.
* as! : as! Is a forced form of a type cast operator. This version will force the downcast to the specified type.
* as?: as? Is a conditional cast operator that attempts to downcast the value to a more specific type and store it in a new constant or variable.
* Conditional cast: A conditional cast casts an instance to the specific type only if it’s possible to do so.
* Downcast: Use downcast to cast an object to a more specific type—for example, casting from a generic Any to an Int, or from a superclass to a subclass.
* Type casting: Is a way to check the type of an object, or to treat that object as a different superclass or subclass from somewhere else in its own class hierarchy.
* Type inspection: is used to determine or check the specific type of data.
* Guard: A guard statement declares a condition that must be true in order for the code after the guard statement to be executed. Using a guard statement for requirements improves the readability of your code, as opposed to using an if statement for the same check.
* Guard let: same as guard.
* Global scope: A global scope variable has been defined at the top-level scope, or global scope, of a program.
* Local scope: A local variable has been defined within a specific block of code or within the local scope of a program.
* Scope: A type, variable, constant, or function is accessible only within the scope it was assigned. By adding a pair of curly braces ({ }) for a function, an if statement, or for a loop, you’re specifying the area within the braces as local scope.
* Variable Showing: Occurs when a variable declared within a local scope has the same name as a variable declared in a global scope. The code that references the variable will refer to the instance in the nearest accessible scope.
* Case: In an enum, the case keyword declares a name for one of the enum’s options. In a switch, the case keyword introduces a pattern to try to match a value against.
* Default: A default option is selected when no other option is available. In a switch statement, the default block is executed if none of the other more-specific cases matches the input. In an if statement, the default option is the final else clause.
* Enum: The enum keyword declares a type made up of a group of related choices. An instance of an enum will be exactly one of the enum’s choices. The keyword enum comes from the word enumeration, which means listing distinct things one by one.
* Enumeration: An enumeration, or enum, is a data type that defines a group of relaed value and enables you to work with those values in a type-safe way within your code.
* Bar button: A bar button item represents an iteam on a UIToolbar or UINavigationItem object.
* Modal presentation: A modal presentation places a new view controller on top of the previous one. On smaller screens, a modal presentation will always appear at full screen you can customize a modal presentation to appear as popover, a form sheet, or a full-screen presentation.
* Modal segue: In a modal segue, one view controller presents another view controller as its child a modal presentation style. Use modal segues to present tasks that the user must complete before continuing.
* Navigation bar: A navigation bar controls navigation of hierarchical content, usually displayed at the top of the screen and used in conjunction with a navigation controller.
* Navigation controller: A navigation controller is a specialized view controller subclass that manages transitions backward and forward through a series of view controllers.
* Pop: Pop refers to removing the most recently added item to a stack. For example, you pop a view controller off the stack to exit from it. (And you’d push it onto a navigation stack to display it.)
* Push: Push refers to the addition of a new item to a stack. For example, you push a view controller onto a navigation stack to display it. (And you’d pop it off of the stack to exit from it).
* Root view controller: The first view controller or scene to be displayed. The bottom of the navigation stack.
* Segue: A segue to a transition from one scene to another in a storyboard.
* Show segue: A show segue varies the way new content is displayed based on the target view controller. For navigation controllers, the show segue pushes new content on top of the current view controller stack. Use a navigation controller and show segues to navigate through hierarchical data.
* Unwind segue: An unwind segue is called when dismissing a scene or view controller, and can be used to pass information to another view in the navigation stack.
* Badge: A badge is an indicator or a description on a tab bar item, typically used to communicate additional information.
* Flat hierarchy: A flat hierarchy refers to navigation where each is on the same contextual level, as with a tab bar controller.
* System item: iOS includes a number of system items that make it easy to add familiar tab bar items—for example, the More item and the Favorites item—to your tab bars.
* Tab bar: A tab bar provides easy access, via multiple tab bar items, to different views in the app.
* Tab bar controller: A tab bar controller allows the user to switch quickly between contexts or tasks at any point in their workflow.
* Tab bar item: A tab bar items one element or specific feature consisting of an icon image and text, within a tab bar.
* Implementation: An implementation puts a decision into effect. You implement a method or function by writing code that performs the work described in the name of the function.
* Override: Override refers to replacing the implementation of a method defined on a superclass.
* State refers to the stored information of a variable program or system at a given time.
* Navigation hierarchy: Navigation hierarchy refers to the workflow through your app. Flat navigation hierarchies are usually built with a tab bar controller, and hierarchical workflows are usually built with a navigation controller. Workflows can be combined to create a larger workflow or navigation hierarchy for a more complex app.
* Workflow: A workflow is a navigation hierarchy that allows the user to complete a given task or access a feature in your app.

**UNIT 3A**

* ARConfiguration: The base class for AR session configurations, such as ARWorldTrackingConfiguration and ARFaceTrackingConfiguration. These configurations establish a correspondence between the real world and a virtual 3D coordinate space.
* ARKit: The framework that integrates the device camera and motion features to produce augmented reality (AR) experiences.
* ARSCNView: A view for displaying AR experiences that augment the camera view with 3D SceneKit content.
* ARSession: An object that coordinates the major process that ARKit performs on your behalf to create an augmented reality experience.
* Origin: The starting coordinate (0,0,0) that can be used to position objects.
* Scene (SceneKit): The SceneKit scene to be displayed in the view.
* SceneKit: A framework that combines a high-performance rendering engine with a descriptive API, allowing 3D content to be controlled and displayed.
* Depth Fighting: When two geometries include identical values in the x, y, or z-axis, resulting in a graphical bug.
* Scene hierarchy: A tree structure that organizes 3D content as a set of nodes.
* SCNLight: A light source that can be attached to a node to illustrate the scene.
* SCNNode: An element of a scene graph that represents a position and transform in a 3D coordinate space. Geometries and lights may be attached to it.
* SCNVector: A three or four-component vector that describes node positions, sizes, and transformations.
* Anchor: The position of a tracked or detected object in the scene that can be used to place additional objects.
* Extent: The estimates width and length of a detected plane.
* Feature point: A point automatically identified by ARKit as part of a continues surface or detectable object.
* Plane detection: The act of detecting flat surfaces in the camera-captured image. The configuration option, planeDetection, specifies which planes should be detected.
* Force: A simulated “push” value that can be applied to physics bodies.
* Hit test: The transition of a screen tap to a ray, finding elements of a scene located along the line segment.
* Physics body: Physics simulation attributes that may be attached to a scene graph node.
* Physics shape: Determines the solid of the physics body for use in collision detection.
* AR Resource Group: A folder within an Xcode project’s asset catalog that contains a list of images that may be used for image detection.
* ARImageAnchor: Information about the position and orientation of an image detected in a world-tracking AR session.
* ARReferenceImage: An image to be recognized in the real-world environment during a word-tracking AR session.
* SCNAction: A reusable animation that can change the attributes of its associated node.

**UNIT 4**

* Adopt: In iOS development, when you adopt a protocol, you are implementing the properties or methods required by the protocol.
* Codable: A protocol that allows classes or structures to be encoded to or decoded from different types of data such as JSON or Plist data.
* Comparable: Comparable allows you to be define how instances of the same type are sorted.
* CustomStringConvertible: Allows you to control how your custom objects are printed to the console.
* Equatable: Allows you to define how instances of the same type are equal to each other.
* Implementation: An implementation puts a decision into effect. You implement a method or function by writing code that performs by the work described in the name of the function.
* Protocol: A protocol is a blueprint of methods, properties, and other requirements that suit a particular task or piece of functionality. HTTP and NSCoding are examples of protocols.
* Read-only: A read-only value can be viewed (read) but never changed (written).
* Read/write: A read/write value can be both viewed (read) and changed (written).
* Active: When an app is in active state, it’s running in the foreground and is receiving events. This is the normal mode for foreground apps.
* App delegate: An app delegate, or AppDelegate instance is an object that creates the window where your app’s content is drawn and provides a place to respond to state transitions within the app.
* App state: The app state indicates the status of the app, which includes not running, inactive, active, background, or suspended.
* Foreground: An app is considered to be in the foreground when it’s the currently running app.
* Abstraction: An abstraction groups things together so you think of the group as a single, easier to remember thing instead of trying to reason about each of the elements separately. Abstraction allows programmers to make decisions and think through code logic without getting overwhelmed by details and complexity.
* Architecture: Architecture refers to the framework for designing an application.
* Controller: A controller object acts as an intermediary between one or more of an app’s view objects and one or more of its model objects. In this way, controller objects act as a conduct through which view objects learn about model changes and vice versa.
* Model: Model refers to all the data types a particular app needs to define and handle.
* Model-View-Controller: The MVC pattern refers to an app design approach in which controllers serve as the communication pipeline between views and the data model.
* View: A view is an object that supports drawing and event handling in Cocoa.
* Bounds: A view’s visible layer should be confined to the bounds or limits of the view.
* Content inset: The content inset property specifies a buffer area around the content of the scroll view, so that controllers, toolbars, or keyboards don’t interfere with the user’s experience of the content.
* Content view: A content view is a view object at the top of a view hierarchy, serving as a container for the subviews below it.
* Accessory View: Table view cells have an accessory view that can be used to display additional information.
* Dequeue: Table views use a special aching system that, whenever possible reuses cells that have already been displayed. The process of choosing and preparing the cell for reuse is called dequeuing.
* Dynamic: Refers to dynamic data which is data that changes.
* Index path: An index path, or NSIndexPath class, represents the path to a specific node in a tree of nested array collections. An index path is commonly used with table views to identify the section and row of a cell.
* Readability margin: Auto Layout enables you to control the readability margin of your views across different device types.
* Reorder control: The reorder control allows users to move cells within a table view.
* Table view cell: A table view cell refers to objects that draw visible rows, and the table view cell caches those objects as long as the rows are visible.
* Zero-based: Arrays are zero-based, which means that the first item in the ordered list is at index zero and is accessed by using 0 as the subscript (array[0]).
* Custom table view cell: You create a custom table view cell when you create a subclass of UITableViewCell allowing you to add properties, outlets, or actions to your subclassed cell.
* Row action: One or more row actions can be displayed when the user swipes a table view cell.
* Static table view: A static table view has a predefined constant collection of cells and can be defined in a table view controller scene in a storyboard. Static table views are commonly used for form input.
* Archiving: Archiving is a method of saving information or data to a file saved on a disk.
* Documents directory: A documents directory is a safe place to store your app’s data since it’s accessible only by your app and can’t be modified by another app.
* Data: A type that represents generic binary data. Types that adopt the Codable or NSCoding protocols can be encoded to Data. Data may represent the contents of a file, such as an image, MP3, or PDF. Many Swift or UIKit types can be initialized using Data. For example, the init(data: Data) initializer for UIImage attempts to decode the raw Data into an image. If successful, the new image value can be used, otherwise the returned optional will be nil.
* Persist, persistence: Refers to data that outlasts the process that created it. To persist data in iOS development, you’ll often save data to disk or to a network service (and fetch the data when the user wants to view it again).
* Plist: A plist or property list is a file format for storing data that can be used in an app. Automatically created in all new Xcode app projects, an Info.Plist file includes required information for building, running, and signing an application.
* Sandboxing: In the sandboxing model, each app has its own environment (its own sandbox) where it can create, modify, or delete data, but it needs explicit permission to access resources outside of the sandbox.
* Serialization: Is the process of translating data structures from one format into another format.
* Unarchiving: Is a method of reading information or data from a file saved on a disk.
* Handler: A completion handler is a closure containing code that will be executed at a later point.
* Date formatter: Using date formatter or DateFormatter instance, you can create string representations of dates (and create dates from strings). By setting the dateStyle and timeStyle properties of the date formatter instance, you can control how the strings will read.

**UNIT 5**

* Capture: Means to identify and grab a piece of data.
* Closure: A closure is a self-contained block of functionality that can be passed around and used throughout your code. Closures in Swift are similar to blocks in C and Objective-C, and to lambdas in other programming languages.
* Filter: The filter function returns a new collection with only the items that match the conditions included in the closure parameter.
* Filter: The filter function returns a new collection with only the items that match the conditions included in the closure parameter.
* Map: The map function loops over a collection and applies the same operation to each item in the collection. The function accepts a closure that defines what action to perform on the items in the collection.
* Reduce: Use the reduce() function to combine all the items in a collection into a single value. For example, if you had an array of Turn objects that contained a score, you could use the reduce function to calculate the total score.
* Sorted: The sorted(by: ) function will run the instruction on every pair of objects in an array, returning a Boolean value until the entire collection is sorted according to the logic in the closure.
* Syntactic sugar: Refers to programming syntax that’s design to make code easier to read or express.
* Mutate: Mutate refers to changing the value of a property or a variable. When writing a method that will modify a structure, you must opt in to mutating behavior for that method. A method with the mutating keyword can mutate (change) the value of the structure.
* An Affine transformation matrix is used to rotate, scale, translate, or skew the objects you draw in a graphics context. Affine transforms are used when animating views.
* Animation: An animation modifies properties of a view over time.
* Concatenate: Concatenation refers to the operation of joining two strings together.
* Core Animation: Provides a general-purpose system for animating views and other visual elements of your app. Core animation technology integrates with views to provide better performance and support for animating their content.
* Live view: Each page in Xcode playground can have its own live view for displaying a view hierarchy in the assistant editor.
* PlaygroundPage: One of the objects in the PlaygroundSupport framework is called PlaygroundPage. Use the PlaygroundPage.current property to access the \_current\_ playground page.
* PlaygroundSupport: This framework allows you to access properties of the playground environment, so you can share playground data, manage live views, and control the execution of a playground.
* Touch-down: Refers to an event that’s triggered when the user’s finger taps down on a control.
* Touch-up: Refers to an event that’s triggered when the user’s finger lifts up from tapping a control.
* Transform: The transform property enables the manipulation of an element, from scaling the size of an image to increasing or decreasing the amount of time it takes for an animation to complete.
* Wireframe: A wireframe is a prototype or mockup of a given view or user interaction.
* API: An application programming interface, or API refers to any code that provides an interface to a specific set of functions—enabling developers to make use of complex code that’s already written.
* Asynchronous: Asynchronous code runs on separate processor threads, with user interface related code running on the main thread and all session data tasks running on a background thread.
* GET: An HTTP request method, GET requests data from a specified online source.
* HTML: HTML or HyperText Markup Language, is a standardized system for writing content that will appear on a website.
* HTTP Header: HTTP headers are included in some network requests as a way to pass extra details to the server that will respond to the request.
* JSON: JavaScript Object Notation or JSON, is a plain-text format for transporting strings, numbers, Boolean values, arrays, and dictionaries, in key value pairs.
* POST: An HTTP method, POST submits data to be processed to a specified resource.
* Query: A query is a request for data.
* Query Item: A query item or an NSURLQuery item represents a single name/value pair for an item in the query portion of a URL. Use query items with the query Items property of an NSURLQuery Components object.
* URL: A Uniform Resource Locator or URL refers to an address of a network resource or website.
* URL Session: The URL Session API retrieves the contents from an URL object directly, either as an NSData object as a file on disk.
* User-Agent: One common header parameter called User-Agent, is a string that identifies the system performing the request.
* Web server: A web server refers to any computer that can listen for and respond to a network request.
* Escaping closure: An escaping closure is said to escape a function when the closure is passed as an argument to the function, but it called after the function returns.
* Throwing function: A throwing function is a special type of Swift function that can return specific types of errors.
* App Transport Security (ATS): App Transport Security (ATS) improves user security and privacy by requiring apps to use secure network connections over HTTPS.
* Concurrency: Concurrency is the idea that a program may run different parts of code—such as networking or animation code—on different threads, or queues at the same time.
* Grand Central Dispatch (GCD): Is an Apple technology that manages the work on multiple processing cores by providing a series of queues or lines that process code based on relative priority levels.
* Dispatch queue: The dispatch queue class allows you to assign work to be executed on different system queues or threads.
* Main Thread or main Queue: You can think of a thread as a queue, or a line, of instructions that the processor will execute in order. As the processor handles each instruction, it moves on to the next level. The main queue is responsible for all user interface updates in an iOS app.
* Background Queue or thread: All session data tasks are executed on a background thread—enabling the main thread to continue updating the user interface and responding to user input while the background thread waits for response to the network request.
* Network activity indicator: The network activity indicator spins in the top-left corner of the status bar when the app is performing a long-running network task.

**UNIT 6**

* App icon: Is a visual badge or representation of an iOS app used to identify the app, used to identify the app on the Home screen, in the App store, in the spotlight search, and in the settings app.
* Brand: A brand is the visual identity of an app.
* Human interface guidelines: Guidelines maintained by Apple on how to build user interfaces that are intuitive and engaging.
* Launch screen: A launch screen appears instantly when your app starts up and is replaced once the initial view controller is loaded.
* Template image: An image that discards colors and adopts the tint color of the app or enclosing view.
* Minimum viable product: MVP is a product with just enough features to release. MPV’s are often used to quickly release a product and get user feedback—before sinking time into building features that may not be useful or desirable.
* Persona: A persona refers to a set of user characteristics or a user type that might use and app in a particular way.
* Scope creep: Gradually adding more and more features to an app idea is known as scope creep—which delay the app’s release and increases its chances of confusing your target users.