# Intermediate Swift programming - Control flows & Optionals

## Overview

- If the text in a certain label is getting truncated, then you can change the `lines = 0` which will not limit the lines to 1, hence creating as many lines as required and hence not truncating the text.
- Initialization of an `unused constant` throws a `warning` asking us to replace it with `_`
- `if or else` conditions are similar to Python. But we have curly braces like Java
- Operators are similar to Java
- `All collections` in Swift `start & end in []`

## If - Else statements

- `if or else` conditions are similar to Python. But we have curly braces like Java

```
if hardness == "soft"{
  print(softTimer)
}
```

- `Operators` are same as Java with the signs

## Switch statements

Useful if we have more than 5 conditions

```
switch hardness {
  case "Soft":
    print(5)
  case "Medium":
    print(7)
  case "Hard":
      print(12)
    default:
        print("Error")
}
```

## Dictionaries

- Dictionaries are similar to any programming language but with `[]`
- When a key is not present in the dictionary it returns `nil`

```
let eggTimes = [
        "Soft": 5,
        "Medium": 7,
        "Hard": 12
    ]

var dict : [String : Int] = [
                                    "A
                                    "I
                                    ]
```

## Optionals !?

- This is a datatype in itself. It means that the variable can have the specific value or no value at all
- This is when a variable is uncertain.

***Example*** - We can have a `var userName` which would store the username. When we initialize, it would contain the value `nil` but at a later time, it will actually store the name.

```
//Create optional
var userName: String? = nil
userName = "Eternity"

//Unwrap an optional
if userName != nil{
    var unwrapped = userName!
    print(unwrapped)
}
```

---

## Scheduled Timer

- Selector - `@objc` method syntax is used and the `updateTimer` method is passed
- timeInterval - After how many seconds is the timer triggered
- repeats - `True` to keep the timer running
- `timer.invalidate()` - Makes the timer stop

```
Timer.scheduledTimer(timeInterval: 1, target: self, selector: #selector(updateTime
```

***Note:*** Declaration of the timer has to be done like `var countdownTimer = Timer()`

---

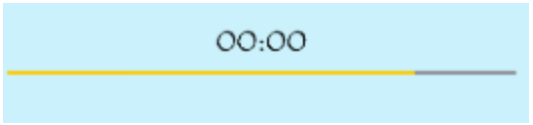
## Progress View

- This shows the progress of an ongoing task

**Important properties**

1. `style` - Change it to `bar` to add height to the progress view
2. `Progress tint` - Color of the completed part
3. `Track tint` - Color of the remaining part
4. `Progress` - This denotes the progress of the bar according to numerical values

`Progress = 0.8`




`Progress = 0.1`