Swift Programming language

Team Members

1) Zuxuan Chen
2) Shejal Shankar
3) Shiddarth Srivastava
4) Qi liu
5) Yanxi Li

History of Swift

- Swift was created by Apple as a successor to Objective-C, with the goal of improving development speed and code safety.

- Swift is the primary language for building applications on various Apple platforms, including iOS, macOS, watchOS, and tvOS.

- Swift is a statically-typed language.

- Swift is Open Source Programming Language.

Names, Binding and Scopes

Name Criteria:

NO mathematical symbols, such as "+", "-", "*", "/“

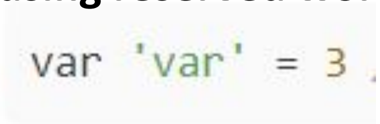
NO arrows

NO reserved words, such as var, let, for, if, etc.

NO Illegal Unicode characters

NO hyphens and tabs

Ways of using reserved words (use ‘’)



```
var 'var' = 3
```

Binding, in Swift

Binding in swift refers to associating a name with a value or a type

FROM value_name TO specific locations in memory

FROM func_name TO specific function body

The work of the compiler, which is not visible to the programmer.

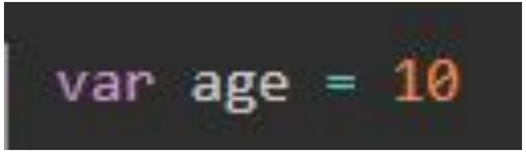
Scopes

- Scopes define the visibility and validity of names in code.
- Swift supports both local and global scopes, similar to other programming languages
- local scopes are usually associated with curly braces {}
- global scopes are the outermost scopes of the entire program, and any name declared in a global scope can be accessed throughout the program.

Data Type of Swift

Commonly used Data Types- Int,Double, Bool,String. Swift also supports custom data types with structs, enums, and classes.

You can declare types without explicitly declaring them, the compiler automatically infers them.

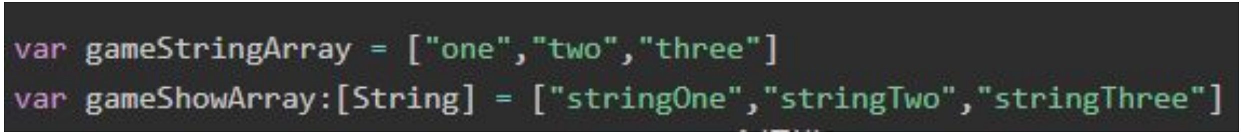


```
var age = 10
```

A var, named age, type: int

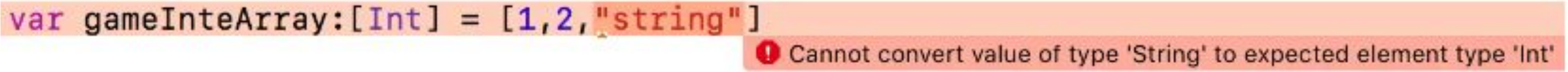
let & var

Arrays



```
var gameStringArray = ["one","two","three"]
var gameShowArray:[String] = ["stringOne","stringTwo","stringThree"]
```

Arrays store multiple values of the same type



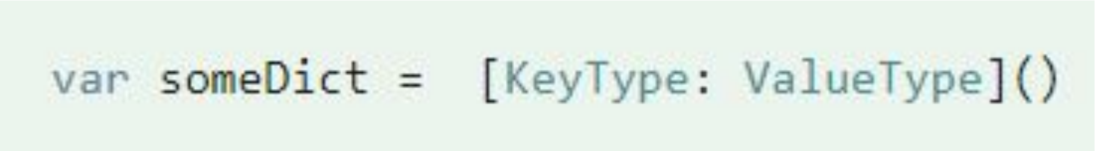
```
var gameInteArray:[Int] = [1,2,"string"]
```

Cannot convert value of type 'String' to expected element type 'Int'

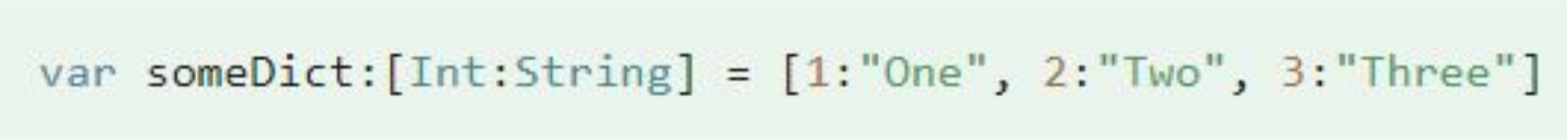
An error is raised here!!!

Dictionary

Give compiler type of KEY & VALUE.



```
var someDict =  [KeyType: ValueType]()
```



```
var someDict:[Int:String] = [1:"One", 2:"Two", 3:"Three"]
```

Expressions and Assignment Statements

- Prefix Expressions
- Binary Expressions
- Suffix Expressions
- Ternary Operator

1)Prefix Expressions → Symbol A

++ Self-increasing(Used to exist)

-- Self-decreasing(Do not exist now)

! Logical non

~ Bitwise inverse

2)Binary Expressions → A Symbol **B**

<< Left Shift

>> Right Shift

+ - * / %

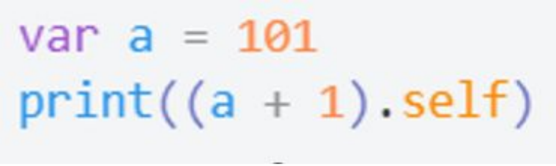
> >= < <=

== === && ||

is Type checking

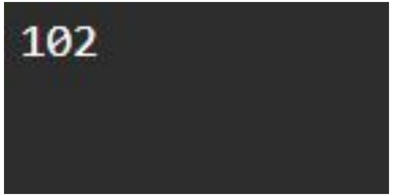
as Type Conversion

3) Suffix Expressions → A Symbol



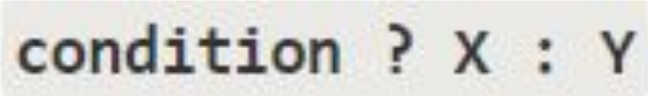
```
var a = 101
print((a + 1).self)
```

→



```
102
```

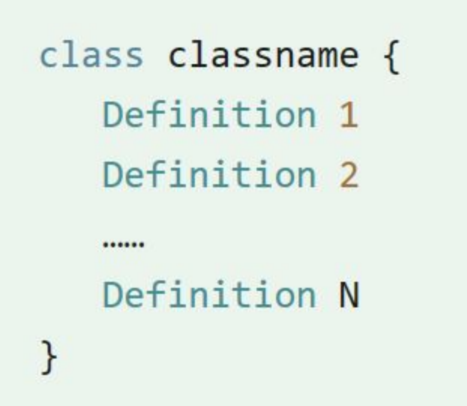
4) Ternary Operator



```
condition ? X : Y
```

Support to OO Programming

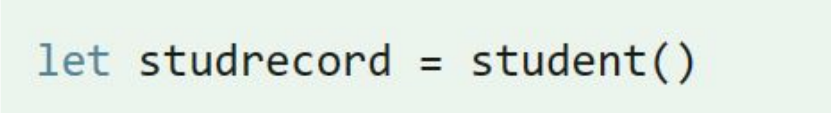
1.Definition Syntax



```
class classname {
   Definition 1
   Definition 2
   ……
   Definition N
}
```

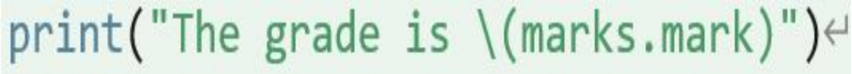
Note: Swift doesn’t require you to create separate interface and implementation files for custom structures and classes.

2. Class Instances



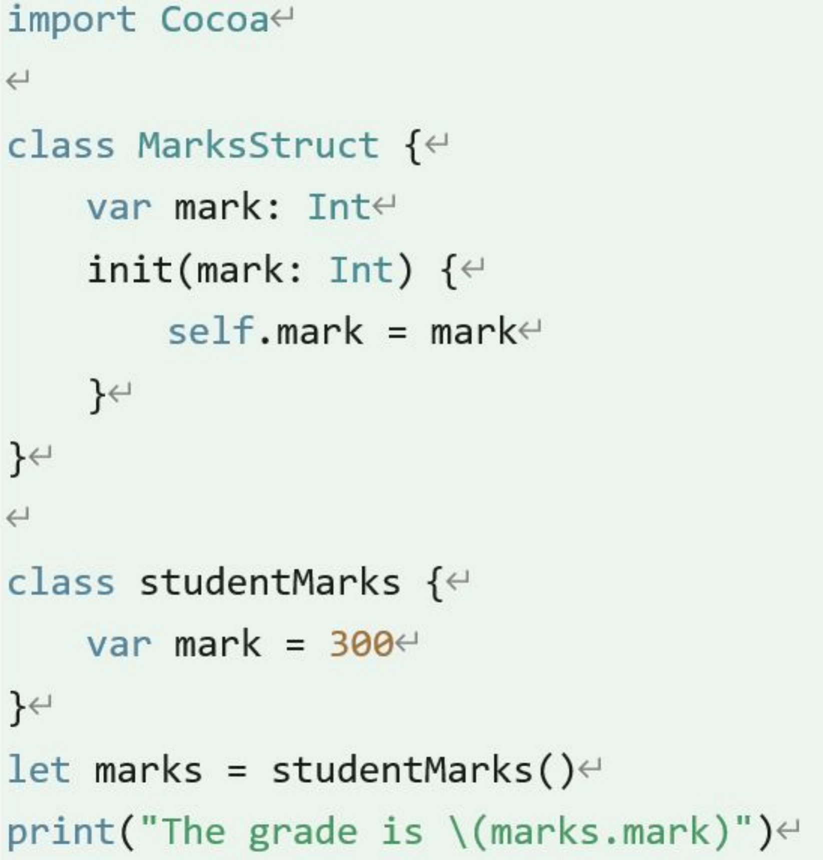
```
let studrecord = student()
```

3. Accessing Properties



```
print("The grade is \(marks.mark)")↵
```

4. Example



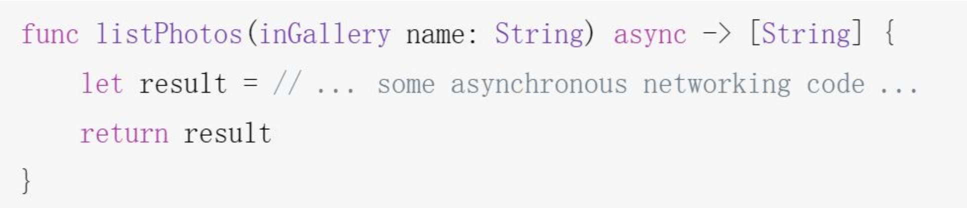
```
import Cocoa

class MarksStruct {
    var mark: Int
    init(mark: Int) {
        self.mark = mark
    }
}

class studentMarks {
    var mark = 300
}
let marks = studentMarks()
print("The grade is \(marks.mark)")
```

Concurrency

Swift has built-in support for writing asynchronous and parallel code in a structured way. Asynchronous code can be suspended and resumed later, although only one piece of the program executes at a time.

```
func fetchData() async -> [String] {
    let data = await fetchRemoteData()
    let processedData = await process(data)
    return processedData
}
```

1. Defining and Calling Asynchronous Functions



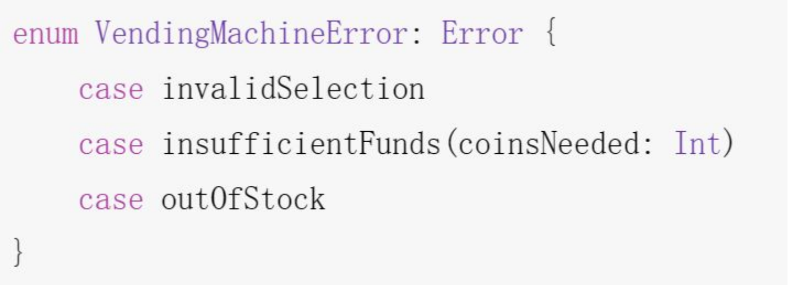
```
func listPhotos(inGallery name: String) async -> [String] {
    let result = // ... some asynchronous networking code ...
    return result
}
```

Note: To indicate that a function or method is asynchronous, you write the keyword in its declaration after its parameters

Exception Handling and Event Handling

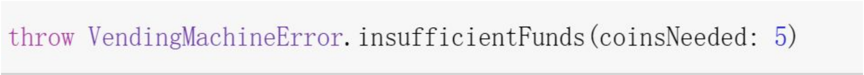
The process of **responding to** and **recovering** from error conditions in your program

Representing and Throwing Errors



```
enum VendingMachineError: Error {
    case invalidSelection
    case insufficientFunds(coinsNeeded: Int)
    case outOfStock
}
```

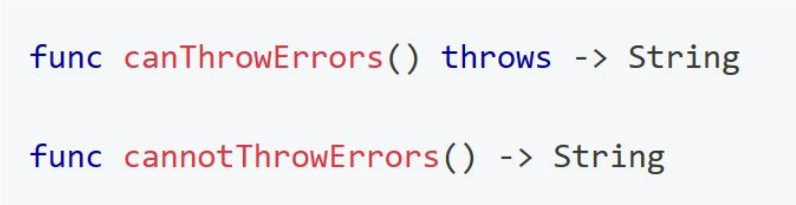
The following code throws an error to indicate that five additional coins are needed by the vending machine



```
throw VendingMachineError.insufficientFunds(coinsNeeded: 5)
```

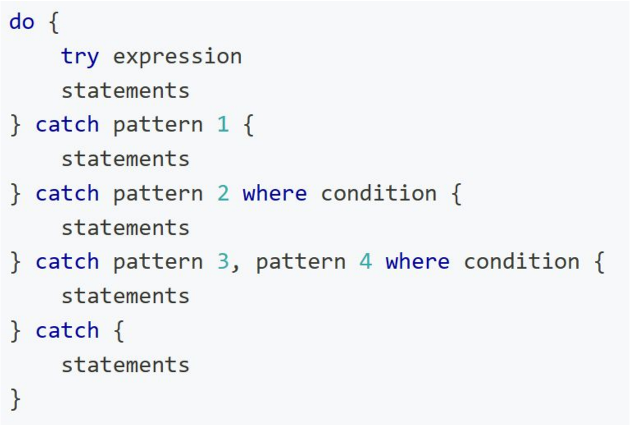
Handling Errors

1. Propagating Errors Using Throwing Functions



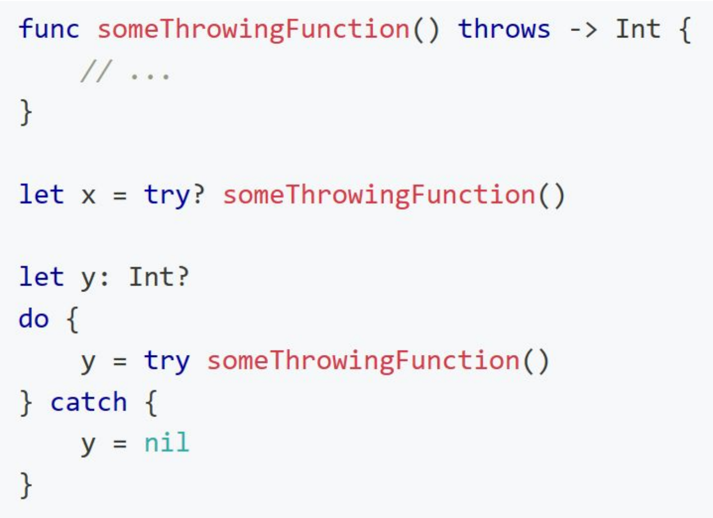
```
func canThrowErrors() throws -> String

func cannotThrowErrors() -> String
```

2. Handling Errors Using Do-Catch



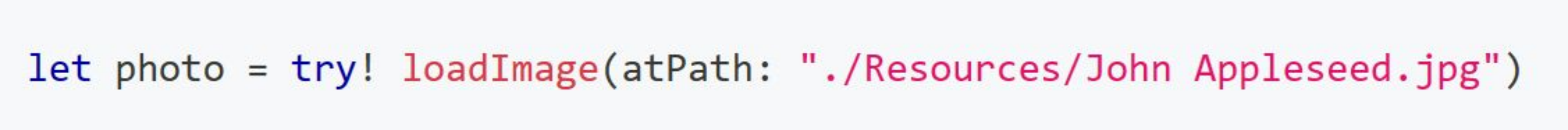
```
do {
    try expression
    statements
} catch pattern 1 {
    statements
} catch pattern 2 where condition {
    statements
} catch pattern 3, pattern 4 where condition {
    statements
} catch {
    statements
}
```

3. Converting Errors to Optional Values



```swift
func someThrowingFunction() throws -> Int {
    // ...
}

let x = try? someThrowingFunction()

let y: Int?
do {
    y = try someThrowingFunction()
} catch {
    y = nil
}
```

4. Disabling Error Propagation



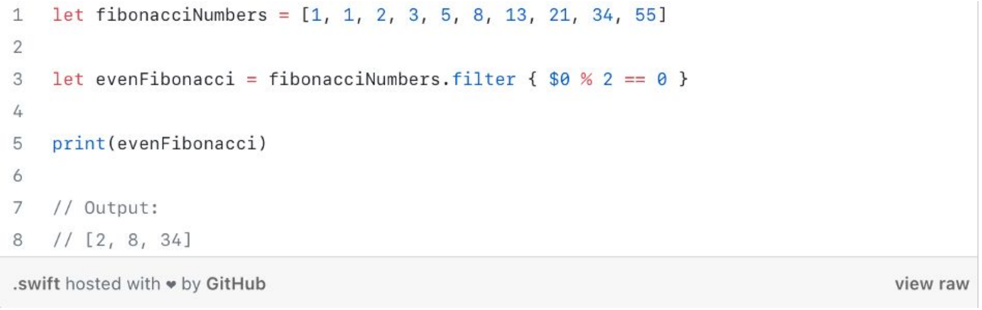
```
let photo = try! loadImage(atPath: "./Resources/John Appleseed.jpg")
```

Functional Programming

- Functional programming is a programming paradigm that treats computation as the evaluation of mathematical functions, avoiding changing state and mutable data.
- Swift also supports higher-order functions, which are functions that either take other functions as parameters or return functions as results. Examples of higher-order functions in Swift include **map, filter**, and **reduce**.
- **Map function:** func map<T>(_ transform: (Element) throws -> T) rethrows -> [T]

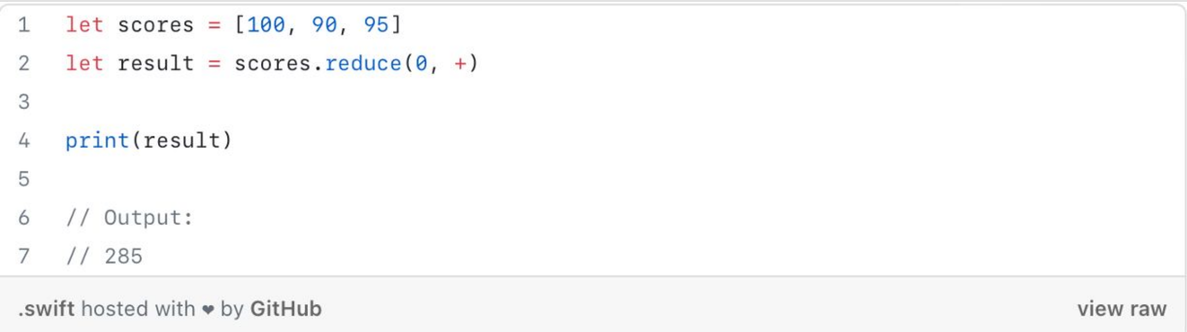
```
let numbers = [1, 2, 3, 4]
let squaredNumbers = numbers.map { $0 * $0 }.
```

filter()- This function loops over every value in a collection and returns a collection with those values that satisfy a condition.



```swift
let fibonacciNumbers = [1, 1, 2, 3, 5, 8, 13, 21, 34, 55]

let evenFibonacci = fibonacciNumbers.filter { $0 % 2 == 0 }

print(evenFibonacci)

// Output:
// [2, 8, 34]
```

reduce() function compresses an array of values to a single value.



```swift
let scores = [100, 90, 95]
let result = scores.reduce(0, +)

print(result)

// Output:
// 285
```

The Project

Objective

The objective of the Map project in Swift is to create a user-friendly and feature-rich map application that provides an intuitive way for users to explore and navigate the world around them. Our goal is to offer a powerful and efficient mapping solution that caters to both casual and professional users.

Constraints

- **Platform:** The project will be developed for iOS 16.0 devices, so it will need to conform to Apple's guidelines and design principles.
- **Performance:** The application should provide smooth and responsive map interactions even on devices with lower hardware capabilities.
- **Data:** Access to mapping data, whether through a third-party service like Google Maps or by creating custom maps, will be necessary. Consider data licensing and terms of use.
- **User Experience:** The app should be designed with a user-friendly interface and intuitive navigation, considering various user profiles, from tourists to professionals.

Features

- **Interactive Maps:** Implement interactive maps with pinch-to-zoom, panning, and rotation for seamless exploration.
- **Location Services:** Utilize GPS and location services to display the user's current location on the map and provide directions.
- **Search and Geocoding:** Allow users to search for places, addresses, and points of interest and display them on the map.
- **Routing and Directions:** Provide route planning and directions between two or more points, including driving, walking, and public transit options.
- **Custom Markers and Overlays:** Allow users to add custom markers, pins, and overlays to the map for personalization.
- **Security and User Privacy:** Implement secure data handling and respect user privacy, especially when using location services.

Technologies Used

- GPS
- Coding Platform: Xcode
- Swift & Swift UI
- Apple MapKit API

Where this project can be applied

- Daily Life navigation
- Route suggestion