

struct and class

- Both **struct** and **class** have ...
... pretty much exactly the same syntax.



struct and class

- Both **struct** and **class** have ...
 - ... pretty much exactly the same syntax.
 - stored **vars** (the kind you are used to, i.e., stored in memory)
 - var isFaceUp: Bool**



struct and class

- Both **struct** and **class** have ...

- ... pretty much exactly the same syntax.

- stored **vars** (the kind you are used to, i.e., stored in memory)

- computed **vars** (i.e. those whose value is the result of evaluating some code)

- ```
var body: some View {
 return Text("Hello World")
}
```



# struct and class

- Both **struct** and **class** have ...

- ... pretty much exactly the same syntax.

- stored **vars** (the kind you are used to, i.e., stored in memory)

- computed **vars** (i.e. those whose value is the result of evaluating some code)

- constant **lets** (i.e. **vars** whose values never change)

- ```
let defaultColor = Color.orange
```

- ```
...
```

- ```
CardView().foregroundColor(defaultColor)
```



struct and class

- Both **struct** and **class** have ...

- ... pretty much exactly the same syntax.

- stored **vars** (the kind you are used to, i.e., stored in memory)

- computed **vars** (i.e. those whose value is the result of evaluating some code)

- constant **lets** (i.e. **vars** whose values never change)

- functions**

- ```
func multiply(operand: Int, by: Int) -> Int {
```

- ```
    return operand * by
```

- ```
}
```

- ```
multiply(operand: 5, by: 6)
```



struct and class

- Both **struct** and **class** have ...

- ... pretty much exactly the same syntax.

- stored **vars** (the kind you are used to, i.e., stored in memory)

- computed **vars** (i.e. those whose value is the result of evaluating some code)

- constant **lets** (i.e. **vars** whose values never change)

- functions

- ```
func multiply(operand: Int, by: Int) -> Int {
 return operand * by
}
```

- ```
multiply(operand: 5, by: 6)
```

- ```
func multiply(_ operand: Int, by otherOperand: Int) -> Int {
 return operand * otherOperand
}
```

- ```
multiply(5, by: 6)
```



struct and class

- Both **struct** and **class** have ...
 - ... pretty much exactly the same syntax.
 - stored **vars** (the kind you are used to, i.e., stored in memory)
 - computed **vars** (i.e. those whose value is the result of evaluating some code)
 - constant **lets** (i.e. **vars** whose values never change)
 - functions**
 - initializers** (i.e. special functions that are called when creating a **struct** or **class**)



struct and class

- Both **struct** and **class** have ...

- ... pretty much exactly the same syntax.

- stored **vars** (the kind you are used to, i.e., stored in memory)

- computed **vars** (i.e. those whose value is the result of evaluating some code)

- constant **lets** (i.e. **vars** whose values never change)

- functions**

- initializers** (i.e. special functions that are called when creating a **struct** or **class**)

```
struct RoundedRectangle {  
    init(cornerRadius: CGFloat) {  
        // initialize this rectangle with that cornerRadius  
    }  
    init(cornerRadius: CGSize) {  
        // initialize this rectangle with that cornerSize  
    }  
}
```



struct and class

- Both **struct** and **class** have ...

- ... pretty much exactly the same syntax.

- stored **vars** (the kind you are used to, i.e., stored in memory)

- computed **vars** (i.e. those whose value is the result of evaluating some code)

- constant **lets** (i.e. **vars** whose values never change)

- functions**

- initializers** (i.e. special functions that are called when creating a **struct** or **class**)

```
struct MemoryGame {  
    init(numberOfPairsOfCards: Int) {  
        // create a game with that many pairs of cards  
    }  
}
```



struct and class

- Both **struct** and **class** have ...

- ... pretty much exactly the same syntax.

- stored **vars** (the kind you are used to, i.e., stored in memory)

- computed **vars** (i.e. those whose value is the result of evaluating some code)

- constant **lets** (i.e. **vars** whose values never change)

- functions**

- initializers** (i.e. special functions that are called when creating a **struct** or **class**)

- So what's the difference between **struct** and **class**?



struct and class

struct

class



struct and class

struct

Value type

class

Reference type



struct and class

struct

Value type

Copied when passed or assigned

class

Reference type

Passed around via pointers



struct and class

struct

Value type

Copied when passed or assigned

Copy on write

class

Reference type

Passed around via pointers

Automatically reference counted



struct and class

struct

Value type

Copied when passed or assigned

Copy on write

Functional programming

class

Reference type

Passed around via pointers

Automatically reference counted

Object-oriented programming



struct and class

struct

Value type

Copied when passed or assigned

Copy on write

Functional programming

No inheritance

class

Reference type

Passed around via pointers

Automatically reference counted

Object-oriented programming

Inheritance (single)



struct and class

struct

Value type

Copied when passed or assigned

Copy on write

Functional programming

No inheritance

"Free" `init` initializes ALL `vars`

class

Reference type

Passed around via pointers

Automatically reference counted

Object-oriented programming

Inheritance (single)

"Free" `init` initializes NO `vars`



struct and class

struct

Value type

Copied when passed or assigned

Copy on write

Functional programming

No inheritance

"Free" `init` initializes ALL `vars`

Mutability must be explicitly stated

class

Reference type

Passed around via pointers

Automatically reference counted

Object-oriented programming

Inheritance (single)

"Free" `init` initializes NO `vars`

Always mutable



struct and class

struct

Value type

Copied when passed or assigned

Copy on write

Functional programming

No inheritance

"Free" `init` initializes ALL `vars`

Mutability must be explicitly stated

Your "go to" data structure

class

Reference type

Passed around via pointers

Automatically reference counted

Object-oriented programming

Inheritance (single)

"Free" `init` initializes NO `vars`

Always mutable

Used in specific circumstances



struct and class

struct

Value type

Copied when passed or assigned

Copy on write

Functional programming

No inheritance

“Free” `init` initializes ALL `vars`

Mutability must be explicitly stated

Your “go to” data structure

Everything you’ve seen so far is a `struct`
(except `View` which is a `protocol`)

class

Reference type

Passed around via pointers

Automatically reference counted

Object-oriented programming

Inheritance (single)

“Free” `init` initializes NO `vars`

Always mutable

Used in specific circumstances

The ViewModel in MVVM is always a `class`
(also, UIKit (old style iOS) is `class`-based)

