Optional Variables

In Swift, variables are either:

1. Read only (let) vs. writable (var)
    a. By default in Swift, usually, variables are always declared as read only (let)
    b. And it is only if you are going to change it that you declare it as (var)

Swift variables are also either:

1. Optional : means that the variable can store a null value (nil)
2. Non-optional: means that the variable CANNOT store null (nil)

In Swift, you cannot assign NULL to a variable, UNLESS you explicitly tell Swift that the variable is able to store NULL values.

By default, variables are declared as non-optional

Let is the default declaration for variable
Non-optional is the default declaration for variables as well

Syntax of an Optional

// Creating an Optional variable using explicit declaration

// read only variable that is optional

let name:String? = "_____"

Name can be assigned any string value OR it can be assigned null

// writable variable that is optional

var name:String? = "_____"

// Creating a non-optional variable

// read only
let name:String = "_____"
// write only
var name:String = "_____"

Name can be assigned any string value, but CANNOT store null

Code Sample

```
// writeable optional variable
var name:String? = "Peter"
print(name)
```

```
Optional("Peter")
Program ended with exit code: 0
```

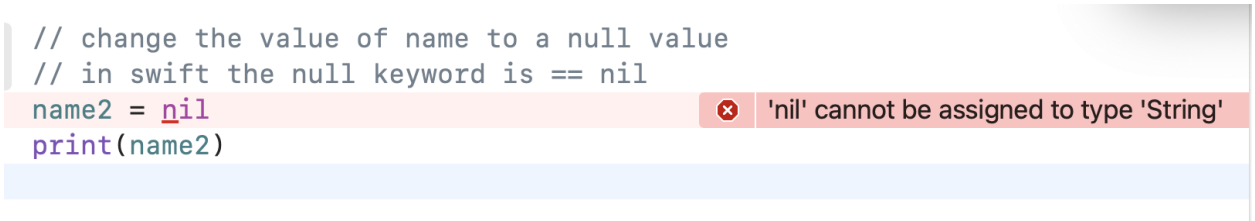
```
// change the value of name to a null value
// in swift the null keyword is == nil
name = nil
print(name)
```

```
nil
Program ended with exit code: 0
```

```swift
// writeable non-optional variable
var name2:String = "Abigail"
print(name2)
```

```
Abigail
Program ended with exit code: 0
```

```swift
// change the value of name to a null value
// in swift the null keyword is == nil
name2 = nil
print(name2)
```




```swift
// example of optionals with other basic data types

// Int
var y:Int? = nil
print(y)
y = 25
// Bool
var z:Bool? = nil
print(z)
z = true

var y2:Int = 25
var z2:Bool = false
```

OOPS

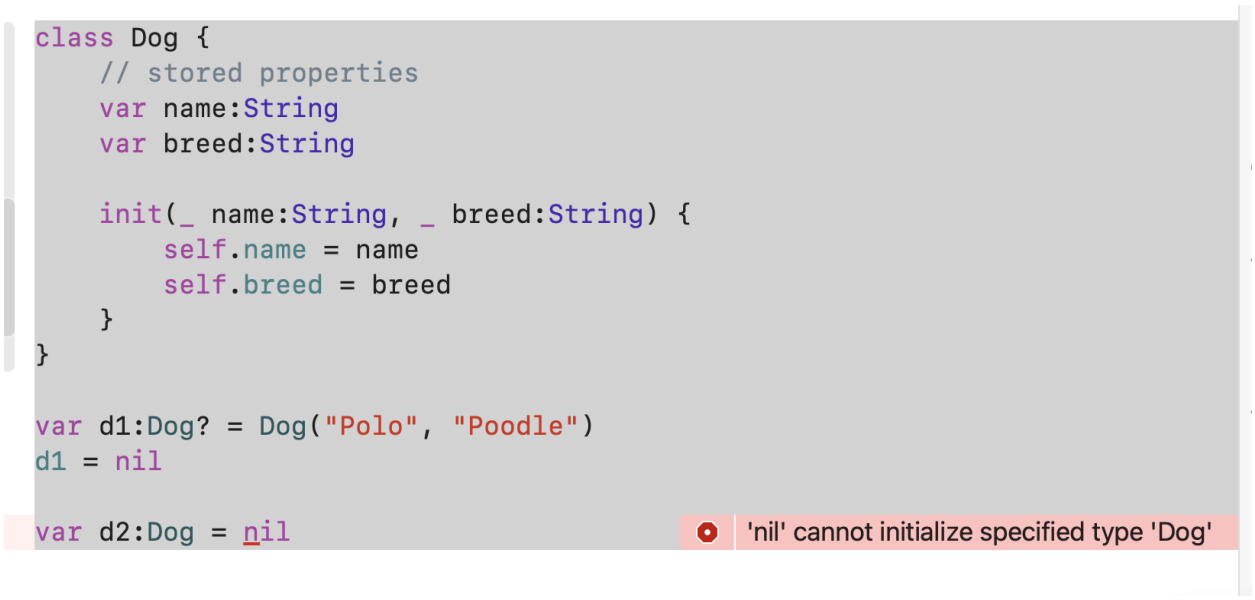
- **Objects can also be declared as optional vs. non optional:**
- **In example below, d1 = optional Dog, d2 = non-optional Dog**
- **As a result, d1 can be assigned nil**
- **D2 CANNOT be assigned nil**

```
class Dog {
    // stored properties
    var name:String
    var breed:String

    init(_ name:String, _ breed:String) {
        self.name = name
        self.breed = breed
    }
}

var d1:Dog? = Dog("Polo", "Poodle")
d1 = nil

var d2:Dog = nil
```

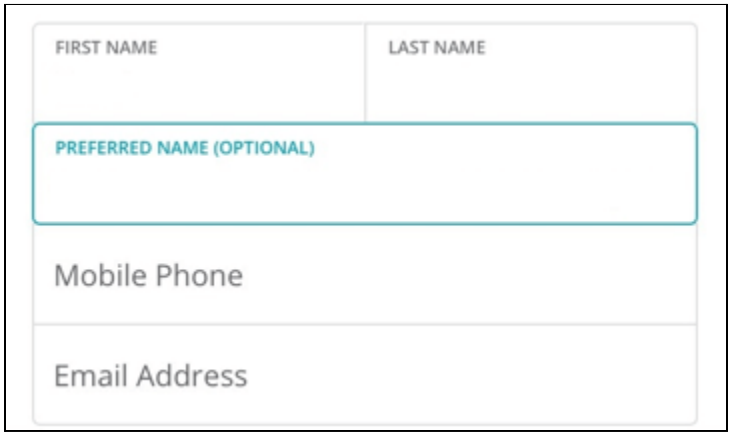


Why do we use optionals?

- Optionals **verify** that a variable contains a valid value **before** you access it.
- If we attempt to use an optional value without verifying that it contains a valid value, we may encounter a runtime error, and your application will crash.

Examples of when data is optional:

- When you fill out internet forms, some field are mandatory vs optional



Here is one way to represent this form data in a class

```
ass SignupForm {
    // stored properties -non optional
    // Because on the website, these fields are manadtory
    //
    let firstName:String
    let lastName:String
    let mobilePhone:String
    let emailAddress:String

    // does it make sense to store preferredName as non-optional?
    // does it make sense to store preferredName as a MANDATORY
variable?
    // yes, no maybe?
    let preferredName:String?

    // initializer
    init(fName:String, prefName:String?, lName:String,
phone:String, email:String) {
        self.firstName = fName
        self.lastName = lName
        self.mobilePhone = phone
        self.emailAddress = email
        self.preferredName = prefName
    }
    // methods
    func displayInfo() {
        print("Name: \(self.firstName) \(self.preferredName)
\(self.lastName)")
        print("Email: \(self.emailAddress)")
        print("Phone: \(self.mobilePhone)")
    }
}
```

```
// example 1: is someone who fills in all the fields
let form1Data:SignupForm = SignupForm(fName: "Peter", prefName:
"Pete", lName: "Smith", phone: "555-999-1234", email:
"peter@gmail.com")
form1Data.displayInfo()
```

Expected output:

```
Name: Peter Optional("Pete") Smith
Email: peter@gmail.com
Phone: 555-999-1234
Program ended with exit code: 0
```

```
// example 2: here is someone who only fills in the mandatory
field, but not the optional one
// (preferred name)
let form2Data:SignupForm = SignupForm(fName: "Abigail", prefName:
"", lName: "Diaz", phone: "888-111-3434", email:
"abigail@gmail.com")
form2Data.displayInfo()

// example 3: here is someone who only fills in the mandatory
field, but not the optional one
// (preferred name)
let form3Data:SignupForm = SignupForm(fName: "Sujay", prefName:
nil, lName: "Waters", phone: "777-222-3333", email:
"sujay@gmail.com")
form3Data.displayInfo()
```

```
Name: Abigail Optional("") Diaz
Email: abigail@gmail.com
Phone: 888-111-3434
Name: Sujay nil Waters
Email: sujay@gmail.com
Phone: 777-222-3333
```

Swift forces you to think about why you are doing what you are doing in your code
And to be INTENTIONAL in the choices you make

Updating the displayInfo()

```
// methods
    func displayInfo() {
        // simplest way is to do an if statment
        if (self.preferredName == nil || self.preferredName == "") {
            print("Name: \(self.firstName) \(self.lastName)")
        }
        else {
            print("Name: \(self.firstName) \(self.preferredName) \(self.lastName)")
        }
        print("Email: \(self.emailAddress)")
        print("Phone: \(self.mobilePhone)")
    }
```

```
Name: Abigail Diaz
Email: abigail@gmail.com
Phone: 888-111-3434
Name: Sujay Waters
Email: sujay@gmail.com
Phone: 777-222-3333
```

When it comes to designing your initializer, it may be advantageous to do your data validation at the time of object creation:

```
// initializer
   init(fName:String, prefName:String?, lName:String, phone:String, email:String) {
      self.firstName = fName
      self.lastName = lName
      self.mobilePhone = phone
      self.emailAddress = email
      // whatever inputs your receive that are "invalid" or "empty strings", you
      // automatically set it to nil within the constructor
      if (prefName == "") {
         self.preferredName = nil
      }
      else {
         self.preferredName = prefName
      }

   }
```

This way, your object is guaranteed to be either nil or meaningful data by the time you use it elsewhere in the class

Returning at 1pm

Write a program to calculate a person's weekly salary.
The person is paid $750 per week for 40 hours of work.
The person may or may not work overtime. → optional variable
If the person works more than 40 hours per week, their overtime hours are billed at $23.50 / hour.

==

Implement this with optionals?
How do you deal with the fact that values can sometimes be nil?

Update 1 - Directly using that optional value in a math calculation does not work

```
var overtimeHours:Double? = nil
var salary:Double = 750
if (overtimeHours == nil) {
    salary = 750
}
else {
    salary = 750 + overtimeHours * 25.30
}

// calculate their final salary
print("Final salary: $\(salary)")
```

```swift
var overtimeHours:Double? = nil
var salary:Double = 750
if (overtimeHours == nil) {
    salary = 750
}
else {
    salary = 750 + overtimeHours * 25.30
}

// calculate their final salary
print("Final salary: $\(salary)")
```

Value of optional type 'Double?' must be unwrapped to a value of type '

Dangers of Forced Unwrapping

Forced unwrapping is very dangerous because it relies on the developer being **absolutely certain** that the variable will contain a value **at the time the variable is accessed.**

Example of when forced unwrapping is okay:

```swift
var overtimeHours:Double? = 10
var salary:Double = 750

// at this point,Swift goes to examine what the value of overtime hours is
// overtime hours could be 750 or it could be nil

// Are you sure that I can take the value stored in overtime hours and
// do math with it?
// Are you sure that if I take the value stored in overtime hours
// and do math, I won't crash the program?
// Hey developer? Are you certiain about this?
salary = 750 + overtimeHours! * 25.30
```

Example of when forced unrwapping is not okay

```
var overtimeHours:Double? = nil
var salary:Double = 750

// at this point,Swift goes to examine what the value of overtime hours is
// overtime hours could be 750 or it could be nil

// Are you sure that I can take the value stored in overtime hours and
// do math with it?
// Are you sure that if I take the value stored in overtime hours
// and do math, I won't crash the program?
// Hey developer? Are you certiain about this?
salary = 750 + overtimeHours! * 25.30
```

Forced unwrapping vs not forced unrwapping

```
// writeable optional variable
var name:String? = "Peter"
print(name!)       // hey swift, i promise that name has some value other than nil
print(name)
```

To fix, you can check for nill before forced unrapping

```
var overtimeHours:Double? = 10
var salary:Double = 750

// obviously, you can use the if statement
if (overtimeHours != nil) {
    salary = 750 + overtimeHours! * 25.30
    print(salary)
}
```

Instead of using force unwrapping, you can use:

- Optional Binding: if-let statements
    - Used only in the *body* of a program
- Guard Let: Early exit
    - Used only in a function
- Nil coalescing: Provides a default value to your optional