옵셔널

nil이란?

```
class Person{

    var name:String

    init(name:String) {
        self.name = name
    }

}
```

```
let person1:Person = Person(name:"joo")
print(person1.name)
```

결과 : “joo"

nil이란?

```
class Person{

    var name:String

    init(name:String) {
        self.name = name
    }


}
```

```
let person2:Person
print(person2.name)
```

```
결과 : ????
```

nil이란?

```swift
class Person{

    var name:String

    init(name:String) {
        self.name = name
    }

}
```

```swift
let person2:Person
print(person2.name)
```

선언만 되고 인스턴스를 생성하지 않은 상태를 nil이라고 한다.

Type Safety

- nil인 상태에서 속성을 참조하거나, 함수를 실행시 발생하는 error로 인한 코드의 불안정성 내포

- Swift의 중요한 특징 중 하나는 Safety!!

- Type Safety를 위해 컴파일러 수준의 nil 체크

- 만약 nil인 변수 선언을 해야할 경우 optional을 사용한다.

- **optional은 두가지 가능성을 가질수 있는데**
  한개는 값이 있음을 나타내고 (!기호 사용)
  또다른 한가지는 nil일 가능성을 내포하고 있다.(?기호 사용)

옵셔널 타입

```
var person:Person
```
→ 에러

!

```
var person:Person!
person = Person()
```

?

```
var person:Person?
person = Person()
```

옵셔널 타입

!

```
var person:Person!
person = Person()
```

```
person.run()
```

만약 person가 초기화가 안되어 있다면! 프로그램이 멈춘다.

?

```
var person:Person?
person = Person()
```

```
person?.run()
```

만약 person가 초기화가 안되어 있다면! run()이 실행이 안된다.

```
person!.run()
```

만약 person가 초기화가 안되어 있다면! 프로그램이 멈춘다.

옵셔널 타입 테스트

- 일반 변수 vs 옵셔널 변수

- ?와 !의 차이

Unwrapping

Optional 변수에 값이 있음을 확인하여 일반 변수로 전환해준다.

- Forced Unwrapping

- Optional Binding

- Early Exit

강제 해제(Forced Unwrapping)

```
func testFuc(optionalStr:String?)
{
    if optionalStr != nil
    {
        let unwrapStr:String = optionalStr!
        print(unwrapStr)

    }
}
```

선택적 해제(Optional Binding)

```
func testFuc(optionalStr:String?)
{
    if let unwrapStr:String = optionalStr
    {
        print(unwrapStr)
    }
}
```

Early Exit

```
guard 조건값 else
{
  //조건값이 거짓일때 실행
}
```

Early Exit

```
func testFuc(optionalStr:String)
{
    guard let unwrapStr:String = optionalStr  else
    {
        return
    }

    print(unwrapStr)
}
```

선택적 해제 - 예제

```
func isNumber(inputNum:String) -> Bool
{
    if let firstNumber = Int(inputNum)
    {
        print(“\(firstNumber)”)
        return true
    }else
    {
        return false
    }
}
```

선택적 해제 - 예제

*문제 : inputNum0이 한개가 아닌 두개라면?

```
func isNumber(inputNum1:String, inputNum2:String) -> Bool
{

}
```

선택적 해제 - 예제

```
func isNumber(inputNum1:String, inputNum2:String) -> Bool
{

    if let firstNumber = Int(inputNum1), let secondNumber =
Int(inputNum1)
    {
        return true
    }else
    {
        return false
    }
}
```

* (,) 콤마를 통해 옵셔널 바인딩을 추가하고, 또 조건도 추가 할수 있다.

클래스 & 객체

Swift Class Architecture

```
class ClassName : superClass
{
  var vName1 = “1”
  var vName2 = 4

  func fName1() - > Any
  {


  }

  func fName2(_ ani:Bool)
  {


  }

}
```

<CalssName.swift>

구조

```
class Hat
{



}
```

프로퍼티

```
class Hat
{
    var color:String = "흰색"
    var shape:String = ""
    var size:Int = 0
}
```

객체 만들기

```
class Hat
{
    var color:String = "흰색"
    var shape:String = ""
    var size:Int = 0
}
```

```
var myHat:Hat = Hat()
```

init

```
    init() {
        // perform some initialization here
    }
```

```
    class Hat
    {
        var color:String = "흰색"
        var shape:String = ""
        var size:Int = 0

        init() {
            size = 52
            shape = "라운드형태"
        }

}
```

Custom init

- init에 parameter를 추가해서 custom하게 만들수 있다.

```
class Hat
{
    var color:String = "흰색"
    var shape:String = ""
    var size:Int = 0

    init(color:String, size:Int) {
        self.color = color
        self.size = size
    }

}
```

self

- 자기 자신 인스턴스를 가르키는 포인트
- 명시적 표시를 위해 사용하거나, 중복을 구분하기 위해 사용한다.

학점 계산기

- 같이 해보아요~

Subject Class

- 과목이름 변수
- 과목점수 변수

Student Class만들기

- 이름
- 과목들(과목 객체 배열)
- 평균점수

Point Calculator Class만들기

- 학생 객체를 받아서 평균을 알려주는 클래스

클래스의 상속

- Subclassing
- 기존에 구현되어있는 클래스를 확장, 변형
- 부모 클래스(super class, parent class)와
  자식 클래스(sub class, child class)로 관계를 표현
- 상속 할 수록 더 확장되는 구조
  - 즉, 자식이 기능이 더 많을 가능성이 크다

클래스의 상속

```
class UniversityStudent: Student {

}
```

UniversityStudent

Student

Person

name
age
eat()

grade
study()

major
goMT()

클래스의 상속

- UniversityStudent클래스를 만들어 Student를 상속받게 만듭니다.

- grade와 point프로퍼티 추가

- 그에 따른 calculator에 추가 함수 만들기

재정의

재정의

- 영어로 Override

재정의

- 영어로 override(오버라이드)
- 부모 클래스에게서 물려받은 성질을 그대로 사용하지 않고
  자식 클래스에게 맞는 형태 또는 행위로 변경하여
  사용할 수 있는 기능

다형성

- **재정의(Override)**와 **중복정의(Overload)**는
  OOP의 **다형성**의 또다른 모습

실습

- 학점계산기 업그레이드 (대학생 버전으로 만들기)