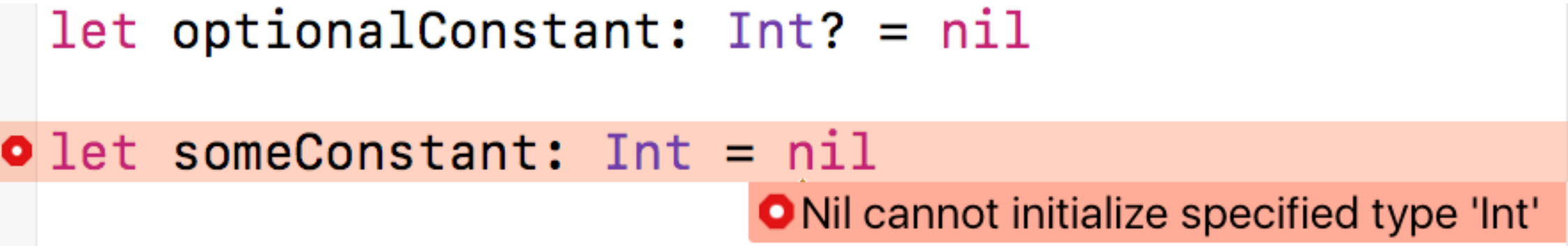
Swift
- Optional -

yagom

Optional

값이 '있을 수도, 없을 수도 있음'


let optionalConstant: Int? = nil
let someConstant: Int = nil
Nil cannot initialize specified type 'Int'

Why??

옵셔널이 왜 필요할까요?

nil의 가능성을 명시적으로 표현

- nil 가능성을 문서화 하지 않아도 코드만으로 충분히 표현가능
  - 문서/주석 작성 시간을 절약
- 전달받은 값이 옵셔널이 아니라면 nil체크를 하지 않더라도 안심하고 사용
  - 효율적인 코딩
  - 예외 상황을 최소화하는 안전한 코딩

any localization. This is useful, for example, when working with fixed-format representations of information that is written out and read back in at a later time.

Important
When working with text that's presented to the user, use the localizedStringWithFormat: method, or the initWithFormat:locale: or initWithFormat:locale:arguments: method, passing `currentLocale` as the locale.

`format`	A format string. See Formatting String Objects for examples of how to use this method, and String Format Specifiers for a list of format specifiers. This value must not be `nil`. **Important** Raises an `NSInvalidArgumentException` if `format` is `nil`.
`...`	A comma-separated list of arguments to substitute into `format`.

A string created by using `format` as a template into which the remaining argument values are substituted without any localization.

iOS (2.0 and later), macOS (10.0 and later), tvOS (9.0 and later), watchOS (2.0 and later)

Foundation

Type Method Reference

```swift
// someOptionalParam can be nil
func someFunction(someOptionalParam: Int?) {
    // ...
}
// someParam must not be nil
func someFunction(someParam: Int) {
    // ...
}

someFunction(someOptionalParam: nil)
someFunction(someParam: nil)
```

Nil is not compatible with expected argument type 'Int'

Optional

enum + general

```
enum Optional<Wrapped> : ExpressibleByNilLiteral {
    case none
    case some(Wrapped)
}

let optionalValue: Optional<Int> = nil
let optionalValue: Int? = nil
```

Optional

Implicitly Unwrapped Optional

암시적 추출 옵셔널

```
var optionalValue: Int! = 100

switch optionalValue {
case .none:
    print("This Optional variable is nil")
case .some(let value):
    print("Value is \(value)")
}
```

Implicitly Unwrapped Optional

```
// 기존 변수처럼 사용 가능
optionalValue = optionalValue + 1

// nil 할당 가능
optionalValue = nil

// 잘못된 접근으로 인한 런타임 오류 발생
optionalValue = optionalValue + 1
```

Optional

Optional

```
var optionalValue: Int? = 100

switch optionalValue {
case .none:
    print("This Optional variable is nil")
case .some(let value):
    print("Value is \(value)")
}
```

Optional

```
// nil 할당 가능
optionalValue = nil

// 기존 변수처럼 사용불가 - 옵셔널과 일반 값은 다른 타입이므로 연산불가
optionalValue = optionalValue + 1
```