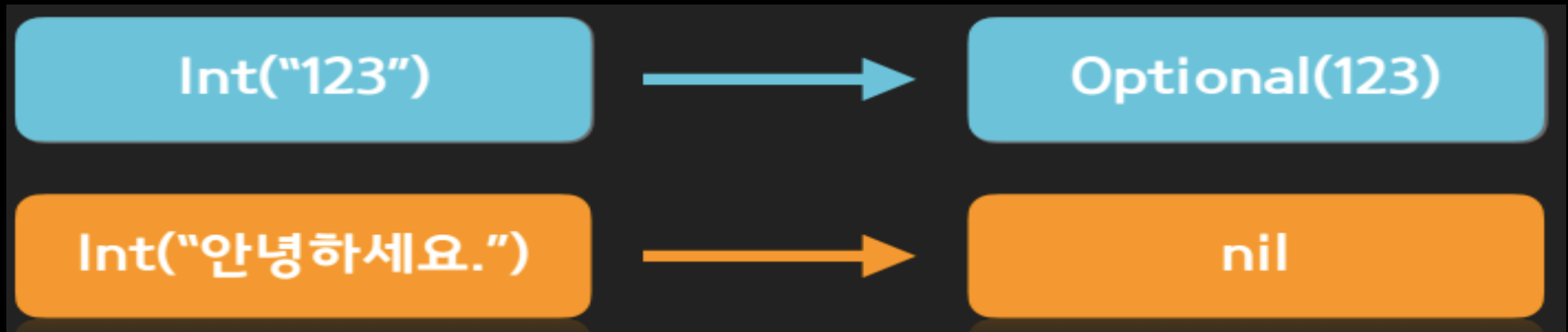


5. Optional




1. 잠재적으로 오류가 발생할 가능성이 있는 상황을 방지하기 위한 안전장치
2. 반환하고자 하는 값을 옵셔널 객체로 다시 한번 감싼 형태를 의미
3. 정상적인 처리 결과에는 제대로 된 값에 옵셔널을 감싼 형태로 반환
4. 비정상적인 처리에는 `nil` 반환

5. Optional 강제 해제

- 옵셔널 강제해제

1. 옵셔널 타입의 변수&상수 뒤에 '!' 붙이면 강제해제
2. nil인 변수 & 상수 강제 해제 시 에러 발생

```
//5. 옵셔널  
var optionalInt1 = Int("27")  
var optionalInt2 = Int("안녕하세요")//nil  
  
let forced1 = optionalInt1!  
let forced2 = optionalInt2!  error: Executi  
//nil인 변수&상수 강제 해제시 에러발생
```

```
var myName : String? = "하태경"  
var haba : String = myName!  
myName = nil  
haba = myName!//런타임 오류
```

3. 다음과 같은 분기문으로 옵셔널을 강제해제 하면 옵셔널을 사용함으로써 얻는 장점이 없다(다른 언어에서 NULL 값을 체크 하는 방식과 비슷).

```
//옵셔널 강제 추출  
if myName != nil{  
    print("My name is \$(myName!)")  
}else {  
    print("myName == nil")  
}
```

5. Optional 비강제 해제

- 옵셔널 비강제해제

```
//옵셔널 바인딩
//myName이 nil이 아니라면, print("\(name)") 실행

myName = nil
if let name1 = myName{
    print("\(name1)")
}else{
    //name1은 if 문 블록안에서만 사용 가능
    //print(name1)
    print("myName is nil")
}

myName = "하태경"

if var name2 = myName{
    //name2 는 if 문 블록안에서만 사용 가능
    //var로 할당하면 블록안에서 변수 값 변경 가능
    name2 = "haba"
    print("\(name2)")
}else{
    // print(name2)
    print("myName is nil")
}

//guard let 을 이용한 옵셔널 비강제 해제
func checkOptional(str:String?){
    guard let name3 = str else{
        print("myName is nil")
        return
    }
    print(name3)
}

//myName = "하태경"
myName = nil
checkOptional(str: myName)
```

if let ~ : 옵셔널에 값이 있다면 옵셔널에서 추출된 값을 일정 블록 안에서 사용할 수 있는 상수나 변수로 할당해서 옵셔널이 아닌 형태로 사용할 수 있도록 해준다. 보통 if 문과 결합하여 사용하며 if let 구문에서 선언한 변수는 if 블록안에서만 사용 가능하다

guard let ~ : 함수 안에서 사용하는 것이 일반적. 조건 [name3 = str] 의 결과값을 Bool형식(true,false)으로 반환하며, 조건이 false일 때 else 블록 안으로 들어가서 구문을 실행한다. 조건이 true일 때는 else{}구문을 무시하고 print(name3)을 실행한다 if let 구문과의 가장 큰 차이점은 guard let 구문에서 변수로 선언한 name3을 guard문 바깥에서도 사용이 가능하다는것이다.(if 문은 if문 안에서만 변수 사용 가능)