4 w.

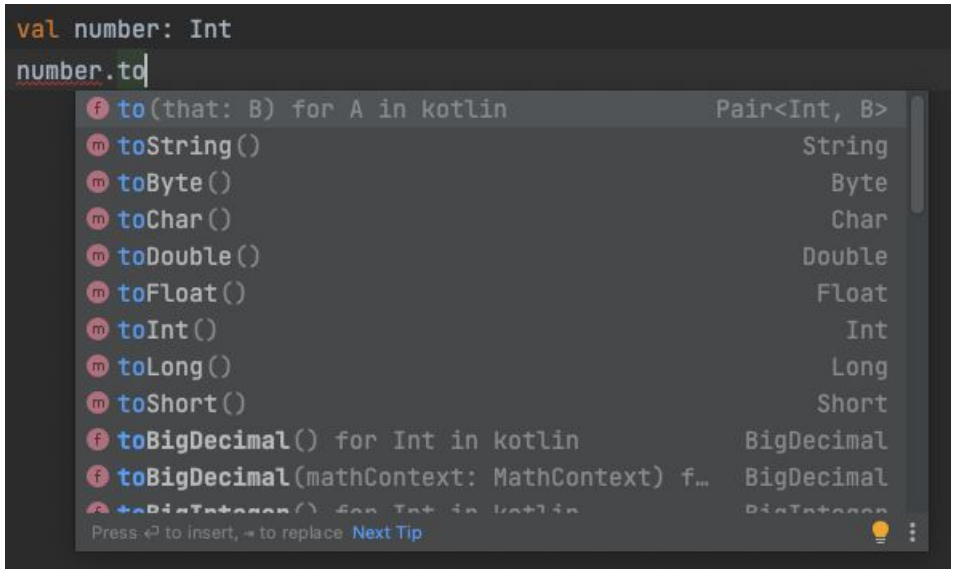
Kotlin

Ivy

Kotlin Type

Kotlin은 Data Type을 객체로만 제공합니다.

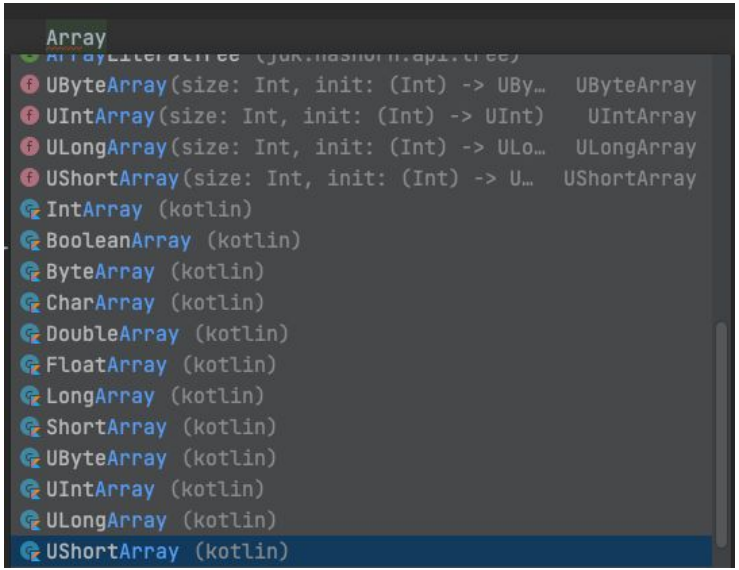
- EX. Number



Kotlin Type

모든 타입의 Array를 제공합니다.

- Array도 대부분의 고차 함수를 지원합니다. (vs Java와 대비되는 점)



Array 지원 메서드

Kotlin	JavaScript
all()	every()
any()	some()
fold()	기본값이 있는 reduce()
reduce()	기본값이 없는 reduce()

기본 operator

(1) Safe Call Operator

```
val layer: Layer? = null
val x = layer?.x

-> compiler 가 아래와 같이 변환한다.
val x = if (layer != null) layer.x else null
```

(2) Not-null assertion operator

```
val l = b!!.length
```

🚨 코드 작성 단계에서 nullable 을 정확히 판단하지 못하는 경우들(특히 Java 기반 코드와 호환되는 경우)도 있다. → `여전히 Kotlin에서도 신경써서 개발하지 않으면 NPE는 발생할 수 있다.`

기본 operator

(3) Elvis Operator

```
fun foo(node: Node): String? {
    val parent = node.getParent() ?: return null
    val name = node.getName() ?: throw IllegalArgumentException("name expected")
    // ...
}
```

```
val parent = node.getParent() ?: return null
```

(4) Safe cast operator

```
val aInt: Int? = a as? Int
```

lambda의 return 방식

- 람다 / 람다식 : `val sum: (Int, Int) -> Int = { x: Int, y: Int -> x + y }`

- run : 인자로 받은 람다를 실행하는 함수
- lambda return
  - return
  - return@run

```
override fun onCreate(savedInstanceState: Bundle?) {
    val uri = intent?.data ?: run {
        finish()
        return
    }
    Log.d(TAG, "landing uri=${uri}")
}
```

lambda의 이름

(1) 자동 이름

- 람다를 인자로 받은 함수 이름

```
animation.addEventListener("enter") {
  val frame = animation.currentFrame as? Number ?: return@addEventListener null
  val frameRate = animation.frameRate as? Number ?: return@addEventListener null
  lottieLayer.currentTime.setImmediate(frame.toFloat() / frameRate.toFloat())
}
```

(2) 기명 람다

- 람다에 이름을 직접 부여하는 방식

```
name@{ return@name }

animation.addEventListener("enter") onComplete@{
  val frame = animation.currentFrame as? Number ?: return@onComplete null
  val frameRate = animation.frameRate as? Number ?: return@onComplete null
  lottieLayer.currentTime.setImmediate(frame.toFloat() / frameRate.toFloat())
}
```

과제 : Scope 함수, inline 함수 활용

```
public inline fun <R> run(block: () → R): R {
```

```
public inline fun <T, R> T.run(block: T.() → R): R {
```

```
public inline fun <T, R> with(receiver: T, block: T.() → R): R {
```

```
public inline fun <T> T.apply(block: T.() → Unit): T {
```

```
public inline fun <T> T.also(block: (T) → Unit): T {
```

```
public inline fun <T, R> T.let(block: (T) → R): R {
```

```
public inline fun <T> T.takeIf(predicate: (T) → Boolean): T? {
```

```
public inline fun <T> T.takeUnless(predicate: (T) → Boolean): T? {
```

```
public inline fun repeat(times: Int, action: (Int) → Unit) {
```