Kotlin : Basics

Mobile Software

2022 Fall

What to do next?

- **Basic types**
- 제어문과 반복문
- 배열

- Kotlin 강의 노트에서는
  - 소스 코드를 별도로 제공하지 않음
  - 실행 결과를 포함하지 않음.
  - 직접 코드를 입력하고, 강의 노트 설명을 확인해야 함.

Basic Types : Numbers

- Numbers : **Kotlin type은 첫 글자가 대문자**
  - **Double**(64), **Float**(32), **Long**(64)
  - **Int**(32), **Short**(16), **Byte**(8)

U: Unsigned (부호 없음)
ULong, UInt, UShort, UByte

  - **상수(literal constants)**
    - **Double** : 123.5, 123.5e10
    - **Float** : 123.5**F**, 123.5**f**
    - **Long** : 123**L**
    - **Hexadecimals : 0x**0F
    - **Binaries : 0b**00001011
  - **형 변환 함수 : to + type ➔ toByte()**
    - **toByte**(), **toShort**( ), **digitToInt**(radix : Int), **toLong**( )
    - **toFloat**( ), **toDouble**( ), **toChar**( )

Numbers 예 (1/2)

```
fun main() {
    println("Byte : " + Byte.MIN_VALUE + ", " + Byte.MAX_VALUE + ", " + Byte.SIZE_BITS)
    println("Short : " + Short.MIN_VALUE + ", " + Short.MAX_VALUE + ", " + Short.SIZE_BITS)
    println("Int : " + Int.MIN_VALUE + ", " + Int.MAX_VALUE + ", " + Int.SIZE_BITS)
    println("Long : " + Long.MIN_VALUE + ", " + Long.MAX_VALUE + ", " + Long.SIZE_BITS)
    println("Float : " + Float.MIN_VALUE + ", " + Float.MAX_VALUE + ", " + Float.SIZE_BITS)
    println("Double : " + Double.MIN_VALUE + ", " + Double.MAX_VALUE + ", " + Double.SIZE_BITS)
}
```

```
Byte : -128, 127, 8
Short : -32768, 32767, 16
Int : -2147483648, 2147483647, 32
Long : -9223372036854775808, 9223372036854775807, 64
Float : 1.4E-45, 3.4028235E38, 32
Double : 4.9E-324, 1.79769313486231570E308, 64
```

잠깐! toInt() is deprecated

- **Deprecated (비 추천)**
  - 문제가 발생해 업그레이드 시점에서 더 이상 지원하지 않음.
  - 기능은 같으면서 이름이나 사용 방법이 바뀐 대안을 사용.

```
fun main() {
    println("4".toInt())
    println('4'.toInt())
}
```

4
52

4가 아닌 52 ➔ 52는 숫자 4의 ASCII code

```
fun main() {
    var str = String.format("%c, %c", Char( code: 52), Char( code: 53))
    println(str)
    println('4'.digitToInt( radix: 10))
    println('4'.digitToInt())
}
```

Numbers 예 (2/2)

```
fun main() {
    var f = 3.14f
    var d = 3.14
    var i:Short = 3
    var fd = f.toDouble()
    var i2f = i.toFloat()

    typeCheck(f)
    typeCheck(d)
    typeCheck(i)
    typeCheck(fd)
    typeCheck(i2f)
}

fun typeCheck(v:Any) {
    when(v) {
        is Short -> println("the type of $v is Short.")
        is Int -> println("the type of $v is Int.")
        is Float -> println("the type of $v is Float.")
        is Double -> println("the type of $v is Double.")
    }
}
```

5개 변수 모두 타입을 추론할 수 있기 때문에 타입 생략 가능.

자동 형 변환 기능은 없음.
➔ 형 변환 함수 사용

Any: 어떤 타입과도 호환 가능

is 는 Java의 **instanceof** 에 해당

when 구문은 Java의 **switch-case** 구문

Basic Types : Char (1/2)

- **Char** 타입 변수는 따옴표로 묶은 문자만 할당할 수 있음.
- ASCII 코드에 해당하는 숫자를 할당할 경우 에러 발생.
  - **toChar( )** 를 적용하여 **Char** 타입으로 변환.

```
val c : Char = 'A'
```

```
val c : Char = 65
```

```
fun main() {
    val code:Int = 65
    val han_code:Char = '\uD55c'

    println(code.toChar() + ", " + (code+1).toChar())
    println(han_code)
}
```

문자 'A'의 ASCII code

한글 음절 '한'의 Unicode

Basic Types : Char (2/2)

```
fun main() {
    for (i in 48 ≤ .. ≤ 60)
        print("${decimalValue(i)}")
    println()
}

fun decimalValue(i:Int) =
    if (i in 48 ≤ .. ≤ 57) i.toChar() else '-'
```

문자 '0'~ '9'를 숫자 0~9로 변환

Basic Types : Boolean

- Boolean
  - 값 : **true**(참), **false**(거짓)
- 논리 연산 기호 : &&, ||, !

```
fun main() {
    var foo:Boolean = true
    val bar = false

    println( foo && bar )    /* false */
    println( foo || bar )    /* true */
    println( !foo )          /* false */

    foo = !foo               // 상태 toggle
    println( foo )           // false
}
```

Basic Types : String (1/2)

- String (**문자열**, *string of characters*)
  - **String객체**.get(index) 또는 **String객체**[index]
    - 문자열의 특정 위치 문자(**Char**)에 접근
  - Strings are **immutable**(변경 불가, read-only).

```
fun main() {
    var foo:String = "My First Kotlin"

    val size = foo.length
    for (i in 0 ≤ until < size)    /* until은 size-1 까지임 */
        print(foo[i])
    println()
    println("first char = ${foo[0]}, last char = ${foo[size-1]}")

    var ch1:Char = foo.get(3)
    var ch2:Char = foo[9]
    println("length=$size, ch1 = $ch1 and ch2 = $ch2")
}
```

Kotlin은 Heap 메모리 중 String pool 공간에 문자열을 저장하고 관리.

Basic Types : Quiz

- 코드 실행 결과는 ?

```
fun main() {
    var foo: String = "My First Kotlin"
    foo = foo.substring(0, 9) + "python"
    println(foo)

    var foo2 = foo.replace( oldValue: "Kotlin",  newValue: "python")
    println(foo2)
    println(foo == foo2)
}
```

.foo, foo2는 같을까요? 다를까요?

Basic Types : String (2/2)

- **String.format**() : 규격화된 문자열(formatted string) 출력

```
import java.lang.Math.PI

fun main() {
    val pi: Float = PI.toFloat()
    val digit = 10
    val str = "Hello"
    val length = 3000

    val lengthStr: String = String.format("Length: %d meters", length)
    println("pi = %.2f, %3d, %s".format(pi, digit, str))
    println(lengthStr)
}
```

데이터 타입 별로 출력 서식을 지정
%f, %d, %s

출력 서식을 아래처럼 바꾸면 결과는?
%.5f, %5d, %10s

string templates (1/2)

- 문자열 템플릿(string template)
  - 앞에서 정의한 변수를 문자열 안에서 참조할 수 있음
    - 변수 이름 앞에 $를 붙임
  - 문자열 템플릿에 식(expression)을 포함할 경우 중괄호로 구분.

```
fun main() {
    var a = 1
    val s1 = "a is $a"

    a = 2
    val s2 = "${s1.replace("is", "was")}, but now is $a"
    println(s2)

    val s = "abc"
    println("$s.length is ${s.length}")
}
```

string templates (2/2)

- 문자열 템플릿에 특수 기호를 포함하려면
  - ${'특수기호'}를 사용하거나
  - escape char(backslash 기호) 사용

```
fun main() {
    val expr = "My First Kotlin"
    val amount = 10

    val lengthStr = "text length: ${expr.length}"
    val priceStr = "price: USD ${'$'}$amount"
    val priceStr2 = "price: USD \$$amount"

    println(lengthStr)
    println(priceStr)
    println(priceStr2)
}
```

${'특수기호'} 사용

escape char 사용

아래 문장을 escape char를 사용해 바꾸세요.

```
println("${'"'}Hey${'"'}, I have only $amount${'$'}.")
```

Safe call (1/2)

- 프로그램 실행 중 **null** 인 객체에 접근하면,
  - **NullPointerException** (NPE) ➔ 실행 중단!!!
- 변수(객체)가 null 값을 갖는지 여부를 직접 지정
  - **null** 값을 허용하도록 하려면 type 뒤에 ?를 붙임

```
var str: String
str = null          // 에러 발생
```

```
var str:String?
str = null
```

```
fun main( ) {
    var str:String? = null
    nullCheck(str)
}

fun nullCheck(s:String?) {
    if (s == null)
        println("\"$s\" is null")
    else
        println("\"$s\" is NOT null")
}
```

Safe call : Quiz

- 코드 실행 결과는 ?

```
fun main( ) {
    var str:String? = null
    nullCheck(str)
    emptyCheck(str)

    var str2:String? = ""
    nullCheck(str2)
    emptyCheck(str2)
}
```

```
fun nullCheck(s:String?) {
    if (s == null)
        println("\"$s\" is null")
    else
        println("\"$s\" is NOT null")
}

fun emptyCheck(s:String?) {
    if (s?.isEmpty() == true)
        println("\"$s\" is empty")
    else
        println("\"$s\" is NOT empty")
}
```

Safe call (2/2)

```
fun main(){
    var str: String? = "Hello, Kotlin"
    str = null      // NPE 예외가 발생하도록 str 값을 null로 변경

    println("str: $str length = ${str.length}")
}
```

Error 발생

```
println("str: $str length = ${str?.length}")
```

변수 이름 뒤에 ?를 붙임
➔ safe call

```
val len = if (str != null) str.length else -1
println("str: $str length = ${len}")
```

```
val len = str?.length ?: -1
println("str: $str length = ${len}")
```

변수 이름 뒤에 ?를 붙이고,
Elvis 연산 기호(?:) 사용

Smart cast와 Type check

```
fun main() {
    var num:Number = 8L
    val str:Any

    typeCheck(num)
    typeCheck(8)
    var ld = num as Long
    typeCheck(ld)

    str = "Hello, Kotlin"
    if (str is String)
        println("\"$str\" is ${str.javaClass}")
}

fun typeCheck(x: Any) {
    if (x is Int)
        println("$x is ${x.javaClass}")
    else if (x !is Int)
        println("$x is NOT Int. The type is ${x.javaClass}")
}
```

Number : 숫자를 대표하는 타입.

Any : 최상위 기본 클래스.
어떤 type으로도 변환 가능.

as 를 사용한 smart cast

smart cast : 타입 추론.
컴파일러가 자동으로 Long 형으로 변환.

What to do next?

- Basic types
- 제어문과 반복문
- 배열

조건문 : if-else (1/2)

```
fun main() {
    val a = 12
    val b = 7

    maxValue(a, b)
}

fun maxValue(x:Int, y:Int): Int {
    if (x > y) {
        println("$x is chosen.")
        return x
    } else {
        println("$y is chosen.")
        return y
    }
}
```

block의 마지막 식을 변수 max에 할당

```
fun main() {
    val a = 12
    val b = 7

    val maxValue = if (a > b) {
        println("$a is chosen.")
        a
    } else {
        println("$b is chosen.")
        b
    }
    println(maxValue)
}
```

```
fun maxValue(x: Int, y: Int)= if (x > y) x else y
```

조건문 : if-else (2/2)

Non-null assertion(!!)
절대 null이어서는 안됨.
Null 이면 exception 발생

```
fun main() {
    print("Enter the score: ")
    val score = readLine()!!.toFloat()
    var grade = 'F'

    if (score >= 90)
        grade = 'A'
    else if (score in 80.0..89.9)
        grade = 'B'
    else if (score in 70.0..79.9)
        grade = 'C'

    println("Your grade is $grade")
}
```

in 연산 기호와
범위 지정(range check) 연산

```
else if (score >= 80.0 && score <= 89.9)
    grade = 'B'
```

조건문 : when (1/3)

```
fun main() {
    checkValue(1)
    checkValue(3)
    checkValue(5)
}

fun checkValue(x: Int) {
    when (x) {
        1 -> println("x is 1")
        2 -> println("x is 2")
        3, 4 -> println("x is 3 or 4")
        else -> {
            println("x is greater than or equal to 5")
        }
    }
}
```

Java의 switch-case 구문에 해당

break 문이 없음

조건문 : when (2/3)

```
fun main() {
    print("Enter the score: ")
    val score = readLine()!!.toFloat()
    var grade = 'F'

    if (score >= 90)
        grade = 'A'
    else if (score in 80.0..89.9)
        grade = 'B'
    else if (score in 70.0..79.9)
        grade = 'C'

    println("Your grade is $grade")
}
```

```
fun main() {
    print("Enter the score: ")
    val score = readLine()!!.toFloat()
    var grade = 'F'

    when (score) {
        in 90.0..100.0 -> grade = 'A'
        in 80.0..89.9 -> grade = 'B'
        in 70.0..79.9 -> grade = 'C'
    }
    println("socre: $score, grade: $grade")
}
```

조건문 : when (3/3) – enum 클래스

```
enum class Color {
    RED, ORANGE, YELLOW, GREEN, BLUE, INDIGO, VIOLET
}
```

무지개 일곱 가지 색

```
fun main() {
    println(getMnemonics(Color.BLUE))
    println(getWarmth(Color.ORANGE))
}
```

```
fun getMnemonics(color: Color): String {
    return when (color) {
        Color.RED -> "Richard"
        Color.ORANGE -> "Of"
        Color.YELLOW -> "York"
        Color.GREEN -> "Gave"
        Color.BLUE -> "Battle"
        Color.INDIGO -> "In"
        Color.VIOLET -> "Vain"
        else -> {
            "Not a defined color"
        }
    }
}
```

```
fun getWarmth(color: Color): String {
    return when (color) {
        Color.RED, Color.ORANGE, Color.YELLOW -> "warm"
        Color.GREEN -> "neutral"
        Color.BLUE, Color.INDIGO, Color.VIOLET -> "cold"
        else -> {
            "Not a defined color"
        }
    }
}
```

Range and loop : for, while, do-while

```
fun main() {
    for (i in 1..100) {
        print(fizzBuzz(i))
        if (i % 10 == 0) println()
    }
    for (i in 100 downTo 1 step 2) {
        print(fizzBuzz(i))
        if (i % 10 == 0) println()
    }
}

fun fizzBuzz(i:Int):String {
    return when {
        i % 15 == 0 -> "FizzBuzz"
        i % 3 == 0 -> "Fizz"
        i % 5 == 0 -> "Buzz"
        else -> "$i"
    }
}
```

while 루프

```
var j = 1
while (j <= 100) {
    print(fizzBuzz(j))
    if (j % 10 == 0) println()
    j++
}
```

do-while 루프

```
var j = 1
do {
    print(fizzBuzz(j))
    if (j % 10 == 0) println()
    j++
} while (j<=100)
```

What to do next?

- Basic types
- 제어문과 반복문
- **배열**

One dimensional arrays

```
fun main() {
    val numbers = arrayOf(1, 2, 3)
    val mixedArray = arrayOf(7, "Kotlin", false)

    val intOnlyArray = arrayOf<Int>(1, 2, 3)
    val intOnlyArray2 = intArrayOf(4, 5, 6, 7)

    var i = intOnlyArray[0]
    var i2 = intOnlyArray2.get(2)
    println("i=$i, i2=$i2")
    println("${intOnlyArray.size}, ${intOnlyArray2.size}")
    println("${intOnlyArray.first()}, ${intOnlyArray2.last()}")

    intOnlyArray[0] = 0
    intOnlyArray2.set(3, 9)
    println(intOnlyArray.contentToString())
    println(intOnlyArray2.contentToString())
}
```

배열 원소의 type에 제한이 없음

배열 원소의 type을 한 가지로 제한
charArrayOf, **boolean**ArrayOf, **long**ArrayOf
short**byte**ArrayOf, …

배열의 원소 값 변경

배열의 모든 원소를 출력

1차원 배열 원소 출력 (1/3)

```
fun main() {
    val intOnlyArray = arrayOf<Int>(1, 2, 3)
    val intOnlyArray2 = intArrayOf(4, 5, 6, 7)

    showElement(intOnlyArray)
    showElement(intOnlyArray2)
}

fun showElement(arr:Array<Int>) {
    for (i in arr)
        print("$i\t")
    println()
}
```

배열 원소를 출력하기 위한 **showElement** 함수는 **arrayOf <Int>** 타입을 갖는 배열만 사용 가능.

다른 타입(intArrayOf)으로 선언한 배열은 에러 발생

해결 방법은?

```
showElement2(intOnlyArray2)

fun showElement2 (arr:IntArray) {
    for (i in arr)
        print("$i\t")
    println()
}
```

타입에 맞게 정의한 함수를 추가로 정의

이런 방법 말고는 없을까?

1차원 배열 원소 출력 (2/3)

```
fun main() {
    val intOnlyArray = arrayOf<Int>(1, 2, 3)
    val intOnlyArray2 = intArrayOf(4, 5, 6, 7)

    intOnlyArray.forEach { element -> print("$element ") }
    println()
    intOnlyArray2.forEachIndexed {
            i, e -> println("intOnlyArray2[$i] = $e")
    }

    val iter:Iterator<Int> = intOnlyArray.iterator()
    while (iter.hasNext()) {
        val e = iter.next()
        print("$e ")
    }
    println()
}
```

배열 객체가 갖고 있는 메소드를 사용
forEach 또는
forEachIndexed

iterator 객체를 생성
hasNext() 와
next() 메소드를 함께 사용

1차원 배열 원소 출력 (3/3)

```
fun main() {
    val words: Array<String> =
        arrayOf("python", "kotlin", "swift")
    val intOnlyArray = arrayOf(4, 5, 6, 7)

    bar(words)
    foo(intOnlyArray)
}

fun foo(arr:Array<Int>) {
    for (i in arr)
        println(i)
}

fun bar(arr:Array<String>) {
    for (i in arr)
        println(i)
}
```

T : type parameter
타입을 가변적으로 지정

```
fun main() {
    val words: Array<String> =
        arrayOf("python", "kotlin", "swift")
    val intOnlyArray = arrayOf(4, 5, 6, 7)

    unified<String>(words)
    unified<Int>(intOnlyArray)
}

fun <T> unified(arr:Array<T>) {
    for (i in arr)
        println(i)
}
```

Two dimensional arrays

```
fun main() {
    val array1 = arrayOf(1, 2, 3)
    val array2 = arrayOf(4, 5, 6)
    val array3 = arrayOf(7, 8, 9)

    val arr2d = arrayOf(array1, array2, array3)

    for (e1 in arr2d) {
        for (e2 in e1)
            print("$e2 ")
        println()
    }

    println(arr2d[0][1]) // 1행 2열
    println(arr2d[1][1]) // 2행 2열
    println(arr2d[2][1]) // 3행 2열
}
```

2차원 배열 선언: 3x3

중첩된
2개의 for 루프