File - C:\Users\pepet\IdeaProjects\Variables\src\main\kotlin\Main.kt

```kotlin
import javax.swing.text.StyledEditorKit.BoldAction

fun main() {

////////////////////////////////////////////////////////////////////////////
//////Day 1/////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////

// Declare Variables
    var myInt: Int = 4
    var myUInt: UInt = 4u
    var myLong: Long = 5L
    var myFloat: Float = 4.2f
    var myDouble: Double = 4.3
    var myHexInt: Int = 0x000A
    var myBinInt: Int = 0b0111
    var myChar: Char = 'D'
    var myByte: Byte = 2
    var myShort: Short = 5
    var myString: String = "inha"

// Type Casting
    myInt = myLong.toInt()

// Bit Operator (shifting, and, or, xor)
    var leftShift  = 1 shl 2 // shift left,  0100
    var rightShift = 0b0100.shr(2) // shift right, 0001
    var INT_MAX: UInt = (1 shl 31).toUInt() // shift left, 2^31, 214748364
    println(INT_MAX)
    var and = 1 and 0x00001111
    var or  = 1 or  0x00001111
    var xor = 1 xor 0x00001111

// String with Double Quotes, or Triple Double Quotes(No need escape letters)
    var myString1: String = "<Sale>\nPrice : \$100,000"
    var myString2: String = """<Sale>
Price : $100,000"""

// Array Declare
    var myArray = arrayOf(1,2,3)
    var mySquareArray1 = Array(10, {k -> k * k }) // {0,1,4,9,16,...,1024}
    var mySquareArray2 = Array(10, { it * it }) // {0,1,4,9,16,...,1024}

// Array Print
    println(myArray.contentToString()) // "[1, 2, 3]"
    println(myArray.joinToString()) // Only possible when element is primitive, like Array<Int>.

// Range operator '..', 'in', 'downTo', 'rangeTo', 'step', 'reversed'
// Remember : Range is defined by Start, End, Delta(step).
    val aToz = "A".."Z"
    val isCapitalLetter = "c" in aToz // false
    val myDescendingOrder1 = 5.downTo(1) // range 5,4,3,2,1
    val myDescendingOrder2 = 5 downTo 1 // range 5,4,3,2,1
    val myAscendingOrder1 = 5.downTo(9) // range 5,6,7,8,9
    val myAscendingOrder2 = 5 downTo 9 // range 5,6,7,8,9
    val my13579_1 = (1..10).step(2) // range 1,3,5,7,9
    val my13579_2 = 1..10 step 2 // range 1,3,5,7,9
    val my97531_1 = my13579_1.reversed() // range 9,7,5,3,1

// for loops
    // Remember : for ('elem' in 'range')

    // 1. .. operator
    for (i in 1..5){
        print(i)
    }; println() // 12345

    // 2. Array
    for (i in intArrayOf(0,1,0,5,3,1,8,6,4,6,1)){
        print(i)
    }; println() // 01053186461

    // 3. Descending Order
    for (i in 5 downTo 1){
        print(i)
    }; println() // 54321

    // 4. Descending Order, step
    for (i in 9 downTo 0 step 2){
        print(i)
    }; println() // 97531

    // 5. String
    val tmpString: String = "InHa"
    for (i in tmpString){
        print(i.toString()+" ")
    }; println() // I n h a

    // 6. When you need index, use 'indices'.
```

File - C:\Users\pepet\IdeaProjects\Variables\src\main\kotlin\Main.kt

```kotlin
    val tmpArray = arrayOf(1,2,3)
    for (i in tmpArray.indices){
        println("Index $i : ${tmpArray[i]}")
    }

// class declare(No need to use 'new')
    class Vector2D(var x: Double, var y: Double){
        constructor() : this(0.0, 0.0)
        fun biggerValue(): Double = if (x>y) x else y // return statement with one-line if-else!
    }
    var myVec = Vector2D(3.0, 4.0)
    println("${myVec.x} ${myVec.y} ${myVec.biggerValue()}")
    var myVec2 = Vector2D()
    println("${myVec2.x} ${myVec2.y} ${myVec2.biggerValue()}")

// How to print many variables(Use '$' in ""!)
    val tmpInt1 = 1;
    val tmpInt2 = 2;
    val tmpInt3 = 3;
    println("$tmpInt1, $tmpInt2, $tmpInt3") // 123

    val DoNotExecuteHere = false;
// Get User Input, and Store in List
    if (DoNotExecuteHere) {
        val myList: List<Int>? = readLine()?.split(" ")?.map { it.toInt() }
        // ? : Means that it's nullable.
        // readLine()? : Get user input as ASCII String.
        // split(" ") : Return List<T> that delimiter is " ".
        // map{code} : Apply 'code' in to every element, and change them.
        // it : Name of Variable in Lambda Function.
        // it.toInt() : Means to convert every element into Int.
        // Ex) input : "1 2" -> result : myList = [1,2]
    }

// Get 2 numbers by user, print sum
    if (DoNotExecuteHere) {
        print(readln().sumOf { it - ' ' } - 32)
        // readln() : Get user input as ASCII String.
        // sumOf : Function that return Sum, which have Lambda Function as it's argument
        // {it-' '} : Subtract ' ' for every char in String. So, it subtracts ' '(32).
        // -32 : '0' is 48. We have to subtract 16 for each number because we subtracted 32 already. So subtract 32 because there are 2 numbers.
        // Ex) input : "1 2" -> result : print 3.
    }

// Get 2 numbers by user, print sum (2)
    if (DoNotExecuteHere) {
        print(readln().split(" ").sumOf { it.toInt() })
        // readln() : Get user input as ASCII String.
        // split(" ") : Return List<T> that delimiter is " ".
        // sumOf : Function that return Sum, which have Lambda Function as it's argument
        // Ex) input : "1 2" -> result : print 3.
    }

////////////////////////////////////////////////////////////////////////////////
//////Day 2/////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////

// Referential Equality, Structural Equality
    // Referential Equality : 2 references point to same instance of memory.
    class Square(width: Double, height: Double) {}
    var myEntity1 = Square(1.0, 4.0)
    var myEntity2 = Square(1.0, 4.0)
    val sameReference = myEntity1 === myEntity2 // false
    // Structural Equality : 2 seperate instance of memory but same value.
    val sameStructure = myEntity1 == myEntity2 // true`

// if statement, if expression
    // if statement example (same as c++ if statement)
    var tmpValue = 1
    var tmpBool = myEntity1 == myEntity2
    if (tmpBool) {
        tmpValue = 10
    } else {
        tmpValue = 20
    }
    // if expression example (same as c++ ? operator)
    tmpValue = if (tmpBool) 10 else 20 // tmpBool ? 10 : 20 (C++)

// Nullable variable
    var myStr1: String = "Not nullable String"
    var myStr2: String? = "Nullable String" // this is nullable!

// Smart cast (Type checking)
    /**
     * // JAVA CODE
     * public void printStringLength(Object obj) {
     *     if (obj instanceof String) {
     *         String str = (String) obj
```

```
178      *         System.out.print(str.length())
179      *     }
180      * }
181      */
182
183     // KOTLIN CODE 1
184     fun printStringLength(any: Any) {
185         if (any is String) {
186             println(any.length)
187         }
188     }
189
190     // KOTLIN CODE 2
191     fun isNotStringOrEmpty(any: Any): Boolean {
192         return any !is String || any.length == 0 // !is operator
193     }
194
195 // Explicit cast (var as type)
196     // code 1.
197     fun returnString1(any: Any): String? {
198         val tmpString = any as String
199         return tmpString
200     }
201     /**
202      * chatGPT Explanation
203      * This code snippet attempts to cast the any parameter to a String type using the unsafe cast operator as.
204      * If any is not a String type, this will result in a ClassCastException at runtime.
205      * This code does not handle nullability, so if any is null, it will also result in a NullPointerException.
206      */
207
208     // code 2.
209     fun returnString2(any: Any): String? {
210         val tmpString = any as String?
211         return tmpString
212     }
213     /**
214      * chatGPT Explanation
215      * This code snippet attempts to cast the any parameter to a nullable String type using the safe cast operator as?.
216      * This means that if any is not a String type, tmpString will be set to null instead of throwing a
    ClassCastException.
217      * This code handles nullability by casting any to a nullable String type, which means that if any is null,
    tmpString will also be null.
218      */
219
220     // code 3.
221     fun returnString3(any: Any): String? {
222         val tmpString = any as? String
223         return tmpString
224     }
225     /**
226      * chatGPT Explanation
227      * This code snippet is similar to Code 2, but it uses the safe cast operator as? instead of as.
228      * This means that if any is not a String type, tmpString will be set to null instead of throwing a
    ClassCastException.
229      * This code also handles nullability by casting any to a nullable String type, which means that if any is null,
    tmpString will also be null.
230      */
231
232     /**
233      * Additional Explanation of chatGPT
234      * The main difference between Code 2 and Code 3 is that Code 3 is more concise, as it combines the safe cast
    operator with the nullable type.
235      * This makes the code more readable and less error-prone, as it reduces the chances of accidentally casting to a
    non-nullable type.
236      */
237
238     // code 4.(Written by chatGPT)
239     fun returnString(any: Any?): String? = any as? String
240     /**
241      * In this version:
242      * The any parameter is explicitly declared as nullable using Any?.
243      * The function uses the safe cast operator as? to attempt to cast any to a String type. If any is not a String,
    the result will be null.
244      * The function returns the result of the cast as a nullable String? type.
245      * By using a single expression with an implicit return type, the function is more concise and easier to read.
246      * This version of returnString function improves type safety and null safety while also being more concise than
    the previous version.
247      * By using the safe cast operator, it avoids the risk of a ClassCastException and returns null if the cast fails,
    making it null-safe as well.
248      */
249
250
251
252
253
254
255 }
256
257
```

```
258
259
260
261
262
```