# LECTURE 9.1: STRINGS

BY LINA HAMMAD & AHMAD BARGHASH

In this lecture, you will learn about how to declare, use and manipulate strings in Kotlin, string templates, and few commonly used string properties and functions with the help of examples.

# KOTLIN STRING

- Strings are a arrays of characters. For example, **"Hello there!"**.
- In Kotlin, all strings are objects of String class.

# HOW TO CREATE A STRING VARIABLE?

- To define a String variable in Kotlin use the **" "**

```
val myString = "Hey there!"
```

- Here, myString is a variable of type String.

- Also, you can declare variable of type String and specify its type in one statement and initialize the variable in another statement later in the program.

```
val myString: String
... .. ...
myString = "Howdy"
```

# HOW TO ACCESS CHARACTERS OF A STRING?

- To access elements (character) of a string, index access operator is used. For example,

```
fun main() {
    val myString = "Hey there!"
    val item = myString[2]

}
```

- Here, item variable contains y, third character of the myString string. It's because indexing in Kotlin starts from 0 not 1.

```
fun main() {
    val myString = "Hey there!"
    var item: Char

    item = myString[0]    // item contains 'H'
    item = myString[9]    // item contains '!'
    item = myString[10]   // Error! String index is out of range
    item =  myString[-1]  // Error! String index is out of range

}
```

# EXAMPLE: ITERATE THROUGH A STRING

- If you need to iterate through elements of a string, you can do it easily by using a for loop.

```
fun main() {
        val myString = "Hey!"
        for (item in myString) {
            println(item)
        }
}
```

The output is:

```
H
e
y
!
```

# STRINGS IN KOTLIN ARE IMMUTABLE

- Like Java, strings are immutable in Kotlin. This means, you cannot change individual character of a string. For example,

```
fun main() {
    var myString = "Hey!"
    myString[0] = 'h'      // Error! Strings
}
```

- However, you can reassign a string variable again if you declared the variable using keyword var. (Remember: Kotlin var Vs val).

```
fun main(args: Array<String>) {
    var myString = "Hey!"
    println("myString  = $myString")
    myString = "Hello!"
    println("myString  = $myString")
}
```

The output is:

```
myString  = Hey!
myString  = Hello!
```

# STRING LITERALS

- A literal is the source code representation of a fixed value. For example, "Hey there!" is a string literal that appears directly in a program without requiring computation (like variables).

- There are two types of string literals in Kotlin:
  1. **Escaped string**
  2. **Raw String**

# ESCAPED STRING

- A escaped string may have escaped characters in them. For example, \n is an escape character which inserts a newline in the text where it appears.

```
val myString = "Hey there!\n"
```

- Here is a list of escape characters supported in Kotlin:
  - \t - Inserts tab
  - \b - Inserts backspace
  - \n - Inserts newline
  - \r - Inserts carriage return  (Send the beginning of the line)
  - \' - Inserts single quote character
  - \" - Inserts double quote character
  - \\ - Inserts backslash
  - \$ - Inserts dollar character

# RAW STRING

- A raw string can contain newlines (not new line escape character) and arbitrary text. A raw string is delimited by a triple quote """. For example,

```
fun main(args: Array<String>) {
    val myString = """
    for (character in "Hey!")
        println(character)
"""
    print(myString)
}
```

The output is:

```
for (character in "Hey!")
    println(character)
```

- You can remove the leading whitespaces of a raw string using **trimMargin() function**.

# EXAMPLE: PRINTING RAW STRING

```
fun main() {
    println("Output without using trimMargin function:")
    val myString = """
    |Kotlin is interesting.
    |Kotlin is sponsored and developed by JetBrains.
"""
    println(myString)
    println("Output using trimMargin function:\n")
    println(myString.trimMargin())
}
```

The output is:

```
Output without using trimMargin function:


    |Kotlin is interesting.
    |Kotlin is sponsored and developed by JetBrains.


Output using trimMargin function:


Kotlin is interesting.
Kotlin is sponsored and developed by JetBrains.
```

- By default, **trimMargin()** function uses | as margin prefix. However, you can change it by passing a new string to this function.

# EXAMPLE: TRIMMARGIN() WITH ARGUMENT

```
fun main(args: Array<String>) {
    val myString = """
    !!! Kotlin is interesting.
    !!! Kotlin is sponsored and developed by JetBrains.
"""
    println(myString.trimMargin("!!! "))
}
```

The output is:

```
Kotlin is interesting.
Kotlin is sponsored and developed by JetBrains.
```

# KOTLIN STRING TEMPLATES

- Kotlin has an awesome feature called string templates that allows strings to contain template expressions.
- A string template expression starts with a dollar sign $.

# EXAMPLE: KOTLIN STRING TEMPLATE

```
fun main(args: Array<String>) {
    val myInt = 5;
    val myString = "myInt = $myInt"
    println(myString)
}
```

The output is:

```
myInt = 5
```

- It is because the expression **$myInt** (expression starting with $ sign) inside the string is evaluated and concatenated into the string.

# EXAMPLE: STRING TEMPLATE WITH RAW STRING

```
fun main(args: Array<String>) {
    val a = 5
    val b = 6
    val myString = """
    |${if (a > b) a else b}
"""
    println("Larger number is: ${myString.trimMargin()}")
}
```

The output is:

```
Larger number is: 6
```

# FEW STRING PROPERTIES AND FUNCTIONS

- Since literals in Kotlin are implemented as instances of String class, you can use several methods and properties of this class.

- **length** property - returns the length of character sequence of a string.
- **compareTo** function - compares this String (object) with the specified object. Returns 0 if the object is equal to the specified object.
- **get** function - returns character at the specified index. You can use index access operator **[ ],** instead of get function as index access operator internally calls get function.
- **plus** function - returns a new string which is obtained by the concatenation of this string and the string passed to this function. You can use + operator instead of plus function as **+** operator calls plus function under the hood.
- **subSequence** Function - returns a new character sequence starting at the specified start and end index.

# EXAMPLE: STRING PROPERTIES AND FUNCTION

```
fun main(args: Array<String>) {
    val s1  = "Hey there!"
    val s2 = "Hey there!"

    var result: String
    println("Length of s1 string is ${s1.length}.")
    result = if (s1.compareTo(s2) == 0) "equal" else "not equal"
    println("Strings s1 and s2 are $result.")

    // s1.get(2) is equivalent to s1[2]
    println("Third character is ${s1.get(2)}.")
    result = s1.plus(" How are you?") // result = s1 + " How are you?"
    println("result = $result")
    println("Substring is \"${s1.subSequence(4, 7)}\"")
}
```

The output is:

```
Length of s1 string is 10.
Strings s1 and s2 are equal.
Third character is y.
result = Hey there! How are you?
Substring is "the
```

# EXAMPLE 1: COMPARING STRINGS

```
fun main() {
    val style = "Kotlin"
    val style2 = "Kotlin"
    if (style == style2)
        println("Equal")
    else
        println("Not Equal")
}
```

The output is:

```
Equal
```

```
fun main(args: Array<String>) {
    val style = "Kotlin"
    val style2 = "Kotlin"
    if (style.equals(style2))
        println("Equal")
    else
        println("Not Equal")
}
```

The output is:

```
Equal
```

- In the above program, we've two strings style and style2. We simply use equality operator (==) or method *equals* to compare the two strings, which compares the value **`Kotlin`** to **`Kotlin`** and prints Equal.

# EXAMPLE 2: COMPARING STRINGS

```
fun main(args: Array<String>) {
    val style = buildString { "Bold" }
    val style2 = buildString { "Bold" }
    if (style === style2)
        println("Equal")
    else
        println("Not Equal")
}
```

The output is:

```
Not Equal
```

```
fun main(args: Array<String>)
{
    val style2 = buildString { "Bold" }
    if ("Bold" === style2)
        println("Equal")
    else
        println("Not Equal")
}
```

The output is:

```
Not Equal
```

```
fun main(args: Array<String>)
{
    if ("Bold" === "Bold")
        println("Equal")
    else
        println("Not Equal")
}
```

The output is:

```
Equal
```

- In the above program, instead of creating a string using just quotes, we've used a helper method buildString to create a String object.
- Notice the difference between == and ===

# EXAMPLE 3: DIFFERENT WAYS TO COMPARE TWO STRINGS

```
fun main(args: Array<String>) {
    val style = buildString { "Bold" }
    val style2 = buildString { "Bold" }
    var result = style.equals("Bold") // true
    println(result)
    result = style2 === "Bold" // false
    println(result)
    result = style === style2 // false
    println(result)
    result = "Bold" === "Bold" // true
    println(result)
}
```

The output is:

```
true
false
false
true
```

# EXAMPLE 4: COMPARING STRINGS

```
fun main(){

    var str1 = "BeginnersBook"
    var str2 = "beginnersbook"
    println("String Equals? : ${str1.compareTo(str2)}")
}
```

The output is:

```
String Equals? : -32
```

```
fun main(){
    var satr1 = "B"
    var str2 = "b"
    println("String Equals? : ${str1.compareTo(str2)}")
}
```

The output is:

```
String Equals? : -32
```

# EXAMPLE 5: SUBSTRINGS AND CONTAIN STRING

```
fun main(){

    var str = “MobileApplication"

    println("Index from 2 to 5: " +str.subSequence(2,5))
}
```

The output is:

```
Index from 2 to 5: bil
```

```
fun main(){

    var str = “GJU.EDU.JO"

    println("Contains GJU.: ${str.contains("GJU.")}")
}
```

The output is:

```
Contains GJU.: true
```

# EXAMPLE 6: STRING REPLACEMENT

```
fun main(args: Array<String>) {
    var sentence = "T    his is b  ett     er."
    println("Original sentence: $sentence")
    sentence = sentence.replace("\\s".toRegex(), "")
    println("After replacement: $sentence")
}
```

The output is:

```
Original sentence: T    his is b  ett     er.
After replacement: Thisisbetter.
```

- In the aboe program, we use String's replaceAll() method to remove and replace all whitespaces in the string sentence.
- We've used regular expression \\s that finds all white space characters (tabs, spaces, new line character, etc.) in the string. Then, we replace it with "" (empty string literal).

# EXAMPLE 7: STRING REPLACEMENT

```
fun main(args: Array<String>) {
    var sentence = "Tomorrow"
    println("Original sentence: $sentence")
    sentence = sentence.replace("^T".toRegex(), "t")
    println("After replacement: $sentence")}
```

The output is:

```
Original sentence: Tomorrow
After replacement: tomorrow
```

```
fun main(args: Array<String>) {
    var sentence = "Tomorrow"
    println("Original sentence: $sentence")
    sentence = sentence.replace("w$".toRegex(), "WW")
    println("After replacement: $sentence")}
```

The output is:

```
Original sentence: Tomorrow
After replacement: TomorroWW
```

# EXAMPLE 8: STRING REPLACEMENT

```
fun main(args: Array<String>) {
    var sentence = "Tomorrow"
    println("Original sentence: $sentence")
    sentence = sentence.replace("[A-Za-z]".toRegex(), "9")
    println("After replacement: $sentence")}
```

The output is:

```
Original sentence: Tomorrow
After replacement: 99999999
```

```
fun main(args: Array<String>) {
    var sentence = "Tomorrow"
    println("Original sentence: $sentence")
    sentence = sentence.replace("[A-Za-z]{1,}".toRegex(), "9")
    println("After replacement: $sentence")}
```

The output is:

```
Original sentence: Tomorrow
After replacement: 9
```

# EXAMPLE 9: STRING REPLACEMENT

```
fun main(args: Array<String>) {
    var sentence = "Tomorrow"
    println("Original sentence: $sentence")
    sentence = sentence.replace("or.*".toRegex(), "oooooo")
    println("After replacement: $sentence")}
```

The output is:

```
Original sentence: Tomorrow
After replacement: Tomoooooo
```

```
fun main(args: Array<String>) {
    var sentence = "Tomorrow"
    println("Original sentence: $sentence")
    sentence = sentence.replace("o..o".toRegex(), "rrrr")
    println("After replacement: $sentence")}
```

The output is:

```
Original sentence: Tomorrow
After replacement: Tomrrrrw
```