Week 1 : Session 4

Control flow Statements : Loops – (For, While, Do-While)

Agenda

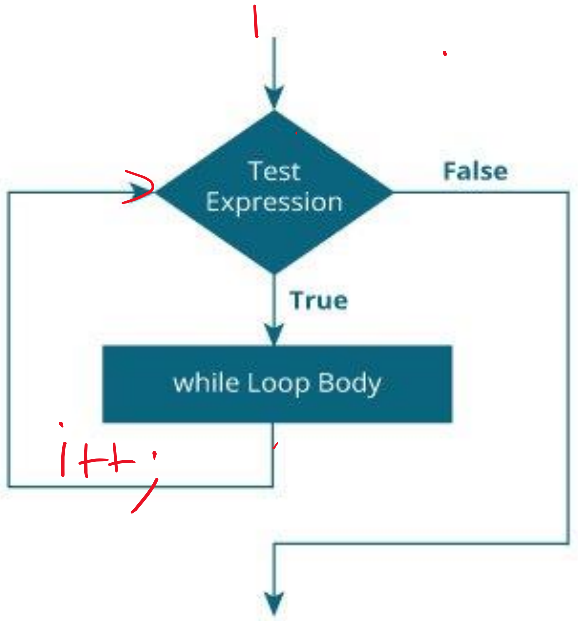
- Kotlin Control Flow - Loops
  - **While Loop**
  - **Do-While Loop**
  - **For Loop**
    - With Range
    - With Array
    - With String
  - **“break” Expression**
  - **“continue” Expression**
- Exercise : Demo Application

Loops : While

- Loop is used in programming to repeat a specific block of code until certain condition is met.
- While loop is called indetermined loop
- Syntax :

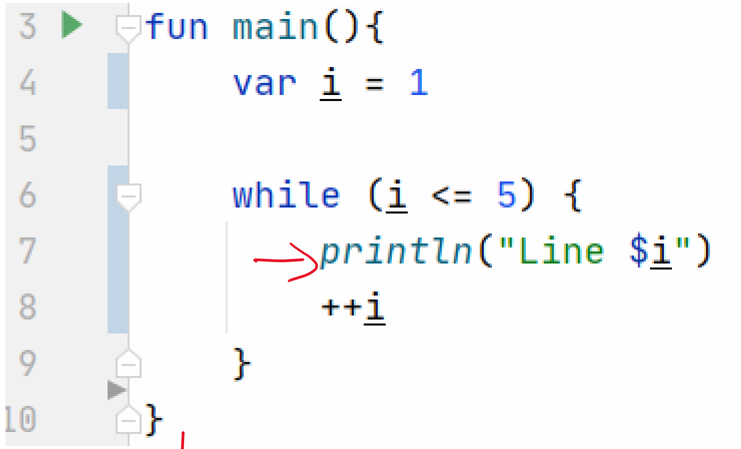
```
while (testExpression) {
    // codes inside body of while loop
}
```

- Steps to write a while loop
  - The test expression inside the parenthesis is a Boolean expression.
  - If the test expression is evaluated to true,
    - statements inside the while loop are executed.
    - then, the test expression is evaluated again.
  - This process goes on until the test expression is evaluated to false.
  - If the test expression is evaluated to false,
  - while loop is terminated.



Loops : While

- Examples

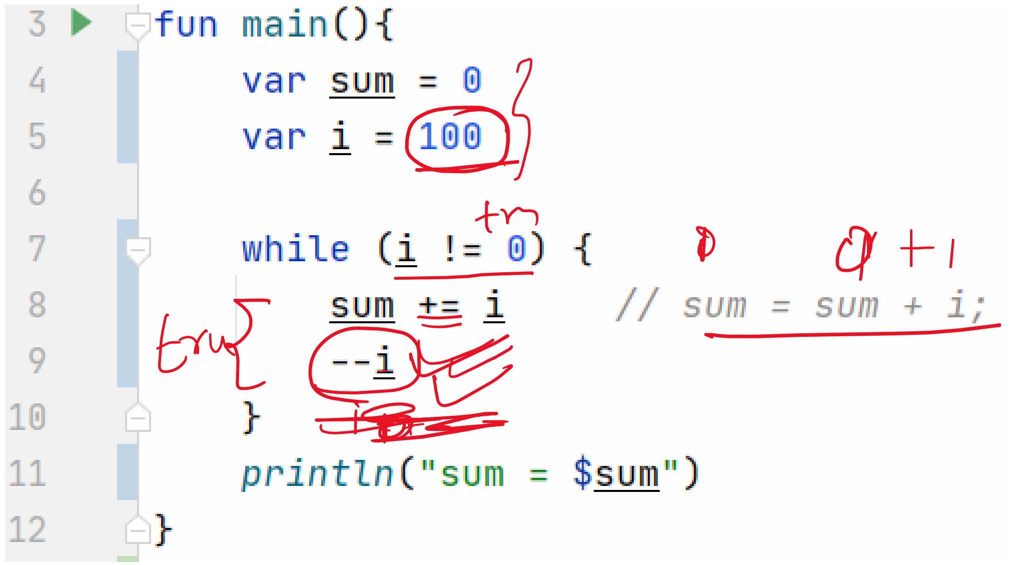


```kotlin
fun main(){
    var i = 1

    while (i <= 5) {
        println("Line $i")
        ++i
    }
}
```

Line 1

⋮

Line 5



```kotlin
fun main(){
    var sum = 0
    var i = 100

    while (i != 0) {
        sum += i      // sum = sum + i;
        --i
    }
    println("sum = $sum")
}
```

true

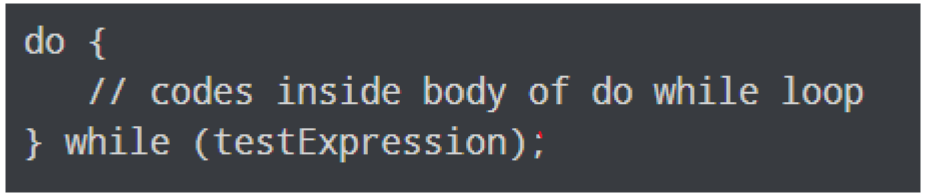
true

0   0 + 1

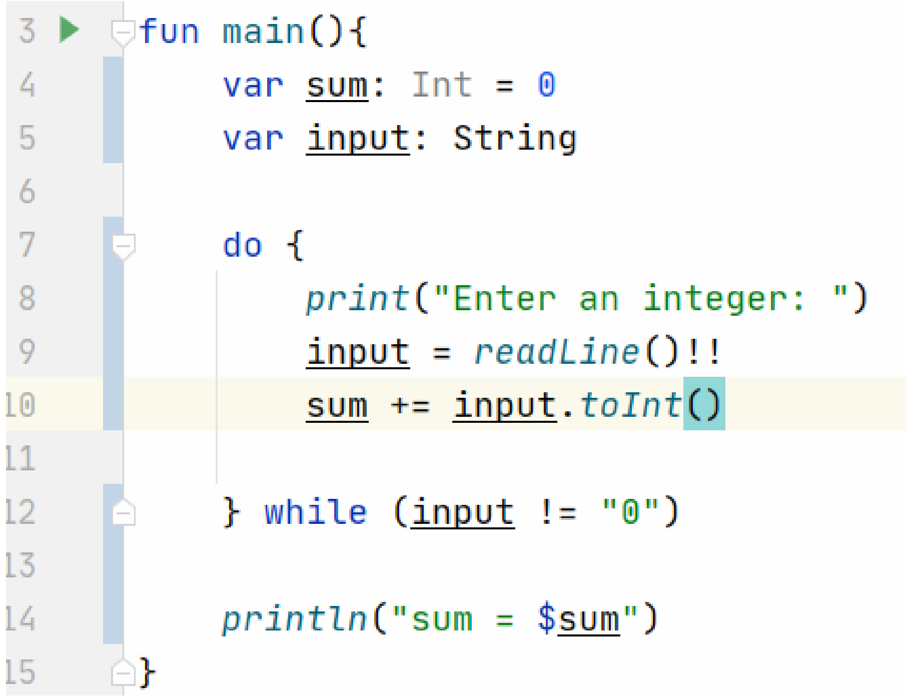
5050

Loops : Do - While

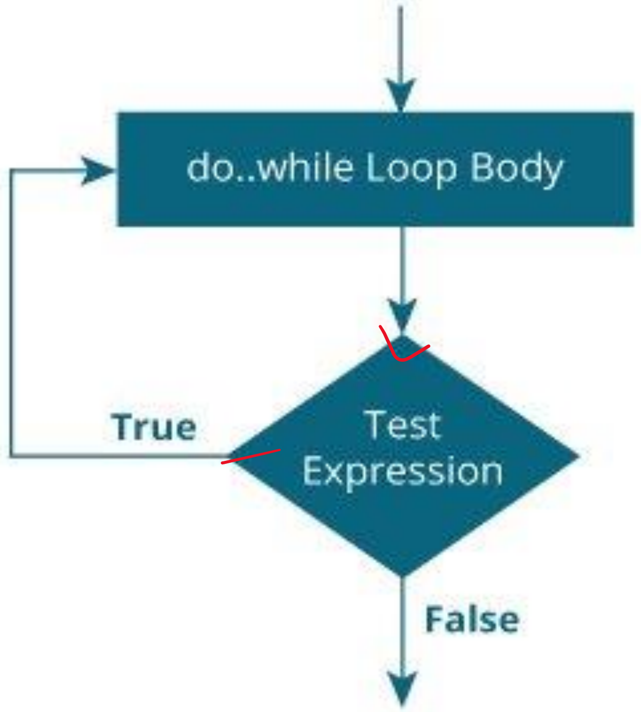
- The do-while loop is similar to while loop except that it tests the condition at the end of the loop.
- The body of do...while loop is executed once before the test expression is checked.



```
do {
   // codes inside body of do while loop
} while (testExpression);
```

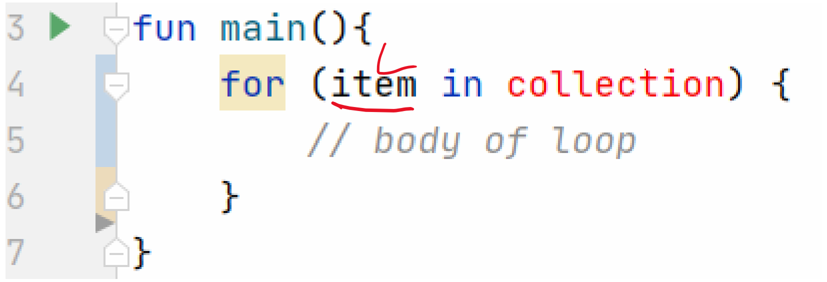


```
fun main(){
    var sum: Int = 0
    var input: String

    do {
        print("Enter an integer: ")
        input = readLine()!!
        sum += input.toInt()

    } while (input != "0")

    println("sum = $sum")
}
```



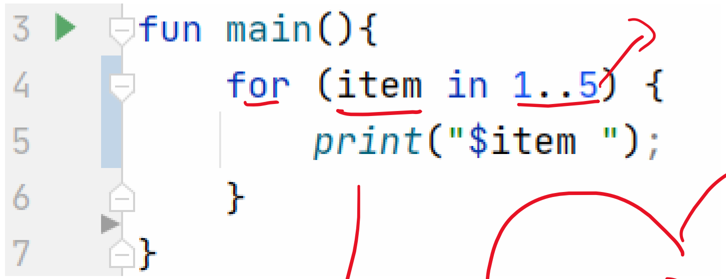
Loops : For

- For loop is used to iterate through ranges, arrays, maps and so on (anything that provides an iterator).
- Syntax of for loop in Kotlin is:



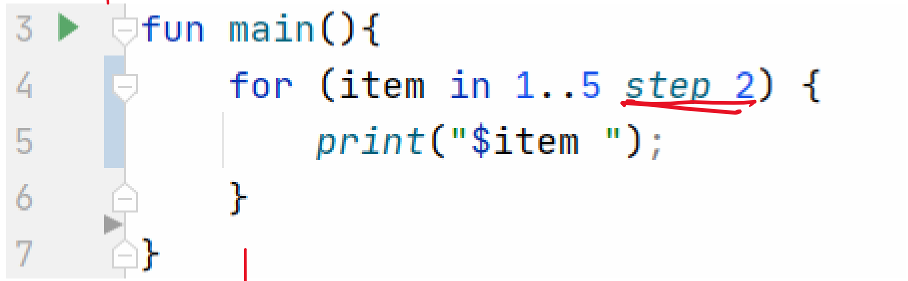
```
fun main(){
    for (item in collection) {
        // body of loop
    }
}
```

- Iterate through range



```
fun main(){
    for (item in 1..5) {
        print("$item ");
    }
}
```

1 2 3 4 5

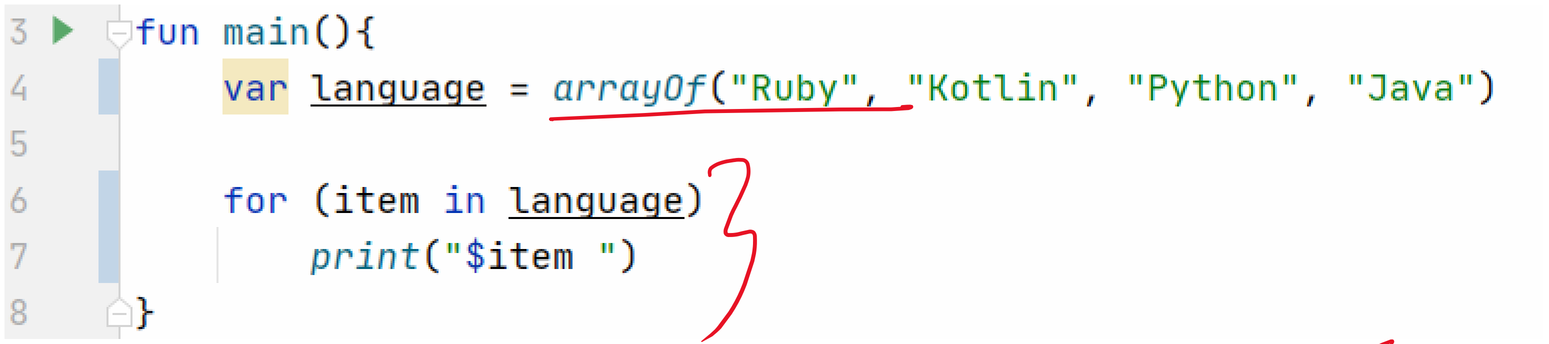


```
fun main(){
    for (item in 1..5 step 2) {
        print("$item ");
    }
}
```

1 3 5

Loops : For

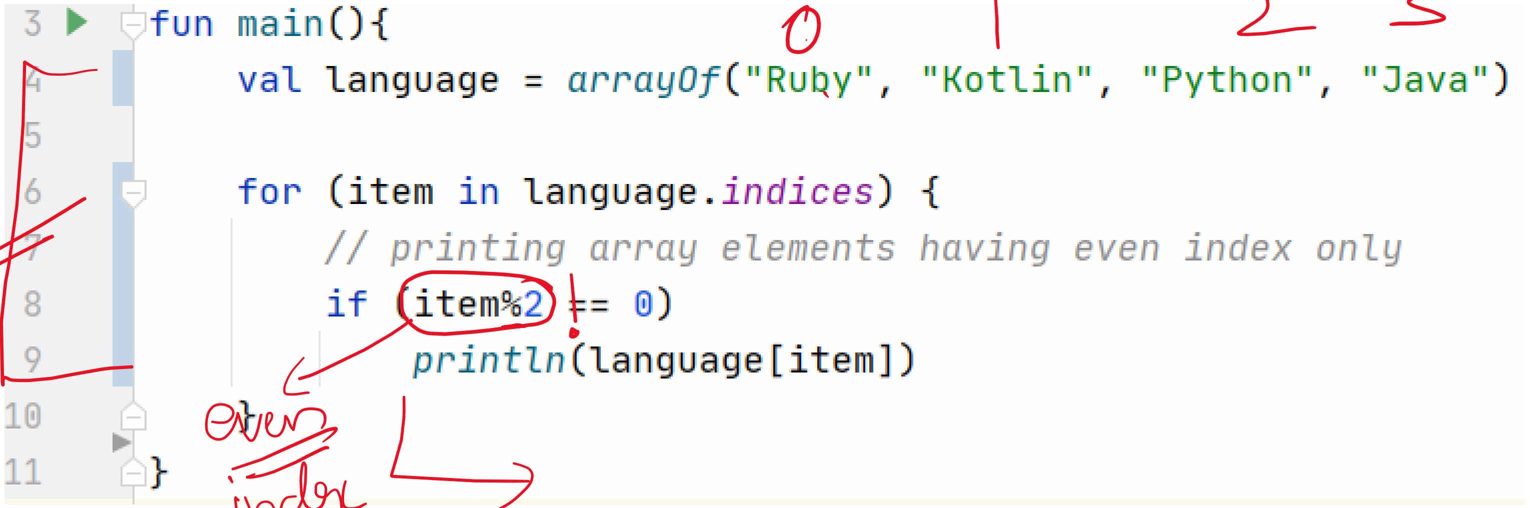
- Iterate through Array



```
fun main(){
    var language = arrayOf("Ruby", "Kotlin", "Python", "Java")

    for (item in language)
        print("$item ")
}
```

- Using index

0 1 2 3



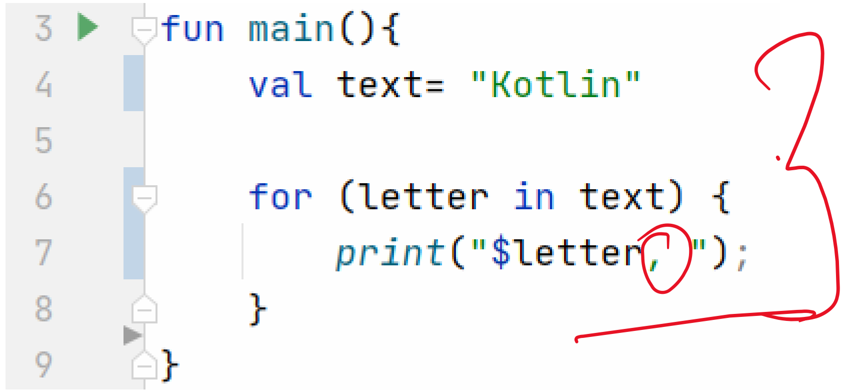
```
fun main(){
    val language = arrayOf("Ruby", "Kotlin", "Python", "Java")

    for (item in language.indices) {
        // printing array elements having even index only
        if (item%2 == 0)
            println(language[item])
    }
}
```

!

even index

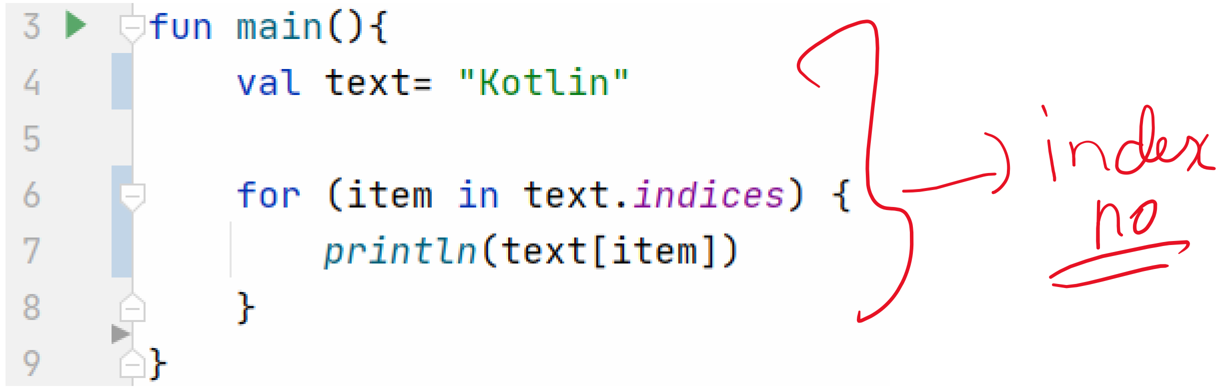
Loops : For

- Iterate through String



```
fun main(){
    val text= "Kotlin"

    for (letter in text) {
        print("$letter, ");
    }
}
```

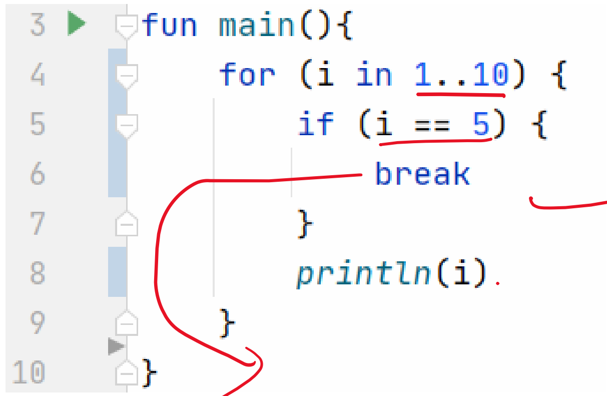
- Using index



```
fun main(){
    val text= "Kotlin"

    for (item in text.indices) {
        println(text[item])
    }
}
```

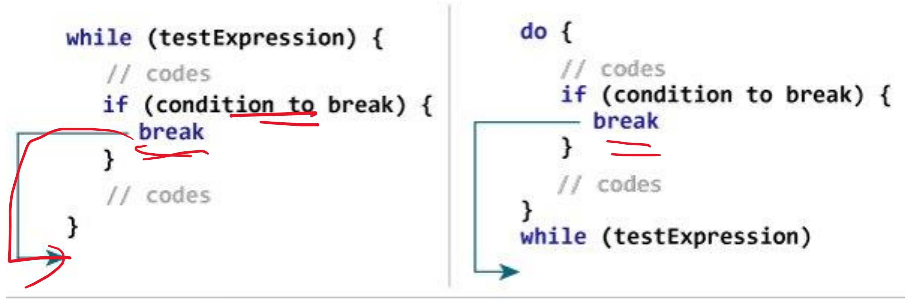
Break Expression

- Break expression is to terminate a loop.
- Unlabelled break
  - It terminates the nearest enclosing loop



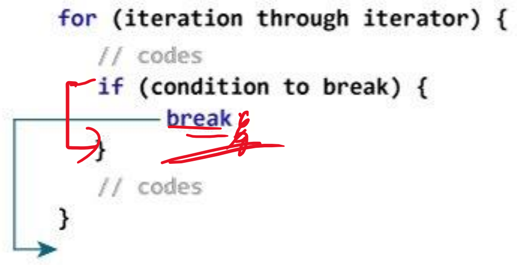
```
fun main(){
    for (i in 1..10) {
        if (i == 5) {
            break
        }
        println(i)
    }
}
```

1,2,3,4



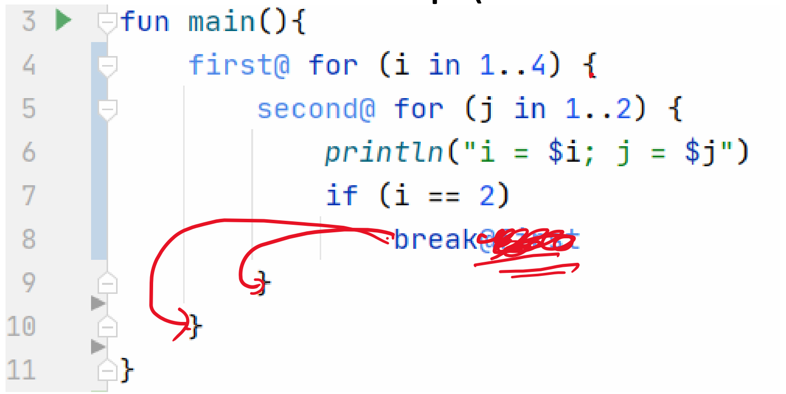
```
while (testExpression) {
    // codes
    if (condition to break) {
        break
    }
    // codes
}
```

```
do {
    // codes
    if (condition to break) {
        break
    }
    // codes
}
while (testExpression)
```

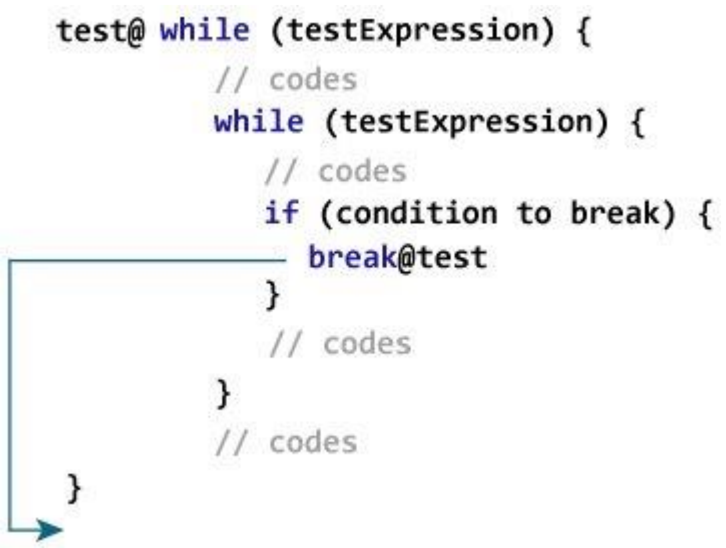


```
for (iteration through iterator) {
    // codes
    if (condition to break) {
        break
    }
    // codes
}
```

- Labelled break
  - terminate the desired loop (can be an outer loop)



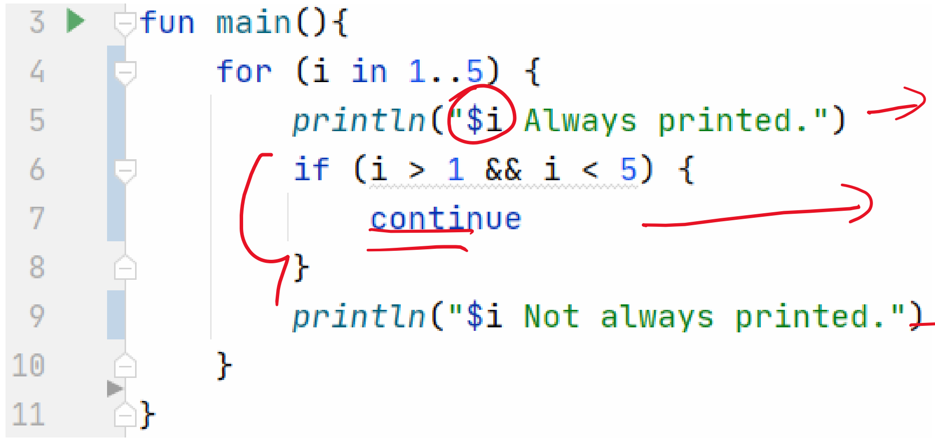
```
fun main(){
    first@ for (i in 1..4) {
        second@ for (j in 1..2) {
            println("i = $i; j = $j")
            if (i == 2)
                break@[illegible]t
        }
    }
}
```



```
test@ while (testExpression) {
    // codes
    while (testExpression) {
        // codes
        if (condition to break) {
            break@test
        }
        // codes
    }
    // codes
}
```

Continue Expression

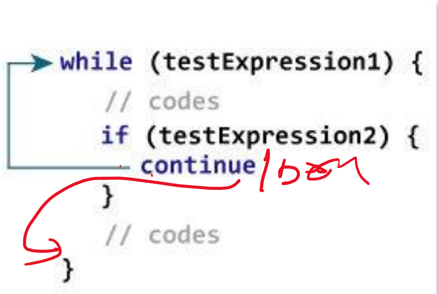
- "continue" is to skip the current iteration of a loop.
- Unlabelled continue
    - It skips to the nearest opening of loop



```kotlin
fun main(){
    for (i in 1..5) {
        println("$i Always printed.")
        if (i > 1 && i < 5) {
            continue
        }
        println("$i Not always printed.")
    }
}
```

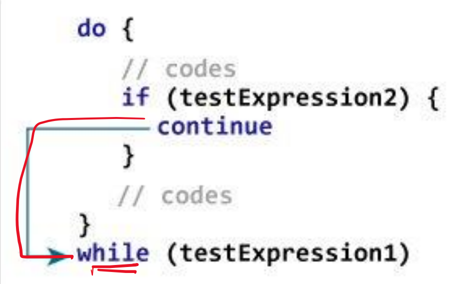
→ 1, 2, 3, 4, 5

→ 2, 3, 4

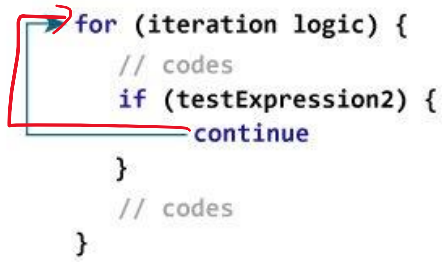


```
while (testExpression1) {
    // codes
    if (testExpression2) {
        continue
    }
    // codes
}
```

loop

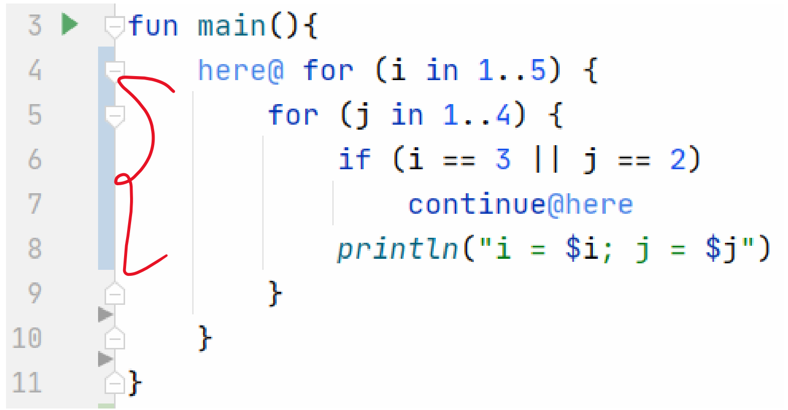


```
do {
    // codes
    if (testExpression2) {
        continue
    }
    // codes
}
while (testExpression1)
```

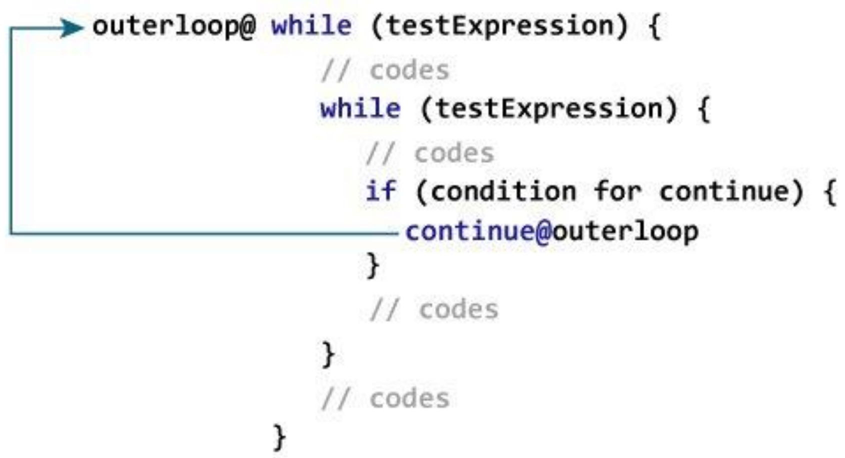


```
for (iteration logic) {
    // codes
    if (testExpression2) {
        continue
    }
    // codes
}
```

- Labelled continue
    - Skips to the desired opening of loop (can be an outer loop)

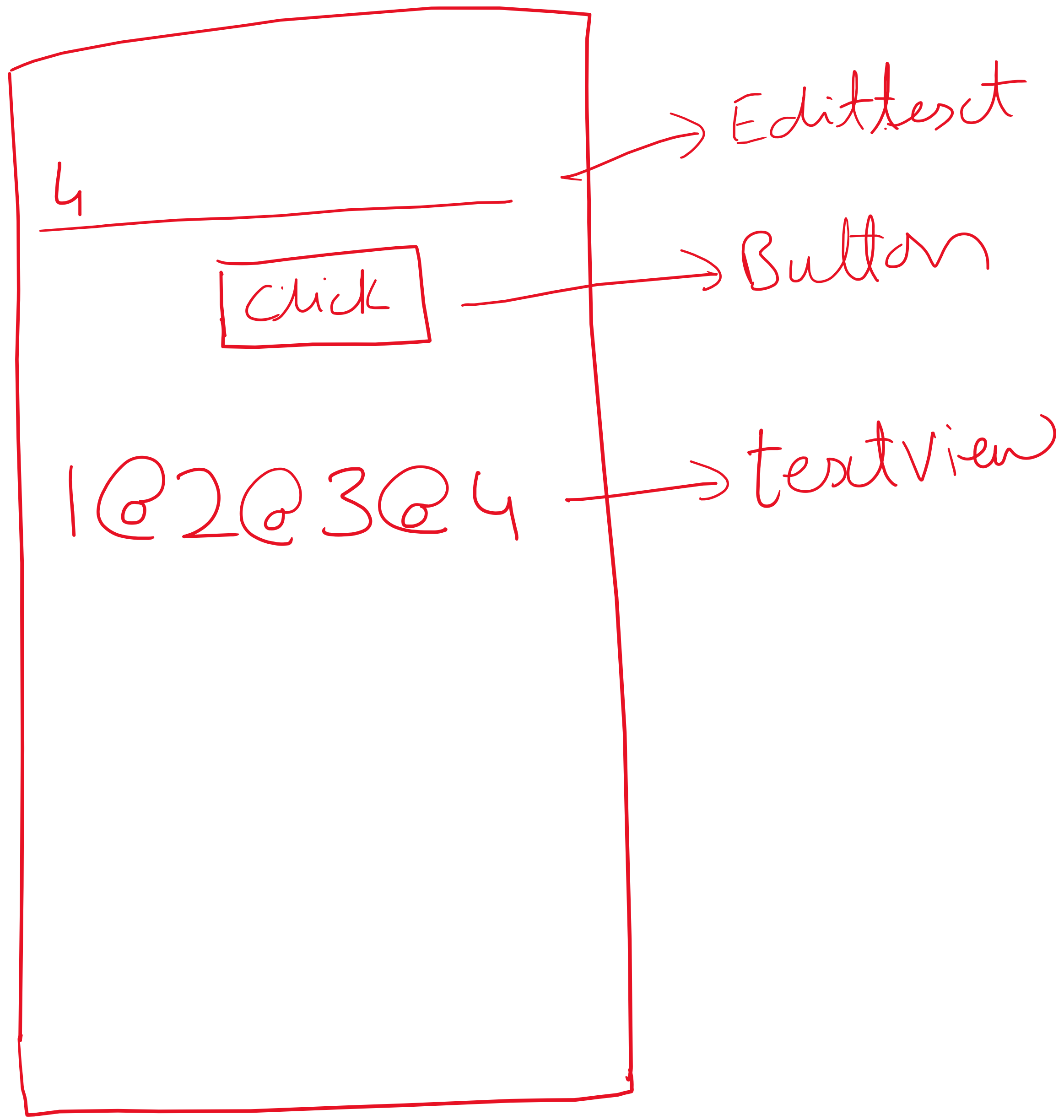


```kotlin
fun main(){
    here@ for (i in 1..5) {
        for (j in 1..4) {
            if (i == 3 || j == 2)
                continue@here
            println("i = $i; j = $j")
        }
    }
}
```



```
outerloop@ while (testExpression) {
    // codes
    while (testExpression) {
        // codes
        if (condition for continue) {
            continue@outerloop
        }
        // codes
    }
    // codes
}
```

Exercise


4
Click
1@2@3@4
Edittext
Button
textview