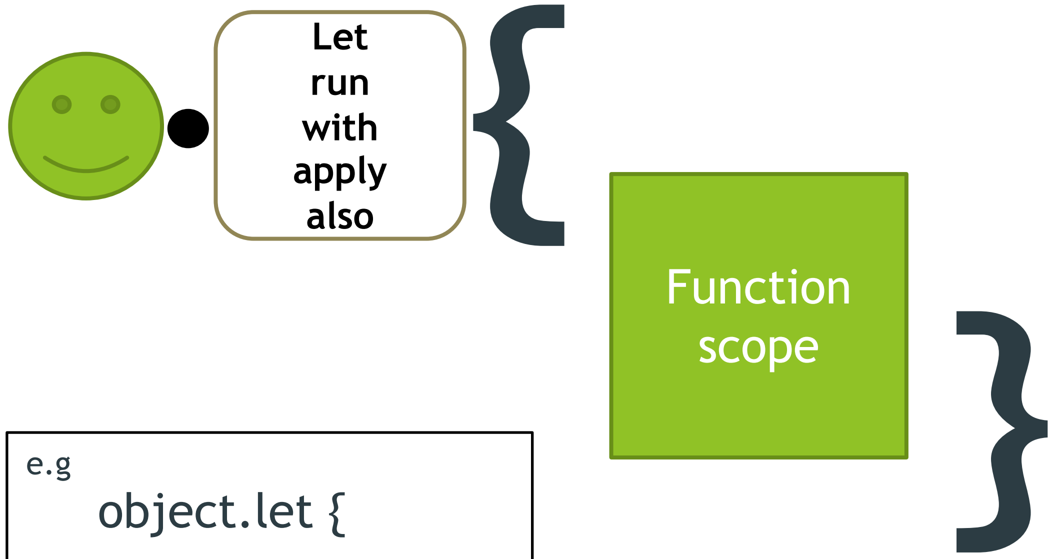
Scope Functions in Kotlin

Scope functions?

Kotlin standard library has extension functions that execute a block of code on Object and they provide a scope using lambda expression that's why they are called **Scope Functions**.

Types

1) let
2) run
3) with
4) apply
5) also



e.g

```
object.let {
            //logic
               }
```

let is mostly used for these scenarios

1) To perform actions on <u>non null object </u>using safe call operator **?.**
2) Multiple functions calls as <u>chain</u>
3) Helps to <u>create local variable </u>with limited scope in lambda
4) Transformation: Let can return anything from object

OR: You can create a local variable in lambda scope

😉 It's easy

```kotlin
person.let { p ->
    print(p.name)
}
```

Person data class

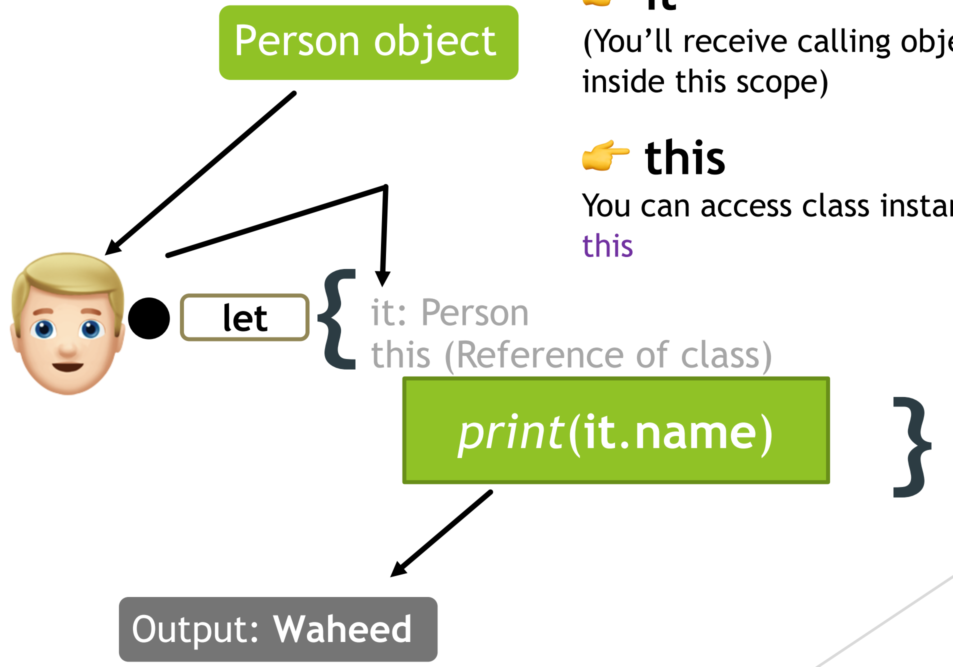
```kotlin
class Person(val name: String, val age: Int)
```

Accessing object properties using let

```kotlin
val person = Person( name: "Waheed",  age: 28)
person.let { it: Person
    print(it.name)
}
```

👉 **it**
(You'll receive calling object as it inside this scope)

👉 **this**
You can access class instance as this



By Waheed Nazir

Let on non null objects using safe call operator ?.

String variable that can hold null value

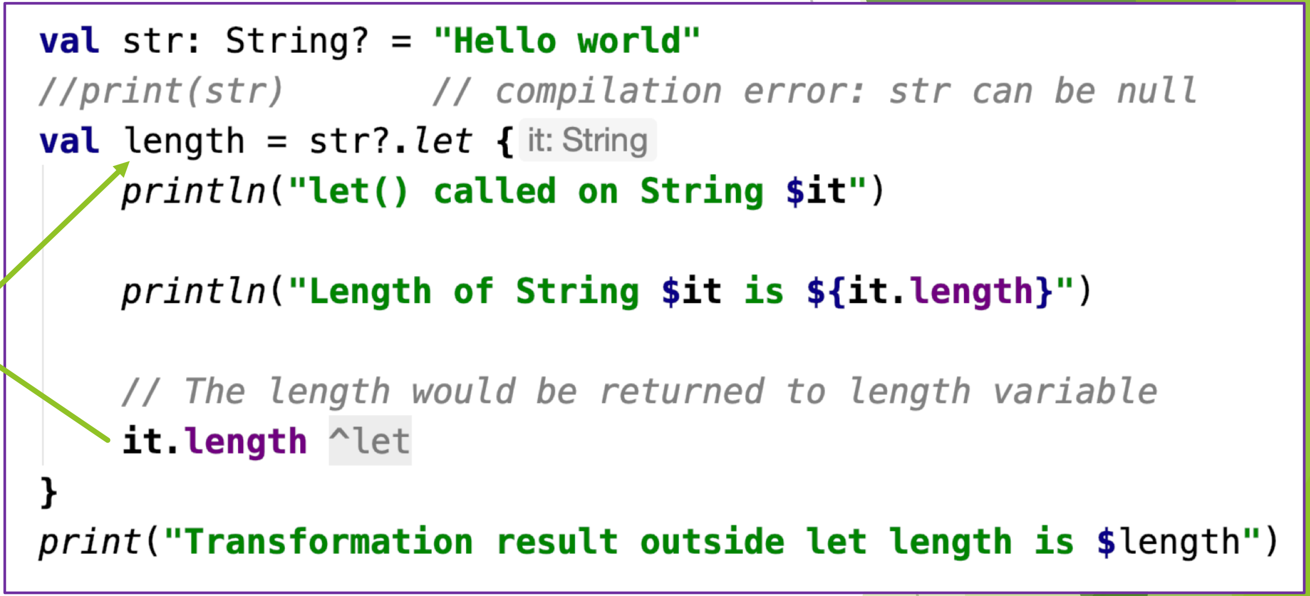
Example: Null check

```
val str: String? = null

str?.let {
// Wouldn't execute because str object is null
print(it)
} // No output
```

Safe Call operator

Example: Non null check, Transformation, Creating variable

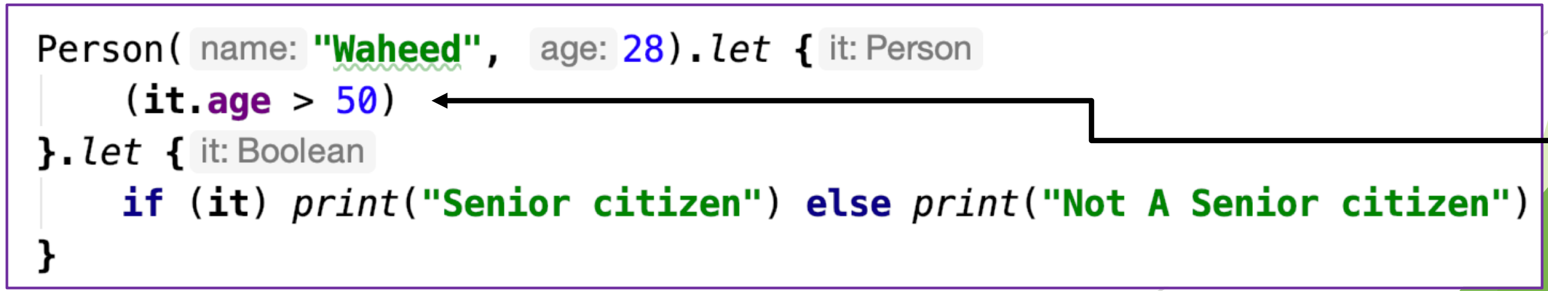
Creating variable from Transformation



```
val str: String? = "Hello world"
//print(str)        // compilation error: str can be null
val length = str?.let { it: String
    println("let() called on String $it")

    println("Length of String $it is ${it.length}")

    // The length would be returned to length variable
    it.length ^let
}
print("Transformation result outside let length is $length")
```

Transformation: Creating length variable, passed from here

Transformation: Chain of calls using let



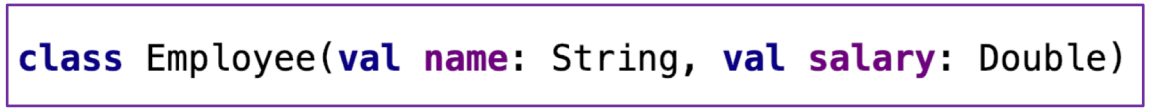
```
Person( name: "Waheed",  age: 28).let { it: Person
    (it.age > 50)
}.let { it: Boolean
    if (it) print("Senior citizen") else print("Not A Senior citizen")
}
```

Applying condition on one property of object and handling result using let.

By Waheed Nazir

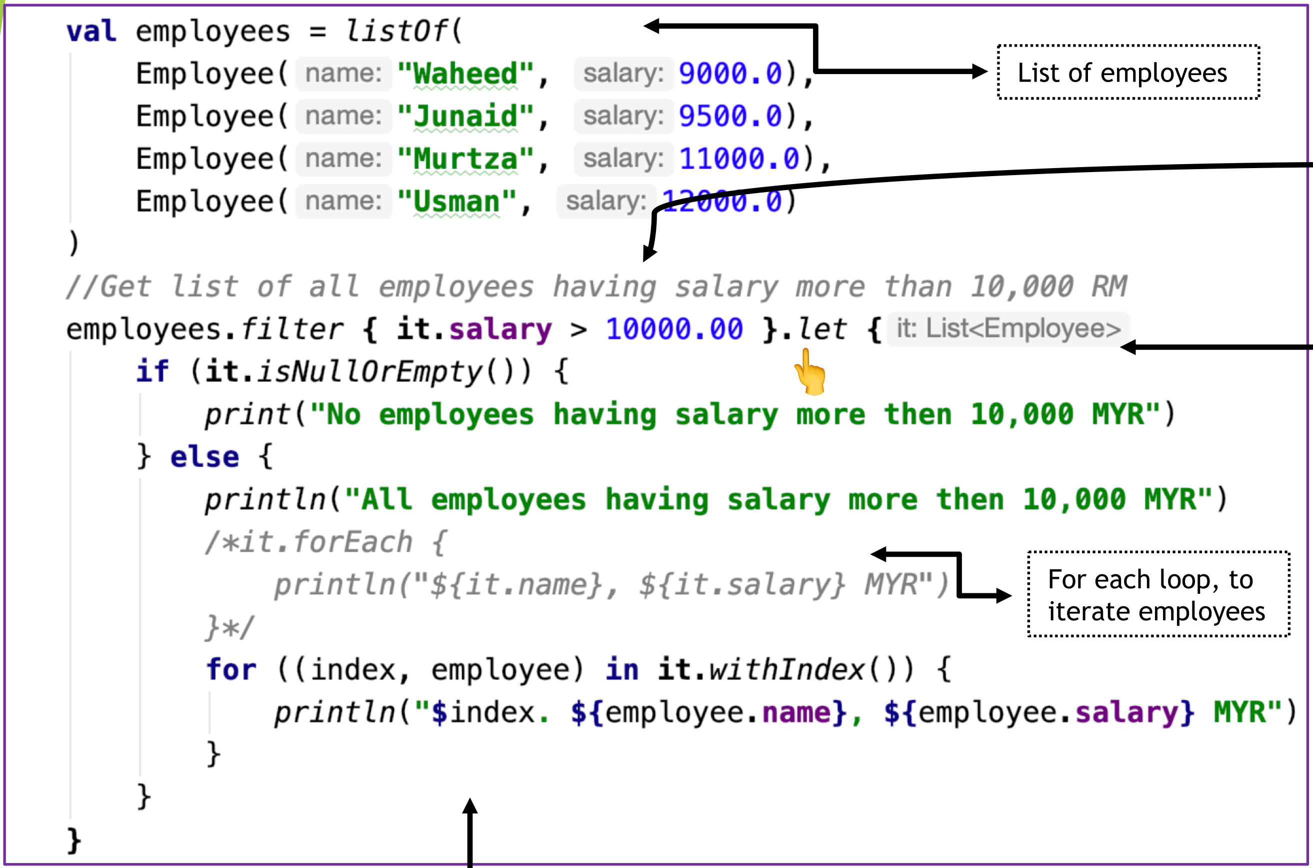
Let examples:

Employee Data class



```kotlin
class Employee(val name: String, val salary: Double)
```

-:Use case:- operations on Employees data e.g. filter, check salaries, total amount paid



```kotlin
val employees = listOf(
    Employee( name: "Waheed",  salary: 9000.0),
    Employee( name: "Junaid",  salary: 9500.0),
    Employee( name: "Murtza",  salary: 11000.0),
    Employee( name: "Usman",  salary: 12000.0)
)
//Get list of all employees having salary more than 10,000 RM
employees.filter { it.salary > 10000.00 }.let { it: List<Employee>
    if (it.isNullOrEmpty()) {
        print("No employees having salary more then 10,000 MYR")
    } else {
        println("All employees having salary more then 10,000 MYR")
        /*it.forEach {
            println("${it.name}, ${it.salary} MYR")
        }*/
        for ((index, employee) in it.withIndex()) {
            println("$index. ${employee.name}, ${employee.salary} MYR")
        }
    }
}
```

List of employees

Predicate/logic to filter data

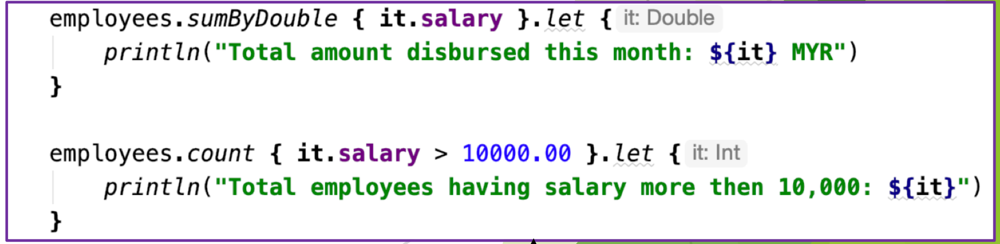
You'll get filter list here in it

For each loop, to iterate employees

For loop with index to iterate filtered list of employees

More operations on employees list

Find total sum of all employee salaries or Total salary amount disbursed



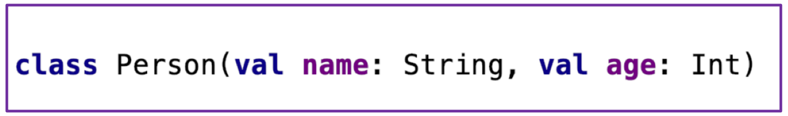
```kotlin
employees.sumByDouble { it.salary }.let { it: Double
    println("Total amount disbursed this month: ${it} MYR")
}

employees.count { it.salary > 10000.00 }.let { it: Int
    println("Total employees having salary more then 10,000: ${it}")
}
```

Count all employees having salary more then 10,000 MYR

run is almost same like let, but it is more focused on target object

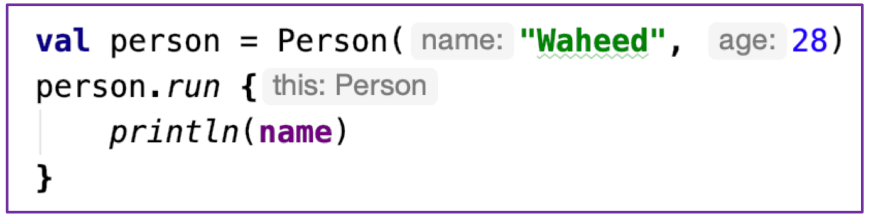
1) To perform actions on non null object using safe call operator **?.**
2) Multiple functions calls as chain
3) Transformation: Run can return anything from object
4) Doesn't allow to create a local variable in lambda scope

Person data class

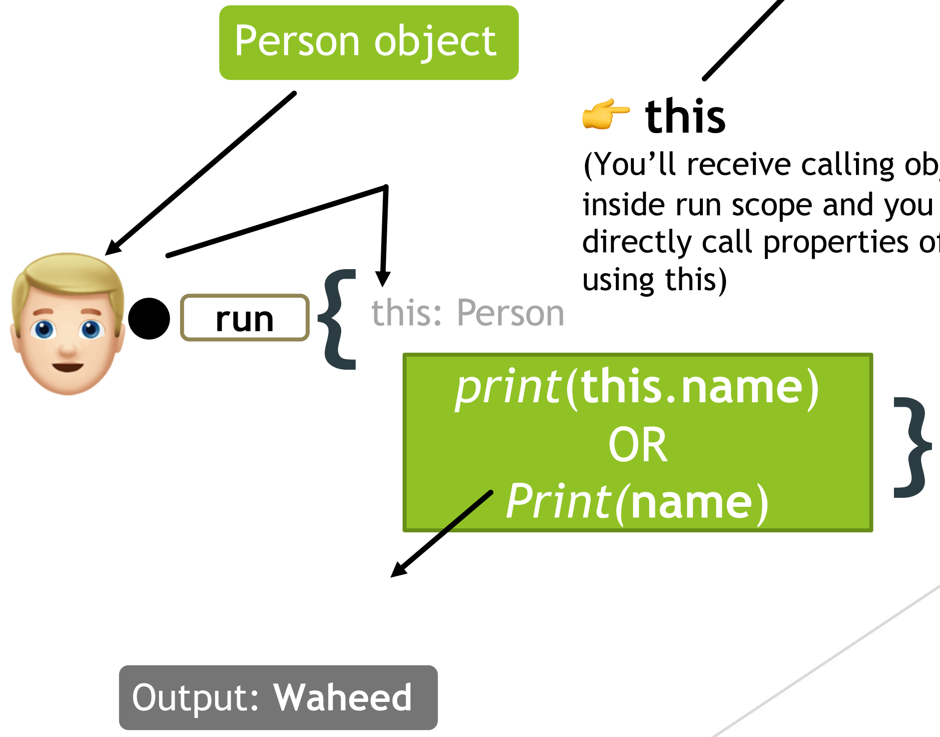


```
class Person(val name: String, val age: Int)
```

Accessing object's properties using run

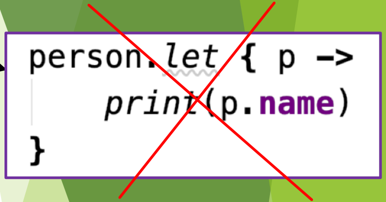


```
val person = Person( name: "Waheed",  age: 28)
person.run { this: Person
    println(name)
}
```



👉 **this**
(You'll receive calling object as this inside run scope and you can directly call properties of object or using this)

You cannot create a local variable in lambda scope

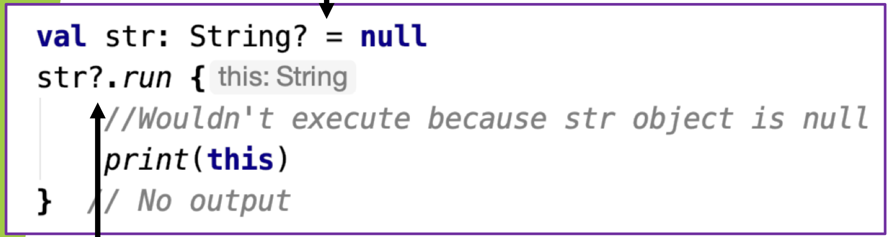


```
person.let { p ->
    print(p.name)
}
```

Run on non null objects using safe call operator ?.

String variable that can hold null value

Example: Null check



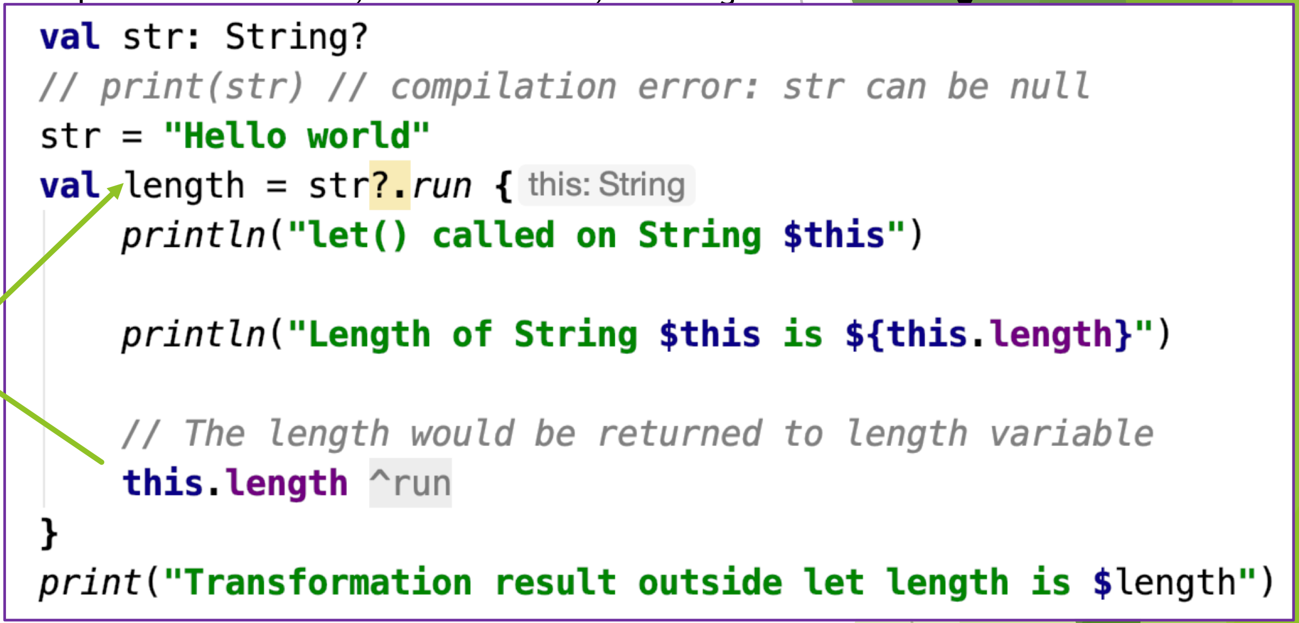
```
val str: String? = null
str?.run { this: String
    //Wouldn't execute because str object is null
    print(this)
} // No output
```

Safe Call operator

Transformation: Creating length variable, passed from here

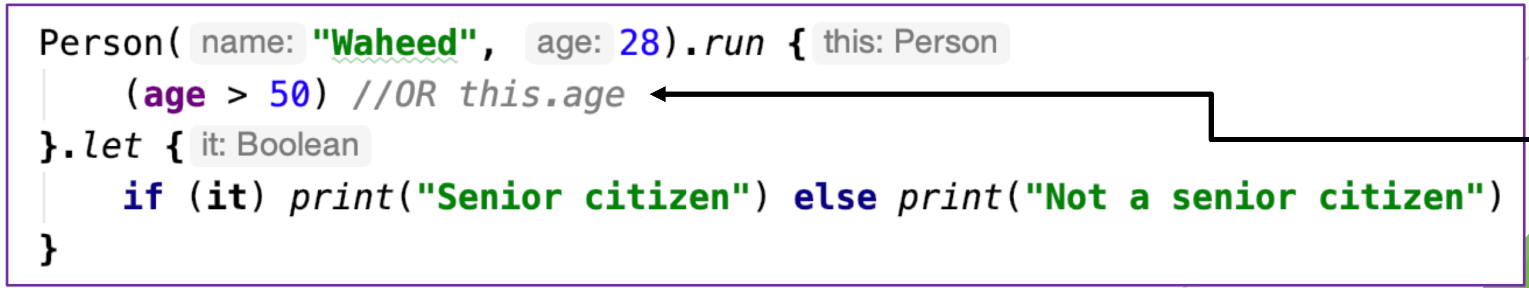
Creating variable from Transformation

Example: Non null check, Transformation, Creating variable



```
val str: String?
// print(str) // compilation error: str can be null
str = "Hello world"
val length = str?.run { this: String
    println("let() called on String $this")

    println("Length of String $this is ${this.length}")

    // The length would be returned to length variable
    this.length ^run
}
print("Transformation result outside let length is $length")
```

Transformation: Chain of calls using run



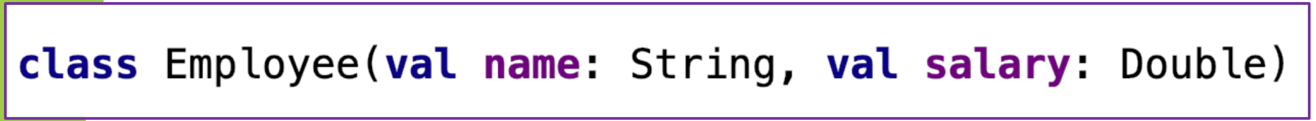
```
Person( name: "Waheed",  age: 28).run { this: Person
    (age > 50) //OR this.age
}.let { it: Boolean
    if (it) print("Senior citizen") else print("Not a senior citizen")
}
```

Applying condition on one property of object and handling result using run.

By Waheed Nazir

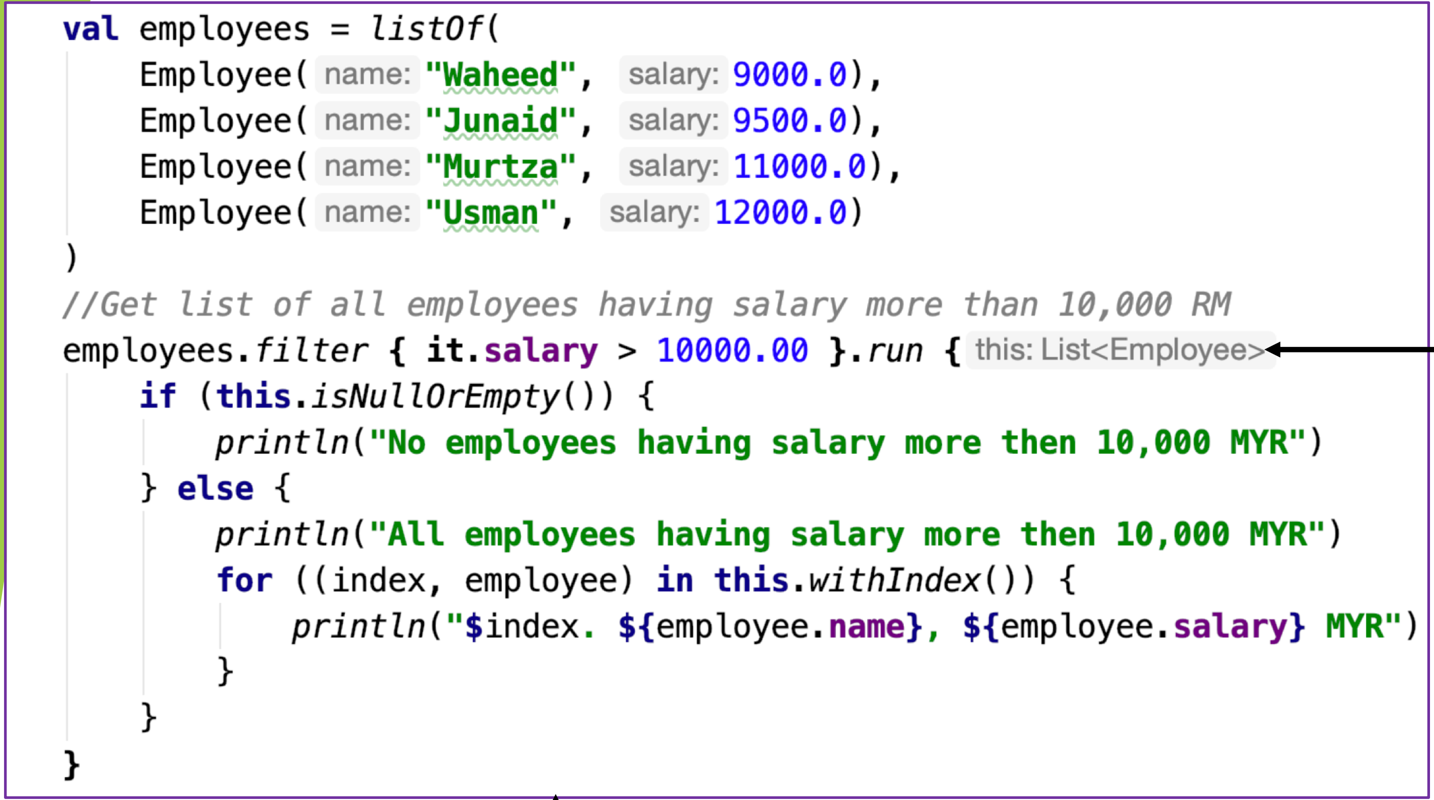
Run examples:

Employee Data class



```
class Employee(val name: String, val salary: Double)
```

-:Use case:-
operations on Employees data e.g. filter, check salaries, total amount paid

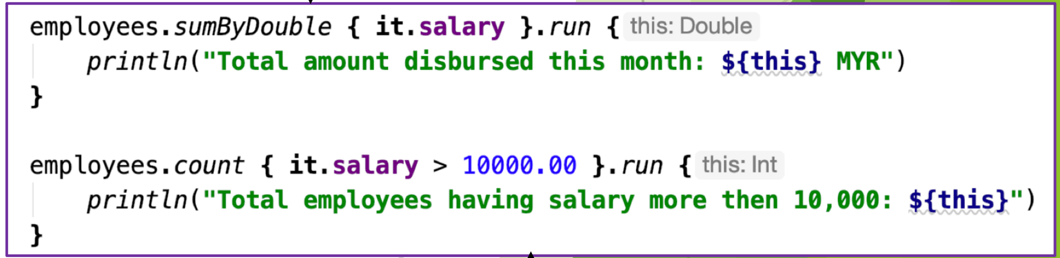


```
val employees = listOf(
    Employee( name: "Waheed",  salary: 9000.0),
    Employee( name: "Junaid",  salary: 9500.0),
    Employee( name: "Murtza",  salary: 11000.0),
    Employee( name: "Usman",  salary: 12000.0)
)
//Get list of all employees having salary more than 10,000 RM
employees.filter { it.salary > 10000.00 }.run { this: List<Employee>
    if (this.isNullOrEmpty()) {
        println("No employees having salary more then 10,000 MYR")
    } else {
        println("All employees having salary more then 10,000 MYR")
        for ((index, employee) in this.withIndex()) {
            println("$index. ${employee.name}, ${employee.salary} MYR")
        }
    }
}
```

You'll get filter list here in it

More operations on employees list

Find total sum of all employee salaries or Total salary amount disbursed



```
employees.sumByDouble { it.salary }.run { this: Double
    println("Total amount disbursed this month: ${this} MYR")
}

employees.count { it.salary > 10000.00 }.run { this: Int
    println("Total employees having salary more then 10,000: ${this}")
}
```

For loop with index to iterate filtered list of employees

Count all employees having salary more then 10,000 MYR

also is mostly used for these scenarios

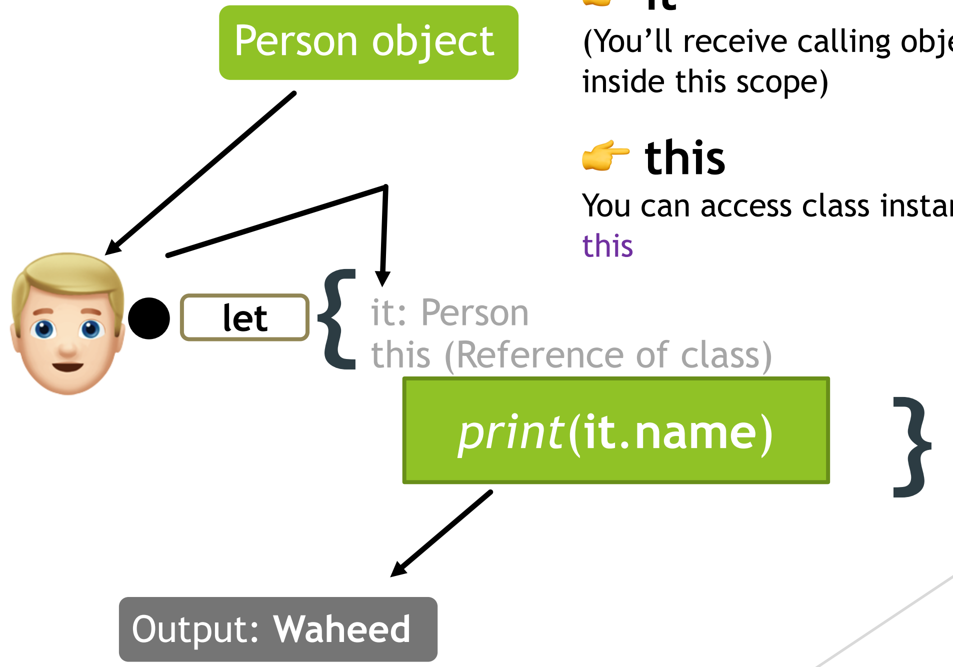
1) To perform actions on non null object using safe call operator **?.**
2) Multiple functions calls as chain
3) Helps to create local variable with limited scope in lambda
4) It returns original object instead of transformation

Person data class

```
class Person(val name: String, val age: Int)
```

Accessing object properties using let

```
val person = Person( name: "Waheed",  age: 28)
person.let { it: Person
    print(it.name)
}
```



👉 **it**
(You'll receive calling object as it inside this scope)

OR: You can create a local variable in lambda scope

😉 It's easy

```
person.let { p ->
    print(p.name)
}
```

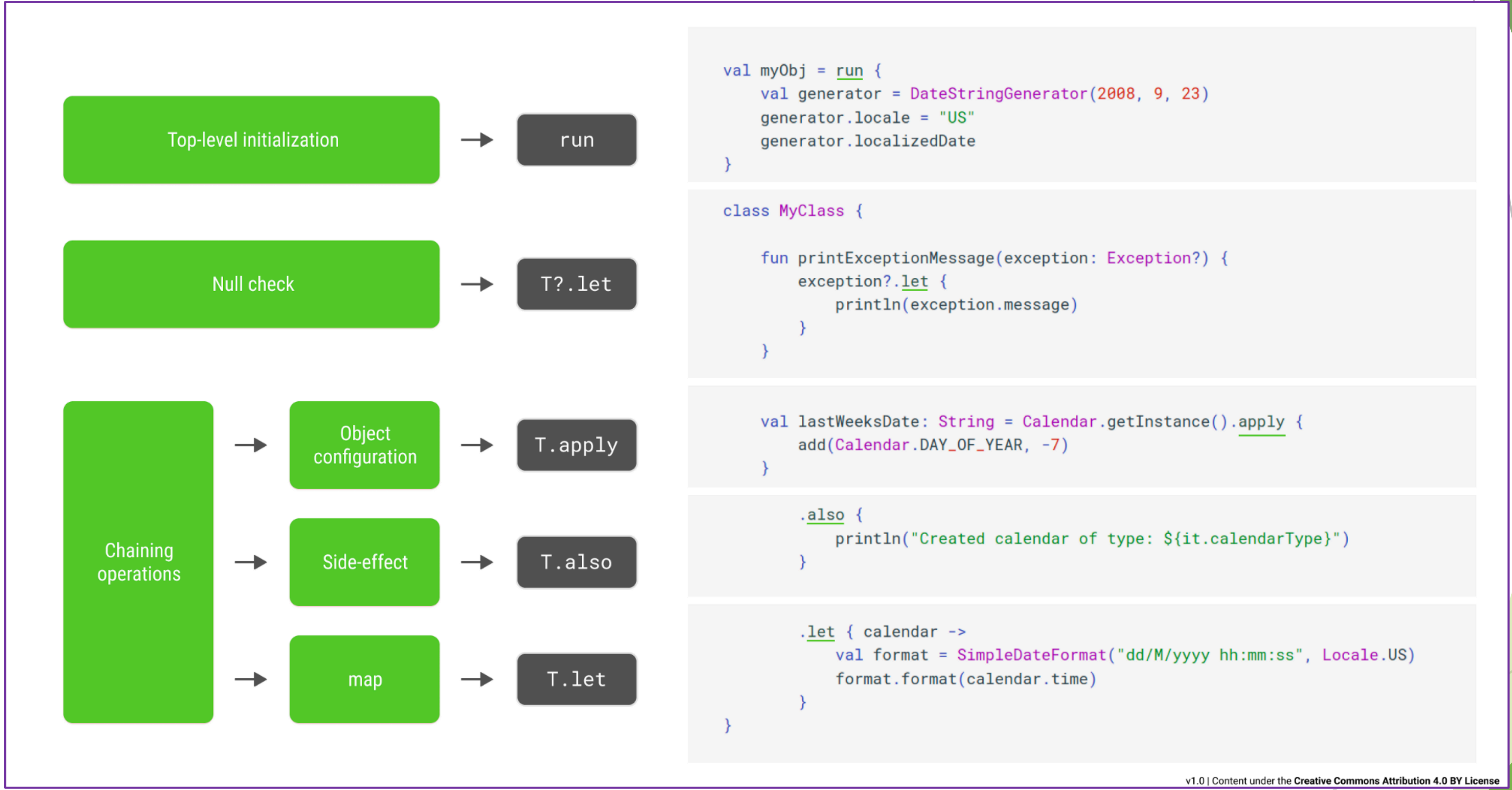
👉 **this**
You can access class instance as this

Comparison Table:

Function	Object reference	Return value	Is extension function?
let	it	Lambda result	Yes
also	it	Context object	Yes
run	this	Lambda result	Yes
apply	this	Context object	Yes
with	this	Lambda result	No

Reference, more details

Cheat-sheet: Jose Alcérreca (Maker & Developer Programs Engineer @ Google – Android)



```
val myObj = run {
    val generator = DateStringGenerator(2008, 9, 23)
    generator.locale = "US"
    generator.localizedDate
}
```

```
class MyClass {

    fun printExceptionMessage(exception: Exception?) {
        exception?.let {
            println(exception.message)
        }
    }
```

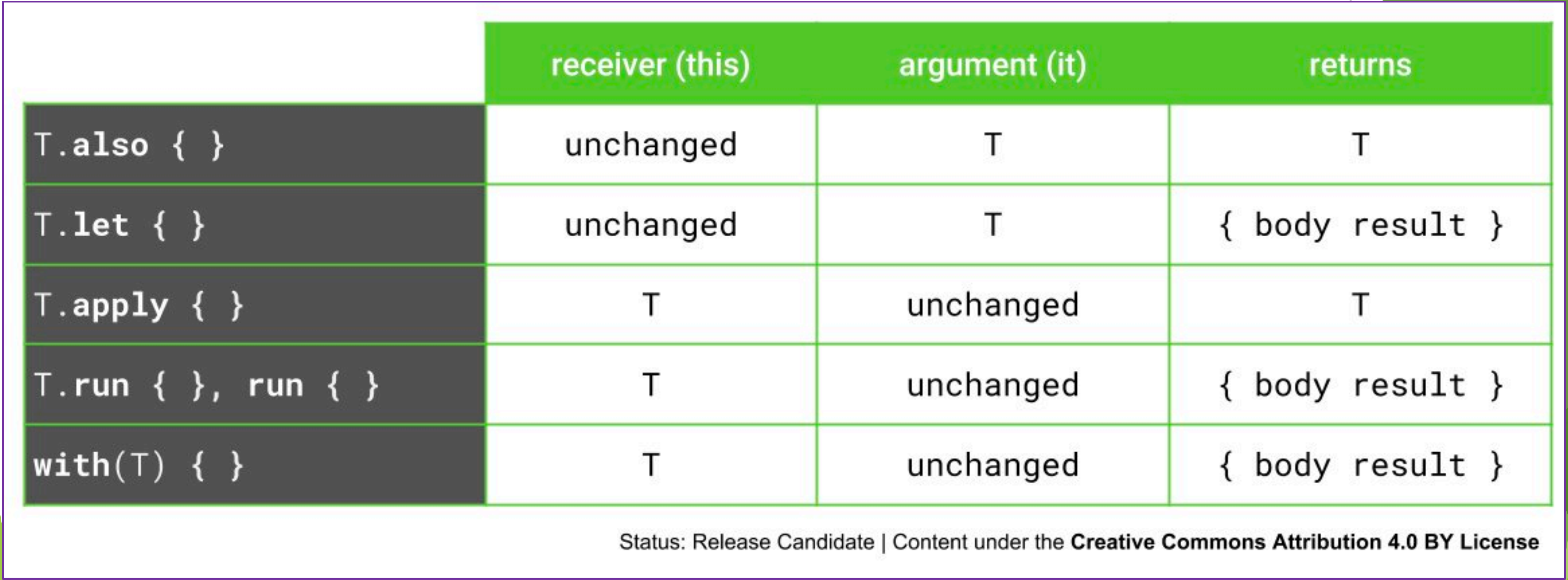
```
    val lastWeeksDate: String = Calendar.getInstance().apply {
        add(Calendar.DAY_OF_YEAR, -7)
    }
```

```
        .also {
            println("Created calendar of type: ${it.calendarType}")
        }
```

```
        .let { calendar ->
            val format = SimpleDateFormat("dd/M/yyyy hh:mm:ss", Locale.US)
            format.format(calendar.time)
        }
}
```

https://medium.com/androiddevelopers/kotlin-standard-functions-cheat-sheet-27f032dd4326

Cheat-sheet:
Jose Alcérreca (Maker & Developer Programs Engineer @ Google – Android)



	receiver (this)	argument (it)	returns
`T.also { }`	unchanged	T	T
`T.let { }`	unchanged	T	{ body result }
`T.apply { }`	T	unchanged	T
`T.run { }, run { }`	T	unchanged	{ body result }
`with(T) { }`	T	unchanged	{ body result }

Status: Release Candidate | Content under the **Creative Commons Attribution 4.0 BY License**

https://medium.com/androiddevelopers/kotlin-standard-functions-cheat-sheet-27f032dd4326

References:

- https://kotlinlang.org/docs/reference/scope-functions.html
- https://medium.com/androiddevelopers/kotlin-standard-functions-cheat-sheet-27f032dd4326
- https://twitter.com/ppvi/status/1081168598813601793/photo/1
- https://kotlinlang.org/docs/reference/functions.html
- https://www.journaldev.com/19467/kotlin-let-run-also-apply-with