Library functions looking like built-in constructs

Under the Hood

Lambdas can be inlined

No performance overhead

How inlining works?
Why inline?..

Useful library functions

`run, let, takeIf, takeUnless, repeat`

`run` function

runs the block of code (lambda) and returns the last expression as the result

```
val foo = run {
    println("Calculating foo...")
    "foo"
}
```

`let` function

allows to check the argument for being non-null, not only the receiver

```
fun getEmail(): Email?

val email = getEmail()
if (email != null) sendEmailTo(email)

email?.let { e -> sendEmailTo(e) }

getEmail()?.let { sendEmailTo(it) }
```

What is the type of `it` in the code below?

```
fun sendEmailTo(email: Email) { /*...*/ }
fun getEmail(): Email?

getEmail().let { sendEmailTo(it) }
```

What is the type of `it` in the code below?

```
fun sendEmailTo(email: Email) { /*...*/ }
fun getEmail(): Email?
```

Email?

```
getEmail().let { sendEmailTo(it) }
```

```
Error: Type mismatch: inferred type is Email?
                      but Email was expected
```

What is the type of `it` in the code below?

```
fun sendEmailTo(email: Email) { /*...*/ }
fun getEmail(): Email?

getEmail()?.let { sendEmailTo(it) }
```

Email

`let` function

```
interface Session {
    val user: User
}

fun analyzeUserSession(session: Session) {
    val user = session.user
    if (user is FacebookUser) {
        println(user.accountId)
    }
}
```

```
(session.user as? FacebookUser)?.let {
    println(it.accountId)
}
```

`takeIf` function

returns the receiver object if it satisfies the given predicate, otherwise returns `null`

```
val number = 42
number.takeIf { it > 10 }
```

What is the result of `takeIf` call below?

```
val number = 42
number.takeIf { it > 10 }
```

What is the result of `takeIf` call below?

```
val number = 42
number.takeIf { it > 10 }
```

42

`takeIf` function

returns the receiver object if it satisfies the given predicate, otherwise returns `null`

```
val number = 42
number.takeIf { it > 10 }    // 42

val other = 2
other.takeIf { it > 10 }     // null
```

`takeUnless` function

returns the receiver object if it **does not** satisfy the given predicate, otherwise returns null

```
val number = 42
number.takeUnless { it > 10 }   // null

val other = 2
other.takeUnless { it > 10 }    // 2
```

`repeat` function

repeats an action for a given number of times

```
repeat(10) {
    println("Welcome!")
}
```

They all are declared as `inline` functions

```
inline fun <R> run(block: () -> R): R = block()

inline fun <T, R> T.let(block: (T) -> R): R = block(this)

inline fun <T> T.takeIf(predicate: (T) -> Boolean): T? =
                                        if (predicate(this)) this else null

inline fun <T> T.takeUnless(predicate: (T) -> Boolean): T? =
                                       if (!predicate(this)) this else null

inline fun repeat(times: Int, action: (Int) -> Unit)  {
    for (index in 0 until times) {
        action(index)
    }
}
```