


СИНТАКСИС В Kotlin

Змінні

- Для створення змінної, використайте **var** або **val** і присвойте їй значення знаком рівності (=).
- Різниця між **var** і **val** полягає в тому, що змінні, оголошені з ключовим словом **var**, можна змінювати/модифікувати, а змінні **val** - ні.
- Щоб запам'ятати **var** (variable - змінна), **val** (value - значення)

kotlin

 Копіювати код

```
fun main() {  
    // Ініціалізація змінної та константи  
    var name = "John"  
    val birthyear = 1975  
  
    // Виведення початкових значень  
    println(name)          // Вивести значення змінної name  
    println(birthyear)     // Вивести значення birthyear  
  
    // Зміна значення змінної name  
    name = "Michael"  
  
    // Виведення зміненого значення  
    println(name)          // Вивести нове значення змінної name  
    // birthyear = 1980     // Це призведе до помилки, оскільки birthyear є константою  
}
```

Змінні

- В Kotlin можна вказувати тип змінної явно, а можна і не вказувати — компілятор присвоїть тип автоматично на основі значення. Однак, змінна може бути null, тоді її тип має бути явно позначений як nullable, додаючи ? після типу. У Kotlin за замовчуванням змінні не можуть мати значення null.
- Приклад без вказання типу (Kotlin сам присвоїть тип String):

kotlin

```
val languageName = "kotlin"
```

- Приклад з явним вказанням типу:

kotlin

```
val languageName: String = "kotlin"
```

- Наступний код не буде компілюватись, оскільки String не може бути null:

kotlin

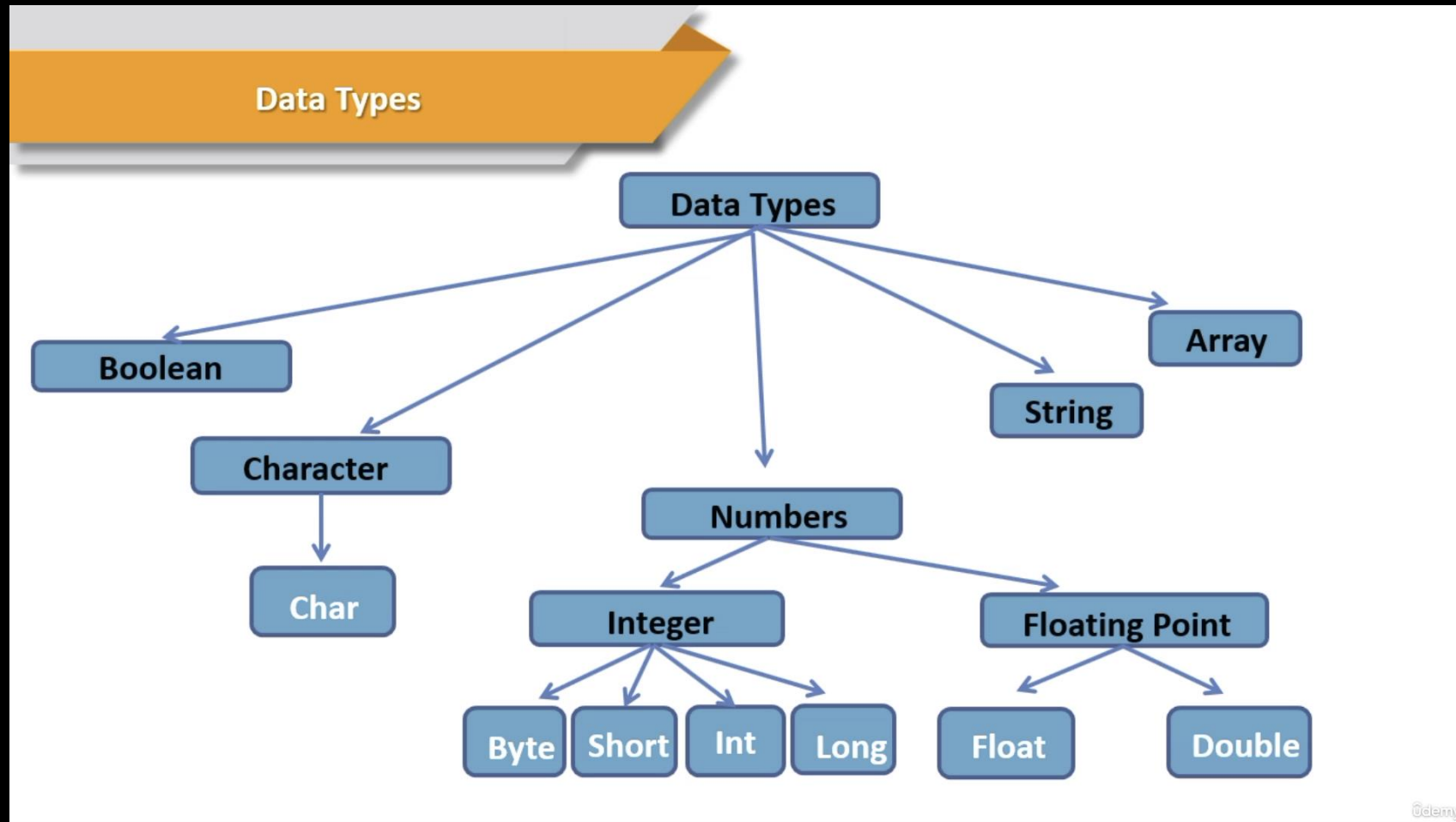
```
// Не компілюється  
val languageName: String = null
```

- Щоб змінна могла містити null, потрібно вказати тип як nullable, додаючи ?:

kotlin

```
val languageName: String? = null
```

Основні типи даних



Основні типи даних

Data Types

Kotlin Data Types

Data Type	Default Size	Range	Default Value
Byte	1 byte (8 Bit)	$[-128, 127]$	0
Short	2 Bytes (16 Bit)	$[-32.768, 32.767]$	0
Int	4 Bytes (32 Bit)	$[-2^{31}, 2^{31}-1]$	0
Long	8 Bytes (64 Bit)	$[-2^{63}, 2^{63}-1]$	0L
Float	4 Bytes (32 Bit)	32-bit floating point	0.0f
Double	8 Bytes (64 Bit)	64-bit floating point	0.0d
Boolean	1 Bit	true or false	false
Char	2 Bytes (16 Bit)	$['\u0000', '\uffff']$ or $[0, 65535]$	<code>'\u0000'</code> (means 0 in ASCII)

Основні типи даних

1. Цілі числа:

- Byte: 8-бітове ціле число зі знаком.
- Short: 16-бітове ціле число зі знаком.
- Int: 32-бітове ціле число зі знаком.
- Long: 64-бітове ціле число зі знаком.

kotlin

```
// При ініціалізації змінних можна явно вказати їх тип.  
fun main() {  
    val byteValue: Byte = 127 // 8-бітове ціле число  
    val shortValue: Short = 32767 // 16-бітове ціле число  
    val intValue: Int = 2147483647 // 32-бітове ціле число  
    val longValue: Long = 9223372036854775807 // 64-бітове ціле число  
  
    println("Byte value: $byteValue")  
    println("Short value: $shortValue")  
    println("Int value: $intValue")  
    println("Long value: $longValue")  
}
```

Основні типи даних

2. Числа з плаваючою комою:

- Float: 32-бітове число з плаваючою точкою. Може містити до 7 десяткових знаків.
- Double: 64-бітове число з плаваючою точкою. Може містити до 15 десяткових знаків.

kotlin

```
// При ініціалізації змінних можна явно вказати їх тип.  
fun main() {  
    val floatValue: Float = 3.14f  
    // 32-бітове число з плаваючою точкою  
  
    val doubleValue: Double = 3.141592653589793  
    // 64-бітове число з плаваючою точкою  
  
    println("Float value: $floatValue")  
    println("Double value: $doubleValue")  
}
```

Основні типи даних

3. Логічний тип (Boolean):

- Boolean: логічний тип даних який зберігає два значення: true або false. Він часто використовується для умовних операторів та контролю виконання програми.

Логічні оператори:

- AND (&&): Повертає true, якщо обидва операнди є true. Наприклад, якщо ви перевіряєте, чи є обидва значення істинними, використовуйте оператор AND.
- OR (||): Повертає true, якщо хоча б одне з двох значень є true. Цей оператор корисний, коли потрібно перевірити кілька умов.
- NOT (!): Інвертує логічне значення. Якщо значення true, після застосування оператора NOT воно стане false, і навпаки.

kotlin

```
// При ініціалізації змінних можна явно вказати їх тип.  
fun main() {  
    val isKotlinFun: Boolean = true  
    // Логічне значення, яке зберігає true або false  
  
    val isFishTasty: Boolean = false  
    // Логічне значення, яке зберігає true або false  
  
    println("Is Kotlin fun? $isKotlinFun")  
    println("Is fish tasty? $isFishTasty")  
}
```


Основні типи даних

4. Символьний тип (Char):

- Char: представляє один символ Unicode. Символи в Kotlin записуються в одинарних лапках, наприклад: 'A', '1'.
- На відміну від рядків (String), тип Char призначений для зберігання лише одного символу.
- Також можна порівнювати символи за допомогою операторів >, <, >=, <=, ==, != оскільки Char представляє числове значення у таблиці Unicode.

kotlin

```
fun main() {  
    val letter: Char = 'A'    // Один символ: літера 'A'  
    val digit: Char = '1'    // Один символ: цифра '1'  
    val symbol: Char = '$'    // Один символ: спеціальний символ '$'  
  
    println("Letter: $letter, Digit: $digit, Symbol: $symbol")  
}
```

kotlin

```
val charA: Char = 'A'  
val charB: Char = 'B'  
println(charA < charB) // Виведе true
```

Основні типи даних

5. Строковий тип (String):

- String: представляє рядок символів (послідовність символів). Рядки записуються в подвійних лапках, наприклад: "Hello, World!". Рядки є незмінними.

Strings	
Useful Methods of String	
Method	Description
length()	returns the length of a String.
equals(Object another)	compares the string with the specified string and if both matches returns true else false.
isEmpty()	returns true if the given string has 0 length. If the length of String is non-zero then returns false.
plus("...")	concatenates the string "..." and other string.
lowercase()	returns a string in lowercase.
uppercase()	returns a string in uppercase.
trim()	removes spaces at the beginning and end of the word

kotlin

```
val greeting: String = "Hello, world!"
```

kotlin

```
val multiLineString = """  
    Це рядок,  
    який займає кілька рядків.  
    """
```

Приклади

Ініціалізація + Конкатенація:

kotlin

```
val modifiedGreeting = greeting + " How are you?"
```

Доступ:

kotlin

```
val firstChar = greeting[0] // 'H'
```

Довжина рядка:

kotlin

```
val length = greeting.length // 13
```

Методи рядків:

kotlin

```
val containsWorld = greeting.contains("World") // true  
val upperCaseGreeting = greeting.toUpperCase() // "HELLO, WORLD!"
```

Додавання змінних у рядки за допомогою символу \$:

kotlin

```
val name = "Kotlin"  
val interpolatedGreeting = "Hello, $name!" // "Hello, Kotlin!"
```

Використання спец. Символів для форматування рядка:

kotlin

```
val escapedString = "Це рядок з новим рядком:\nІ ось новий рядок."
```

Основні типи даних

6. Масив (Array):

- Array: узагальнений клас, що представляє масиви. Масиви в Kotlin створюються за допомогою конструктора `arrayOf()`, наприклад: `val arr = arrayOf(1, 2, 3)`.

kotlin

```
val arr = arrayOf(1, 2, 3)
```

Приклади

Ініціалізація масиву:

kotlin

```
val arr = arrayOf(1, 2, 3)
```

Довжина масиву:

kotlin

```
val length = arr.size // 3
```

Доступ до елементів масиву:

kotlin

```
val firstElement = arr[0] // 1
```

Створення масиву певного типу:

kotlin

```
val intArray = IntArray(3)
```

Зміна елементів масиву:

kotlin

```
arr[0] = 10 // Змінює перший елемент масиву на 10
```

Заповнення певними значеннями:

kotlin

```
val filledArray = IntArray(5) { it * 2 } // [0, 2, 4, 6, 8]
```

Перетворення типів

Для перетворення числового типу даних в інший тип необхідно скористатися однією з наступних функцій: `toByte()`, `toShort()`, `toInt()`, `toLong()`, `toFloat()`, `toDouble()` або `toChar()`.

kotlin

```
fun main() {  
    val originalInt: Int = 42  
  
    // Перетворення в різні типи  
    val doubleValue = originalInt.toDouble()  
    val stringValue = originalInt.toString()  
    val longValue = originalInt.toLong()  
    val shortValue = originalInt.toShort()  
    val byteValue = originalInt.toByte()  
    val parsedInt = stringValue.toIntOrNull()  
    val backToInt = doubleValue.toInt()  
}
```

Оператори

Оператори в програмуванні - це спеціальні символи, які виконують операції над одним, двома або більше операндами та повертають результат. Оператори можуть бути арифметичними, логічними, порівняння, присвоєння тощо, і відіграють ключову роль у виконанні обчислень та контролі потоку програми.

Оператори

Арифметичні оператори

Арифметичні оператори виконують базові математичні операції над числовими значеннями. Вони використовуються для додавання, віднімання, множення, ділення та інших обчислень.

Оператор	Назва	Опис	Приклад
+	Додавання	Додає два значення	$x + y$
-	Віднімання	Віднімає одне значення від іншого	$x - y$
*	Множення	Множить два значення	$x * y$
/	Ділення	Ділить одне значення на інше	x / y
%	Остача від ділення	Повертає залишок від ділення	$x \% y$
++	Інкремент	Збільшує значення на 1	$++x$
--	Декремент	Зменшує значення на 1	$--x$

Оператори

Оператори присвоєння

Оператори присвоєння використовуються для збереження значення в змінну. Базовий оператор присвоєння - це `=`, який встановлює значення змінної. Оператори `+=`, `-=`, `*=`, `/=`, та `%=` поєднують арифметичні операції з присвоєнням, дозволяючи скоротити запис. Ці оператори допомагають писати код більш компактно та зручно.

Оператор	Приклад	Те ж саме, що й
<code>=</code>	<code>x = 5</code>	<code>x = 5</code>
<code>+=</code>	<code>x += 3</code>	<code>x = x + 3</code>
<code>-=</code>	<code>x -= 3</code>	<code>x = x - 3</code>
<code>*=</code>	<code>x *= 3</code>	<code>x = x * 3</code>
<code>/=</code>	<code>x /= 3</code>	<code>x = x / 3</code>
<code>%=</code>	<code>x %= 3</code>	<code>x = x % 3</code>

Оператори

Оператори порівняння

Оператори порівняння використовуються для порівняння двох значень і повертають булевий результат (істина або хибна). Вони грають важливу роль у прийнятті рішень у програмах, наприклад, в умовних конструкціях (if, switch) і циклах.

Оператор	Назва	Приклад
==	Рівність	<code>x == y</code>
!=	Не рівність	<code>x != y</code>
>	Більше ніж	<code>x > y</code>
<	Менше ніж	<code>x < y</code>
>=	Більше або рівне	<code>x >= y</code>
<=	Менше або рівне	<code>x <= y</code>

Оператори

Логічні оператори

Логічні оператори використовуються для об'єднання двох або більше умов у булевих виразах. Вони повертають булеве значення (істина або хиба) і зазвичай використовуються в умовних конструкціях і циклах, щоб контролювати виконання коду в залежності від декількох умов.

Оператор	Назва	Опис	Приклад
&&	Логічне "і"	Повертає true, якщо обидва вирази істинні	$x < 5 \ \&\& \ x < 10$
	Логічне "або"	Повертає true, якщо один з виразів істинний	$x < 5 \ \ x < 4$
!	Логічне "не"	Змінює результат: повертає false, якщо результат істинний	$!(x < 5)$

Основні структури

- If...else
- While loop
- do-while loop
- When
- For loop
- Break and Continue

Основні структури

1. Умовна конструкція *if...else*

if в Kotlin може використовуватися як оператор для виконання умовної логіки. Він може повертати значення, що робить його дуже гнучким.

Синтаксис

kotlin

```
if (умова) {  
    // Блок коду виконується, якщо умова істинна  
} else if (інша умова) {  
    // Блок коду виконується, якщо інша умова істинна  
} else {  
    // Блок коду виконується, якщо жодна з умов не виконана  
}
```

Приклади

Перевірка числа на парність

```
kotlin

val num = 7
if (num % 2 == 0) {
    println("$num є парним")
} else {
    println("$num є непарним")
}
```

Визначає оцінку студента
залежно від отриманого балу.

```
kotlin

val grade = 85
if (grade >= 90) {
    println("Оцінка: A")
} else if (grade >= 80) {
    println("Оцінка: B")
} else if (grade >= 70) {
    println("Оцінка: C")
}
```

Використання if...else як вираз

```
kotlin

val age = 18
val canVote = if (age >= 18) "Може голосувати" else "Не може голосувати"
```

Основні структури

2. Цикл **while** виконує блок коду доти, поки умова є істинною.

Синтаксис

kotlin

```
while (умова) {  
    // блок коду, який буде виконано  
}
```

kotlin

```
fun main() {  
    var count = 0  
    while (count < 5) {  
        println("Count: $count")  
        count++  
    }  
}
```

Приклади

Пошук елемента cherry в масиві

kotlin

```
val items = arrayOf("apple", "banana", "cherry", "date")
var found = false
var i = 0
while (i < items.size && !found) {
    if (items[i] == "cherry") {
        println("Found 'cherry' at index $i")
        found = true
    }
    i++
}
```

Вивід всіх елементів масиву

kotlin

```
val numbers = arrayOf(10, 20, 30, 40, 50)
var index = 0
while (index < numbers.size) {
    println("Element at index $index: ${numbers[index]}")
    index++
}
```


Основні структури

3. Цикл ***do...while*** працює схожим чином, але з однією важливою відмінністю: блок коду виконається принаймні один раз, незалежно від того, істинна умова чи ні. Після виконання коду умова перевіряється, і якщо вона істинна, цикл продовжується.

Синтаксис

kotlin

```
do {  
    // блок коду, який буде виконано  
} while (умова)
```

kotlin

```
var count = 10  
  
do {  
    println("Цей текст буде виведено один раз, навіть якщо умова false")  
} while (count < 5)
```

Приклади

Цикл запитує введення користувача, і виконується доти, поки користувач не введе "exit".

kotlin

```
var input: String?
do {
    println("Введіть текст (або 'exit' для виходу):")
    input = readLine()
} while (input != "exit")
```

Постійний запит до користувача, поки той не введе правильний пароль

kotlin

```
val correctPassword = "kotlin123"
var enteredPassword: String?
do {
    println("Введіть пароль:")
    enteredPassword = readLine()
} while (enteredPassword != correctPassword)
println("Пароль правильний, доступ дозволено.")
```

Основні структури

4. Конструкція **when** в Kotlin — це оператора **switch** в інших мовах програмування. Її використовують для перевірки значення змінної на відповідність кільком умовам. **when** може повертати значення, а також підтримує різні типи умов, як-то порівняння з діапазоном, типовою відповідністю тощо.

Визначення дня тижня

```
kotlin

val day = 3
val result = when (day) {
    1 -> "Понеділок"
    2 -> "Вівторок"
    3 -> "Середа"
    else -> "Невідомий день"
}
println(result) // Виведе: Середа
```

Синтаксис

```
kotlin

when (змінна) {
    значення1 -> // код для значення1
    значення2 -> // код для значення2
    else -> // код для всіх інших випадків
}
```

Визначення оцінки по балам

```
kotlin

val score = 85
val grade = when (score) {
    in 90..100 -> "Відмінно"
    in 70..89 -> "Добре"
    else -> "Погано"
}
println(grade) // Виведе: Добре
```

Основні структури

5. Цикл **for** використовується для ітерації по массивах, списках і т.п.

Синтаксис

kotlin

```
for (елемент in колекція) {  
    // код, який буде виконаний для кожного елемента  
}
```

kotlin

```
fun main() {  
    for (i in 1..5) {  
        println("i: $i")  
    }  
}
```

kotlin

```
fun main() {  
    val numbers = listOf(1, 2, 3, 4, 5)  
    for (number in numbers) {  
        println(number)  
    }  
}
```

Основні структури

6. Оператор ***break*** припиняє виконання циклу повністю і передає управління наступній частині коду після циклу.

Цикл `for` переривається, коли `i` дорівнює 5. `break` зупиняє цикл і виводить значення до 5.

kotlin

```
for (i in 1..10) {  
    if (i == 5) {  
        println("Зупинка на числі $i")  
        break  
    }  
    println(i)  
}
```

Синтаксис

kotlin

```
for (i in 1..5) {  
    if (i == 3) break  
    println(i)  
}  
  
// Результат:  
// 1  
// 2
```

Основні структури

7. Оператор ***continue*** припиняє виконання циклу повністю і передає управління наступній частині коду після циклу.

Цикл `for` пропускає всі числа, які не є кратними 3, і виводить лише ті, що діляться на 3.

kotlin

```
for (i in 1..10) {  
    if (i % 3 != 0) {  
        continue // Пропускає числа, що не діляться на 3  
    }  
    println(i)  
}
```

Синтаксис

kotlin

```
for (i in 1..5) {  
    if (i == 3) continue  
    println(i)  
}  
  
// Результат:  
// 1  
// 2  
// 4  
// 5
```

Функції

Функція — це блок коду, який виконує певну задачу і може повертати значення. Функції в Kotlin визначаються за допомогою ключового слова `fun`, після якого йде ім'я функції, список параметрів у круглих дужках, та, за потреби, тип повернення.

Синтаксис

kotlin

```
fun ім'яФункції(параметр1: Тип, параметр2: Тип): ТипПовернення {  
    // Тіло функції  
    return значення  
}
```

- `fun` — ключове слово для визначення функції.
- `ім'яФункції` — назва функції, за якою можна її викликати.
- параметри — значення, які передаються в функцію при її виклику.
- `ТипПовернення` — тип значення, яке функція повертає. Якщо функція нічого не повертає, тип повернення `Unit`, але його можна опустити.
- `return` — оператор для повернення значення (необов'язковий для `Unit`).

Приклади

Найпростіший приклад функції

kotlin

```
fun greet() {  
    println("Hello, World!")  
}  
  
greet() // Викликає функцію і виводить "Hello, World!"
```

Функція add приймає два параметри типу Int, додає їх і повертає результат.

kotlin

```
fun add(a: Int, b: Int): Int {  
    return a + b  
}  
  
val sum = add(5, 3) // Результат: 8  
println(sum)
```

Функція greetUser приймає параметр name типу String і виводить персоналізоване привітання.

kotlin

```
fun greetUser(name: String) {  
    println("Hello, $name!")  
}  
  
greetUser("Alice") // Виведе: "Hello, Alice!"
```

У функції greetUserWithDefault параметр name має типове значення "Guest". Якщо параметр не передається при виклику, використовується значення за замовчуванням.

kotlin

```
fun greetUserWithDefault(name: String = "Guest") {  
    println("Hello, $name!")  
}  
  
greetUserWithDefault() // Виведе: "Hello, Guest!"  
greetUserWithDefault("John") // Виведе: "Hello, John!"
```


Приклади

Функція multiply визначена як вираз і повертає добуток двох чисел. Така форма функції не потребує ключового слова return або тіла в фігурних дужках.

kotlin

```
fun multiply(a: Int, b: Int) = a * b

val product = multiply(4, 5) // Результат: 20
println(product)
```

Рекурсивна функція factorial обчислює факторіал числа n. Функція викликає сама себе, доки не досягне базового випадку (n == 0).

kotlin

```
fun factorial(n: Int): Int {
    return if (n == 0) 1 else n * factorial(n - 1)
}

val result = factorial(5) // Результат: 120
println(result)
```

Лямбда-функції

Лямбда-функції в Kotlin — це функції, які не мають імені. Вони використовуються для передачі функціональності як параметри іншим функціям або для зберігання у змінних.

Лямбда-функція `square` обчислює квадрат переданого числа.

kotlin

```
val square: (Int) -> Int = { x -> x * x }  
println(square(4)) // Виведе: 16
```

Синтаксис

kotlin

```
val myLambda: (Int, Int) -> Int = { a, b -> a + b }
```

- `val myLambda`: оголошення змінної для зберігання лямбда-функції.
- `(Int, Int) -> Int`: тип лямбда-функції, що приймає два параметри типу `Int` і повертає значення типу `Int`.
- `{ a, b -> a + b }`: тіло лямбда-функції, яке додає два числа.

Приклади

Лямбда функція з типом параметрів

kotlin

```
val greet: (String) -> Unit = { name -> println("Hello, $name!") }  
greet("Alice") // Виведе: "Hello, Alice!"
```

Лямбда функція без параметрів

kotlin

```
val printHello: () -> Unit = { println("Hello!") }  
printHello() // Виведе: "Hello!"
```

Лямбда функція з функціями вищих порядків

kotlin

```
fun performOperation(a: Int, b: Int, operation: (Int, Int) -> Int): Int {  
    return operation(a, b)  
}  
  
val result = performOperation(3, 4) { x, y -> x + y }  
println(result) // Виведе: 7
```

Приклади

Лямбда-функція використовується для подвоєння значень у масиві numbers за допомогою методу map, а потім результат перетворюється назад у масив.

kotlin

```
val numbers = arrayOf(1, 2, 3, 4, 5)
val doubled = numbers.map { it * 2 }.toTypedArray()
println(doubled.joinToString()) // Виведе: 2, 4, 6, 8, 10
```

Лямбда-функція використовується для фільтрації парних чисел з масиву numbers за допомогою методу filter, а потім результат перетворюється назад у масив.

kotlin

```
val numbers = arrayOf(1, 2, 3, 4, 5)
val evenNumbers = numbers.filter { it % 2 == 0 }.toTypedArray()
println(evenNumbers.joinToString()) // Виведе: 2, 4
```

end...