

Лабораторная работа №3: Циклы в Kotlin

Цель: Научиться использовать циклы, генерацию случайных чисел в Kotlin. Закрепить навыки взаимодействия с пользователем через консольный ввод, а также работу с логикой и структурой программы.

Цикл — это конструкция, которая позволяет выполнять один и тот же блок кода **многократно**, пока выполняется заданное условие.

Зачем нужны циклы?

Представьте, что вам нужно:

- Вывести числа от 1 до 100.
- Перебрать все товары в корзине.
- Повторять вопрос, пока пользователь не введёт правильный ответ.

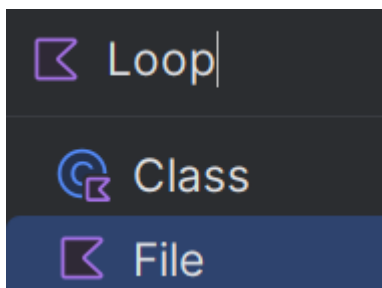
Без циклов пришлось бы писать один и тот же код сотни раз. С циклами — всего несколько строк!

В Kotlin (и большинстве языков) есть 3 основных типа циклов:

1. **while** — повторяет код, пока условие **true**.
2. **do-while** — сначала выполняет код, потом проверяет условие.
3. **for** — перебирает элементы диапазона или коллекции.

Шаг 1. Цикл **while**

Откройте предыдущую лабораторную работу. Создайте в папке `src` новый файл и назовите его **Loop**:



Синтаксис цикла **while**:

```
while (условие) {  
    // Код, который будет повторяться  
}
```

- **условие** — выражение, которое возвращает `true` или `false`.
- Цикл выполняется, пока условие истинно (`true`).

- Как только условие становится ложным (false) — цикл останавливается.

Давайте подсчитаем числа от 1 до 5.

В начале определим переменную `number` и присвоим ей значение 1.

```
val number = 1
```

После создадим цикл в котором в качестве условия передадим выражение `number <= 5`, при таких условиях он будет продолжать выполнять действия:

```
while (number <= 5) {  
  
}
```

И в нём мы выведем значение числа `number`:

```
while (number <= 5) {  
    println("Число: $number")  
}
```

Теперь мы хотим после каждого вывода увеличиваем `number` на 1:

```
while (number <= 5) {  
    println("Число: $number")  
    number++  
}
```

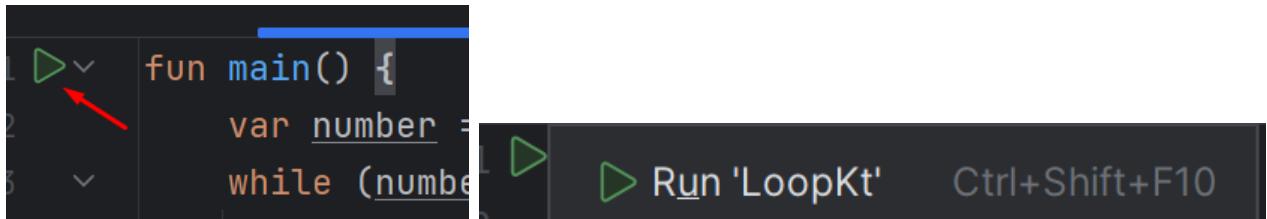
Обратите внимание, у нас переменная стала подчёркиваться красным цветом, наведите и прочитайте ошибку:

```
number++  
'val' cannot be reassigned.  
Change to 'var' Alt+Shift+Enter
```

Наша переменная не может быть переназначена, и сама IDE подсказывает нам, что нужно изменить переменную на `var`. Воспользуемся подсказкой:

```
var number = 1  
while (number <= 5) {  
    println("Число: $number")  
    number++  
}
```

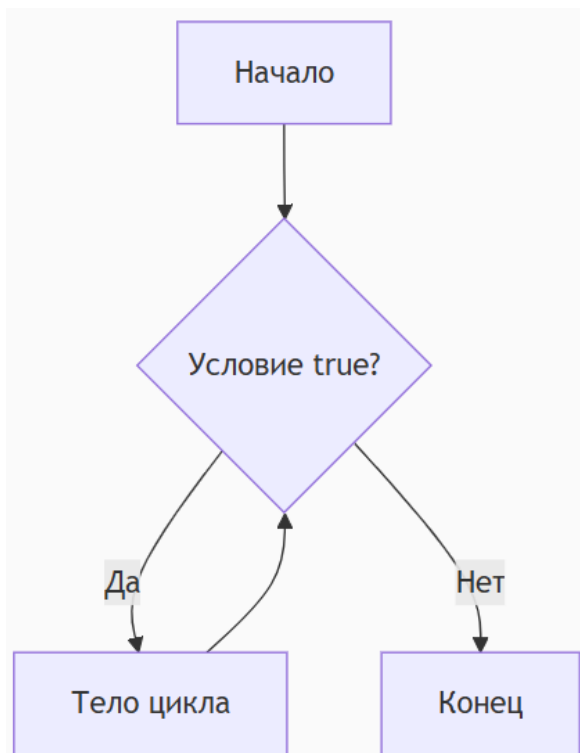
Теперь, когда `number` станет 6, условие станет ложным, и цикл остановится. Запустите программу и убедитесь, что она выполняет нужные нам действия:



Вывод:

```
Число: 1
Число: 2
Число: 3
Число: 4
Число: 5
```

Для лучшего понимания цикла **while**, представьте его как **охранника**, который проверяет пропуск (условие) перед каждым проходом:



Шаг 2. Операторы `continue` и `break`

Оператор **break** немедленно **останавливает выполнение цикла**, даже если условие всё ещё истинно.

Его нужно использовать, когда достигнуто нужное значение, и дальше выполнять цикл бессмысленно.

Рассмотрим на примере:

```
fun main() {  
    var number = 1  
    while (number <= 10) {  
        println(number)  
        number++  
    }  
}
```

Теперь, когда мы дойдём до числа 5, мы хотим выйти из программы и сообщить про это. Добавим условие с if:

```
while (number <= 10) {  
    println(number)  
    number++  
    if (number == 5) {  
        println("Останавливаемся на $number")  
        break  
    }  
}
```

Запустите программу и проверьте вывод:

```
1  
2  
3  
4  
Останавливаемся на 5
```

Оператор **continue** прерывает только текущую итерацию и переходит к следующей.

Его нужно использовать, когда вы хотите пропустить ненужное значение, но продолжить цикл дальше.

Давайте немного изменим наш предыдущий пример:

1. Поменяем значение на 0 и условие на < 10
2. Переместил `number++` в начало цикла
3. Проверяем условие ДО вывода числа
4. Используем `continue` для пропуска вывода

```

fun main() {
    var number = 0
    while (number < 10) {
        number++
        if (number == 3) {
            println("Останавливаемся на $number")
            continue
        }
        println(number)
    }
}

```

Таким образом наш цикл начинается с `number = 0`, затем сразу увеличивает значение переменной на 1. Если `number == 3` - пропускаем вывод, во всех остальных случаях - выводим число.

Сделайте коммит и пуш вашего проекта на GitHub.

Цикл `while` часто используется, когда нужно написать **бесконечный цикл**, если условие сохраняется `true`.

Давайте создадим простую программу, которая будет считывать ввод пользователя, и если введено слово «выход», то программа завершит цикл. В противном случае будет повторять введенную строку:

```

fun main() {
    println("Напишите что-нибудь (для выхода введите 'выход'):")

    while (true) {
        val input = readLine()
        if (input == "выход") {
            println("До свидания!")
            break
        }
        println("Вы ввели: $input")
    }
}

```

Циклы можно использовать для обработки символов, строк и любых других типов данных. Программа ниже отображает английские буквы в одну строку:

```

fun main() {
    var letter = 'A'
    while (letter <= 'Z') {
        print(letter)
        letter++
    }
}

```

Программа берет первую букву А и затем выполняет цикл, пока буква не станет Z. Программа также выводит текущий символ и переходит к следующему. Таким образом, вывод:

ABCDEFGHIJKLMNOPQRSTUVWXYZ

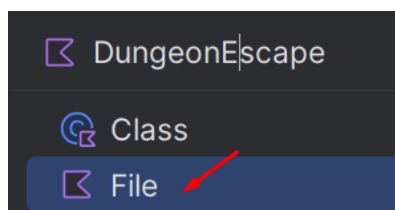
Шаг 3. Побег из подземелья

Решим следующую задачу:

- Игрок находится в подземелье на уровне 0.
- Его цель — дойти до выхода на уровне 5.
- На каждом уровне программа:
 - сообщает, где находится игрок,
 - предлагает действие: идти вперёд (1), осмотреться (2) или сдаться (0).
- Если игрок выбирает 0, игра завершается (break).
- Если игрок выбирает некорректное число — сообщение об ошибке (continue) и запрос снова.

Попробуйте сперва решить эту задачу **самостоятельно**.

В начале создадим файл **DungeonEscape** для нашей задачи:



Объявляем точку входа в нашу программу:

```

fun main() {
}

```

Сперва нам нужно создать переменную **level**, которая отслеживает текущий уровень игрока с начальным значением — 0:

```
var level = 0
```

И выведем приветственное сообщение:

```
println("Добро пожаловать в Подземелье!")
```

Далее создадим цикл **while**:

```
while () {  
  
}
```

Игра продолжается **пока уровень меньше 5**. Как только игрок дойдёт до 5 — цикл завершится (игрок прошёл подземелье):

```
while (level < 5) {
```

Выводим текущий уровень и меню:

```
println("\nВы на уровне $level.")  
println("Выберите действие: [1] Вперёд | [2]  
Осмотреться | [0] Сдаться")
```

Пользователю каждый раз показывается, на каком он уровне и какие действия доступны.

Далее получим пользовательский ввод из консоли:

```
val input = readLine()
```

И создадим обработку ввода через **when**:

```
when (input) {  
    "1" -> {}  
    "2" -> {}  
    "0" -> {}  
    else -> {}  
}
```

Теперь пройдемся по каждому варианту.

Вариант "1" — Идти вперёд:

```
"1" -> {  
    level++  
    println("→Вы переходите на уровень $level.")  
}
```

Увеличиваем уровень на 1. Показываем сообщение, что игрок продвинулся дальше.

Вариант "2" — Осмотреться:

```
"2" -> {  
    println("Вы осматриваетесь. Тут ничего  
    интересного.")  
    continue  
}
```

Игрок не переходит на следующий уровень. Используется `continue`, чтобы сразу перейти к **следующей итерации** цикла — снова вывести меню.

Вариант "0" — Сдаться:

```
"0" -> {  
    println("Вы сдались. Игра окончена.")  
    break  
}
```

Ввод другого значения:

```
else -> {  
    println("Неверный ввод. Попробуйте снова.")  
    continue  
}
```

Если пользователь ввёл что-то непонятное — сообщаем об этом и возвращаемся к началу цикла.

И теперь под конструкцией **when**, выведем для игрока, который достиг пятого уровня – поздравление:

```
if (level == 5)  
    println("Поздравляем! Вы выбрались из подземелья!")
```


Шаг 4. Цикл `do...while`

Сначала выполняется цикл `do...while`, после этого программа проверяет условие. Если условие равно `true`, она повторяет цикл до тех пор, пока условие не станет `false`. Поскольку `do...while` циклы проверяют условие после выполнения, цикл также известен как **цикл после проверки**. В отличие от `while` цикла, который проверяет условие перед выполнением, `do...while` цикл является циклом с условием выхода. Таким образом, тело всегда выполняется по крайней мере один раз.

Этот цикл состоит из трех частей: ключевого слова **`do`**, тела и **`while (condition)`**:

```
do {  
    // Блок кода  
} while (условие)
```

`do...while` **выполняет тело хотя бы один раз**, а потом проверяет условие.

Давайте напишем следующую программу:

```
fun main() {  
    var number: Int  
    do {  
        print("Введите число больше 10: ")  
        number = readln().toInt()  
    } while (number <= 10)  
  
    println("Спасибо! Вы ввели $number.")  
}
```

Как работает наш цикл:

- Цикл **гарантированно запрашивает хотя бы один ввод**.
- Пока пользователь вводит 10 или меньше — цикл повторяется.
- Как только введено число > 10 — программа продолжает выполнение.

Чтобы лучше понять данный цикл давайте напишем программу, которая запрашивает пароль у пользователя, пока он не введет правильный пароль **"qwerty"** сперва на цикле `while`:

```

fun main() {
    print("Введите пароль: ")
    var password: String? = readln()

    while (password != "qwerty") {
        print("Введите пароль: ")
        password = readln()
    }

    println("Доступ разрешён!")
}

```

А теперь давайте перепишем его через цикл do...while.

```

fun main() {
    var password: String?

    do {
        print("Введите пароль: ")
        password = readln()
    } while (password != "qwerty")

    println("Доступ разрешён!")
}

```

Мы убрали первый вывод в консоль. На практике программисты не используют do...while так часто, как while.

Таким образом цикл do...while лучше использовать, когда нужно **гарантировать хотя бы одну итерацию**.

Шаг 5. Цикл for

Цикл for в Kotlin — довольно часто используемый, удобный и понятный цикл. Подойдёт для перебора диапазонов, коллекций, строк и многого другого.

Общий синтаксис:

```

for (переменная in диапазон/коллекция) {
    // тело цикла
}

```

Разберём на примере перебора диапазона чисел:

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

Мы выводим шаг с 1 по 5 включительно, используя диапазон. Цикл for является аналогом цикла foreach в других языках (например, C#).

Давайте переберём элемент списка.

Допустим у нас есть список с фруктами, и мы хотим вывести каждый отдельно на консоль:

```
val fruits = listOf("apple", "banana", "cherry")
```

Внутри цикла создаём локальную переменную и проверяем что она входит в наш список, и затем выводим на экран каждый элемент.

```
for (fruit in fruits) {  
    println("Фрукт: $fruit")  
}
```

Шаг 6. Метод split()

Метод split() **разбивает строку на список подстрок**, используя один или несколько символов-разделителей.

Давайте разберём на примере. У нас есть строка, и мы хотим по пробелу между словами разделить её:

```
val sentence = "Kotlin is awesome"
```

Используем метод split, и передаём ему пробел в качестве разделителя:

```
val words = sentence.split(...delimiters = " ")
```

Выведем результат:

```
println(words)
```

Результат:

```
[Kotlin, is, awesome]
```

Рассмотрим другой пример, у нас есть строка, в которой слова разделены запятыми:

```
val data = "apple,banana,orange"
```

Через разделитель сохраним в отдельную переменную:

```
val fruits = data.split(...delimiters = ",")
```

Затем через цикл пройдемся по элементам и выведем каждый отдельный элемент:

```
for (fruit in fruits) {  
    println(fruit)  
}
```

Результат:

```
apple  
banana  
orange
```

Если у нас несколько разных разделителей, то мы можем указать их все:

```
val messy = "word1,word2;word3|word4"  
val parts = messy.split(...delimiters = ",", ";", "|")  
println(parts)
```

Также мы можем сохранять по индексу отдельные элементы:

```
val fullName = "Иванов Иван"  
val parts = fullName.split(...delimiters = " ")  
val lastName = parts[0]  
val firstName = parts[1]  
println("Фамилия: $lastName, Имя: $firstName")
```

Давайте напишем программу, которая запрашивает у пользователя строку с числами, разделёнными пробелами и выводит сумму этих чисел.

Вначале спросим у пользователя ввести числа через пробел:

```
print("Введите числа через пробел: ")
```

Затем сохраним их в строку:

```
val input = readln()
```

По пробелу разделим нашу строку, чтобы получить отдельные числа:

```
val numbers = input.split(...delimiters = " ")
```

Создадим переменную для сохранения суммы наших чисел:

```
var sum = 0
```

После через цикл `for` пройдемся по нашей коллекции и сохраним результат сложения, не забыв перевести `number` к типу `Int`:

```
for (number in numbers) {  
    sum += number.toInt()  
}
```

И выведем результат на консоль:

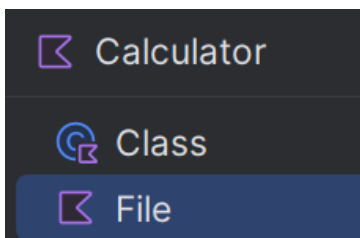
```
println("Сумма чисел: $sum")
```

Шаг 7. Калькулятор

Давайте с вами создадим калькулятор, который будет спрашивать у пользователя два числа, операцию, и выводить результат:

```
Введите первое число, знак операции и второе число через пробел: 23 + 12  
23.0 + 12.0 = 35.0
```

В начале создадим в проекте новый файл с названием **Calculator**:



В нём создадим точку входа в программу **main**:

```
fun main() {  
  
}
```

Далее попросим у пользователя ввести нужные нам операторы:

```
print("Введите первое число, знак операции и второе число через пробел: ")
```

Сохраним ввод в переменную `input`, и разделим вводимые значения пробелами через `split`:

```
val input = readln().split(...delimiters = " ")
```

Далее по индексу сохраним три вводимых оператора в переменные:

```
val symbol = input[1]
val firstNumber = input[0].toDouble()
val secondNumber = input[2].toDouble()
```

Создаём переменную для сохранения значения между двумя числами:

```
var result = 0.0
```

Создаём **when** и передаём в качестве условия переменную `symbol`:

```
when (symbol) {
}
```

Внутри **when** в соответствии с вводимым символом мы будем выполнять соответствующие операции:

```
"/" -> result = firstNumber / secondNumber
"*" -> result = firstNumber * secondNumber
"+" -> result = firstNumber + secondNumber
"-" -> result = firstNumber - secondNumber
else -> println("Некорректный ввод")
```

И после конструкции **when** выведем результат наших действий:

```
println("$firstNumber $symbol $secondNumber = $result")
```

Шаг 8. Random

Случайное число — это число, которое **невозможно предсказать заранее**, как, например, результат броска кубика. В программировании **генераторы случайных чисел** используются повсеместно:

- в играх (рандомный урон, случайный лут);

- в криптографии (создание ключей);
- в машинном обучении (перемешивание данных);
- в тестировании (генерация случайных данных).

Kotlin предоставляет удобные инструменты для генерации случайных чисел через пакет **kotlin.random**.

Генерация случайного Int:

```
import kotlin.random.Random  
  
fun main() {  
    // генерирует целое значение между Int.MIN_VALUE и Int.MAX_VALUE  
    println(Random.nextInt())  
}
```

Аналогично мы можем сделать тоже самое для типов **Long**, **Float** и **Double**:

```
import kotlin.random.Random  
  
fun main() {  
    println(Random.nextLong())  
    println(Random.nextFloat())  
    println(Random.nextDouble())  
}
```

Обратите внимание, что, если вы запустите его снова, числа будут другими. Это потому, что генератор дает нам каждый раз новое число: они могут иногда повторяться, но вероятность этого очень мала.

Как вы видите, некоторые числа оказались довольно большими, другие маленькими, а одно вообще отрицательным. Диапазоны по умолчанию для целых чисел и чисел с плавающей точкой различны и выбраны для охвата наиболее распространенных случаев. Но что, если нам нужны только целые числа от 0 до 100 или числа с плавающей точкой от 0,0 до 5,0? Тогда мы можем указать их явно:

```
Random.nextInt(until = 100)  
Random.nextInt(from = 1, until = 100)  
Random.nextLong(until = 100)  
Random.nextLong(from = 1, until = 100)  
Random.nextDouble(until = 5.0)  
Random.nextDouble(from = 0.0, until = 5.0)
```

Теперь мы можем быть уверены, что отрицательных чисел не будет, все целые числа будут меньше 100, а некоторые числа с плавающей точкой будут больше 1,0.

Обратите внимание, что `nextFloat` это единственная функция, которая не позволяет указать пользовательский диапазон: `nextDouble` обеспечивает лучшую точность.

Давайте решим **задачу**, в которую вы возможно играли в детстве. Вы загадываете случайное число от 1 до 100, и просите вашего друга угадать его.

Вот пример того, как будет выглядеть задача:

```
Угадай число от 1 до 100:
Введи число: 50
Загаданное число меньше.
Введи число: 25
Загаданное число меньше.
Введи число: 12
Загаданное число меньше.
Введи число: 6
Загаданное число больше.
Введи число: 9
Загаданное число меньше.
Введи число: 7
Загаданное число больше.
Введи число: 8
Поздравляю! Ты угадал число: 8
```

Начнём писать код, вначале убедимся, что мы подключили пакет **kotlin.random**:

```
import kotlin.random.Random
```

Затем мы сгенерируем случайное число от 1 до 100 включительно и выведем название игры:

```
fun main() { new *
    val target = Random . nextInt ( from = 1, until = 101)
    println("Угадай число от 1 до 100")
```

Используем цикл **while**, который продолжается, пока пользователь не угадает:


```
while (true) {  
  
}
```

Внутри него попросим ввести пользователя число и сохраним его в переменную:

```
while (true) {  
    print("Введи число: ")  
    val input = readln().toInt()  
}
```

Дальше внутри when будем осуществлять проверку больше ли нашего число искомого и будем выводить подсказку, при условии, что мы угадали число выведем поздравление и исходное число, а также выйдем из цикла:

```
when {  
    input < target -> println("Загаданное число больше.")  
    input > target -> println("Загаданное число меньше.")  
    else -> {  
        println("Поздравляю! Ты угадал число: $target")  
        break  
    }  
}
```

Теперь рассмотрим следующую задачу. У нас есть два кубика, мы их бросаем, и хотим найти сумму бросков:

```
Первый кубик: 3  
Второй кубик: 5  
Сумма: 8
```

Мы помним, что на кубике есть 6 граней, значит числа должны быть в диапазоне от 1 до 6. Поэтому сгенерируем их:

```
fun main() {  
    new *  
    val dice1 = Random.nextInt(from = 1, until = 7)  
    val dice2 = Random.nextInt(from = 1, until = 7)
```

Далее находим их сумму:

```
val sum = dice1 + dice2
```

Выводим результаты:

```
println("Первый кубик: $dice1")
println("Второй кубик: $dice2")
println("Сумма: $sum")
```

И последняя задача, которую попробуйте решить самостоятельно:

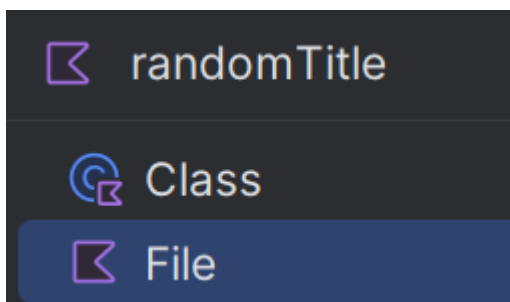
Сгенерируйте три случайных числа с плавающей точкой от 0.0 до 10.0 и найдите их среднее.

```
Числа: 2.7740373774809712, 7.1288238988576875, 5.373605183053365
Среднее: 5.092155486464008
```

Шаг 9. Рандомный титул

Создадим с вами генерацию случайного игрового титула

Создайте новый файл с именем **randomTitle**:



Создайте точку входа в программу метод **main**.

Далее подготовим массивы с частями титула (можете придумать свои):

```
val part1 = arrayOf("Опытный", "Безумный", "Легендарный",
    "Скрытый", "Гигачад")
val part2 = arrayOf("стрелок", "геймер", "воин",
    "волшебник", "трейдер")
val part3 = arrayOf("из CSGO", "на максималках", "из
    будущего", "в бане у Габена", "с проклятым лутом")
```

- **Что делаем:** Создаем 3 массива строк, где:
 - part1 — прилагательные (характеристики).
 - part2 — существительные (роли/классы).
 - part3 — дополнения (локации/особенности).
- **Зачем:** Эти части будут комбинироваться для генерации уникального титула. Затем сгенерируем случайные индексы:

```
val rand1 = (Math.random() * part1.size).toInt()
val rand2 = (Math.random() * part2.size).toInt()
val rand3 = (Math.random() * part3.size).toInt()
```

- **Что делаем:**

1. Math.random() генерирует число от 0.0 до 1.0.
2. Умножаем его на размер массива (part1.size = 5), получая число от 0.0 до 5.0.
3. Приводим к целому числу (toInt()), чтобы получить индекс от 0 до 4.

- **Пример:** Для part1 может получиться rand1 = 2 → "Легендарный".

И наконец соберём наш титул:

```
val phrase = "${part1[rand1]} ${part2[rand2]} ${part3[rand3]}"
println("Ваш титул: $phrase")
```

- **Что делаем:**

Конкатенируем (соединяем) три случайные части в одну строку через пробел.

- **Пример результата:**

"Безумный геймер из будущего".

Самостоятельные задания

Задание 1. Угадай число

Напишите программу, которая загадывает случайное число от 1 до 50. Пользователь должен его угадать, программа подсказывает — «больше» или «меньше». Используйте цикл while.

Задание 2. Счётчик гласных

Напишите функцию countVowels, которая принимает строку и возвращает количество гласных букв в ней.

Задание 3. Обратный отсчёт

Вывести числа от N до 1 (N вводит пользователь).

Задание 4. Генератор пароля

Напишите функцию, генерирующую случайный пароль длиной от 8 до 16 символов. Пароль должен содержать:

- Заглавные и строчные буквы
- Цифры
- Спецсимволы (!@#\\$%^&* и т.п.)

Задание 5. Мини-опрос

Создайте опрос, в котором пользователь вводит имя и 3 ответа на простые вопросы. После завершения программа выводит:

- Имя
- Все ответы
- Случайный отзыв, например: "Ты крутой!" или "Хорошая работа!" (выберите случайно из listOf())

Задание 6. Сумма чисел

Вывести сумму чисел от 1 до N (N вводит пользователь).

Задание 7. Кубик D6

Сымитировать 10 бросков шестигранного кубика (числа от 1 до 6).

Задание 8. Слот-машина

- При каждом запуске программа выводит 3 случайных числа от 0 до 5.
- Если все три числа одинаковые — вывести "Джекпот!".

Задание 9. Банковский счёт

- На счету случайная сумма от 100 до 1000 рублей.
- Каждый день снимается случайная сумма (от 10 до 100 рублей).
- Выводить остаток на счету каждый день, пока деньги не закончатся.

Задание 10. Прогноз погоды

- Сгенерировать температуру для каждого дня недели (от -10 до +30°C).
- Вывести дни, когда температура была ниже нуля.