

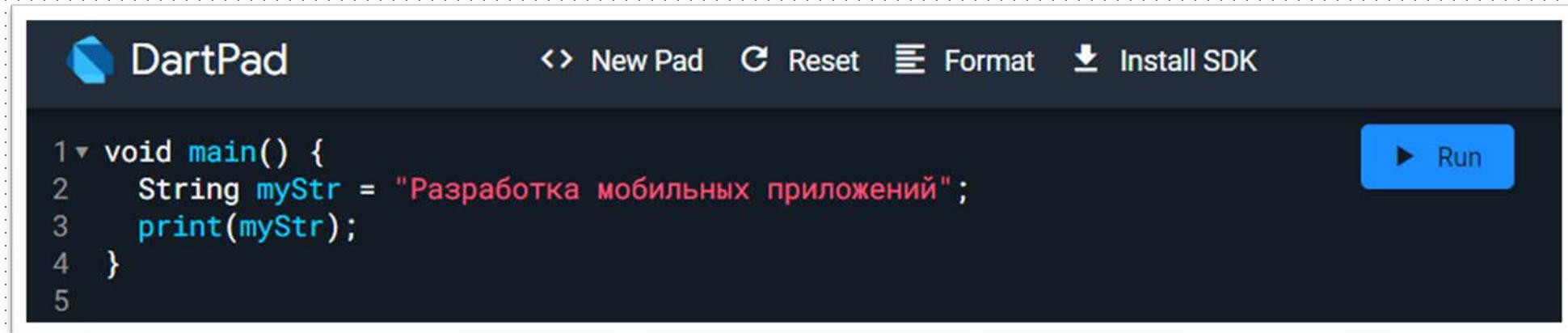
# Особенности языка - Dart

# Dart?

- В 2011 году [Dart](#) появился как замена [JavaScript](#). Это значит, что знакомый с web-программированием, найдёт схожие с [JavaScript](#) возможности в [Dart](#). На сегодняшний день язык прошёл достаточно долгий путь, получил поддержку [null-safety](#), статическую типизацию и [Dart-VM](#).
- Чаще всего его можно увидеть в паре с фреймворком [Flutter](#) в сфере разработки кроссплатформенных приложений, также сейчас есть перспективные попытки использовать [Dart](#) для написания серверов.

# Переменные

- В [Dart](#) есть много встроенных типов. На перечислении всех мы останавливаться не будем, подробнее о них можно прочитать в документации языка.
- Создание переменной мало чем отличается от других языков. Пример:



DartPad

New Pad Reset Format Install SDK

```
1 void main() {  
2     String myStr = "Разработка мобильных приложений";  
3     print(myStr);  
4 }  
5
```

Run

The screenshot shows the DartPad interface. At the top, there's a toolbar with 'New Pad', 'Reset', 'Format', and 'Install SDK'. Below the toolbar is a code editor window containing the following Dart code:

```
1 void main() {  
2     String myStr = "Разработка мобильных приложений";  
3     print(myStr);  
4 }  
5
```

To the right of the code editor is a blue 'Run' button with a play icon.

- Код выше создаёт переменную типа `String`, присваивает ей значение «Разработка мобильных приложений» и выводит результат в консоль.



[Документация языка](#)

**var**

- **Dart** поддерживает вывод типов ([type-inference](#)), так что указывать тип переменной сразу необязательно. На помощь приходит ключевое слово **var**:

The screenshot shows the DartPad interface. The code editor contains the following Dart code:

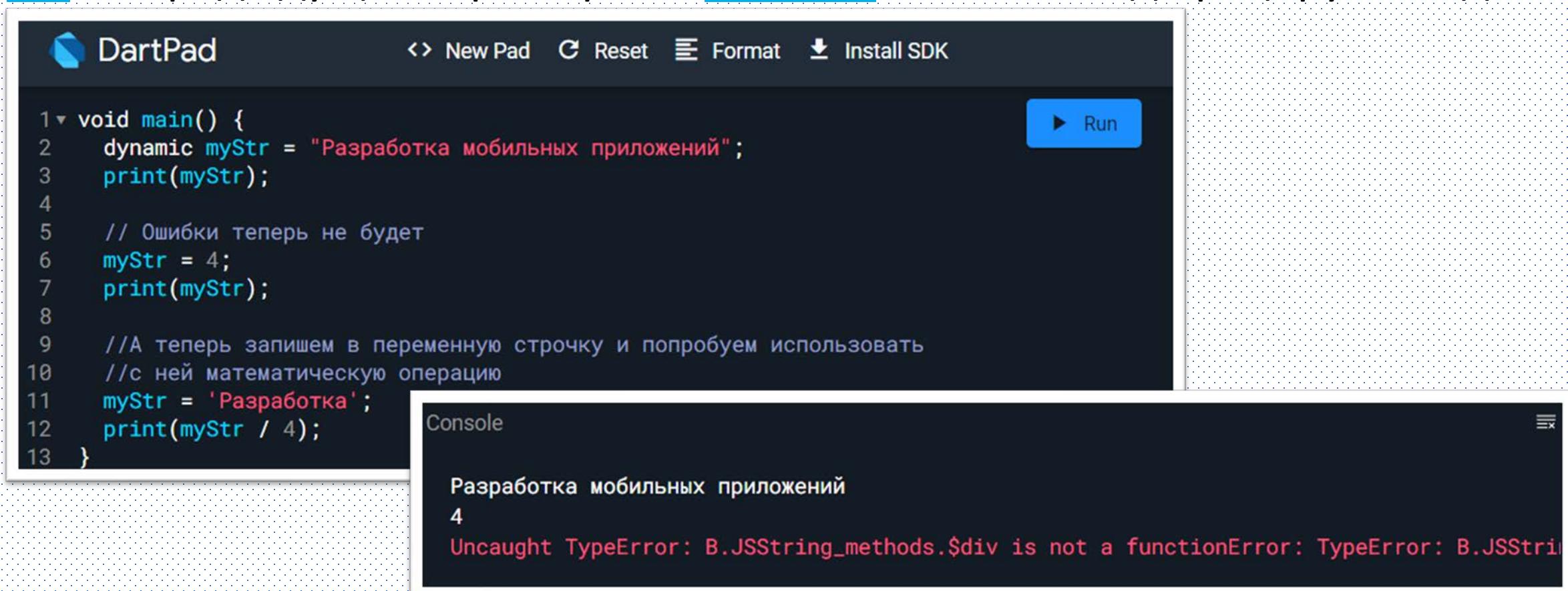
```
1 void main() {  
2     var myStr = "Разработка мобильных приложений";  
3     print(myStr);  
4  
5     //Код ниже выбросит compile time исключение,  
6     //потому что на строке 2 для переменной был вычислен тип String  
7     myStr = 4;  
8     print(myStr);  
9 }
```

The code uses `var` to declare `myStr` without specifying its type. It prints the string "Разработка мобильных приложений". Then it attempts to assign the integer value 4 to `myStr`, which would normally cause a runtime error because the variable has already been assigned a string value. Instead, the code runs successfully, demonstrating type inference.

**error** line 7 • A value of type 'int' can't be assigned to a variable of type 'String'. (view docs) Try changing the type of the variable, or casting the right-hand type to 'String'.

## dynamic

- В [Dart](#) есть ключевое слово [dynamic](#), отключающее проверки типов для переменной. Им стоит пользоваться, если тип переменной не известен до запуска программы и его нельзя вывести. Далее заменим [var](#) из предыдущего примера на [dynamic](#) и слегка модифицируем код:



DartPad

New Pad Reset Format Install SDK

Run

```
1 void main() {  
2     dynamic myStr = "Разработка мобильных приложений";  
3     print(myStr);  
4  
5     // Ошибки теперь не будет  
6     myStr = 4;  
7     print(myStr);  
8  
9     //А теперь запишем в переменную строчку и попробуем использовать  
10    //с ней математическую операцию  
11    myStr = 'Разработка';  
12    print(myStr / 4);  
13 }
```

Console

```
Разработка мобильных приложений  
4  
Uncaught TypeError: B.JSString_methods.$div is not a function  
Error: TypeError: B.JSString_methods.$div is not a function
```

- Пользоваться стоит с осторожностью, потому что любые ошибки, когда используете данную переменную, можно заметить только во время работы программы.
- Стоит заметить, что в случае с var вычисление типа переменной всегда происходит на этапе компиляции. Следовательно, если переменная, помеченная var, не будет инициализирована при объявлении, то такая переменная будет иметь «неизвестный тип» (null), что позволит присвоить ей значение любого типа.
- Рассмотрим описанное поведение на примере ниже:



<> New Pad ⌂ Reset Ⓛ Format ⬇ Install SDK

```
1 void main() {  
2     // Чтобы избежать такого неоднозначного поведения,  
3     // достаточно явно указать тип переменной  
4     // String? myVar;  
5     var myVar;  
6  
7     print('Not initialized variable:');  
8     print(myVar.runtimeType);  
9     print(myVar);  
10  
11    //Теперь присвоим целочисленное значение|  
12    myVar = 1;  
13    print('Integer variable:');  
14    print(myVar.runtimeType);  
15    print(myVar);  
16  
17    //И нам также не мешает этой же переменной присвоить Строку  
18    myVar = '2';  
19    print('String variable:');  
20    print(myVar.runtimeType);  
21    print(myVar);  
22 }  
23
```

▶ Run

### Console

```
Not initialized variable:  
Null  
null  
Integer variable:  
int  
1  
String variable:  
String  
2
```

info

line 5 • An uninitialized variable should have  
an explicit type annotation. ([view docs](#))

Try adding a type annotation.

## **final and const**

- Во всех примерах выше переменные можно модифицировать и переопределять после объявления. Но чаще всего при разработке появляется потребность запретить изменение переменной после определения. Например, хорошей практикой считается делать свойства модели, и следовательно, саму модель неизменяемыми. А константные объекты позволяют выиграть в производительности.

### **final**

- Ключевое слово [final](#) запрещает переопределение переменной после инициализации.
- [Важно](#). Переменная, помеченная [final](#), — не константа, просто её значение нельзя переопределить после инициализации.
- Далее создадим переменную [final](#) и попытаемся изменить её значение.



<> New Pad ⌂ Reset ┓ Format └ Install SDK

```
1▼ void main() {  
2    final myVar;  
3  
4    myVar = 5;  
5  
6    print('Integer variable: $myVar');  
7  
8    myVar = 6;  
9    print('Integer variable 2: $myVar');  
10  
11   myVar = 'Some string';  
12   print('String variable: $myVar');  
13 }
```

▶ Run

error

line 8 • The final variable 'myVar' can only be set once. [\(view docs\)](#)

Try making 'myVar' non-final.

error

line 11 • The final variable 'myVar' can only be set once. [\(view docs\)](#)

Try making 'myVar' non-final.

info

line 2 • An uninitialized variable should have an explicit type annotation. [\(view docs\)](#)

Try adding a type annotation.

- Действительно, переменная myVar получит значение 5, а при дальнейших попытках переприсвоить ей значение мы увидим исключение на этапе компиляции.
- При этом можно модифицировать внутреннее состояние значения, присвоенного переменной final.

The screenshot shows the DartPad interface. On the left, the code editor contains the following Dart code:

```
1 void main() {  
2   final myList = [5, 8, 10];  
3   myList.add(7);  
4   print(myList);  
5 }  
6
```

On the right, there is a blue "Run" button. Below the code editor, a "Console" window displays the output: [5, 8, 10, 7].

## late

- Ключевое слово `late` позволяет объявить переменную и не инициализировать её значение сразу.
- Это полезно, когда хотим объявить переменную non-null (o null-safety), а значение ей задать позже.

The screenshot shows the DartPad interface. In the top bar, there is a logo, the text "DartPad", and several buttons: "New Pad", "Reset", "Format", and "Install SDK". Below the bar is a code editor window containing the following Dart code:

```
1 //Без late будет исключение compile-time
2 late String myVar;
3
4 void main() {
5     myVar = 'приложений';
6     print('Разработка мобильных $myVar');
7 }
```

To the right of the code editor is a "Run" button. Below the code editor is a "Console" window with the text "Разработка мобильных приложений".

## const

- Ключевое слово `const` помечает константы. Их значения известны ещё на этапе компиляции, и их нельзя модифицировать или переопределять во время исполнения.

The screenshot shows a DartPad interface. On the left, there is a code editor with the following Dart code:

```
1 void main() {  
2     const myVar = 12;  
3  
4     myVar = 5;  
5     print('Integer variable: $myVar');  
6  
7     myVar = 6;  
8     print('Integer variable 2: $myVar');  
9  
10    myVar = 'Разработка мобильных приложений';  
11    print('String variable: $myVar');  
12 }  
13
```

On the right, the output window displays four errors:

- line 4 • Constant variables can't be assigned a value. (view docs)  
Try removing the assignment, or remove the modifier 'const' from the variable.
- line 7 • Constant variables can't be assigned a value. (view docs)  
Try removing the assignment, or remove the modifier 'const' from the variable.
- line 10 • Constant variables can't be assigned a value. (view docs)  
Try removing the assignment, or remove the modifier 'const' from the variable.
- line 10 • A value of type 'String' can't be assigned to a variable of type 'int'. (view docs)

**В итоге получаем сразу несколько ошибок до компиляции:**

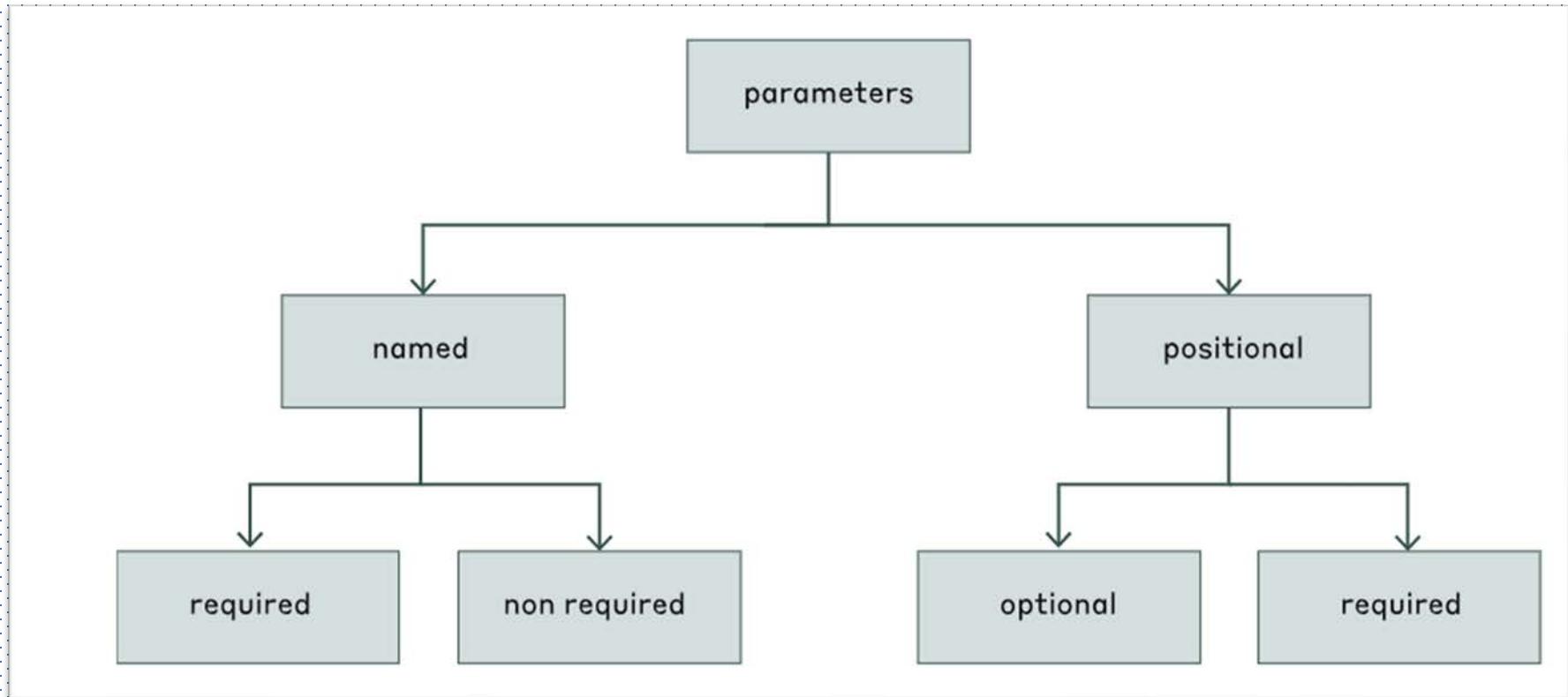
- Переменную `const` нельзя оставить неинициализированной;
- Переменную `const` нельзя переопределить;
- Переменной `const` нельзя изменить тип, что очевидно.

# ФУНКЦИИ

- [Dart](#), следуя за тенденциями, предоставляет большой инструментарий по работе с функциями.

## Параметры функции

- Все виды параметров условно можно распределить на следующей диаграмме.



## positional — параметры функции.

- Их всегда нужно передать, даже если один из них nullable (то есть необязательный):

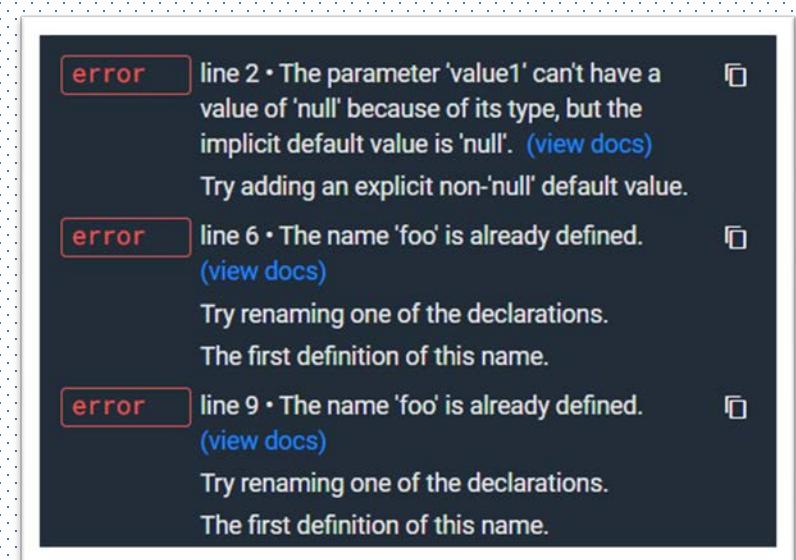
DartPad interface showing code execution. The code defines a function `foo` that takes two parameters: `int value` and `String? value1`. In the `main` function, it calls `foo(5, null)`. A blue "Run" button is visible on the right.

```
1 void foo(int value, String? value1) {}
2
3▼ void main() {
4    foo(5, null);
5
6}
```

**optional positional** — параметры, которые необязательно передавать при вызове. Но нельзя, чтобы у таких параметров non-nullable не было значения.

DartPad interface showing code execution. The code defines a function `foo` that takes two parameters: `int value` and `[String value1]`. It includes comments explaining the usage of optional parameters and their default values. A blue "Run" button is visible on the right.

```
1 // Это ошибка, у value1 гарантированно должно быть значение
2 void foo(int value, [String value1]) {}
3
4 // У optional параметров могут быть значения по-умолчанию.
5 // Важно: значение default — всегда константа
6 void foo(int value, [String value1 = '']) {}
7
8 // Либо можно сделать их nullable, тогда их значение по умолчанию — null
9 void foo(int value, [String? value1]) {}
10
11▼ void main() {
12    foo(19);
13 }
```



- named — параметры, которые позволяют создавать функции с огромным количеством параметров и не ошибиться при вызове.

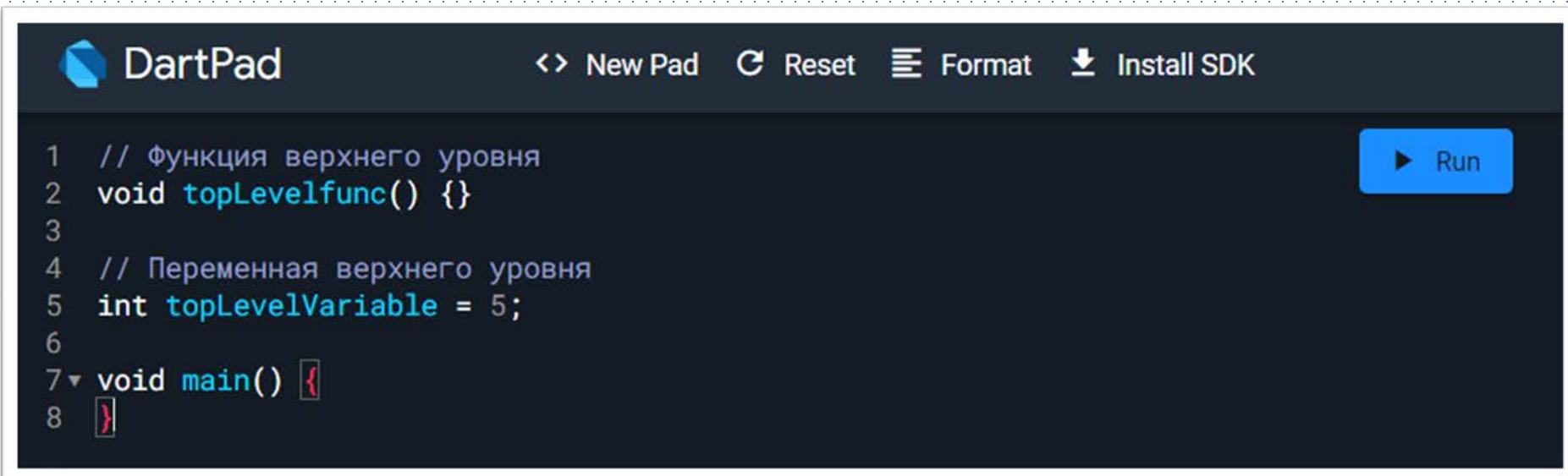
DartPad interface with the following code:

```
1 // неудобно и сложно
2 void foo1(int value1, bool flag, int value2, int value3) {}
3 // удобно и читаемо
4 void foo2({required int value1, required bool flag, required int value2,
5           required int value3}) {}
6
7 void main() {
8   foo1(5, true, 2, 1);
9
10  foo2(value1: 5, flag: true, value2: 2, value3: 3);
11 }
```

The code illustrates two functions: `foo1` and `foo2`. `foo1` has four parameters: `value1`, `flag`, `value2`, and `value3`. `foo2` has three required parameters: `value1`, `flag`, and `value2`, plus one optional parameter `value3`. The `main` function calls both functions with different argument lists. The DartPad interface shows the code in a dark-themed editor with syntax highlighting and a toolbar with "Run" and other options.

## Функции верхнего уровня и переменные

- Это понятие носит много имён: глобальные функции, функции высшего порядка, «просто» функции.
- Для удобства мы будем называть их функциями верхнего уровня.
- Но суть у них одна — они и переменные могут существовать вне контекста какого-то класса.
- Простейший пример такой функции — main(). Это не только точка входа в программу, но ещё и функция верхнего уровня.



DartPad interface with the following code:

```
1 // Функция верхнего уровня
2 void topLevelFunc() {}
3
4 // Переменная верхнего уровня
5 int topLevelVariable = 5;
6
7 void main() {
8 }
```

The code defines a top-level function `topLevelFunc`, a top-level variable `topLevelVariable` initialized to 5, and the `main` function which is currently expanded to show its body.

# Стрелочные функции

- Стрелочные функции позволяют описывать функции в одну строку:

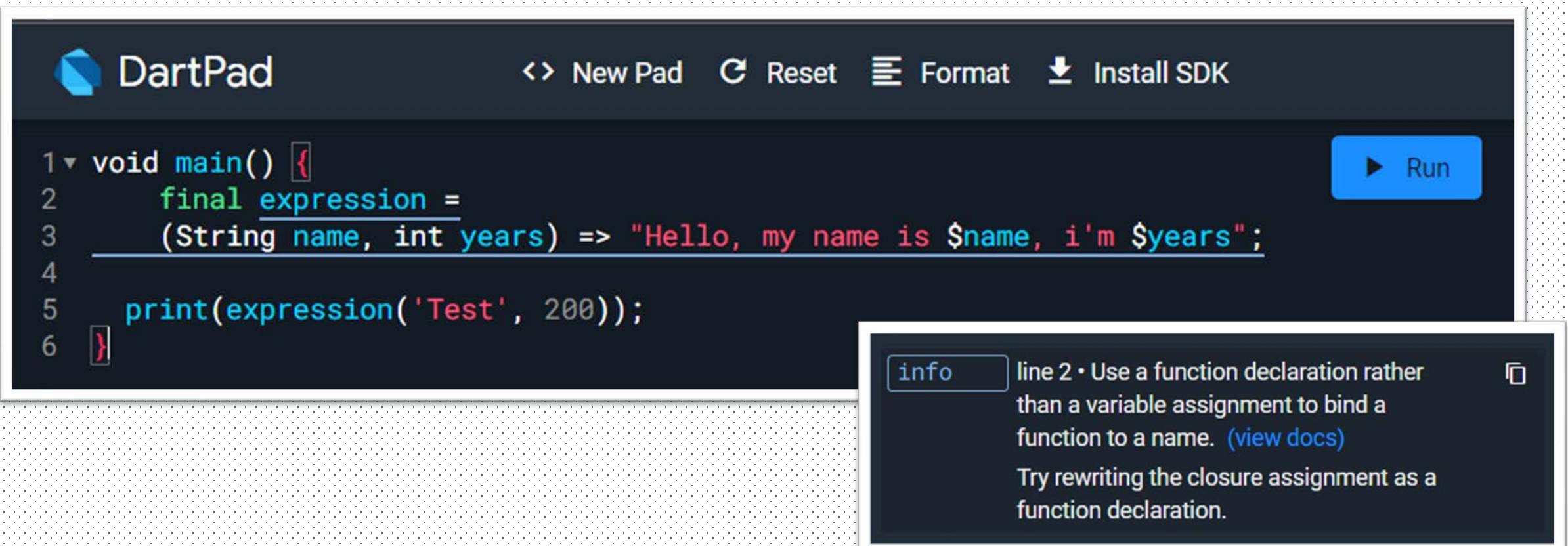
DartPad interface with the following code:

```
1 int getInteger() => 5; // Код вернет 5
2
3 // 1. Стрелочная функция может и не возвращать какого-то конкретного значения
4 // 2. А void может быть опущен
5 // И то, и другое - плохой тон, хоть и так можно.
6 printInteger() => print(getInteger());
7
8 // Ниже эквивалентный код без стрелочных функций
9▼ int getInteger() {
10    return 5;
11 }
12
13▼ printInteger() {
14    print(getInteger());
15 }
```

The screenshot shows the DartPad interface with the code editor containing the provided Dart code. The code demonstrates the use of arrow functions for both returning values and performing side effects like printing. A 'Run' button is visible on the right side of the editor.

## Анонимные функции

- Из функционального подхода в [Dart](#) перекочевали анонимные функции — их можно воспринимать как объект типа [Function](#).



DartPad

New Pad Reset Format Install SDK

```
1 void main() {  
2     final expression =  
3         (String name, int years) => "Hello, my name is $name, i'm $years";  
4  
5     print(expression('Test', 200));  
6 }
```

Run

info line 2 • Use a function declaration rather than a variable assignment to bind a function to a name. ([view docs](#))

Try rewriting the closure assignment as a function declaration.

The screenshot shows a DartPad interface with the following code:

```
void main() {  
    final expression =  
        (String name, int years) => "Hello, my name is $name, i'm $years";  
  
    print(expression('Test', 200));  
}
```

A tooltip from the Dart editor highlights line 2: "line 2 • Use a function declaration rather than a variable assignment to bind a function to a name. ([view docs](#))". It suggests rewriting the closure assignment as a function declaration.

У типа [Function](#) есть метод [call](#), он и есть вызов самой функции. Его полезно использовать, если переменная-функция может быть [null](#).

DartPad

New Pad Reset Format Install SDK

```
1 typedef PrinterFunc = String Function(String name, int years);
2
3 void main() {
4   PrinterFunc? expression = (String name, int years) => "Hello, my name is $name, i'm $years";
5
6   print(expression?.call('Test', 200));
7 }
```

Run

warning line 6 • The receiver can't be null, so the null-aware operator '?' is unnecessary. (view docs)

Try replacing the operator '?' with '..'.

info line 4 • Use a function declaration rather than a variable assignment to bind a function to a name. (view docs)

Try rewriting the closure assignment as a function declaration.

## Каскадный вызов методов

- Функция, позволяет вызывать последовательно методы одной переменной:

The screenshot shows the DartPad interface. The main editor window contains the following Dart code:

```
1 void main() {  
2     final list = [6, 19, 20];  
3     print(list..removeLast()..remove(0));  
4 }
```

The code uses the spread operator (`..`) to chain two `remove` methods on the `list` variable. The `Run` button is visible in the top right of the editor. To the right, a `Console` window displays the output: `[6, 19]`.

Результат каскадного вызова — исходное значение. Чтобы это лучше понять, рассмотрим следующий пример: [toLowerCase\(\)](#) и [toUpperCase\(\)](#) возвращают изменённое значение [value](#), но при каскадном вызове программа не выведет результат этих функций.

The screenshot shows the DartPad interface. The code editor contains the following Dart code:

```
1 void main() {  
2  
3     final String value = 'Разработка мобильных приложений';  
4  
5     // 'Разработка мобильных приложений' - исходное значение value  
6     print(value..toUpperCase()..toLowerCase());  
7  
8     // 'разработка мобильных приложений' - результат toLowerCase()  
9     print(value.toUpperCase().toLowerCase());  
10  
11 }
```

The 'Run' button is highlighted in blue. To the right, the 'Console' window displays the output of the program:

```
Разработка мобильных приложений  
разработка мобильных приложений
```

## Замыкания

- ЭТОТ МЕХАНИЗМ ПОЗВОЛЯЕТ АНОНИМНОЙ ФУНКЦИИ, ОБЪЯВЛЕННОЙ ВНУТРИ ДРУГОЙ, ПОЛУЧАТЬ ДОСТУП К ВЫШЕСТОЯЩЕМУ «[КОНТЕКСТУ](#)».

The screenshot shows a DartPad interface. On the left is the code editor with the following Dart code:

```
1 int counter = 1;
2
3 typedef RobotBuilder = String Function(String name);
4
5 RobotBuilder robotFactory(String specifications) {
6   final model = 'TEST - $counter';
7   counter++;
8   return (String name) => '$model, $name, $specifications';
9 }
0
1 void main() {
2   final firstRobot = robotFactory('Test 1!');
3   final secondRobot = robotFactory('Test 2!');
4   final thirdRobot = robotFactory('Test 3!');
5
6   print(firstRobot('1'));
7   print(secondRobot.call('2'));
8   print(thirdRobot('3'));
9 }
```

On the right is the 'Console' window displaying the output:

```
Console
TEST - 1, 1, Test 1!
TEST - 2, 2, Test 2!
TEST - 3, 3, Test 3!
```

- ФУНКЦИЯ ВЕРХНЕГО УРОВНЯ [robotFactory](#) ВОЗВРАЩАЕТ АНОНИМНУЮ ФУНКЦИЮ ТИПА [RobotBuild](#).
- САМА АНОНИМНАЯ ФУНКЦИЯ ФОРМИРУЕТ СТРОКУ ИЗ ПЕРЕДАННОГО [name](#) И ДОСТУПНЫХ ИЗ КОНТЕКСТА [robotFactory](#) ПЕРЕМЕННЫХ [model](#) И [specifications](#).

## Генераторы

- В Dart есть синхронные и асинхронные генераторы.
- Синхронные генераторы, например, пригодятся, когда вам нужно сгенерировать много тестовых данных:

The screenshot shows the DartPad interface with a code editor and a console window.

**Code Editor (Left):**

```
1 // Функции помеченные sync* всегда возвращают Iterable
2 Iterable<int> generateList(int length) sync* {
3     int counter = 0;
4     while (counter < length) {
5         // Функция "запомнит" это место и продолжит
6         // выполнение отсюда при последующем вызове
7         yield counter++;
8     }
9 }
10
11 void main() {
12     final myList = <int>[];
13
14     for (final value in generateList(10)) {
15         myList.add(value);
16     }
17
18     print(myList);
19 }
```

**Console (Right):**

```
Console
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```