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.

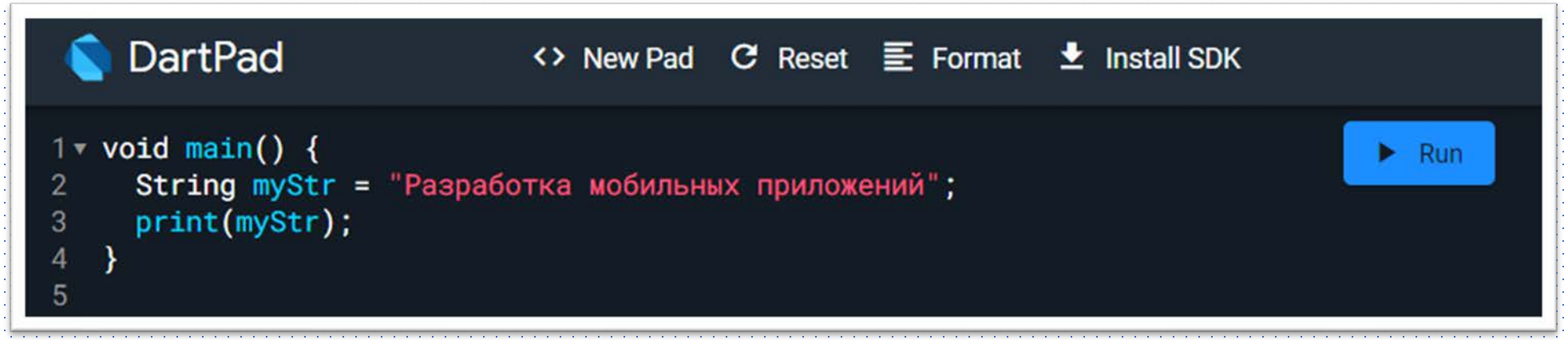
- Чаще всего его можно увидеть в паре с фреймворком Flutter в сфере разработки кроссплатформенных приложений, также сейчас есть перспективные попытки использовать Dart для написания серверов.

Переменные

- В Dart есть много встроенных типов. На перечислении всех мы останавливаться не будем, подробнее о них можно прочитать в документации языка.

Документация языка

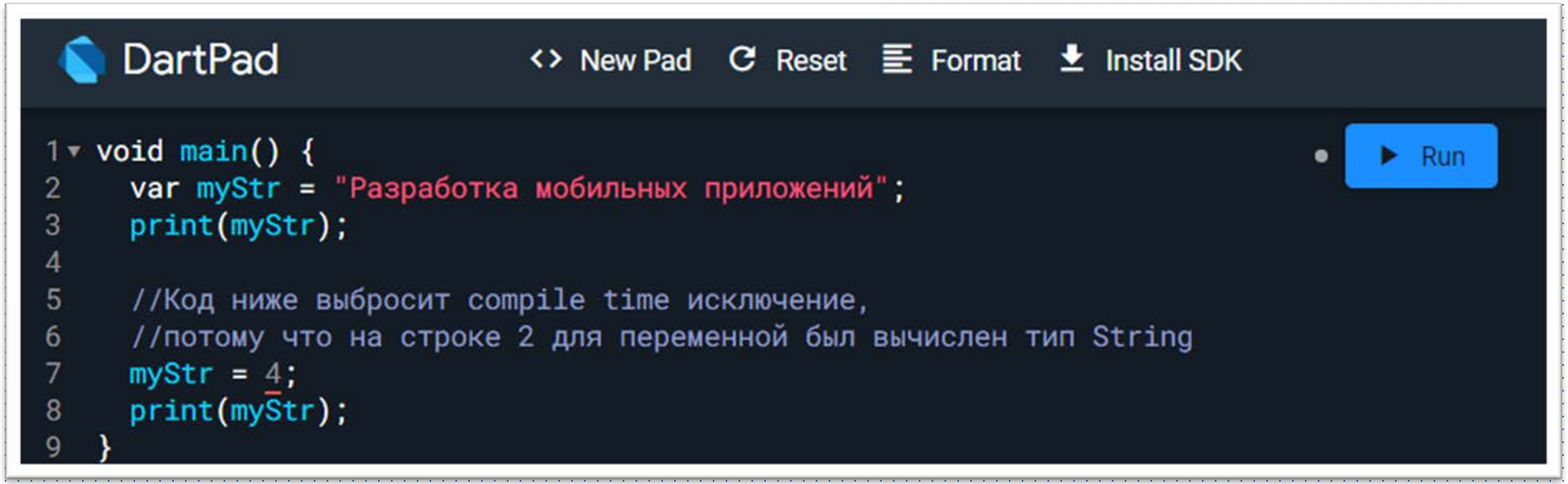
- Создание переменной мало чем отличается от других языков. Пример:

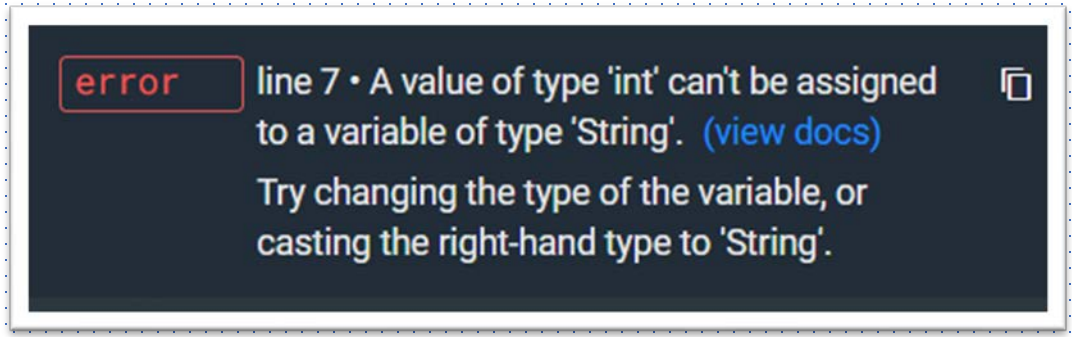


- Код выше создаёт переменную типа String, присваивает ей значение «Разработка мобильных приложений» и выводит результат в консоль.

var

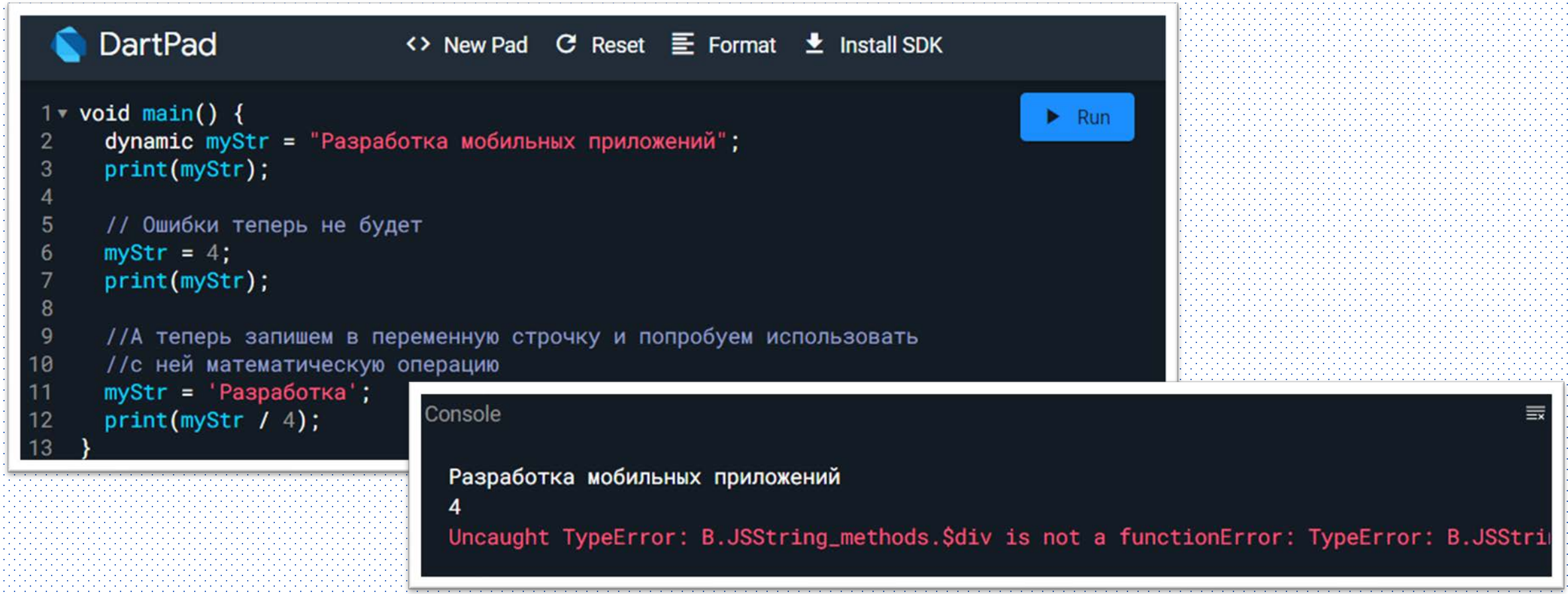
- Dart поддерживает вывод типов (type-inference), так что указывать тип переменной сразу необязательно. На помощь приходит ключевое слово var:



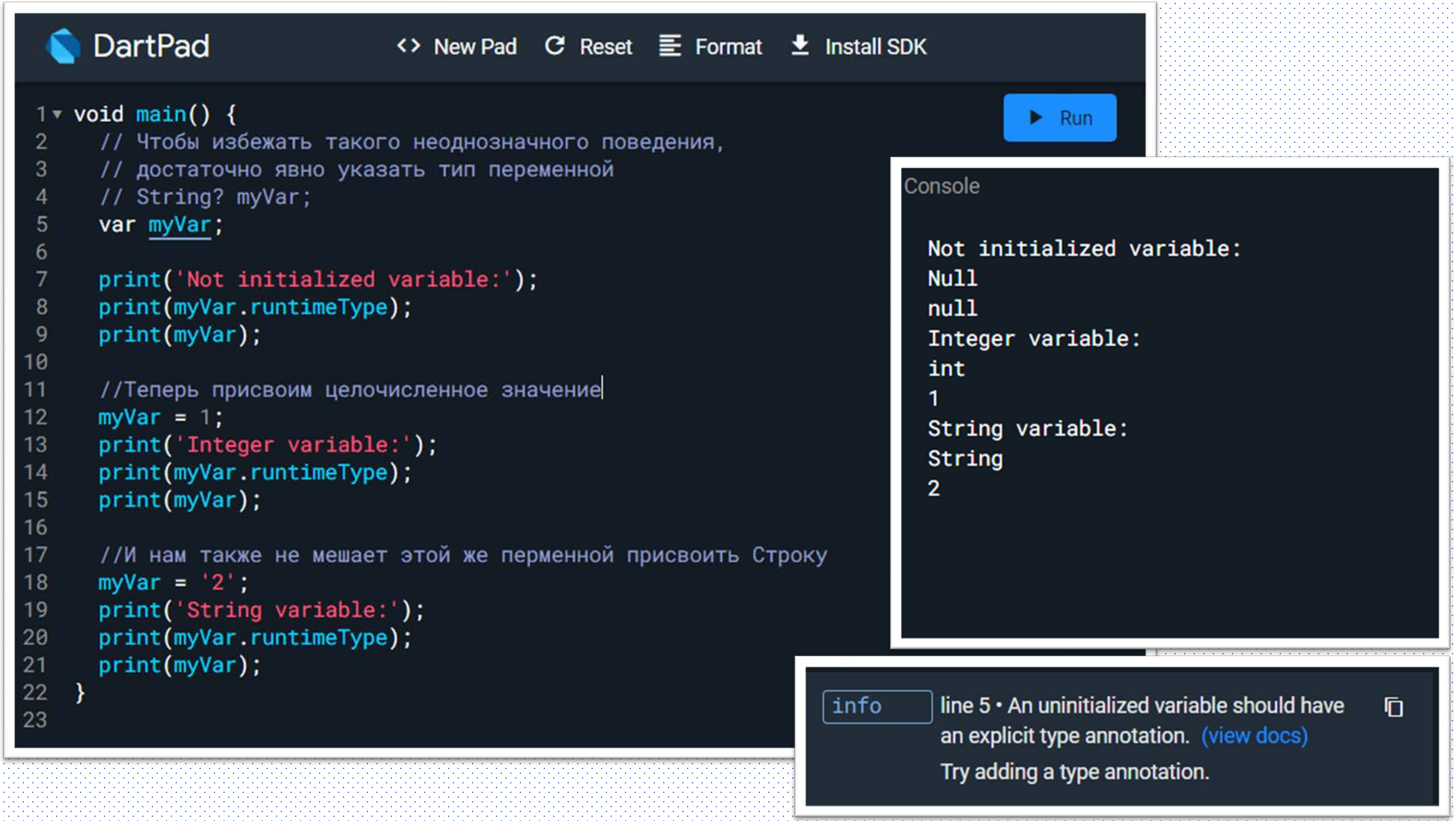


dynamic

- 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:



- 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:

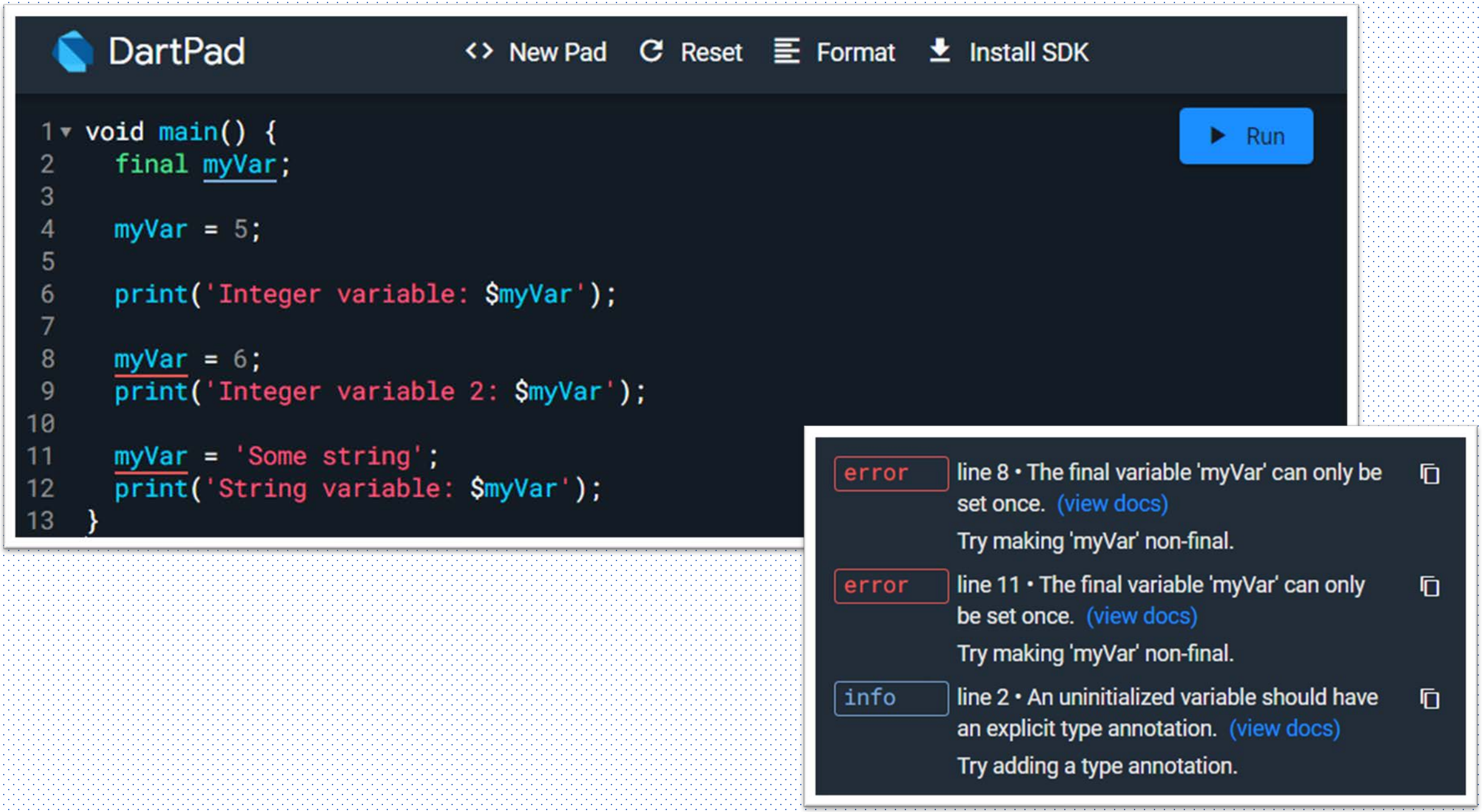

DartPad
New Pad
Reset
Format
Install SDK
Run
void main() {
// Чтобы избежать такого неоднозначного поведения,
// достаточно явно указать тип переменной
// String? myVar;
var myVar;
print('Not initialized variable:');
print(myVar.runtimeType);
print(myVar);
//Теперь присвоим целочисленное значение
myVar = 1;
print('Integer variable:');
print(myVar.runtimeType);
print(myVar);
//И нам также не мешает этой же перменной присвоить Строку
myVar = '2';
print('String variable:');
print(myVar.runtimeType);
print(myVar);
}
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

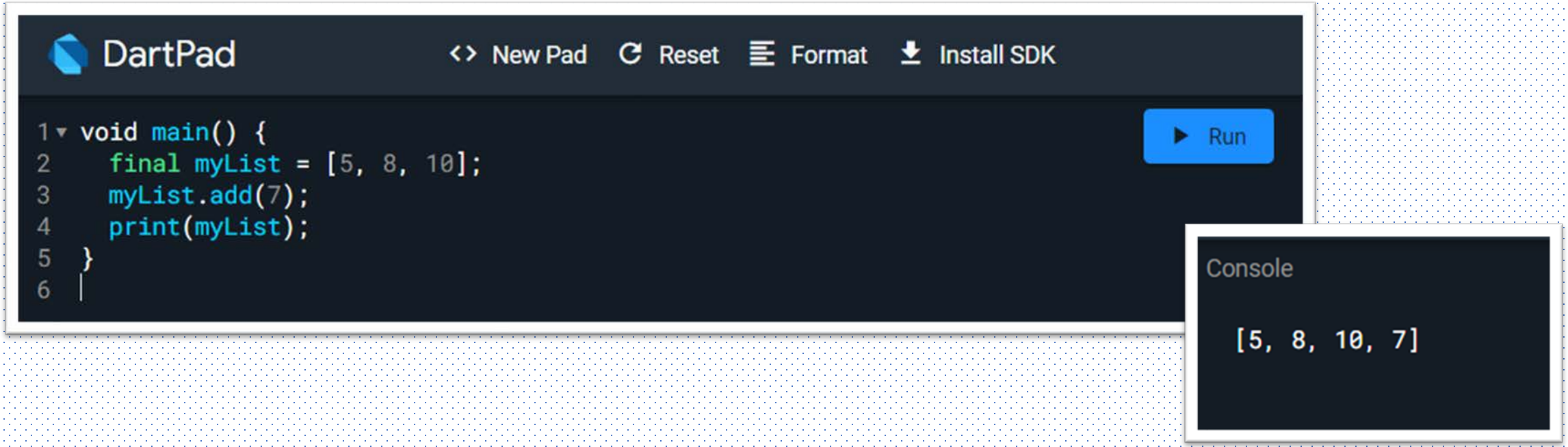
- Во всех примерах выше переменные можно модифицировать и переопределять после объявления. Но чаще всего при разработке появляется потребность запретить изменение переменной после определения. Например, хорошей практикой считается делать свойства модели, и следовательно, саму модель неизменяемыми. А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

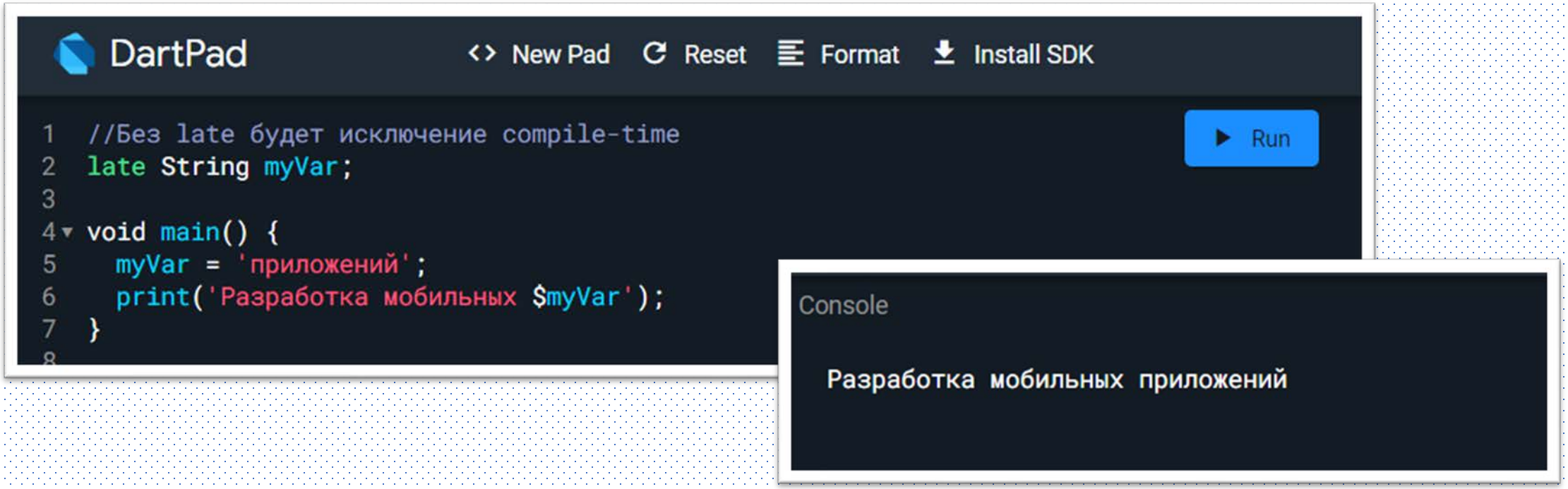

DartPad
New Pad
Reset
Format
Install SDK
Run
1 void main() {
2 final myVar;
3
4 myVar = 5;
5
6 print('Integer variable: $myVar');
7
8 myVar = 6;
9 print('Integer variable 2: $myVar');
10
11 myVar = 'Some string';
12 print('String variable: $myVar');
13 }
error
line 8 • The final variable 'myVar' can 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.



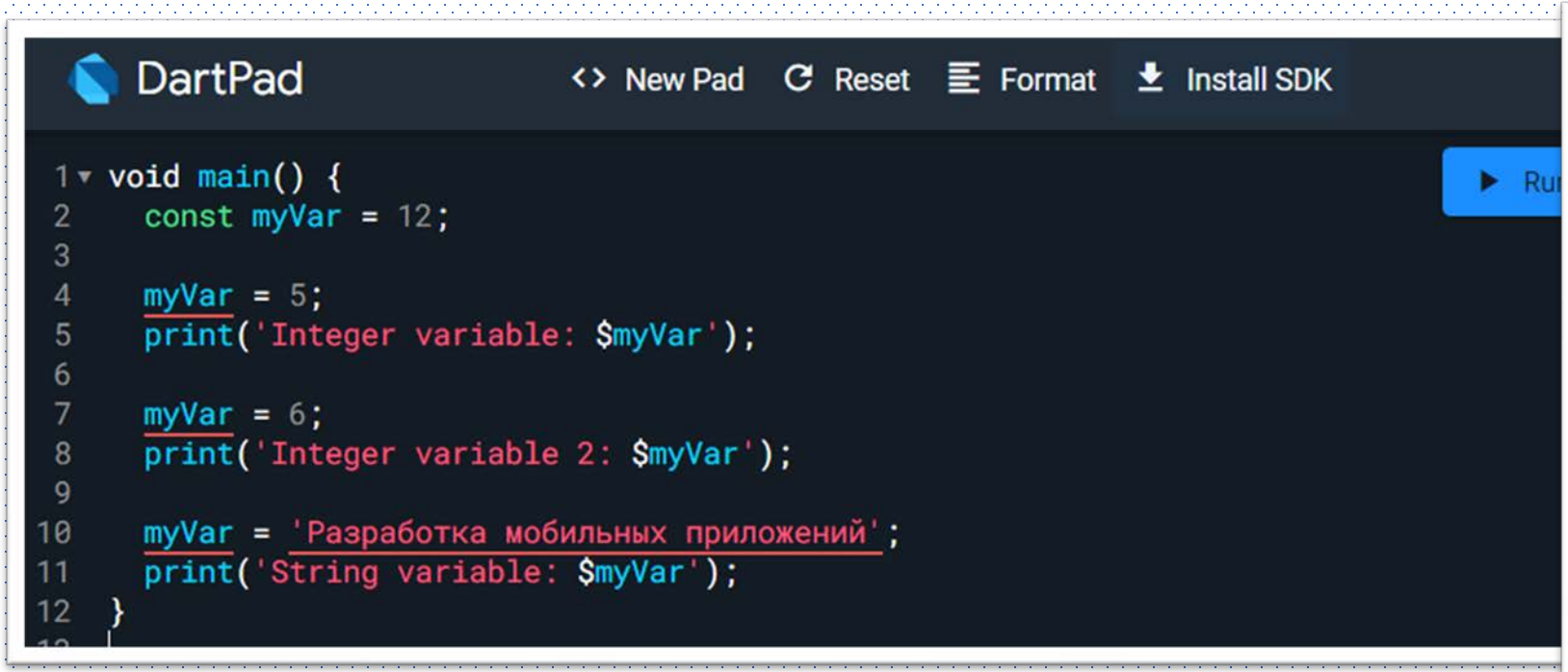
late

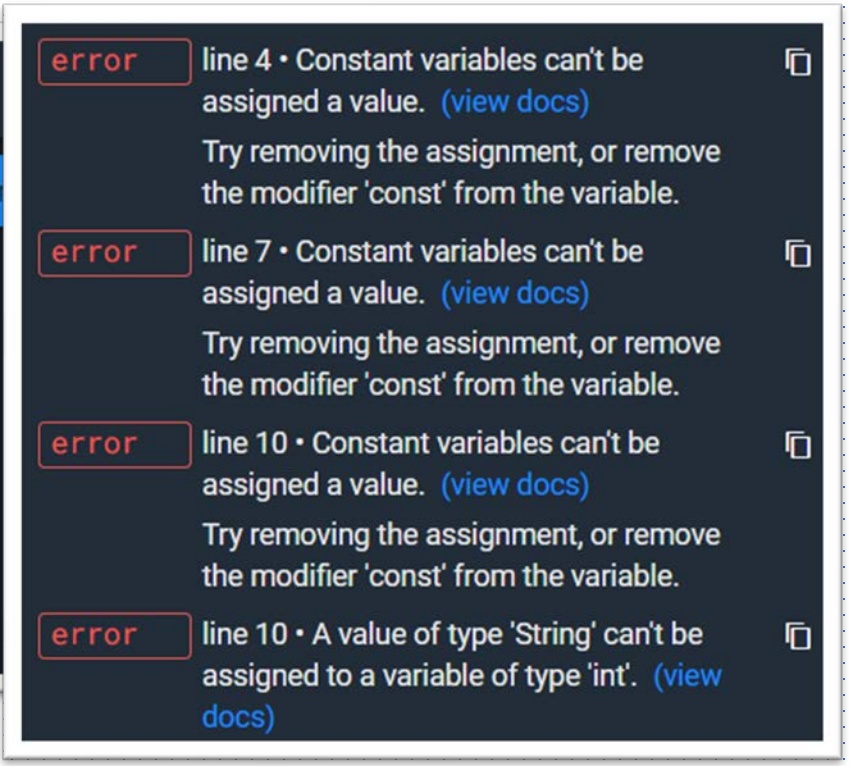
- Ключевое слово late позволяет объявить переменную и не инициализировать её значение сразу.
- Это полезно, когда хотим объявить переменную non-nullable (о null-safety), а значение ей задать позже.



const

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





В итоге получаем сразу несколько ошибок до компиляции:

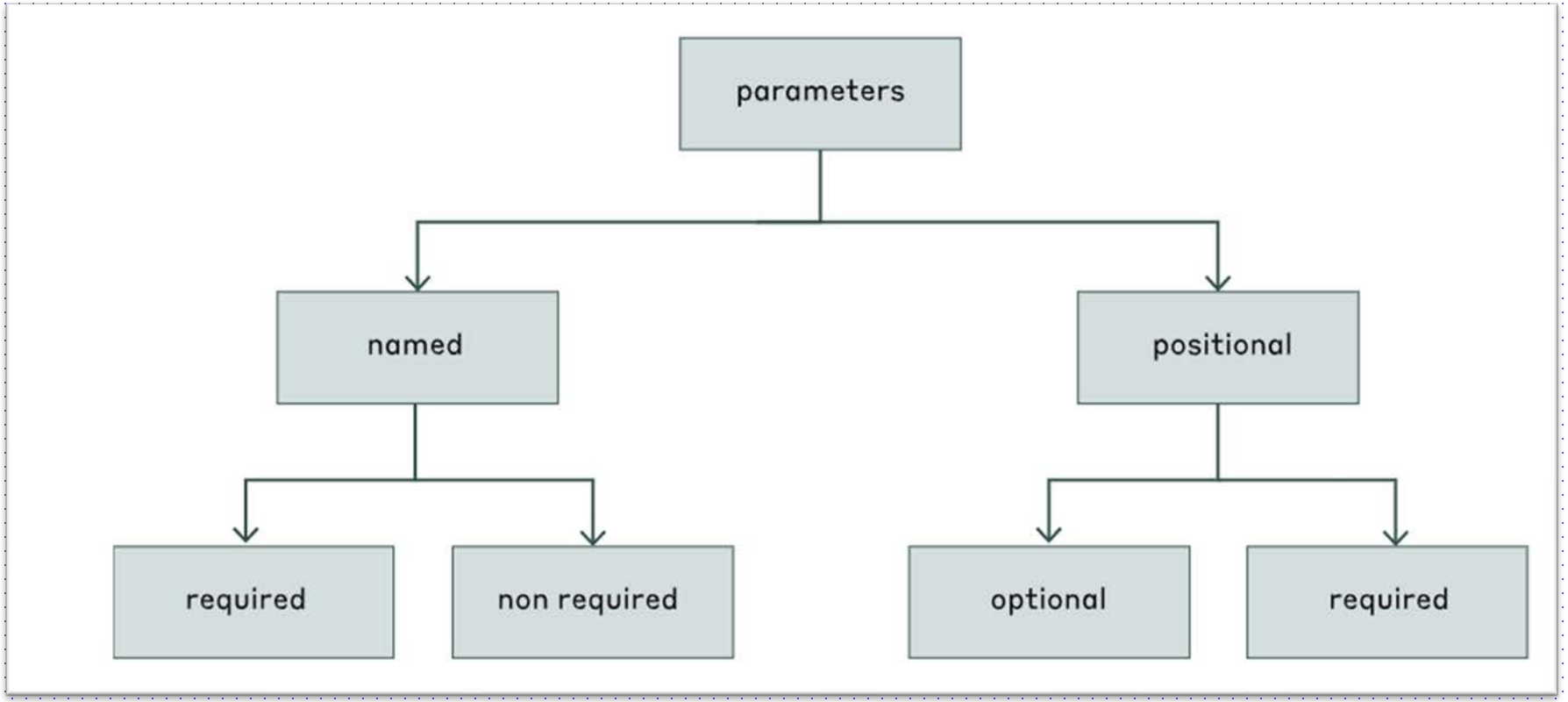
- Переменную const нельзя оставить неинициализированной;
- Переменную const нельзя переопределить;
- Переменной const нельзя изменить тип, что очевидно.

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 New Pad Reset Format Install SDK Run

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

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

DartPad New Pad Reset Format Install SDK Run

```
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 }
```

error line 2 • The parameter 'value1' can't have a value of 'null' because of its type, but the implicit default value is 'null'. (view docs)
Try adding an explicit non-'null' default value.

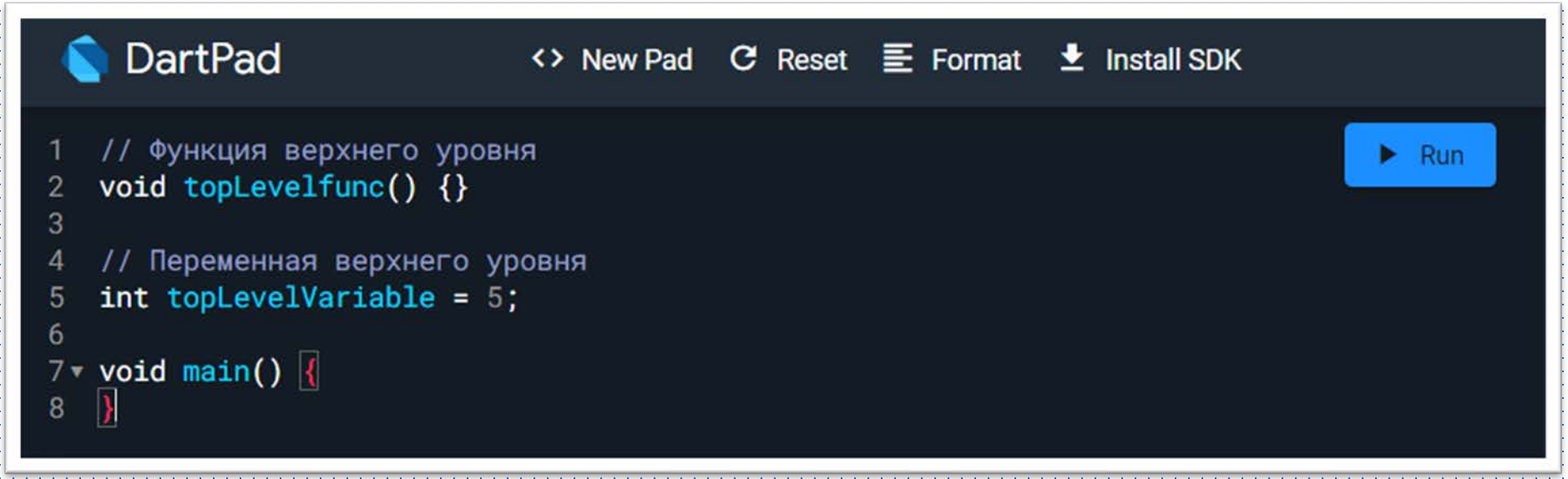
error line 6 • The name 'foo' is already defined. (view docs)
Try renaming one of the declarations.
The first definition of this name.

error line 9 • The name 'foo' is already defined. (view docs)
Try renaming one of the declarations.
The first definition of this name.

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

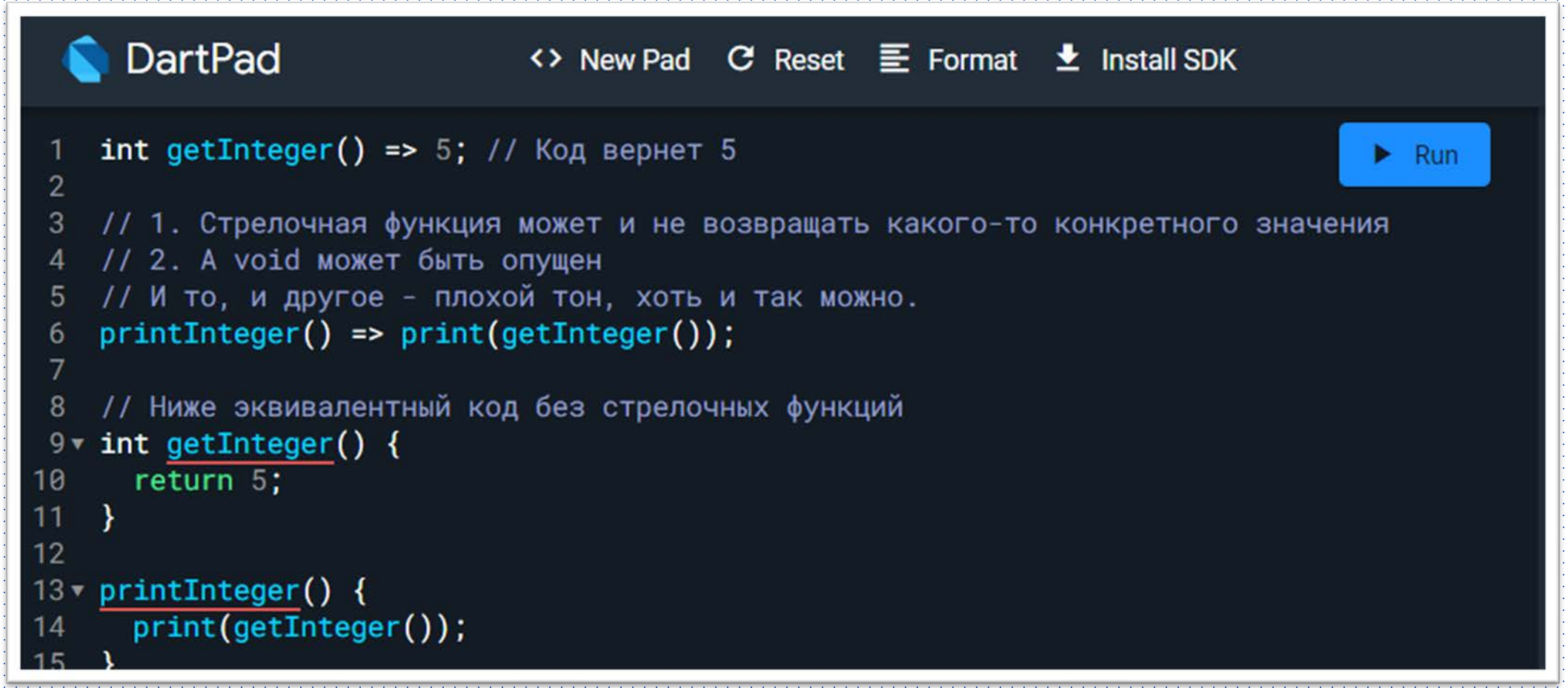
Функции верхнего уровня и переменные

- 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.



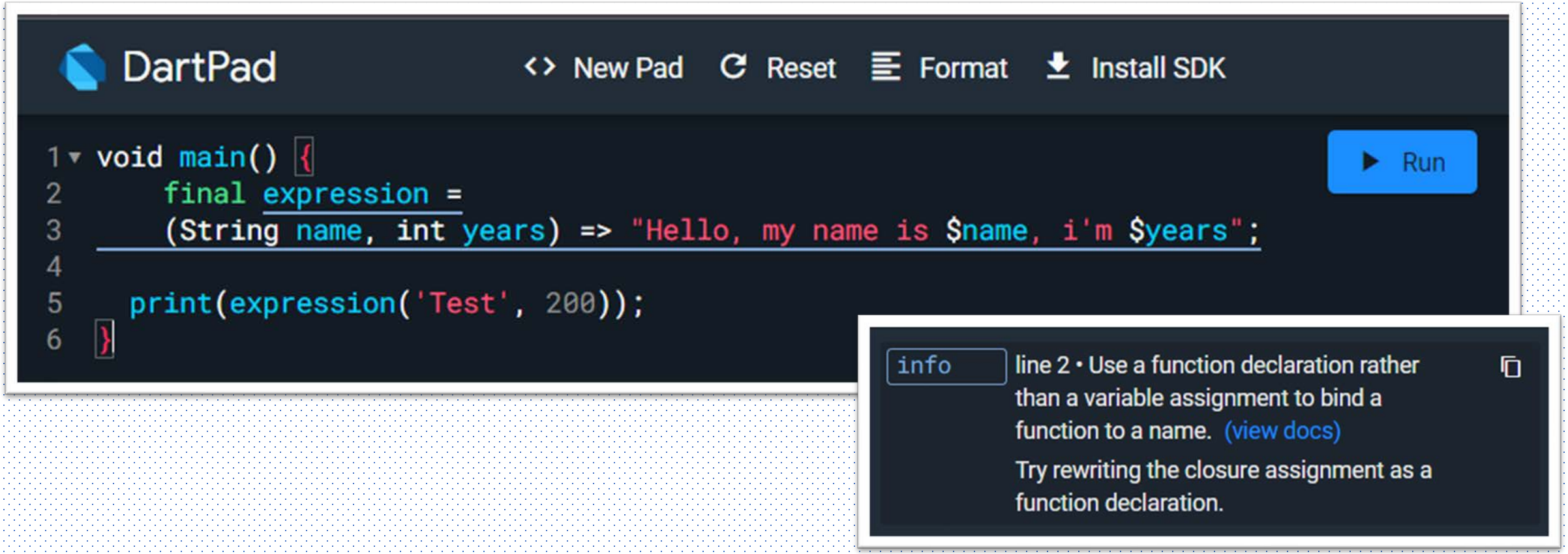
Стрелочные функции

- Стрелочные функции позволяют описывать функции в одну строку:

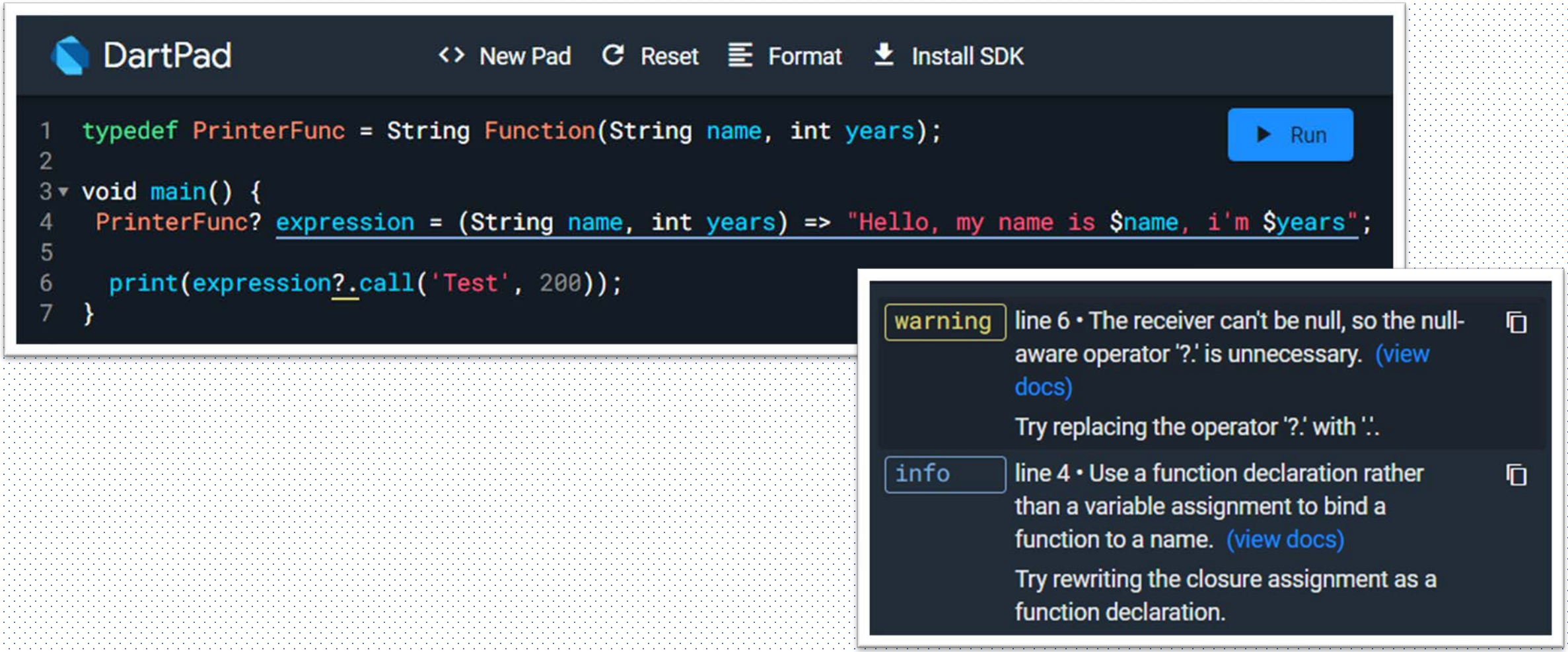


Анонимные функции

- Из функционального подхода в Dart перекочевали анонимные функции — их можно воспринимать как объект типа Function.

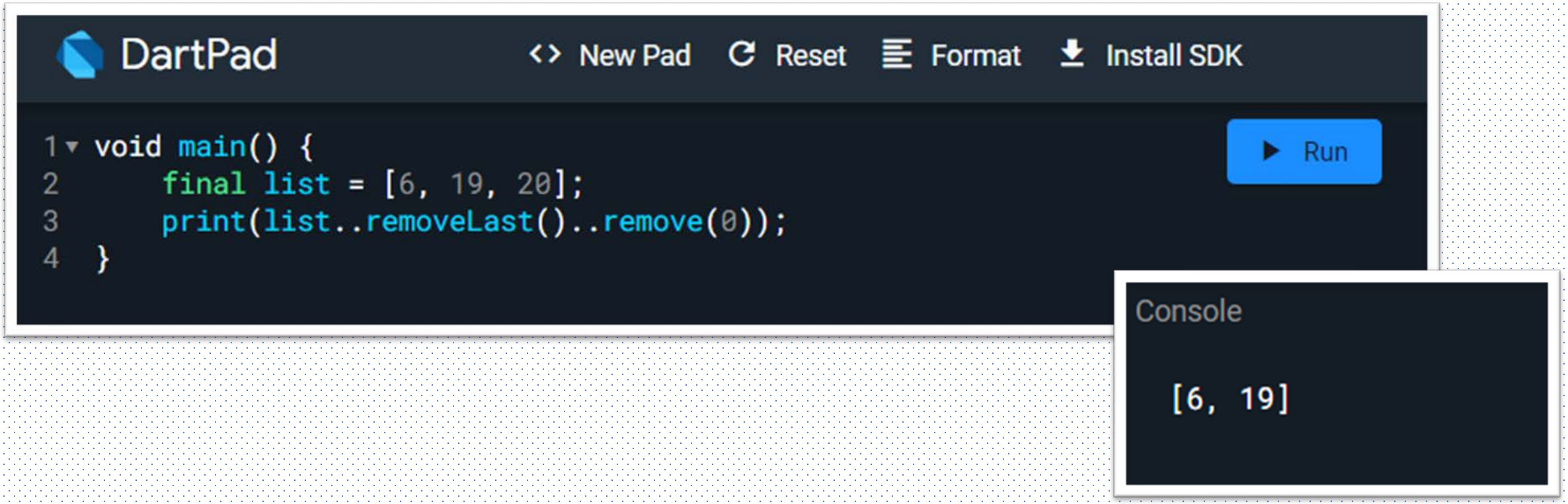


У типа Function есть метод call, он и есть вызов самой функции. Его полезно использовать, если переменная-функция может быть null.



Каскадный вызов методов

- Функция, позволяет вызывать последовательно методы одной переменной:



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.

DartPad

New Pad Reset Format Install SDK

Run

```dart
void main() {

    final String value = 'Разработка мобильных приложений';

    // 'Разработка мобильных приложений' - исходное значение value
    print(value..toUpperCase()..toLowerCase());

    // 'разработка мобильных приложений' - результат toLowerCase()
    print(value.toUpperCase().toLowerCase());

}
```

Console

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

Замыкания

- Этот механизм позволяет анонимной функции, объявленной внутри другой, получать доступ к вышестоящему «контексту».

DartPad <> New Pad | Reset | Format | Install SDK | Run

```dart
int counter = 1;

typedef RobotBuilder = String Function(String name);

RobotBuilder robotFactory(String specifications) {
  final model = 'TEST - $counter';
  counter++;
  return (String name) => '$model, $name, $specifications';
}

void main() {
  final firstRobot = robotFactory('Test 1!');
  final secondRobot = robotFactory('Test 2!');
  final thirdrobot = robotFactory('Test 3!');

  print(firstRobot('1'));
  print(secondRobot.call('2'));
  print(thirdrobot('3'));
}
```

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: