Dart Basics

- HelloWorld in Dart without VS Code
- Create an exe-file from Dart code
- Use command line arguments in Dart
- Built-In types in Dart (int, double, String, bool)
- Nullable types and “sound null safety”
- Variable declaration with var or final
- Functions with positional or named parameters or both
- Functions with the “arrow syntax”

HelloWorld in Dart without VS Code

1) Open a Text Editor (e.g. Notepad) and type

```
void main() {
  print('Hello World!');
}
```

2) Save this in a file hello.dart.

3) Open a Command Prompt in the directory where you saved hello.dart.

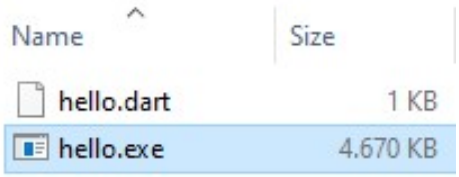
4) Type “dart run hello.dart”:

```
C:\flutter\code\dart_basics>dart run hello.dart
Hello World!
```

Create an exe-file from Dart code

```
C:\flutter\code\dart_basics>dart compile exe hello.dart
Info: Compiling with sound null safety.
Generated: c:\flutter\code\dart_basics\hello.exe
```

This generates a 4 MB executable:



It can be executed on Windows, even if no Dart SDK was installed on that machine:

```
C:\flutter\code\dart_basics>hello.exe
Hello World!
```

Use command line arguments in Dart

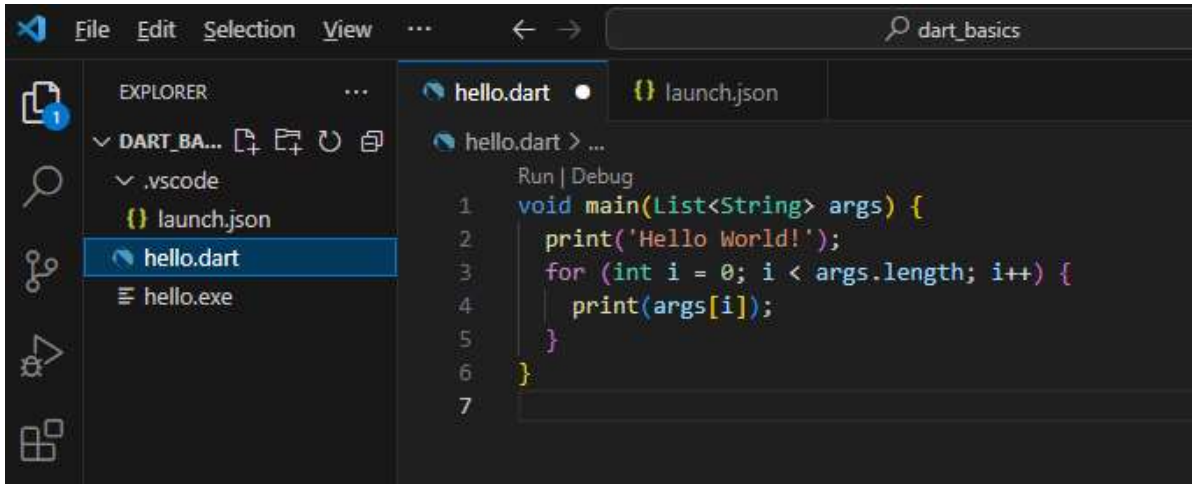
```
void main (List<String> args) {
  print('Hello World!');
  for (int i=0; i<args.length; i++) {
    print(args[i]);
  }
}
```

```
C:\flutter\code\dart_basics>dart run hello.dart a bb ccc 1234
Hello World!
a
bb
ccc
1234
```

Switch from Notepad to VS Code

1) Open the directory with your hello.dart file in VS Code

(either via menu „File/Open Folder…“ in VS Code
or via context menu in Windows Explorer
or by typing “code .” in the Commad Prompt where you executed the last dart commands)

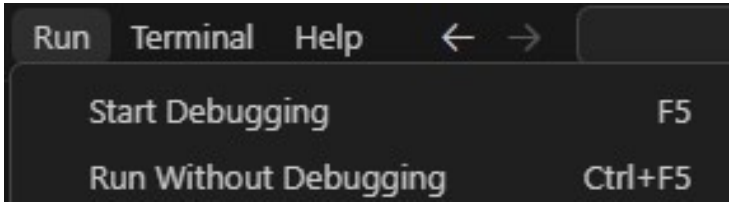


Colors and IntelliSense make life easier

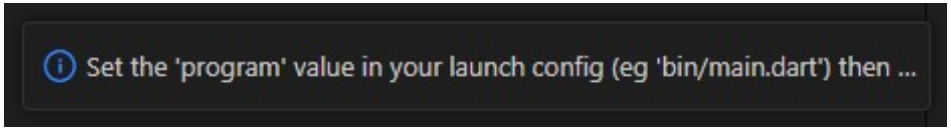
Tip: Shortcut Shift + Alt + F formats the document (see context menu of the editor).

Run or Debug in VS Code

When you select in VS Code one of the menu entries

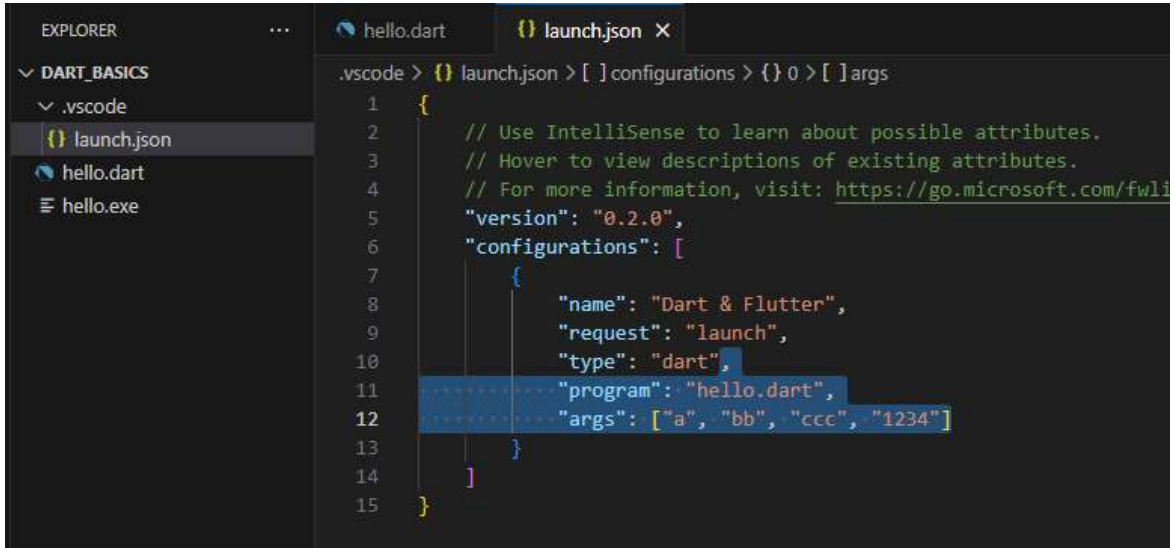


Then VS Code comes up with the message

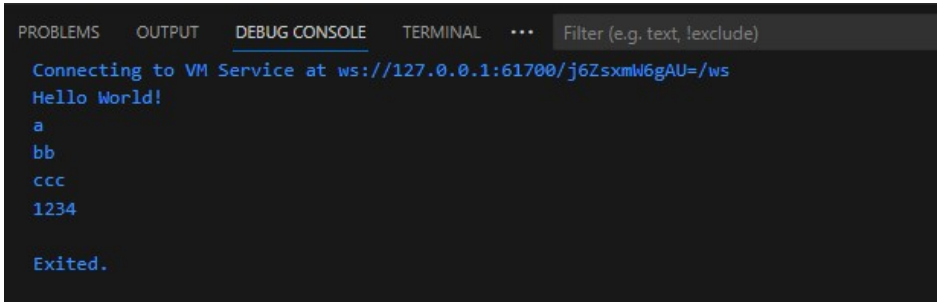


and it creates a new folder .vscode and therein a file “launch.json”:

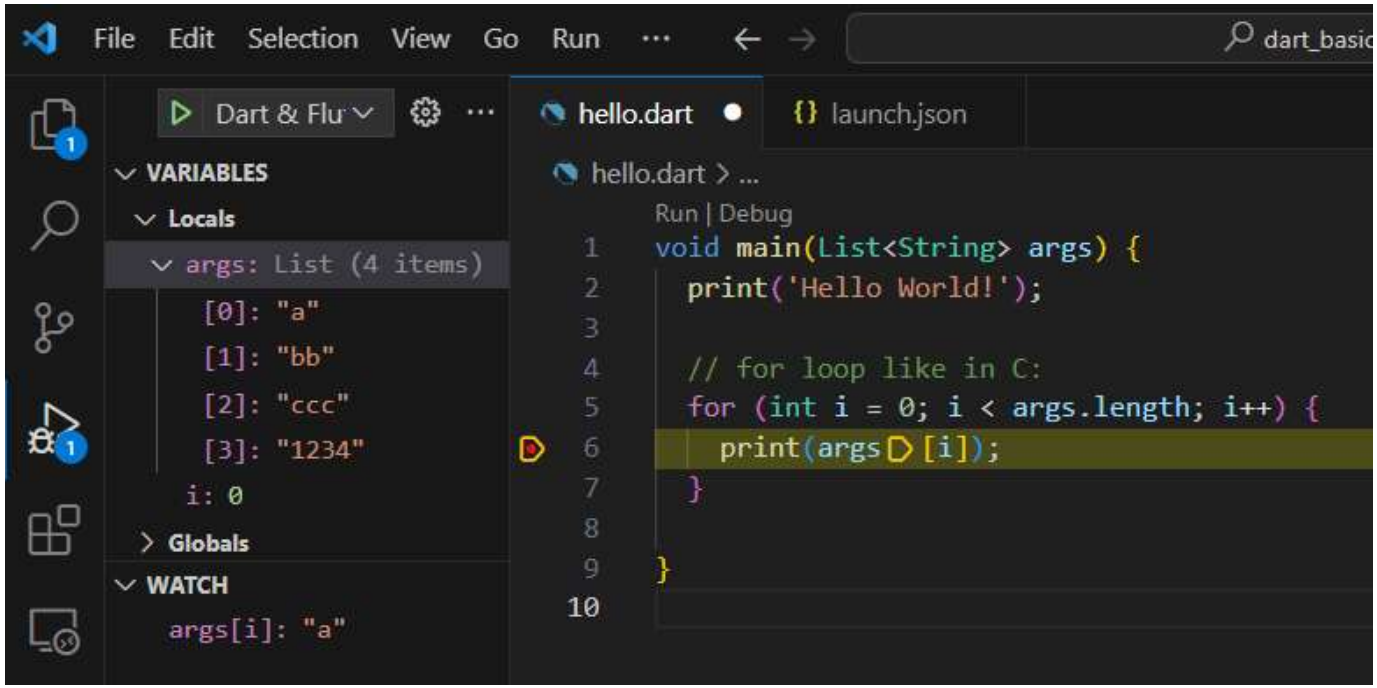
Run or Debug in VS Code (continued)



Add the blue marked text as shown above, save “launch.json”, open “hello.dart” and press F5.



Debugging in VS Code



Shortcuts in VS Code:

F5: Start Debugging
F9: Toggle Breakpoint
F10: Step over

Built-in Types

Built-in types

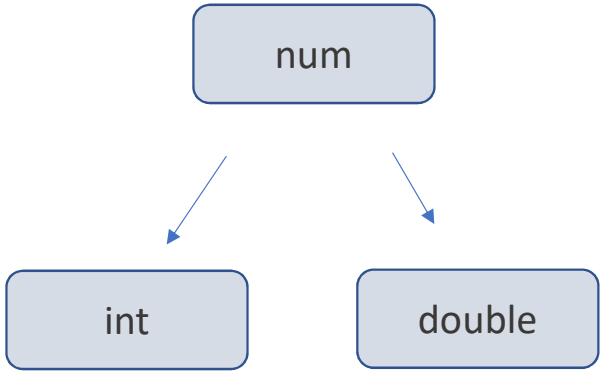
The Dart language has special support for the following:

- Numbers (`int`, `double`)
- Strings (`String`)
- Booleans (`bool`)
- Records (`(value1, value2)`)
- Lists (`List`, also known as *arrays*)
- Sets (`Set`)
- Maps (`Map`)
- Runes (`Runes`; often replaced by the `characters` API)
- Symbols (`Symbol`)
- The value `null` (`Null`)

This support includes the ability to create objects using literals. For example, `'this is a string'` is a string literal, and `true` is a boolean literal.

For some more info see https://www.flutter.de/artikel/dart-basics-datentypen
BTW: Günther has till now nearly no experience with Records, Sets, Runes and Symbols !

Numbers (int and double)




```dart
/// Adds [other] to this number.
///
/// The result is an [int], as described by [int.+],
/// if both this number and [other] is an integer,
/// otherwise the result is a [double].
num operator +(num other);
```

```dart
int i = 23;
double d = 24;

double d1 = 2.34;
int i1 = 2.34;      // double cannot be assigned to int

d = i;              // int cannot be assigned to double
d = i.toDouble();

i = d;              // double cannot be assigned to int
i = d.toInt();      // toInt() truncates the decimal places

d = i + i;          // int cannot be assigned to double
d = i + d;
i = i + d;          // double cannot be assigned to int
```

Strings

```dart
String s = 'hello world';

s = "It's me"; // you can either use " or ' to surround strings
s = 'It\'s me'; // backslash is the "escape character" in strings
s = 'c:\\flutter\\sdk'; // \\ stands for \ inside the string
s = '1st line\n2nd line'; // \n is new line for multi-line strings

s = "hello";
s = s + " world"; // "hello world"
s += "!"; // "hello world!""

double d = 1.234567;
s = d;                  // double cannot be assigned to string
s = d.toString();

s = "d is " + d.toString(); // "d is 1.234567"
s = "d is " + d.toStringAsFixed(2); // "d is 1.23"

num n = 3;
s = "n is " + n.toStringAsFixed(2); // "n is 3.00"
```

String Interpolation

You can access the value of an expression inside a string by using `${expression}`.

```
String greeting = "Hello";
String person = "Fritz";

print("${greeting} ${person} !");   // prints Hello Fritz !
```

If the expression is an identifier, the `{}` can be skipped.

```
print("$greeting $person !");
```

If the variable inside the `{}` isn't a string, the variable's `toString()` method is called:

```
s = "d is $d !";                    // "d is 1.234567 !"
s = "d + 1 is ${d + 1} !";          // "d + 1 is 2.234567 !"
s = "d is ${d.toStringAsFixed(1)}"; // "d is 1.2"
```

The text above was copied from https://shailen.github.io/blog/2012/11/14/dart-string-interpolation/

Number systems and shift operation for int

```
i = 30;
print(i.toRadixString(16));
print(i.toRadixString(8));
print(i.toRadixString(7));
print(i.toRadixString(2));
i = i >> 1;
// next line makes the same as last line:
// i >>= 1;
print(i.toRadixString(2));
print(i);
i = i << 3;
print(i.toRadixString(2));
print(i);
```

```
1e
36
42
11110
```

```
1111
15
```

```
1111000
120
```

Using “hex” in Assembler:

```
R0194   6A90    STH     (R9,LEIN)
        0570R
R0198   20C0    LIH     (R12,LAKTRT)
        02CER
R019C   21C0    AIH     (R12,32)
        0020
R01A0   28F0    BU      (15,BHIASC)
        000OF
R01A4   0004    ADT     (4)
R01A6   056AR   ADT     (LPARFL)
R01A8   7A10    BL      (FEHWAN)
        0098R
```

What is defined where ?

Object o;

o.toString();
int i = o.hashCode;
Type t =
o.runtimeType;

num n;

n.toInt();
n.ToDouble();
n.toStringAsFixed(2);
int i = n.ceil();
int i = n.floor();

int i;

i.toRadixString(2);
i.isEven;
i.isOdd;

Parse strings for numeric values

```
print(int.tryParse("2"));
print(int.tryParse("a"));
```

```
2
null
```

```
int parsed = int.tryParse("2");
```

A value of type 'int?' can't be assigned to a variable of type 'int'.

```
int? parsed = int.tryParse("2");
print(parsed.isEven);
```

The property 'isEven' can't be unconditionally accessed because the receiver can be 'null'.

```
int? parsed = int.tryParse("2");
if (parsed != null) {
  print(parsed.isEven);
}
```

Nullable Types in Dart

Since version 3.0, Dart provides “sound null safety” („solide null Sicherheit”).

It should avoid null pointer exceptions often seen in Java or C++, e.g. in

Temp.java

```
1  public class Temp {
2
3      public static void main(String[] args) {
4
5          foo(null);
6
7      }
8
9      public static void foo(String s) {
10         System.out.println(s.toLowerCase());
11     }
12 }
13
```

Problems Javadoc Declaration Console

```
<terminated> Temp [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_
Exception in thread "main" java.lang.NullPointerException
        at Temp.foo(Temp.java:10)
        at Temp.main(Temp.java:5)
```

nullable.dart > ...

Run | Debug

```
1  void main(List<String> args) {
2    foo1(null);
3  }
4
5  void foo1(String? s) {
6    print(s.toLowerCase());
7  }
8
```

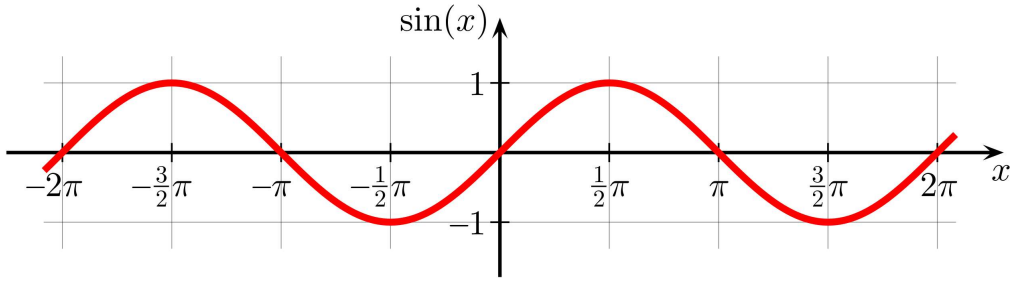
```
The method 'toLowerCase' can't be
unconditionally invoked because the receiver
can be 'null'.
Try making the call conditional (using '?.')
```

Operator ?. (conditional access)

In the expression “s?.toLowerCase()” the method toLowerCase is only called when variable s is not null. When s is null, the whole expression is null:

```
Run | Debug
void main(List<String> args) {
  foo2(null);      // prints "null"
  foo2("Hello");   // prints "hello"
}

void foo2(String? s) {
  print(s?.toLowerCase());

  //String lower = s?.toLowerCase();     // compile-error
  String? lower = s?.toLowerCase();
}
```

Math reminder



Kreisfunktionen

φ	x	sin x	tan x	cot x	cos x
171°53′	**3,00**	0,1411	0,1425	7,015	0,9900
172 28	,01	,1312	,1324	7,555	,9914
173 02	,02	,1213	,1222	8,184	,9926
173 36	,03	,1114	,1121	8,924	,9938
174 11	,04	,1014	,1019	9,809	,9948
174°45′	3,05	0,0915	0,0918	10,89	0,9958
175 20	,06	,0815	,0818	12,23	,9967
175 54	,07	,0715	,0717	13,94	,9974
176 28	,08	,0616	,0617	16,22	,9981
177 03	,09	,0516	,0516	19,37	,9987
177°37′	**3,10**	0,0416	0,0416	24,03	0,9991
178 11	,11	,0316	,0316	31,64	,9995
178 46	,12	,0216	,0216	46,30	,9998
179 20	,13	,0116	,0116	86,26	0,9999
179 55	,14	0,0016	0,0016	627,9	1,0000
180°29′	3,15	0,0084	0,0084	118,9	1,0000
181 03	,16	,0184	,0184	54,32	0,9998
181 38	,17	,0284	,0284	35,19	,9996
182 12	,18	,0384	,0384	26,02	,9993
182 46	,19	,0484	,0484	20,64	,9988
183°21′	**3,20**	0,0584	0,0585	17,10	0,9983
183 55	,21	,0684	,0685	14,60	,9977
184 30	,22	,0783	,0786	12,73	,9969
185 04	,23	,0883	,0886	11,28	,9961
185 38	,24	,0982	,0987	10,13	,9952

Copied from https://upload.wikimedia.org/wikipedia/commons/thumb/a/a2/Sine.svg/2560px-Sine.svg.png

Library math.dart

This library provides trigonometric, exponential, logarithmic and other functions.

To use them you need to import “math.dart”:

math_functions.dart > ...

```dart
import 'dart:math';

Run | Debug
void main() {
  double d = sin(3.14);
  print(d);
}
```

Context-Menu “Go to definition (F12)” for sin shows:

C: > FlutterSDK > flutter_windows_3.3.10-stable > flutter > bin > cache > dart-sdk > lib > math > math.dart > tan

```dart
194
195  /// Converts [radians] to a [double] and returns the sine of the value.
196  ///
197  /// If [radians] is not a finite number, the result is NaN.
198  external double sin(num radians);
```

```dart
void main() {
  double d = sin(3.14);
  print(d);
  print(d.toStringAsFixed(10));
  d = sin(pi);
  print(d);
  print(d.toStringAsFixed(10));
}
```

```
0.0015926529164868282
0.0015926529
1.2246467991473532e-16
0.0000000000
```

Definition von pi in math.dart:

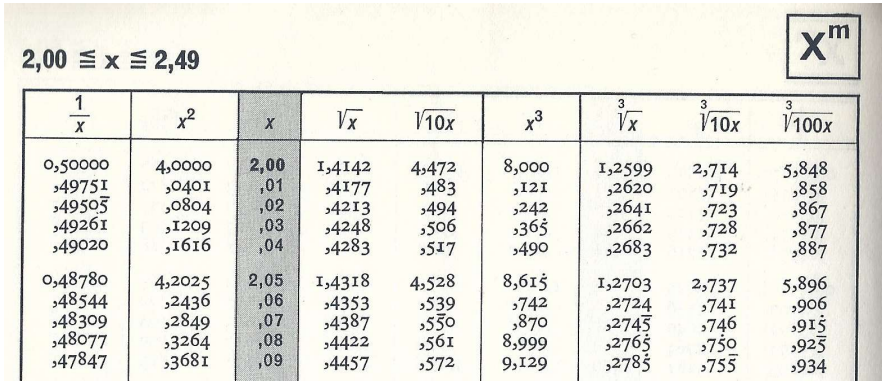
```dart
/// The PI constant.
const double pi = 3.1415926535897932;
```

Exercise

Use dart to create a file “wurzel.exe”.

When this file is executed in a Windows command prompt,
it calculates and prints for each of the given command arguments the square root.

Test wurzel.exe also for negative numbers as arguments and
for arguments that are no numbers.



2,00 ≦ x ≦ 2,49 x^m

$\frac{1}{x}$	x^2	x	$\sqrt{x}$	$\sqrt{10x}$	x^3	$\sqrt[3]{x}$	$\sqrt[3]{10x}$	$\sqrt[3]{100x}$
0,50000	4,0000	**2,00**	1,4142	4,472	8,000	1,2599	2,714	5,848
,49751	,0401	,01	,4177	,483	,121	,2620	,719	,858
,49505	,0804	,02	,4213	,494	,242	,2641	,723	,867
,49261	,1209	,03	,4248	,506	,365	,2662	,728	,877
,49020	,1616	,04	,4283	,517	,490	,2683	,732	,887
0,48780	4,2025	2,05	1,4318	4,528	8,615	1,2703	2,737	5,896
,48544	,2436	,06	,4353	,539	,742	,2724	,741	,906
,48309	,2849	,07	,4387	,550	,870	,2745	,746	,915
,48077	,3264	,08	,4422	,561	8,999	,2765	,750	,925
,47847	,3681	,09	,4457	,572	9,129	,2785	,755	,934

Possible solution

wurzel.dart > main

```dart
import 'dart:math';

Run | Debug
void main(List<String> args) {
  for (int i = 0; i < args.length; i++) {
    String arg = args[i];
    double? d = double.tryParse(arg);
    if (d == null) {
      print("argument '$arg' is not a number.");
    } else if (d < 0) {
      print("cannot calculate square root from negative number '$arg'.");
    } else {
      String rounded = sqrt(d).toStringAsFixed(4);
      print("square root of '$arg' rounded to 4 decimal places is '$rounded'.");
      // instead of the last 2 lines you could also write the following quite long line
      //print("square root of $arg rounded to 4 decimal places is '${sqrt(d).toStringAsFixed(4)}'.");
    }
  }
}
```

```
C:\flutter\code\dart_basics>dart compile exe wurzel.dart
Generated: c:\flutter\code\dart_basics\wurzel.exe

C:\flutter\code\dart_basics>wurzel 4 10 -2 x
square root of '4' rounded to 4 decimal places is '2.0000'.
square root of '10' rounded to 4 decimal places is '3.1623'.
cannot calculate square root from negative number '-2'.
argument 'x' is not a number.
```

Booleans

```dart
bool b = false;
print("not b is ${!b}"); // "not b is true"

int i = 5;
b = (i > -1);
b = (i > -1) && (i < 1);
b = (i <= -1) || (i >= 1);
```

```dart
String result;
if (i.isEven) {
  result = "gerade";
} else {
  result = "ungerade";
}
print (result);
```

shorter with
conditional expression
(also called “ternary operator”
or “conditional expression”)


```dart
print (i.isEven ? "gerade" : "ungerade");
```

Variable declaration with var

```
int x = 1;
double y = 1.23;
List<String> names = ["Franz", "Frank"];
```

Instead we can write:

```
var x = 1;
var y = 1.23;
var names = ["Franz", "Frank"];
```

VS Code shows the type:

```
List<String> names
var
Type: List<String>
var
var names = ["Franz", "Frank"];
```

Object has getter “runtimeType” (see slide 14):

```
print(x.runtimeType );
print(y.runtimeType);
print(names.runtimeType);
```

```
int
double
List<String>
```

Final variables

Variables declared as “final” cannot be modified later in code:

```dart
final int myInt = 1;
final List<String> myList = ["a", "bb"];
```

```dart
myInt = 5;
```



```
The final variable 'myInt' can only be set once.
Try making 'myInt' non-final. dart(assignment_to_final_local)
```

```dart
myList = ["x", "yy"];  // error: The final variable 'myList' can only be set once.
myList.add("ccc");     // that's ok, we do not assign a new list
```

“final” can be used similar to “var”:

```dart
final myInt = 1;
final myDouble = 1.5;
final myList = ["a", "bb"];

print(myInt.runtimeType);
print(myDouble.runtimeType);
print(myList.runtimeType);
```

```
int
double
List<String>
```

final vs. const

"const" variables must be directly initialized:

```
const int cInt;         // error: The constant 'cInst' must be initialized.

const int cInt1 = 23;
```

"final" variables can be initialized later in code:

```
final int fInt;

fInt = 23;
```

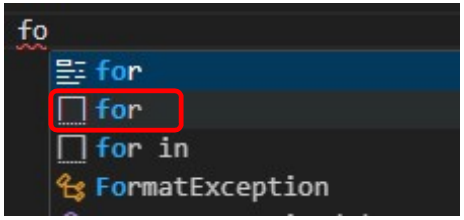
Both "const" and "final" variables cannot be modified after their initialization:

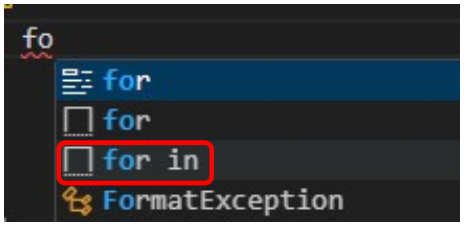
```
cInt1 = 1;
```

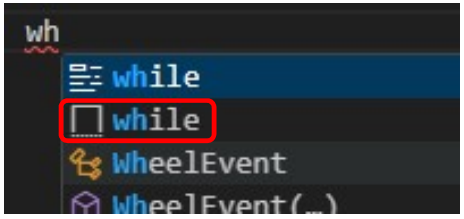
```
Constant variables can't be assigned a value.
Try removing the assignment, or remove the modifier 'const' from the
variable. dart(assignment_to_const)
```

IntelliSense can reduce your typing for loops

IntelliSense:







Generated code:

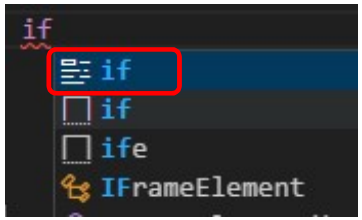
```
for (var i = 0; i < count; i++) {

}
```

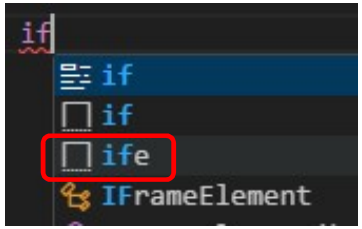
```
for (var element in collection) {

}
```

```
while (condition) {

}
```

IntelliSense for “if” and “if-else”

IntelliSense:





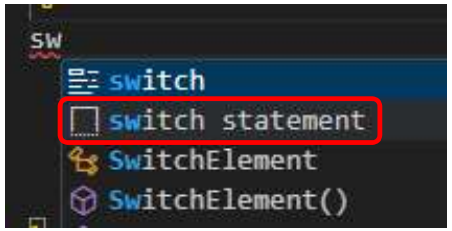
Generated code:

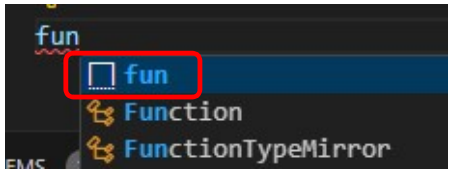
```
if (condition) {

}
```

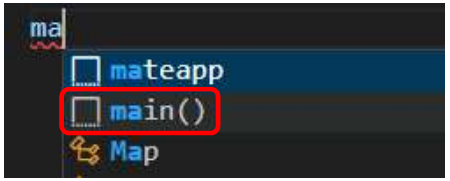
```
if (condition) {

} else {

}
```

More IntelliSense samples

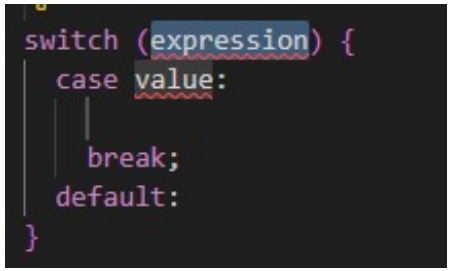
IntelliSense:

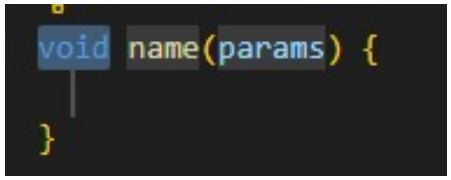


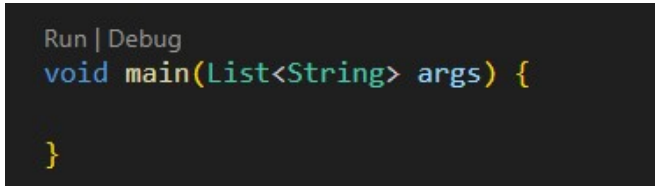




Generated code:







Functions with positional parameters

```dart
void testPositionalParams() {
  usePositionalParams(1, 1.5, "hello");
  //usePositionalParams(1, 0.5); // 3 positional arguments expected by 'usePositionalParams', but 2 found.

  useOptionalPositionalParam(2, 2.5);
  useOptionalPositionalParam(2, 2.5, "hi");

  useNullableOptionalPositionalParam(3, 3.3);
  useNullableOptionalPositionalParam(3, 3.3, "servus");
}

void usePositionalParams(int i, double d, String s) {
  print("usePositionalParams: i: $i, d: $d, s: $s");
}

void useOptionalPositionalParam(int i, double d, [String s = "abc"]) {
  print("useOptionalPositionalParam: i: $i, d: $d, s: $s");
}

void useNullableOptionalPositionalParam(int i, double d, [String? s]) {
  print("useNullableOptionalPositionalParam: i: $i, d: $d, s: $s");
}
```

Output:

```
usePositionalParams: i: 1, d: 1.5, s: hello
useOptionalPositionalParam: i: 2, d: 2.5, s: abc
useOptionalPositionalParam: i: 2, d: 2.5, s: hi
useNullableOptionalPositionalParam: i: 3, d: 3.3, s: null
useNullableOptionalPositionalParam: i: 3, d: 3.3, s: servus
```

Functions with named parameters

```dart
void testNamedParams() {
  //useNamedParameters();  // error: The named parameter 'i' is required, but there's no corresponding argument.
   useNamedParams(i: 3);
   useNamedParams(d: 2.5, i: 5);
   useNamedParams(i: -1, d: 5, s: "hi");
}

void useNamedParams({required int i, double? d, String s = 'test'}) {
  print("useNamedParams: i: $i, d: $d, s: $s");
  if (d != null && d < 4) {
    print('BTW: d was less than 4');
  }
}
```

Output:

```
useNamedParams: i: 3, d: null, s: test
useNamedParams: i: 5, d: 2.5, s: test
BTW: d was less than 4
useNamedParams: i: -1, d: 5.0, s: hi
```

Sample in Flutter:

```dart
Text('Hello World!',
    style: TextStyle(
        color: Colors.red,
        fontSize: 24,
        fontStyle: FontStyle.italic)),
```

```dart
(new) TextStyle TextStyle({
  bool inherit = true,
  Color? color,
  Color? backgroundColor,
  double? fontSize,
  FontWeight? fontWeight,
  FontStyle? fontStyle,
  double? letterSpacing,
  double? wordSpacing,
```

Functions with positional and named parameters

```dart
void testPositionalAndNamedParams() {
  //usePositionalAndNamedParams(i: 1, d: 0.5);  // error: The named parameter 'i' isn't defined.
  //usePositionalAndNamedParams(d: 0.5);        // error: 1 positional argument expected, but 0 found.
  usePositionalAndNamedParams(3, d: 3.3);
  usePositionalAndNamedParams(10, s: "hello", d: 9);
}

void usePositionalAndNamedParams(int i, {required double d, String s = "abc"}) {
  print ("usePositionalAndNamedParams: i: $i, d: $d, s: $s");
}
```

Output:

```
usePositionalAndNamedParams: i: 3, d: 3.3, s: abc
usePositionalAndNamedParams: i: 10, d: 9.0, s: hello
```

Was used e.g. in

```dart
Image.asset("assets/images/snoopy_laptop.jpg",
  width: 140), // Image.asset
```

Functions with the “arrow syntax”

Here’s an example of implementing a function:

```
bool isNoble(int atomicNumber) {
  return _nobleGases[atomicNumber] != null;
}
```

For functions that contain just one expression, you can use a shorthand syntax:

```
bool isNoble(int atomicNumber) => _nobleGases[atomicNumber] != null;
```

The `=> expr` syntax is a shorthand for `{ return expr; }`. The `=>` notation is sometimes referred to as *arrow* syntax.

Note: Only an *expression*—not a *statement*—can appear between the arrow (=>) and the semicolon (;). For example, you can’t put an if statement there, but you can use a conditional expression.

for “conditional expression” see Slide 22.

Text was copied from https://dart.dev/language/functions

Small sample for using the “arrow syntax”

```dart
void main() {
  var weekDay = getWorkDayOfDate(1950, 12, 20);
  print(weekDay);
}

int getWorkDayOfDate(int year, int month, int day) {
  return DateTime(year, month, day).weekday;
}
```

```dart
int getWorkDayOfDate(int year, int month, int day) =>
    DateTime(year, month, day).weekday;
```

Documentation for .weekday

```dart
int get weekday
```

dart:core

The day of the week [monday]..[sunday].

In accordance with ISO 8601 a week starts with Monday, which has the value 1.

```dart
final moonLanding = DateTime.parse('1969-07-20 20:18:04Z');
print(moonLanding.weekday); // 7
assert(moonLanding.weekday == DateTime.sunday);
```

Advanced Exercise

Create your own function to calculate the square root of a double.

Idea: use interval nesting

Example for sqrt(5) :

1st interval: a = 1, e = 5; in the middle is 3; 3*3 = 9; too big; use a=1, e=3;

2nd interval: a = 1, e = 3; in the middle is 2; 2*2 = 4; too small; use a=2, e=3;

3rd interval: a = 2, e = 3; in the middle is 2.5; 2.5*2.5 = 6.25; too big; use a=2, e=2,5

4th interval: a = 2, e = 2.5; in the middle is 2.25; 2.25*2.25 = 5.0625; too big; use a=2, e=2.25

5th interval: a = 2, e = 2.25; in the middle is 2.125; 2.125 * 2.125 = 4.515625; too small; use a=2.125, e=2.25

…