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
- Use mathematical functions from library math.dart
- Functions with positional or named parameters or both

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:

Name	Size
hello.dart	1 KB
hello.exe	4.670 KB

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

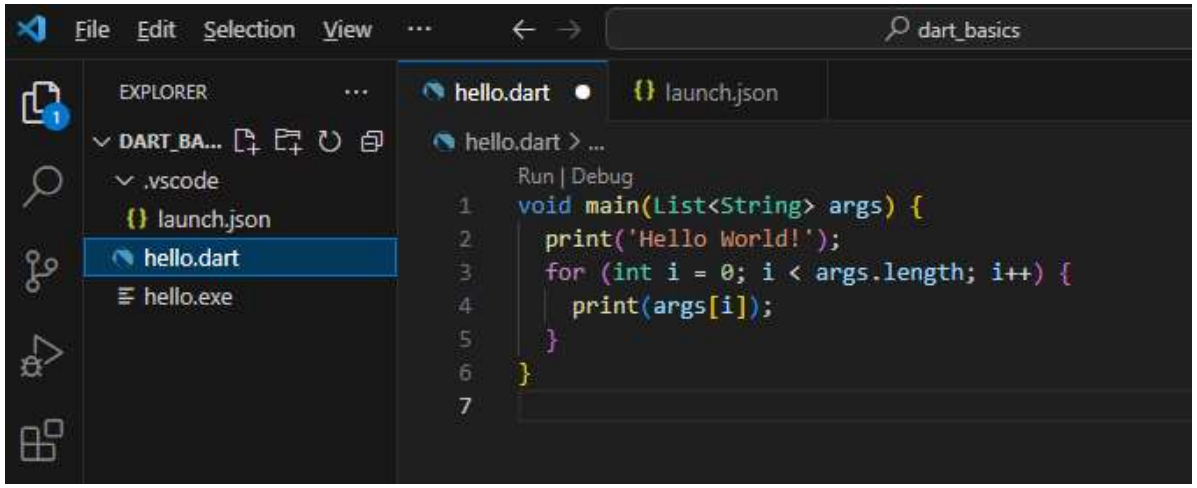
```
void main (List<String> args) {
  print('Hello World!');
  for (int i=0; i<args.length; i++) {
    print(args[i]);
  }
}
```

```
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 Command Prompt where you executed the last dart commands)

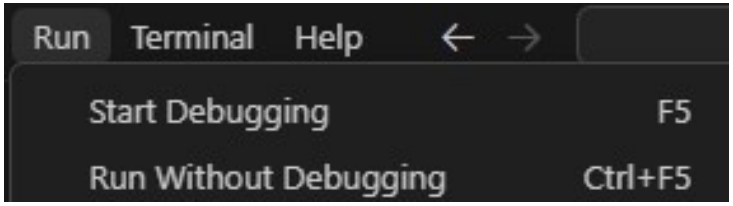


Colors and IntelliSense make life easier

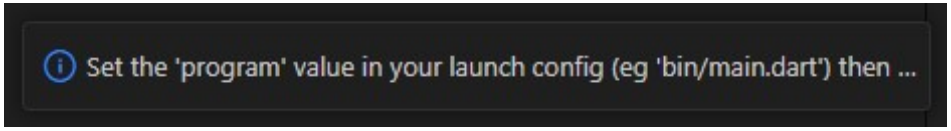
Tip: Shortcut Shift + Alt + F formats the document (see context menu of the editor).

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)

EXPLORER

DART_BASICS
- .vscode
  - launch.json
- hello.dart
- hello.exe

hello.dart | launch.json

.vscode > launch.json > [ ] configurations > {} 0 > [ ] args

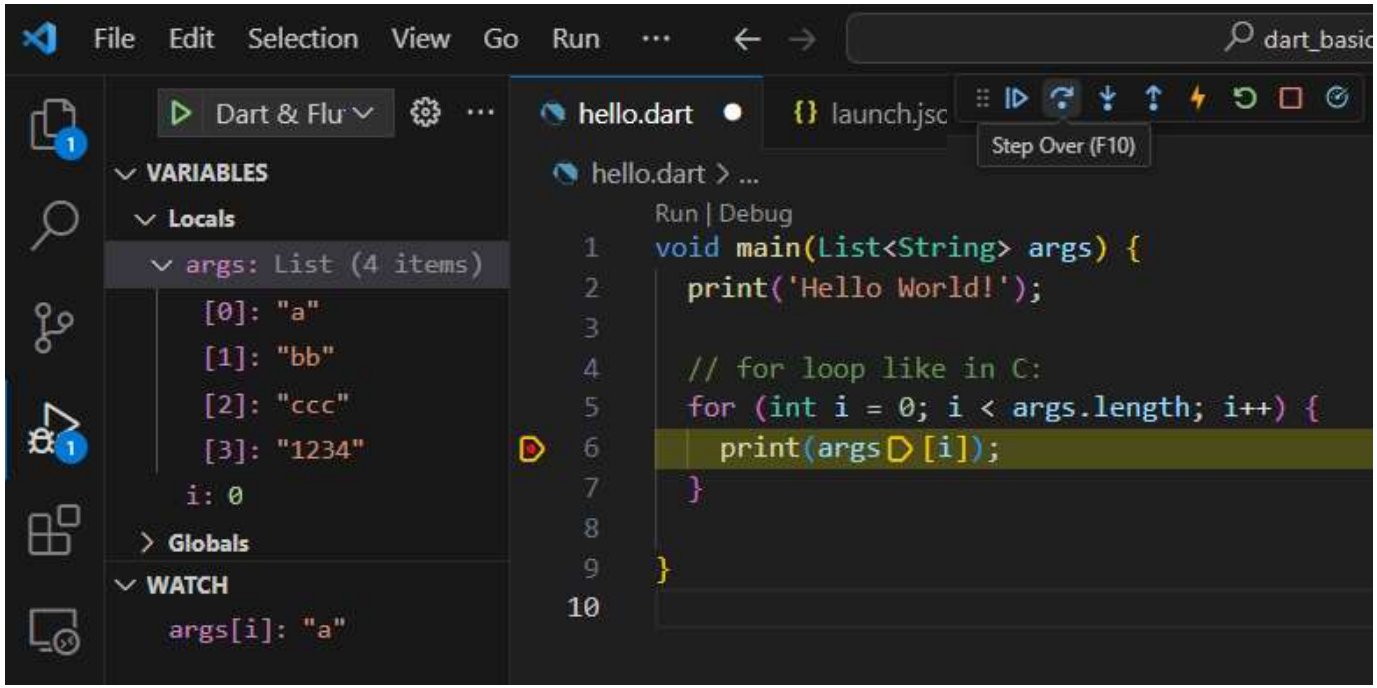
```
1  {
2      // Use IntelliSense to learn about possible attributes.
3      // Hover to view descriptions of existing attributes.
4      // For more information, visit: https://go.microsoft.com/fwli
5      "version": "0.2.0",
6      "configurations": [
7          {
8              "name": "Dart & Flutter",
9              "request": "launch",
10             "type": "dart",
11             "program": "hello.dart",
12             "args": ["a", "bb", "ccc", "1234"]
13         }
14     ]
15 }
```

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

PROBLEMS | OUTPUT | DEBUG CONSOLE | TERMINAL | ... | Filter (e.g. text, !exclude)

```
Connecting to VM Service at ws://127.0.0.1:61700/j6ZsxmW6gAU=/ws
Hello World!
a
bb
ccc
1234

Exited.
```

Debugging in VS Code



Shortcuts in VS Code:

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

Built-in Types

Built-in types

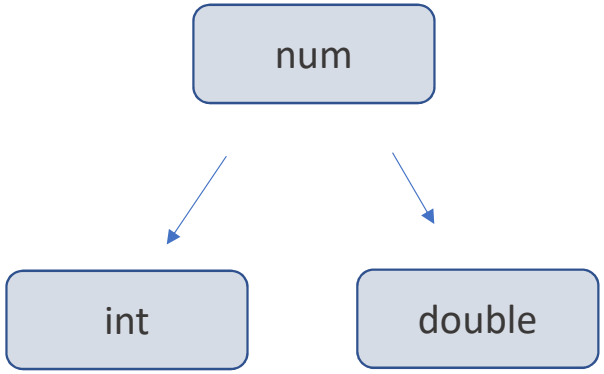
The Dart language has special support for the following:

- 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.

Copied from https://dart.dev/language/built-in-types
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)




```
/// 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);
```

```
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

```
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
```

0xa: 10, 0xb: 11, 0xc: 12, 0xd :13, 0xe: 14, 0xf: 15, 0x10: 16, 0x11: 17 …

1*16 + 1*8 + 1*4 + 1*2 + 0*1 = 30

```
1111
15
```

```
1111000
120
```

Big numbers on Win64

```
// max. positive integer on Win 64
//           1234567890123456
int iMax = 0x7fffffffffffffff;
print (iMax);
int iNext = iMax + 1;
print (iNext);
int iNextNext = iNext + 1;
print (iNextNext);
```

```
9223372036854775807
-9223372036854775808
-9223372036854775807
```

```
double d = iMax.toDouble();
print (d);
double dNext = d + 1;
print (dNext);
```

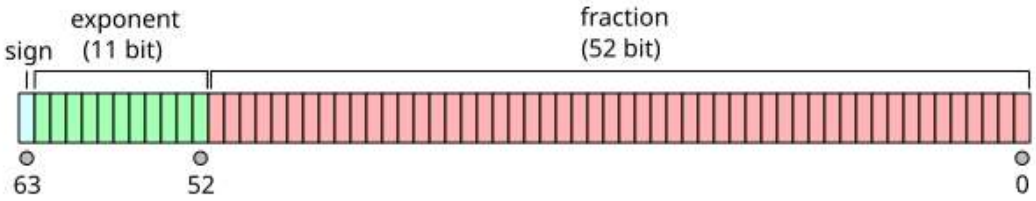
```
9223372036854776000.0
9223372036854776000.0
```

Precision of doubles

```
double d1 = 12345678901234567890.0;
print(d1);
print(d1.toStringAsExponential());
```

```
12345678901234567000.0
1.2345678901234567e+19
```

Aus Wikipedia (https://en.wikipedia.org/wiki/Double-precision_floating-point_format):



The 11 bit width of the exponent allows the representation of numbers between 10^{-308} and 10^{308}, with full 15–17 decimal digits precision.

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);
n.toStringAsExponential();
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

```java
public class Temp {

    public static void main(String[] args) {

        foo(null);

    }

    public static void foo(String s) {
        System.out.println(s.toLowerCase());
    }
}
```

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

```dart
void main(List<String> args) {
  foo(null);
}

void foo(String s) {
  print(s.toLowerCase());
}
```

```
The argument type 'Null' can't be assigned
to the parameter type
'String'. dart(argument_type_not_assignable)
```

Nullable Types in Dart

nullable.dart > ...

Run | Debug

```dart
void main(List<String> args) {
  foo1(null);
}

void foo1(String? s) {
  print(s.toLowerCase());
}
```

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

Run | Debug

```dart
void main(List<String> args) {
  foo1(null);
}

void foo1(String? s) {
  if (s != null) {
    print(s.toLowerCase());
  }
}
```

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());
}
```

```
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();
}
```

Booleans

```
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);
```

```
true
false
true
```

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

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

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

Object has getter “runtimeType” (see slide 16):

```
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:

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

```
myInt = 5;
```

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

```
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”:

```
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;
```

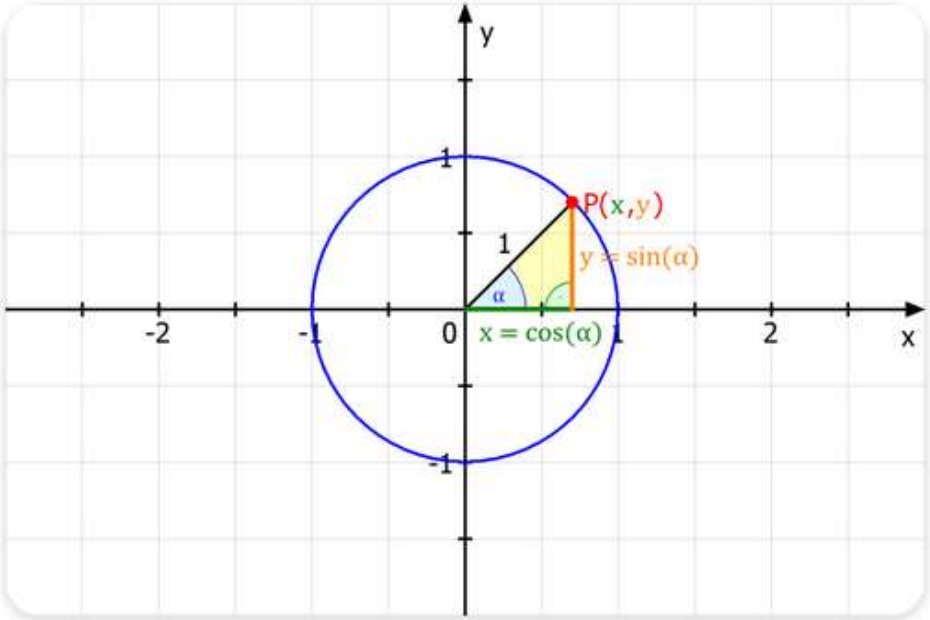
Both “const” and “final” variables cannot be modified after their initialization:

```
cInt1 = 1;
```

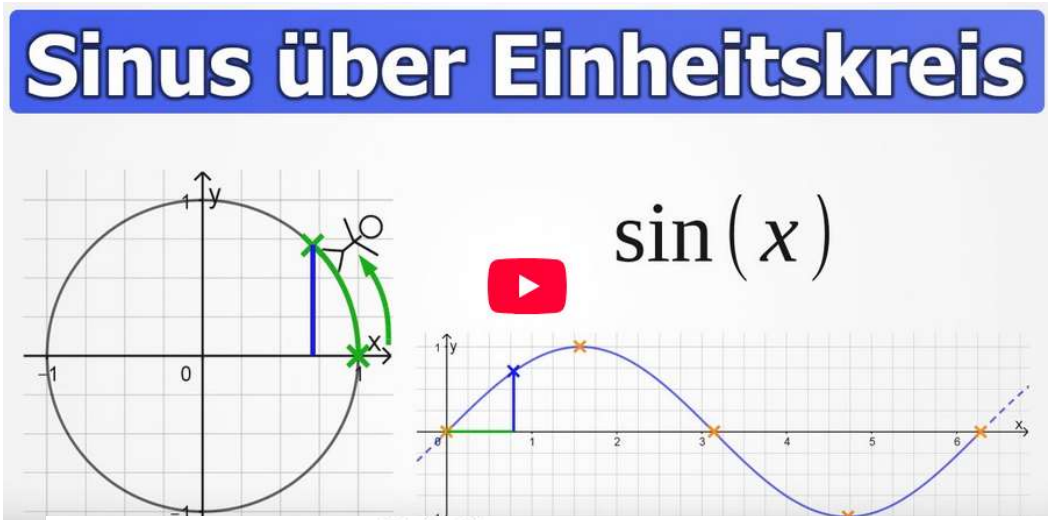
→

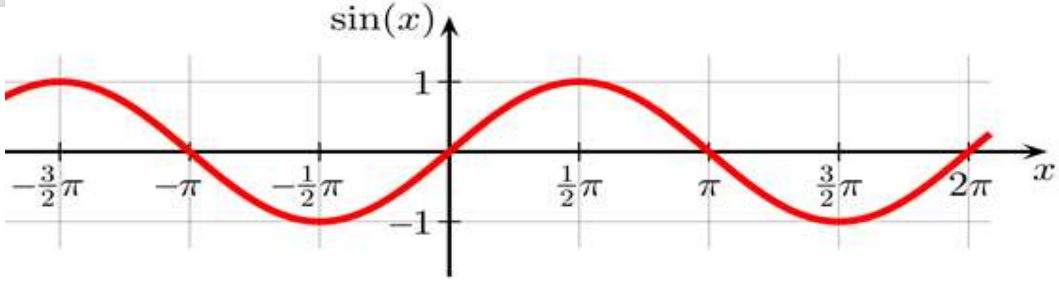
```
Constant variables can't be assigned a value.
Try removing the assignment, or remove the modifier 'const' from the
variable. dart(assignment_to_const)
```

Math reminder



Definition von Sinus und Cosinus am Einheitskreis.





Copied from https://studyflix.de/mathematik/einheitskreis-2061 , https://www.youtube.com/watch?app=desktop&v=t7kSAoPasvQ&t=38s , 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”:

```dart
import 'dart:math';

Run | Debug
void main() {
  var d = pi;
  var x = sin(d);
  print(x);
  print(x.toStringAsFixed(9));

  var y = sin(pi / 2);
  print(y);
}
```

```
1.2246467991473532e-16
0.000000000
```

```
1.0
```

Definition of pi in math.dart:

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

IntelliSense can reduce your typing for loops

IntelliSense:

```
fo
  for
  for
  for in
  FormatException
```

```
fo
  for
  for
  for in
  FormatException
```

```
wh
  while
  while
  WheelEvent
  WheelEvent(…)
```

Generated code:

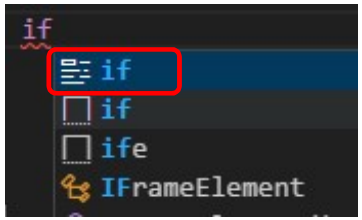
```
for (var i = 0; i < count; i++) {

}
```

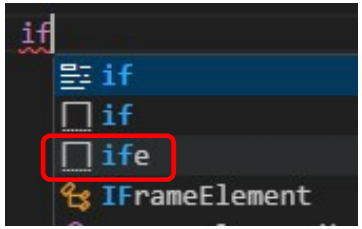
```
for (var element in collection) {

}
```

```
while (condition) {

}
```

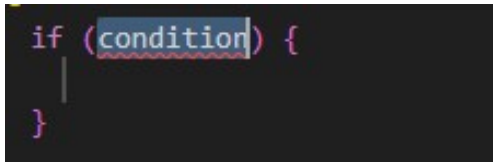
IntelliSense for “if” and “if-else”

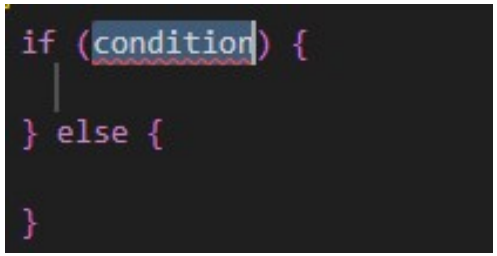
IntelliSense:





Generated code:





More IntelliSense samples

IntelliSense:

```
sw
  switch
  switch statement
  SwitchElement
  SwitchElement()
```

```
fun
  fun
  Function
  FunctionTypeMirror
```

```
ma
  mateapp
  main()
  Map
```

Generated code:

```
switch (expression) {
  case value:

    break;
  default:
}
```

```
void name(params) {

}
```

```
Run | Debug
void main(List<String> args) {

}
```

Functions with positional parameters

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

  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
IconButton(
    onPressed: onButtonPressed,
    iconSize: 50,
    color: Colors.green,
    icon: Icon(Icons.download)),
```

```dart
(new) IconButton IconButton({
  Key? key,
  double? iconSize,
  VisualDensity? visualDensity,
  Color? color,
  Color? disabledColor,
  required void Function()? onPressed,
  MouseCursor? mouseCursor,
  FocusNode? focusNode,
  bool autofocus = false,
  required Widget icon,
```

Functions with positional and named parameters

```
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

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

```
(new) Image Image.asset(
  String name, {
  Key? key,
  AssetBundle? bundle,
  bool excludeFromSemantics = false,
  double? scale,
  double? width,
  double? height,
  Color? color,
```

Exercise

Create a function that calculates the factorial **n!** of an integer n :

$$n! = 1 * 2 * 3 * \ldots * (n-1) * n$$

Call this function in your main for n=1, n=2, n=3 and n=10 and print the factorials:

```
1! is 1
2! is 2
3! is 6
10! is 3628800
```

Possible solution:

```
Run | Debug
void main(List<String> arguments) {
  print('Hello World!');
  print('1! is ${getFactorial(1)}');
  print('2! is ${getFactorial(2)}');
  print('3! is ${getFactorial(3)}');
  print('10! is ${getFactorial(10)}');
}

int getFactorial(int n) {
  var result = 1;
  for (var i = 2; i <= n; i++) {
    result *= i;
  }
  return result;
}
```

Factorial calculator in the web:
https://www.calculatorsoup.com/calculators/discretemathematics/factorials.php

Issues with big n

```
  print (getFactorial(50));
}

int getFactorial(int n) {
  int result = 1;
  for (var i = 2; i <= n; i++) {
    result *= i;
  }
  return result;
}
```

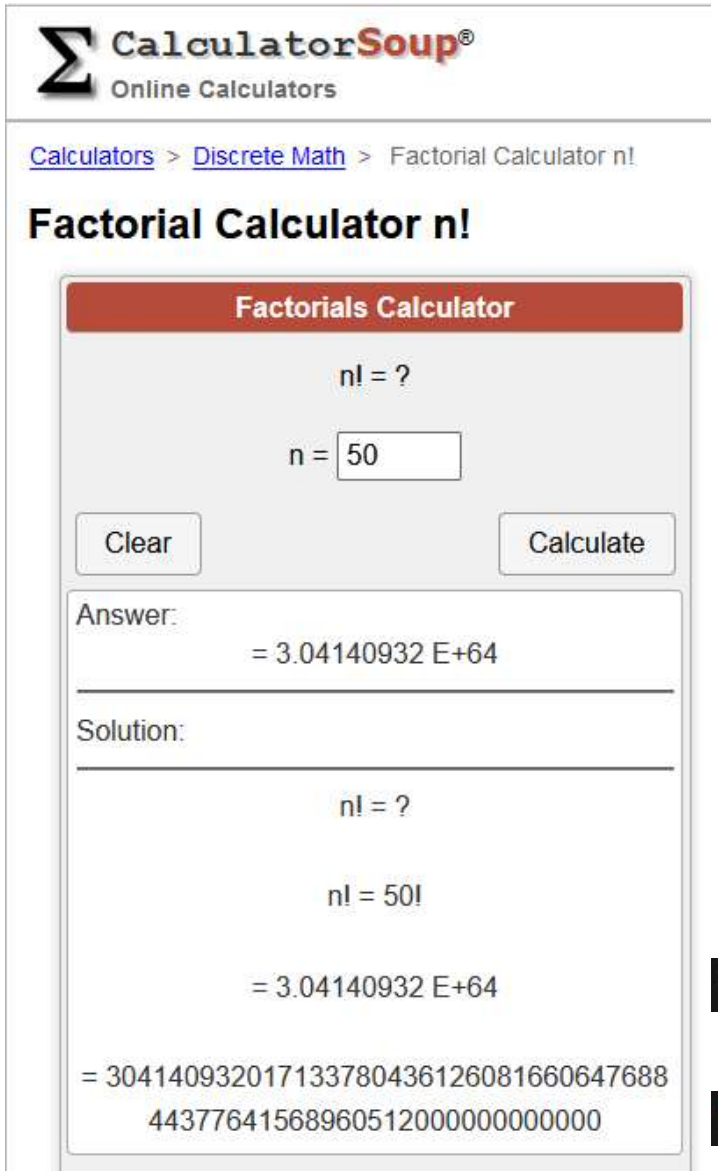
```
-3258495067890909184
```

```
  print (getFactorial(50));
}

double getFactorial(int n) {
  double result = 1;
  for (var i = 2; i <= n; i++) {
    result *= i;
  }
  return result;
}
```

```
3.0414093201713376e+64
```

```
  print (getBigIntFactorial(50));
}

BigInt getBigIntFactorial(int n) {
  BigInt result = BigInt.from(1);
  for (var i = 2; i <= n; i++) {
    result *= BigInt.from(i);
  }
  return result;
}
```

```
30414093201713378043612608166064768844377641568960512000000000000
```



Factorial calculator in the web:

https://www.calculatorsoup.com/calculators/discretemathematics/factorials.php

```
3.0414093201713376e+64
```

```
30414093201713378043612608166064768844377641568960512000000000000
```

Functions with the “arrow syntax”

```
void main(List<String> args) {
  // How many PS are 86 kW ?
  print(kWtoPS(86).toStringAsFixed(0));   // 117
}

double kWtoPS (double kW) {
  return kW * 1.35962;
}
```

If a function body has only 1 line, you can use the arrow syntax:

```
double kWtoPS(double kW) => kW * 1.35962;
```

From https://dart.dev/language/functions:

The `=> expr` syntax is a shorthand for `{ return expr; }`.

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 21.

Anonymous functions with the arrow syntax

```
OutlinedButton(
    onPressed: () {
      print("OK was pressed");
    },
    child: Text("OK")), // OutlinedButton
```

```
OutlinedButton(
    onPressed:() => print("OK was pressed"),
    child: Text("OK")), // OutlinedButton
```

IntelliSense offers both when you enter “:” after “onPressed”

```
OutlinedButton(
    onPressed:
  () =>
  () {}
```

Final optional exercise

Create an exe file **factorial.exe** which calculates the factorials of its command line arguments. Here a sample output:

```
C:\flutter\code\dart_basics>factorial 5 5x 50 500
5! is 120 (this number has 3 digits)

the argument '5x' is no integer

50! is 30414093201713378043612608166064768844377641568960512000000000000 (this number has 65 digits)

500! is 1220136825991110068701238785423046926253574342803192842192413588385845373153881997605496447502203281863
0136164771482035841633787220781772004807852051593292854779075719393306037729608590862704291745478824249127263
4430567017327076946106280231045264421887878946575477714986349436778103764427403382736539747138647787849543848 9
5955375379904232410612713269843277457155463099772027810145610811883737095310163563244329870295638966289116589 7
4769572087926928871281780070265174507768410719624390394322536422605234945850129918571501248706961568141625359 0
5669342381300885624924689156412677565448188650659384795177536089400574523894033579847636394490531306232374906 6
4450488246650759467358620746379251842004593696929810222639719525971909452178233317569345815085523328207628200 2
3402626907898342451712006207714640979456116127629145951237229913340169552363850942885592018727433795173014586 3
5757082835578015873543276888868012039988238470215146760544540766353598417443048012893831389688163948746965881 7
5045069263653381750554781286400000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000 (this number has 1135 digits)
```

Hint: to get the number of characters in a string s, use “s.length”.

Possible solution

```
void main(List<String> args) {
  for (var arg in args) {
    int? i = int.tryParse(arg);
    if (i == null) {
      print("the argument '$arg' is no integer");
    } else {
      var factorialAsString = getBigIntFactorial(i).toString();
      print("$arg! is $factorialAsString (this number has ${factorialAsString.length} digits)");
    }
    print ("");  // print an empty line after the output of each argument
  }
}

BigInt getBigIntFactorial(int n) {
  BigInt result = BigInt.from(1);
  for (var i = 2; i <= n; i++) {
    result *= BigInt.from(i);
  }
  result = result * BigInt.from(1);
  return result;
}
```

Same as in slide 34

To create the exe file, use command “dart compile exe factorial.dart” (see Slide 3).