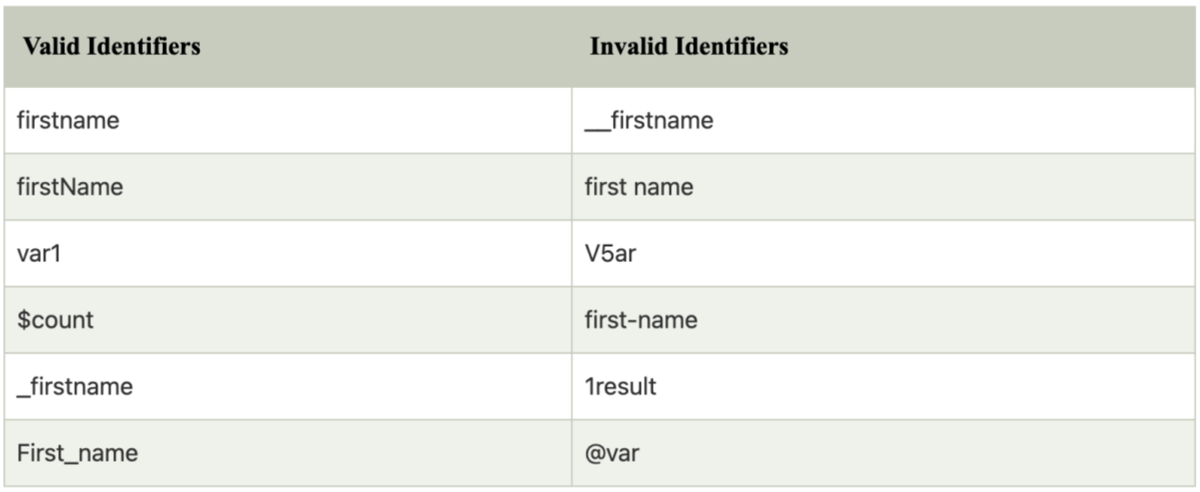
Identifier, Keywords and Variables:

Dart Identifiers

Identifiers are the name which is used to define variables, methods, class, and function, etc. An Identifier is a sequence of the letters([A to Z],[a to z]), digits([0-9]) and underscore(_), but remember that the first character should not be a numeric. There are a few rules to define identifiers which are given below.

- The first character should not be a digit.
- Special characters are not allowed except underscore (_) or a dollar sign ($).
- Two successive underscores (__) are not allowed.
- The first character must be alphabet(uppercase or lowercase) or underscore.
- Identifiers must be unique and cannot contain whitespace.
- They are case sensitive. The variable name **Joseph** and **joseph** will be treated differently.

Below is the table of valid and invalid identifiers.



Valid Identifiers	Invalid Identifiers
firstname	__firstname
firstName	first name
var1	V5ar
$count	first-name
_firstname	1result
First_name	@var

Dart Printing and String Interpolation

The **print()** function is used to print output on the console, and **$expression** is used for the string interpolation. Below is an example.

```
void main()
{
    var name = "Peter";
    var roll_no = 24;
    print("My name is ${name} My roll number is ${roll_no}");
}
```

Semicolon in Dart

The semicolon is used to terminate the statement that means, it indicates the statement is ended here. It is mandatory that each statement should be terminated with a semicolon(;). We can write multiple statements in a single line by using a semicolon as a delimiter. The compiler will generate an error if it is not use properly.

```
var msg1 = "Hello World!";
var msg2 = "How are you?"
```

Dart Whitespace and Line Breaks

The Dart compiler ignores whitespaces. It is used to specify space, tabs, and newline characters in our program. It separates one part of any statement from another part of the statement. We can also use space and tabs in our program to define indentation and provide the proper format for the program. It makes code easy to understand and readable.

Block in Dart

The block is the collection of the statement enclosed in the curly braces. In Dart, we use curly braces to group all of the statements in the block. Consider the following syntax.

Dart Comments

Comments are the set of statements that are ignored by the Dart compiler during the program execution. It is used to enhance the readability of the source code. Generally, comments give a brief idea of code that what is happening in the code. We can describe the working of variables, functions, classes, or any statement that exists in the code. Programmers should use the comment for better practice. There are three types of comments in the Dart.

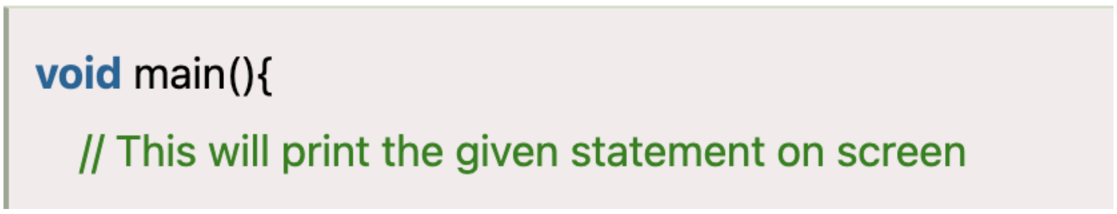
Types of Comments

Dart provides three kinds of comments

- Single-line Comments
- Multi-line Comments
- Documentation Comments

Single-line Comment

We can apply comments on a single line by using the // (double-slash). The single-line comments can be applied until a line break.



```
void main(){
   // This will print the given statement on screen
```

The // (double-slash) statement is completely ignored by the Dart compiler and retuned the output.

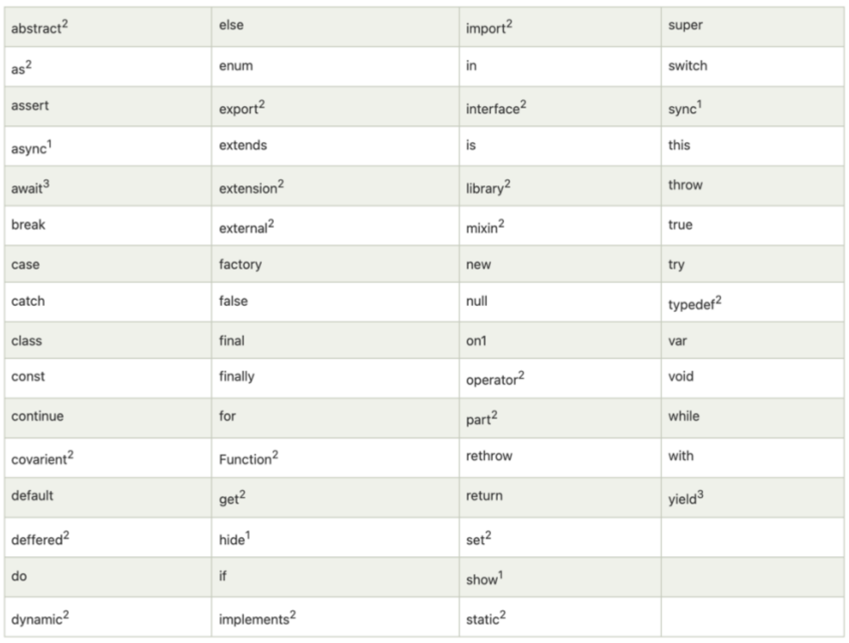
Multi-line Comment

Sometimes we need to apply comments on multiple lines; then, it can be done by using /*…..*/. The compiler ignores anything that written inside the /*…*/, but it cannot be nested with the multi-line comments. Let's see the following example.

```
void main(){
  /* This is the example of multi-line comment
  This will print the given statement on screen */
```

Dart Keywords

Dart Keywords are the **reserve words** that have special meaning for the compiler. It cannot be used as the variable name, class name, or function name. Keywords are case sensitive; they must be written as they are defined. There are 61 keywords in the Dart. Some of them are common, you may be already familiar and few are different. Below is the list of the given Dart keywords.



abstract[2]	else	import[2]	super
as[2]	enum	in	switch
assert	export[2]	interface[2]	sync[1]
async[1]	extends	is	this
await[3]	extension[2]	library[2]	throw
break	external[2]	mixin[2]	true
case	factory	new	try
catch	false	null	typedef[2]
class	final	on1	var
const	finally	operator[2]	void
continue	for	part[2]	while
covarient[2]	Function[2]	rethrow	with
default	get[2]	return	yield[3]
deffered[2]	hide[1]	set[2]	
do	if	show[1]	
dynamic[2]	implements[2]	static[2]	

In the above list of keywords, we have a few keywords which are marked with the **superscript(1,2 and 3)**. Following, we are defining the reason for superscript.

- **Subscript 1 -** These keywords are called **contextual keywords**. They have special meaning and used in particular places.
- **Subscript 2 -** These keywords are called **built-in identifiers**. These types of keywords are used to porting of JavaScript code for Dart, these keywords are treated as a valid identifier, but they cannot be used in the class name function name, or import prefixes.
- **Subscript 3 -** These are newly added keyword related to the **asynchrony**

Dart Variable

Variable is used to store the value and refer the memory location in computer memory. When we create a variable, the Dart compiler allocates some space in memory. The size of the memory block of memory is depended upon the type of variable. To create a variable, we should follow certain rules. Here is an example of a creating variable and assigning value to it.

var name='shyam';

Here the variable called **name** that holds 'shyam' string value. In Dart, the variables store references. The above variable stores reference to a String with a value of shyam.

Rule to Create Variable

Creating a variable with a proper name is an essential task in any programming language. The Dart has some rules to define a variable. These rules are given below.

- The variable cannot contain special characters such as whitespace, mathematical symbol, runes, Unicode character, and keywords.
- The first character of the variable should be an alphabet([A to Z],[a to z]). Digits are not allowed as the first character.

- Variables are case sensitive. For example, - variable age and AGE are treated differently.
- The special character such as #, @, ^, &, * are not allowed expect the underscore(_) and the dollar sign($).
- The variable name should be retable to the program and readable.

How to Declare Variable in Dart

We need to declare a variable before using it in a program. In Dart, The **var** keyword is used to declare a variable. The Dart compiler automatically knows the type of data based on the assigned to the variable because Dart is an infer type language. The syntax is given below.

Syntax -

```
var <variable_name>  = <value>;
```

or

```
var <variable_name>;
```

Example -

```
var name = 'Andrew'
```

In the above example, the variable **name** has allocated some space in the memory. The semicolon(;) is necessary to use because it separates program statement to another.

Type Annotations

As we had pointed out, The Dart is an infer language but it also provides a type annotation. While declaring the variable, it suggests the type of the value that variable can store. In the type annotation, we add the data type as a prefix before the variable's name that ensures that the variable can store specific data type. The syntax is given below.

Syntax -

```
<type> <variable_name>;
```

or

```
<type> <name> = <expression>;
```

Example -

```
int age;
String msg = "Welcome to JavaTpoint";
```

In the above example, we have declared a variable named **age** which will store the integer data. The variable named **msg** stored the string type data.

Declaring the variable with Multiple Values

Dart provides the facility to declare multiple values of the same type to the variables. We can do this in a single statement, and each value is separated by commas. The syntax is given below.

Syntax -

```
<type> <var1,var2....varN>;
```

Example -

```
int i,j,k;
```

Default value

Uninitialized variables that have a nullable type have an initial value of null. (If you haven't opted into null safety, then every variable has a nullable type.) Even variables with numeric types are initially null, because numbers—like everything else in Dart—are objects.

```
int? lineCount;
assert(lineCount == null);
```

If you enable null safety, then you must initialize the values of non-nullable variables before you use them:

```
int lineCount = 0;
```

You don’t have to initialize a local variable where it’s declared, but you do need to assign it a value before it’s used.

Late variables

Dart 2.12 added the late modifier, which has two use cases:

- Declaring a non-nullable variable that’s initialized after its declaration.
- Lazily initializing a variable.

Often Dart’s control flow analysis can detect when a non-nullable variable is set to a non-null value before it’s used, but sometimes analysis fails. Two common cases are top-level variables and instance variables: Dart often can’t determine whether they’re set, so it doesn’t try.

If you’re sure that a variable is set before it’s used, but Dart disagrees, you can fix the error by marking the variable as late:

Note: If you fail to initialize a late variable, a runtime error occurs when the variable is used.

When you mark a variable as late but initialize it at its declaration, then the initializer runs the first time the variable is used. This lazy initialization is handy in a couple of cases:

- The variable might not be needed, and initializing it is costly.
- You’re initializing an instance variable, and its initializer needs access to this.

In the following example, if the temperature variable is never used, then the expensive _readThermometer() function is never called:

```
// This is the program's only call to _readThermometer().
late String temperature = _readThermometer(); // Lazily initialized.
```

Final and const

If you never intend to change a variable, use final or const, either instead of var or in addition to a type. A final variable can be set only once; a const variable is a compile-time constant. (Const variables are implicitly final.)

Note: Instance variables can be final but not const.
Const is used in class with static variables only.

Here's an example of creating and setting a final variable:

```
final name = 'Bob'; // Without a type annotation
final String nickname = 'Bobby';
```

You can't change the value of a final variable:

```
name = 'Alice'; // Error: a final variable can only be set once.
```

Use const for variables that you want to be **compile-time constants**. If the const variable is at the class level, mark it static const. Where you declare the variable, set the value to a compile-time constant such as a number or string literal, a const variable, or the result of an arithmetic operation on constant numbers:

```
const bar = 1000000; // Unit of pressure (dynes/cm2)
const double atm = 1.01325 * bar; // Standard atmosphere
```

The const keyword isn't just for declaring constant variables. You can also use it to create constant *values*, as well as to declare constructors that *create* constant values. Any variable can have a constant value.

```
var foo = const [];
final bar = const [];
const baz = []; // Equivalent to `const []`
```