Dart Function

Dart function is a set of codes that together perform a specific task. It is used to break the large code into smaller modules and reuse it when needed. Functions make the program more readable and easier to debug. It improves the modular approach and enhances the code reusability.

The function provides the flexibility to run a code several times with different values. A function can be called anytime as its parameter and returns some value to where it called.

Defining a Function

A function can be defined by providing the name of the function with the appropriate parameter and return type. A function contains a set of statements which are called function body.

```
return_type func_name (parameter_list)

{
   //statement(s)
  return value;
}
```

- **return_type** - It can be any data type such as void, integer, float, etc. The return type must be matched with the returned value of the function.
- **func_name** - It should be an appropriate and valid identifier.
- **parameter_list** - It denotes the list of the parameters, which is necessary when we called a function.
- **return value** - A function returns a value after complete its execution.

```
void main() {
  int sum(int a, int b) {
    int c;
    c = a + b;
    return c;
  }

  var result = sum(10, 20);
  print("The sum
is:$result");
}
```

Passing Arguments to Function

When a function is called, it may have some information as per the function prototype is known as a parameter (argument). The number of parameters passed and data type while the function call must be matched with the number of parameters during function declaration. Otherwise, it will throw an error. Parameter passing is also optional, which means it is not compulsory to pass during function declaration. The parameter can be two types.

Actual Parameter - A parameter which is passed during a function definition is called the actual parameter.

Formal Parameter - A parameter which is passed during a function call is called the formal parameter.

Return a Value from Function

A function always returns some value as a result to the point where it is called. The **return** keyword is used to return a value. The return statement is optional. A function can have only one return statement.

Function types

Functions are categorized into multiple types based on return type and arguments.

- **Functions without arguments and return type**:

This example contains a function with no arguments and a return type

```
void main() {

printMsg();

}
void printMsg() {

 print("Welcome John");

 }
```

- **Functions without arguments and with return type**:

This example shows a function without arguments and returns a type

```
void main() {

print(getMessage());

}
String getMessage() {

return "welcome";

}
```

•Functions with arguments and without return type:

This example shows a method with arguments and without a return type.

```
void main() {

printMsg("john");

}
void printMsg(String name) {

print("Welcome ${name}");

}
```

- **Functions with arguments and return type:**

```
void main() {

print(substraction(12, 2));

 print(substraction(40, 20));

}
int substraction(int one, int two) {

var result = one - two;

return result;
}
```

In Dart, **positional parameters** are specified by their position in the parameter list, while **named parameters** are identified by a name.

The main differences between the two are:

- **Positional parameters** must be passed in the same order as they appear in the function signature, whereas named parameters can be passed in any order.
- **Positional parameters** are defined by their position in the parameter list, whereas named parameters are defined by their name.
- **Named parameters** are optional by default and are enclosed in curly braces ({ }) in the function signature.
- **positional parameters** are optional when they have a default value and are enclosed in square brackets ([ ]) in the function signature.

To tell if an optional parameter was specified or not, you can use the null value. If an optional parameter has not been specified, its value will be null. You can also use the required keyword to indicate that a named parameter is required and must be provided by the caller. If a required named parameter is not provided, an error will be thrown at runtime.

Dart Functional optional arguments

Functional arguments can be options, which means the caller is not required to send arguments.
There are multiple types of optional argument types

- Optional Named parameters
- Optional Positional Arguments
- Optional Arguments with the default value

optional Named parameters

In Named arguments, Each argument is named with a variable.
Named arguments are enclosed in {}.

```
void functionname(argument, {type arguments}) {
// statements
 }
```

arguments in { } contain a type and are separated by a comma.
Caller function syntax the function called with passing normal arguments with value and named arguments with argument: value syntax if the caller is not passed with optional arguments, the value null is passed.

```
void main() {

 printNumbers(10, number2= 20);

 }

void printNumbers(number1, {int number2, int number3}) {
print(number1);

 print(number2);

print(number3);
 }
```

```
10
20
null
```

Named Optional Parameters

Named optional parameters are very similar to positional optional parameters, but they are defined and called differently. When a function with them is called, the order they are supplied with does not matter. They are defined with curly braces {} and colons to separate their name from their default value. Like positional optional parameters, commas separate them. Also like positional optional parameters, they do not need to have default values. In cases when named optional parameters are not supplied with a value, they will also have a default value of null.

```
void repeat(String word, {int repetitions = 1, String exclamation = ""})
{
  for (int i = 0; i < repetitions; i++) {
    print(word + exclamation);
  }
}

void main() {
  repeat("Dog"); // legal
  repeat("Dog", repetitions: 2, exclamation: "!"); // legal
  repeat("Dog", repetitions: 2); // legal
  repeat("Dog", exclamation: "!"); // legal, even without repetitions
  repeat("Dog", exclamation: "!", repetitions: 2); // legal, even out of
order
}
```

```
Dog Dog! Dog! Dog Dog Dog! Dog! Dog!
```

Positional Optional Arguments

In this, optional arguments are passed inside square brackets[]. positional arguments are assigned based on position and order defined inside [ ].

```
void functionname(argument, [type arguments]) {
// statements
}
```

```
void main() {

printNumbers(10, 20);

}
void printNumbers(number1, [int number2, int number3]) {
print(number1);

 print(number2);

print(number3);
}
```

```
10
20
null
```

Optional Arguments with a default value

Optional arguments are passed with default values in the function declaration.

Syntax:

```
void functionname(argument, {type arguments=defaultvalue}) {
 // statements
}
```

```
void main() {

printNumbers(10, 20);

} void printNumbers(number1, [int? number2, int number3=40]) {

print(number1);

print(number2);

 print(number3);
 }
```

```
10
20
40
```

```
void repeat(String word, [int repetitions = 1, String exclamation = ""])
{
  for (int i = 0; i < repetitions; i++) {
    print(word + exclamation);
  }
}

void main() {
  repeat("Dog"); // legal
  repeat("Dog", 2, "!"); // legal
  repeat("Dog", 2); // legal
  //repeat("Dog", "!"); // ILLEGAL
  //repeat("Dog", "!", 2); // ILLEGAL
}
```

Dart function as parameter

A Dart function can be passed to other functions as a parameter. Such a function is called a *higher-order* function.

```
int inc(int x) => ++x;
int dec(int x) => --x;
int apply(int x, Function f) {
  return f(x);
}

void main() {
  int r1 = apply(3, inc);
  int r2 = apply(2, dec);
  print(r1);
  print(r2);
}
```

```
4
1
```

Dart nested function

A nested function, also called an inner function, is a function defined inside another function.

```
void main() {
  String Message(String name, String occupation) {
    return "$name is a $occupation";
  }

  var name = "John Doe";
  var occupation = "gardener";
  var msg = Message(name, occupation);
  print(msg);
}
```

In the example below, we nest square in the circleArea function.

```
double circleArea(double radius) {
  // outer function
  double square(double r) {
    // inner/nested function
    return r * r;
  }

  return 3.14 * square(radius);
}
void main() {
  double radius = 5;
  double area = circleArea(radius);
  print('The area of circle with radius $radius is $area.');
}
```

```
The area of circle with radius 5.0 is 78.5.
```

Dart Anonymous Function

Dart provides the facility to specify a nameless function or function without a name. This type of function is known as an **anonymous function, lambda, or closure**. An anonymous function behaves the same as a regular function, but it does not have a name with it. It can have zero or any number of arguments with an optional type annotation.

We can assign the anonymous function to a variable, and then we can retrieve or access the value of the closure based on our requirement.

An Anonymous function contains an independent block of the code, and that can be passed around in our code as function parameters.

```
var addNum = (int a, int b) => a + b;

main() {
  print(addNum(15, 7));
}
```

Note: When using the Lambda function syntax, only one expression can be returned, and it must be a single-line expression.

Dart Recursion

Dart Recursion is the method where a function calls itself as its subroutine. It is used to solve the complex problem by dividing it into sub-part. A function which is called itself again and again or recursively, then this process is called recursion.

The iterators can be an option to solve problems, but recursion is recommended to the programmers to deal with complex problems because it is an effective approach of problem-solving technique. It requires less time and code to evaluate the same complex task.

Recursion makes many calls to the same function; however, there should be a base case to terminate the recursion.

Recursion uses the divide and conquers technique to solve a complex mathematical computation task.
It divides the large task into small chunks.

```
int factorial(int num) {
  //base case of recursion.
  if (num <= 1) {
    // base case
    return 1;
  } else {
    return num * factorial(num - 1); //function call itself.  }
}
void main() {
  var num = 5;
  var fact = factorial(num);
  print("Factorial Of 5 is: ${fact}");
}
```

```
factorial(5)           # 1st call with 5
5 * factorial(4)       # 2nd call with 4
5 * 4 * factorial(3)   # 3rd call with 3
5 * 4 * 3 * factorial(2)  # 4th call with 2
5 * 4 * 3 * 2 * 1      # return from 2nd call
120                    # return from 1st call
```