
# 2019-2020 Fall

# CS 315

# Homework 2

Section: 3
Name: Munib Emre
Surname: Sevilgen
Student ID: 21602416

# Dart

Example in Dart:

```
// This example is appropriate for the homework because
// it tests everything about the four issues
void main(){

 // Optional positional parameters with []
 // Always last in the parameter list
 // Default value is null unless default value provided
 foo1(a, [b, c= "formal c"]) {
     if (a != null)
       print("a: " + a);
     if (b != null)
       print("b: " + b);
     if (c != null)
       print("c: " + c);
 }
 // Error: Too many positional arguments: 3 allowed, but 4 found.
 // foo("actual a", "actual b","actual c", "actual d")
 print("Foo1:");
 foo1("actual a", "actual b","actual c");
 print("----------");
 print("Foo1:");
 foo1("actual a", "actual b");
 print("----------");
 print("Foo1:");
 foo1("actual a", "actual c");
 print("----------");
 print("Foo1:");
 foo1("actual a");
 print("----------");
 // Error: Too few positional arguments: 1 required, 0 given.
 // foo();

 // Optional named parameters with {}
 // Always last in the parameter list
 // Default value is null unless default value provided
 foo2(a, {b, c= "formal c"}) {
     if (a != null)
       print("a: " + a);
     if (b != null)
       print("b: " + b);
     if (c != null)
       print("c: " + c);
 }
 print("Foo2:");
```

```
 foo2("actual a", b: "actual b", c: "actual c");
 print("----------");
 print("Foo2:");
 foo2("actual a", b: "actual b");
 print("----------");
 print("Foo2:");
 foo2("actual a", c: "actual c");
 print("----------");
 print("Foo2:");
 foo2("actual a");
 print("----------");
 // Error: Too few positional arguments: 1 required, 0 given.
 // foo2();


 // Variable Number of Actual Parameters cannot be used directly
 // It can be done by list or map
 foo3(List list) {
   for(var i = 0; i < list.length ; i++) {
     print(i.toString() + ": " + list[i]);
   }
 }
 print("Foo3:");
 foo3(["actual a", "actual b", "actual c"]);
 print("----------");

 // Primitives are passed by value, Objects are passed by reference
 foo4(List a, int b){
   a.add("b of foo");
   b++;
 }
 List a = ["a of main"];
 int b = 1;

 print("Before foo4: a = " + a.toString() + ", b = " + b.toString());
 foo4(a, b);
 print("After foo4: a = " + a.toString() + ", b = " + b.toString());
}
```

The output of the example in Dart:

```
Foo1:
a: actual a
b: actual b
c: actual c
----------
Foo1:
```

```
a: actual a
b: actual b
c: formal c
----------
Foo1:
a: actual a
b: actual c
c: formal c
----------
Foo1:
a: actual a
c: formal c
----------
Foo2:
a: actual a
b: actual b
c: actual c
----------
Foo2:
a: actual a
b: actual b
c: formal c
----------
Foo2:
a: actual a
c: actual c
----------
Foo2:
a: actual a
c: formal c
----------
Foo3:
0: actual a
1: actual b
2: actual c
----------
Before foo4: a = [a of main], b = 1
After foo4: a = [a of main, b of foo], b = 1
```

- The parameter correspondence is done by both keywords and positions. In the foo2 function, the parameter a using positional correspondence, and the b and c are using the keyword correspondence by putting them in the curly brackets. However, when the parameters in the curly brackets mean also that they are optional parameters. [1]
- The default values can be assigned to the formal parameters if they are optional parameters as can be seen in the foo1 and foo2 functions. The optional parameters can be achieved by using square brackets that means the positional correspondence and curly brackets that means keyword correspondence. [1]

- The variable number of parameters can be achieved by optional parameters. If the parameters have not a default value then they will be null as can be seen from the fourth call of both foo1 and foo2 functions.[1] Moreover, the List object can be used to pass variable number parameters as can be seen in the foo3 function. It is not in the language by default.[2]
- The primitive types are passed by value, but Objects are passed by reference as can be seen in the foo4 function. The variable a which is a List is changed but b which is int is not changed.[3]

# JavaScript

Example in JavaScript:

```
<!DOCTYPE html>
<html lang="en">
<body>
<script>
// This example is appropriate for the homework because
// it tests everything about the four issues

// Can have default values
// Has only positional paratmeters
// If not enough actual parameter other formals without default value
are undefined
// Optional parameters cen be used directly
function foo1(a, b, c = "formal c") {
   console.log("a: " + a);
   console.log("b: " + b);
   console.log("c: " + c);
}

console.log("Foo1:");
foo1("actual a", "actual b", "actual c");
console.log("----------");
console.log("Foo1:");
foo1("actual a", "actual b");
console.log("----------");
console.log("Foo1:");
foo1("actual a", "actual c");
console.log("----------");
console.log("Foo1:");
foo1("actual a");
console.log("----------");
console.log("Foo1:");
foo1();
console.log("----------");
```

```
// With object literals named variable value pair can be used
function foo2(params){
   for (var key in params)
       console.log(key + ": " + params[key]);
}
console.log("Foo2:");
foo2({a: "actual a", b: "actual b"});
console.log("----------");

// Variable number of actual arameters is possible with ...args at
the end
function foo3(a, ...args) {
   console.log("a: " + a)
   for(i = 0 ; i < args.length ; i++) {
       console.log("" + i + ": " + args[i]);
   }
}
console.log("Foo3:");
foo3("actual a","actual b","actual c");
console.log("----------");

// Primitives are Passed by Value but Objects are Passed by Reference
function  foo4(a, b){
   a[1]="b of foo";
   b++;
}

a = ["a of main"];
b = 1;
console.log("Before foo4: a = " + a + ", b = " + b);
foo4(a, b);
console.log("After foo4: a = " + a + ", b = " + b);

</script>
</body>
</html>
```

The output of the example in JavaScript:

```
Foo1:
a: actual a
b: actual b
actual c
----------
Foo1:
a: actual a
b: actual b
```

```
c: formal c
----------
Foo1:
a: actual a
b: actual c
c: formal c
----------
Foo1:
a: actual a
b: undefined
c: formal c
----------
Foo1:
a: undefined
b: undefined
c: formal c
----------
Foo2:
a: actual a
b: actual b
----------
Foo3:
a: actual a
0: actual b
1: actual c
----------
Before foo4: a = a of main, b = 1
After foo4: a = a of main,b of foo, b = 1
```

- The parameter correspondence is done by positions. In the foo1 function, the parameters using positional correspondence. However, keyword correspondence can be achieved by using object literals which keeps key-value pairs as in the foo2 function. [4]
- The default values can be assigned to the formal parameters. When the parameters have not any default value and actual parameter then it is undefined as can be seen in the second and third call of foo1 function.
- The variable number of parameters can be achieved by ...args at the end of the parameters as can be seen in the foo3 function. The args array can keep any number of parameters. Moreover, the parameters can be used as optional. If they have not a default value then they take undefined value.
- The primitive types are passed by value, but Objects are passed by reference as can be seen in the foo4 function. The variable a which is the list is changed but b which is int is not changed. [5]

# Perl

Example in Perl:

```
# This example is appropriate for the homework because
# it tests everything about the four issues

# Parameters are got by parameter array therefore this has positional
parameters
# Optional parameter can be done with the array
# Default values can be assigned with the if statements
sub foo1 {
   ($a, $b, $c) = @_;
   # Default values can be assigned by this
   if (not defined($c) ) {
       $c = "formal c";
   }
   print "a: $a\n";
   print "b: $b\n";
   print "c: $c\n";
}

print "Foo1:\n";
foo1("actual a","actual b","actual c");
print "----------\n";
print "Foo1:\n";
foo1("actual a","actual b");
print "----------\n";
print "Foo1:\n";
foo1("actual a", "actual c");
print "----------\n";
print "Foo1:\n";
foo1("actual a");
print "----------\n";
print "Foo1:\n";
foo1();
print "----------\n";

# Named variables can be used by hash
sub foo2 {
   ($a, $hash) = @_;
   print "a: $a\n";
   foreach $key (keys %hash) {
       $value = $hash{$key};
       print "$key: $value\n";

   }
}
```

```
$hash{"b"} = "actual b";
$hash{"c"} = "actual c";

print "Foo2:\n";
foo2("actual a", $hash);
print "----------\n";

# Variable Number of Actual Parameters can be used by arrays
sub foo3 {
   my ($a, $params) = @_;
   print "a: $a\n";
   for (my $index = 0; $index <= $#params; $index++) {
       print "$index: $params[$index]\n";
   }
}
@params = ("actual b","actual c");
print "Foo3:\n";
foo3("actual a", $params);
print "----------\n";

# Pass parameters by reference as default
# Pass by value can be done by creating local variables with my
sub foo4 {
   $a, my $b = @_;
   $a = "a of foo4";
   $b = "b of foo4";
   $_[2] = "c of foo4";
}

$a = "a of global";
$b = "b of global";
$c = "c of global";

print "Before foo4: a = $a, b = $b, c = $c\n";
foo4($a, $b, $c);
print "After foo4: a = $a, b = $b, c = $c\n";
```

The output of the example in Perl:

```
Foo1:
a: actual a
b: actual b
c: actual c
----------
Foo1:
a: actual a
b: actual b
```

```
c: formal c
----------
Foo1:
a: actual a
b: actual c
c: formal c
----------
Foo1:
a: actual a
b:
c: formal c
----------
Foo1:
a:
b:
c: formal c
----------
Foo2:
a: actual a
c: actual c
b: actual b
----------
Foo3:
a: actual a
0: actual b
1: actual c
----------
Before foo4: a = a of global, b = b of global, c = c of global
After foo4: a = a of foo4, b = b of global, c = c of foo4
```

- The parameters are passed by the array. Therefore it can be said that the correspondence is done by positions as the indexes of the array. Although it is not in the language by default, the keyword correspondence can be achieved by using the hash to pass the parameters as can be seen in the foo2 function. [6]
- The default values cannot be assigned to the formal parameters directly. By using the if statement that checks the parameter is defined or not the default values can be achieved as can be seen in the foo1 function. It can be said that the parameter c has a default value.
- The variable number of parameters can be achieved because of the array of parameters. It can be said that the parameter c in the foo1 function is optional. Moreover, by using an array of parameters, there can be any number of parameters as can be seen in the function foo3. The params array can keep any number of parameters.
- Pass by reference is the default as can be seen in the foo4 function. If a value is assigned to the parameter array elements or the variable without my keyword and an element of the array is assigned, there is pass by reference as can be seen with the a

and c parameters. However, if the assigned variable is declared with my keyword, then it can be said that there is the pass by value as can be seen with the b parameter. [7]

# PHP

Example in PHP:

```
<html>
<body>

<?php
   // This example is appropriate for the homework because
   // it tests everything about the four issues

   // Can have default values
   // Has only positional paratmeters
   // If not enough actual parameter other formals without default
value are undefined
   // Optional parameters cen be used directly

   function foo1($a, $b, $c="formal c") {
       echo "a: $a\n";
       echo "b: $b\n";
       echo "c: $c\n";
   }
   echo "Foo1:\n";
   foo1("actual a", "actual b", "actual c");
   echo "----------\n";
   echo "Foo1:\n";
   foo1("actual a", "actual b");
   echo "----------\n";
   echo "Foo1:\n";
   // Warning: Missing argument 2 for foo1(), called in
sevilgen_munib_emre_php.php on line 13 and defined on line 3
   foo1("actual a");
   echo "----------\n";


   // Variable number of params is possible with *args at the end
   function foo2($a, ...$args) {
       echo "a: $a\n";
       for($index = 0 ; $index < count($args) ; $index++ ) {
           echo "$index: $args[$index]\n";
       }
   }
   echo "Foo2:\n";
```

```
    foo2("actual a", "actual b", "actual c");
    echo "----------\n";

    // Default is pass by value
    // Pass by refernce with & at the beginning
    function foo3($a, &$b) {
        $a = "a in foo3";
        $b = "b in foo3";
    }

    $a = "a in global";
    $b = "b in global";
    echo "Before foo3: a = $a, b = $b\n";
    foo3($a, $b);
    echo "After foo3: a = $a, b = $b\n"

?>

</body>
</html>
```

The output of the example in PHP:

```
Foo1:
a: actual a
b: actual b
c: actual c
----------
Foo1:
a: actual a
b: actual b
c: formal c
----------
Foo1:
PHP Warning:  Missing argument 2 for foo1(), called in
/home/cs/emre.sevilgen/CS315Homework2/sevilgen_munib_emre_php.php on
line 21 and defined in
/home/cs/emre.sevilgen/CS315Homework2/sevilgen_munib_emre_php.php on
line 10
a: actual a
PHP Notice:  Undefined variable: b in
/home/cs/emre.sevilgen/CS315Homework2/sevilgen_munib_emre_php.php on
line 12
b:
c: formal c
----------
Foo2:
```

```
a: actual a
0: actual b
1: actual c
----------
Before foo3: a = a in global, b = b in global
After foo3: a = a in global, b = b in foo3
```

- The parameter correspondence is done by positions. In the foo1 function, the parameters using positional correspondence. [8]
- The default values can be assigned to the formal parameters. [8] When the parameters have not any default value and actual parameter then it is empty as can be seen in the third call of foo1 function. There is a warning about that in the output.
- The variable number of parameters can be achieved by …$args at the end of the parameters as can be seen in the foo2 function. [8] The args array can keep any number of parameters. Moreover, the parameters can be used as optional. If they have not a default value then they take empty value.
- The default is the pass by value as can be seen in the foo3 function by the parameter a. The pass by reference can be used by the ampersand sign in front of the formal parameter as can be observed by the parameter b in foo3. [8]

# Python

Example in Python:

```python
# This example is appropriate for the homework because
# it tests everything about the four issues

# Both positional and keyword parameter is possible
# Optional parameters should be at the end and must have default
value
def foo1(a, b, c = "formal c"):
   print("a:", a)
   print("b:", b)
   print("c:", c)

print("Foo1:")
foo1("actual a", "actual b", "actual c")
print("----------")
print("Foo1:")
foo1("actual a", "actual b")
print("----------")
# TypeError: foo() takes at least 2 arguments (1 given)
# foo("actual a")
# print("----------")
print("Foo1:")
foo1(a = "actual a", b = "actual b", c = "actual c")
```

```
print("----------")
print("Foo1:")
foo1(b = "actual b", a = "actual a")
print("----------")

# Variable number of params is possible with *args at the end
def foo2(a, *args):
   print("a:", a)
   for i in range(len(args)):
       print(str(i) + ": " + args[i])

print("Foo2:")
foo2("actual a", "actual b", "actual c")
print("----------")

# Pass by assignment actual is assigned to the formal
def foo3(a, myList):
   a += 1
   myList.append("c of foo3")

a = 1
myList = ["b of global"]
print("Before foo3: a =", a, "- list =", myList)
foo3(a, myList)
print("After foo3: a =", a, "- list =", myList)
```

The output of the example in Python:

```
Foo1:
a: actual a
b: actual b
c: actual c
----------
Foo1:
a: actual a
b: actual b
c: formal c
----------
Foo1:
a: actual a
b: actual b
c: actual c
----------
Foo1:
a: actual a
b: actual b
c: formal c
```

```
----------
Foo2:
a: actual a
0: actual b
1: actual c
----------
Before foo3: a = 1 - list = ['b of global']
After foo3: a = 1 - list = ['b of global', 'c of foo3']
```

- The parameter correspondence is done by both keywords and positions. In the first two calls of the foo1 function, the parameters use positional correspondence, at the others the parameters use the keyword correspondence. [9]
- The default values can be assigned to the formal parameters and they will be optional parameters as can be observed by the parameter c in the foo1 function. [9]
- The variable number of parameters can be achieved by optional parameters. The parameters should have a default value. Moreover, the *args parameter can be used to pass variable number parameters as can be seen in the foo2 function. The args array can have any number of parameters. [9]
- The pass by assignment is used in Python. The actual parameter is assigned to the formal parameter. Therefore, in the foo3 function, the integer a is not changed because it is immutable. However, the list is changed because its reference assigned to the formal parameter and it is mutable. [10]

# References


[1] “A Tour of the Dart Language.” Dart. Accessed December 12, 2019. https://dart.dev/guides/language/language-tour.

[2] Nasution, Faris. “Dart How to Make a Function That Can Accept Any Number of Args.” Stack Overflow. Accessed December 12, 2019. https://stackoverflow.com/questions/16262393/dart-how-to-make-a-function-that-can-accept-any-number-of-args/16266780.

[3] “Are Arguments Passed by Value or Reference?” Google Groups. Google. Accessed December 12, 2019. https://groups.google.com/a/dartlang.org/forum/#!topic/misc/jyn35RSuT9Q

[4] Maben, Robin. “Is There a Way to Provide Named Parameters in a Function Call in JavaScript?” Stack Overflow.Accessed December 12, 2019. https://stackoverflow.com/questions/11796093/is-there-a-way-to-provide-named-parameters-in-a-function-call-in-javascript

[5] JavaScript Function Parameters. Accessed December 12, 2019. https://www.w3schools.com/js/js_function_parameters.asp

[6] Justin. “Named Function Parameters in PHP.” Stack Overflow. Accessed December 12, 2019. https://stackoverflow.com/questions/19126860/named-function-parameters-in-php

[7] “Passing Parameters to Subroutine.” Perl Tutorial. Accessed December 12, 2019. https://www.perltutorial.org/passing-parameters-to-subroutine/
[8] “Function Arguments - Manual.” php. Accessed December 12, 2019. https://www.php.net/manual/en/functions.arguments.php
[9] “PEP 0 -- Index of Python Enhancement Proposals (PEPs).” Python.org. Accessed December 12, 2019. https://www.python.org/dev/peps/
[10] “Python Course.” Python Tutorial: Passing Arguments. Accessed December 12, 2019. https://www.python-course.eu/python3_passing_arguments.php