**Python:** (Please run the python file using python filename.py in dijkstra)

**File name:** haque\_faaiz\_python.py

**Code:**

def func1(a , b): #Parameter Correspondence

result = a - b

print result

print "func1 first call"

func1(b=10,a=5) #keyword

print "func1 second call"

func1(10,5) #positional

def func2( a = 10, b = 5 ): #Formal Parameter Default Values

result = a - b

print result

print "func2 first call"

func2() #default values

print "func2 second call"

func2(3,2) #specified

print "func2 third call"

func2(3) #one default value

def func3( zeroes = [0] ): #Formal Parameter Default Values and Passing

zeroes.append(0)

print zeroes

print "func3 first call"

func3() #First call

print "func3 second call"

func3() #Second

print "func3 third call"

func3() #Third

def func4( \*values ): #Variable number of actual parameters

print len(values), "values: " , values, "in func4"

func4("1"), "after func4 first call" #1 value

func4(3,5,0), "after func4 second call" #3 values

def func5( a ): #Parameter passing: doesn't effect arg of caller

print a, id(a), "in func5"

a = 2

print a, id(a), "in func5"

func5(1)

def func6( array): #Parameter passing: doesn't effect arg of caller

array = [0,0]

print array, "in func6"

array = [1,2,3]

func6(array)

print array, "after func6 call"

def func7(array) : #Parameter passing: we copy its reference

array += [0,0]

print array, "in func7"

array = [1,2,3]

func7(array), "after func7 first call"

print array

array = [1,2,3]

func7(array[:]) #SHALLOW COPY of array

print array, "after func7 second call"

**Outputs of the program:**

*func1 first call*

*-5*

*func1 second call*

*5*

*func2 first call*

*5*

*func2 second call*

*1*

*func2 third call*

*-2*

*func3 first call*

*[0, 0]*

*func3 second call*

*[0, 0, 0]*

*func3 third call*

*[0, 0, 0, 0]*

*1 values: ('1',) in func4*

*3 values: (3, 5, 0) in func4*

*1 29039720 in func5*

*2 29039696 in func5*

*[0, 0] in func6*

*[1, 2, 3] after func6 call*

*[1, 2, 3, 0, 0] in func7*

*[1, 2, 3, 0, 0]*

*[1, 2, 3, 0, 0] in func7*

*[1, 2, 3] after func7 second call*

**Explanations:**

In func1(): I investigated the parameter correspondence using func1. As you can see in the func’s arguments two variables are specified a and b, and in the func body the computation of variable1 subtracted by variable 2 is printed. I called the function two times. First with func1(b=10,a=5). Since the function is defined as func1(a,b) the value taken is according to the specification of the arguments in the call. As in, a takes the value of 5 and b takes the value of 10. This is known as keyword correspondence the result of the output is -5. An advantage of this type of parameter correspondence is parameters can appear in any order, so it’s a very flexible routine. However, the disadvantage is the user must know the name of the parameters of the function, and this could be quite complicated to learn in large programs. In the second function call there is no variables specified, rather just two values. So in this case the program reads them as in order, so a takes value of 10 , and b takes value of 5. This is known as positional correspondence of parameters, this method is safe and effective.

In func2(): In this function I investigated the formal parameter default values. The values of a and b are this time specified in the function definition as a=10 and b=5. The body is the same as func1. I called the func3 3 times. Func2(): without specifying any values the two values stated in the functions definition are used, 10 and 5. Func2(3,2) with specifying values the two default values are overwritten and values of a and b are 3 and 2. Func2(3) the first value is overwritten and the second takes it default value, so the values of and b are 3 and 5. A disadvantage of this is we cannot have a case where we specify the value of b but use the default value of a due to their positioning in the functions definition.

In func3(): In this function I investigated the default values again and see how they are updated. The function keeps appending a 0 to an already specified list in the parameter. So initially the list contains 0 zero. For each function call the default value is appended, so each time it is called again the functions size keeps increases as another 0 is added. This corresponds to the outputs after calling the function 3 times, [0,0] then [0,0,0] then [0,0,0,0]

In func4(): In this function I investigated the variable number of actual parameters being used. In python the star token is used to do this. When the definition is specified using a star and we pass a list, we can find its corresponding number of arguments and the arguments being used.

In func5(): Shows an example where we can find the address of a particular variable and how that address differs for different variables, and that callee’s using certain parameters can pass by value to not change their initial value and address. The call is done with an integer 1 and we can see that integers value and address, if we change the integers value we assign a new address to that integer but used by the same variable.

In func6(): Pass by value: We passed [1,2,3] by value but we can easily change it in the body, however since its pass by value it doesn’t change the list’s value in the global scope. It changed to [0,0] in the functions scope but remained [1,2,3] outside

In func7(): Pass by reference: We pass [1,2,3] and we append [0,0] to the array so we’re left with [1,2,3,0,0] in the function body. Since we passed a reference we still have the same array outside in the global scope. This is very advantageous in some programming applications. If we call the same function using a shallow copy of the array, then we will lose the data and won’t have the same value outside the function call. Shallow copy is done by as seen in the coding file: func7(array[:])

**Javascript:** (Please run in a webpage)

Note: everything is executed in the main, so to see how specific things are working comment code excluding that function. Each function is labelled with its comment to see what it is investigating.

**File name:** haque\_faaiz\_javascript.html

**Code:**

**Outputs:**

In func1 result: 5

In func1 result: 5

In func2 result: 5

In func2 result: 7

In func3 x: 10

After func3 call, x: 3

In func4 x[2]: 4

After func4 call, x[2]: 4

**Explanations:**

In func1(): I investigated the parameter correspondence. I called the func1 twice where 2 arguments are given and variable 2 is subtracted from variable1. (variable1-variable2) and printed the result. In the case of the first func1 call I gave two values, and the positional method was applied where the first value corresponds to the first argument etc, so a becomes 10 and b becomes 5 in the code example. This is a safe and efficient method. For the keyword correspondence it does not work in javascript. In my example in the second func1 call: func1(b=10,a=5) where a is defined first and b is defined second. It still took a’s value as 10 and b’s as 5 . So we can conclude javascript doesn’t support keyword correspondence.

In func2(): I investigated the parameter default values. I first created func2 with arguments and specified values of a=10 and b=5. When first calling, I didn’t specify any values for a and b, and by default the result was given 5, as the code body is the same as f1’s. So the default values declared in the function’s definition were used. When specifying one value, the first value is overwritten like in python, so if I passed 12 as one argument it took a as 12 and b took its default value giving a result of 7. Again a limitation is that the second value cannot be changed without changing its preceding value.

In func3(): I investigated passing by value. I passed a value of x=3 outside the function in main scope, and inside the function I changed its value to 10, and inside the function it printed as 10. When leaving the function call in main scope, the value of x wasn’t changed since we passed by value. So it was still 10 outside the function

In func4(): I investigated passing by reference. In javascript you can only stimulate pass by reference on arrays and objects. So I passed an array and incremented its 3rd element by 1. And when leaving the function call the element remained the same since we incremented through its reference.

**PHP:** (Please run using php filename.php in dijkstra)

Note: everything is executed in the main, so to see how specific things are working comment code excluding that function. Each function is labelled with its comment to see what it is investigating.

**File name:** haque\_faaiz\_php.php

**Code:**

<?php

function func1( $a, $b ) #Parameter correspondence

{

$result = $a - $b;

print "In func1 result: $result\n";

}

func1(10,5); #Positional

func1($b=10,$a=5); #Keyword

function func2($a = 10, $b = 5) #Default values

{

$result = $a - $b;

print "In func2 result: $result\n";

}

func2(); #Default values used

func2(5);

function func3($a, $b ) #Pass by return

{

$result = $a - $b;

return $result;

}

$result = func3(3,2);

print "After func3 call result: $result\n";

function func4($x) #Pass by value

{

$x = 10;

print "x in func4: $x \n";

}

$x = 5;

func4($x);

print "x after func4 call: $x \n";

function func5(&$x) #Pass by reference

{

$x = 10;

print "x in func5: $x \n";

}

$x = 5;

func5($x);

print "x after func5 call: $x \n";

function func6() #Variable Number Of Actual Parameters

{

var\_dump(func\_num\_args());

var\_dump(func\_get\_args());

}

$array = array(5, hello, cat );

call\_user\_func\_array('func6',$array);

?>

**Outputs:**

*In func1 result: 5*

*In func1 result: 5*

*In func2 result: 5*

*In func2 result: 0*

*After func3 call result: 1*

*x in func4: 10*

*x after func4 call: 5*

*x in func5: 10*

*x after func5 call: 10*

*PHP Notice: Use of undefined constant hello - assumed 'hello' in /home/cs/faaiz.haque/guvenir/public\_html/haque\_faaiz\_php.php on line 53*

*PHP Notice: Use of undefined constant cat - assumed 'cat' in /home/cs/faaiz.haque/guvenir/public\_html/haque\_faaiz\_php.php on line 53*

*int(3)*

*array(3) {*

*[0]=>*

*int(5)*

*[1]=>*

*string(5) "hello"*

*[2]=>*

*string(3) "cat"*

*}*

**Explanations:**

In func1(): I investigated the parameter correspondence of parameters. I have a body where 2 arguments are subtracted from one another and the result is printed. A-b, where a is the first parameter and b is the second. I called the function twice, in the first call I gave values 10, 5 and a took the result of 10 and b took the result of 5 showing the positional correspondence. This method is quite safe and effective. In my second function call I tried to set b=10, and a =5 in this order but since keyword correspondence is not supported in php it still took a as 10 and b as 5 giving results 5 in both cases. This method is very limited since the programmer has to know each variable name in order to execute it. This could be a reason why php didn’t include this method in its implementation

In func2(): I investigated the formal parameter default values. I set two values already given in the function definition as a 10 and b 5. The code of func2 was a replica of func1. So then I called the function without specifying any values. In this case it took the default values of 10 and 5 specified. I then called with one value specified, which overwrote the default value of the first parameter, being a. The value of b took its old default value. A limitation to be noticed is we cannot change the value of b and leave a to its default value.

In func3(): I simply investigated the passing of parameters through return. When we return a variable that was computed in the function body this variable can be used in the global scope out of the function call, and set to other variables.

In func4(): I investigated pass-by-value, where a parameter is passed and a copy of its value is simply passed. IN my example, I passed the variable x with an initial value of 5, then in the function body I attempt to change this value to 10. In the function I print this variable and get the value of 10, however upon leaving the function’s scope, in the global scope trying to print the same variable we will not get 10 since we did not change the function defined in the main scope. We will retain the variables old value of 5, this is pass-by value

In func5(): I investigated pass-by-reference using the key character: &. Now with the replica of the function body of func4 where we investigated pass-by-value, in this case in the function’s definition the argument x has the reference keyword before it. Now when we change the variable x which was initially 5, to 10 in the function body, upon leaving the function definition we again get 10 outside the function call. This is because we passed a reference and when we changed the variable we were changing the same variable that was referred in the global scope.

In func6(): I investigated variable number of actual parameters. In this case an array is supplied as an argument to a predefined function called call\_user\_func\_array. This returns the number of each actual parameter in the array. For the number 5 it showed its type int, and the value. For the strings it showed the amount of characters and their values. This function is used by aiding through predefined functions: func\_get\_num and func\_get\_args.

**References:**

<https://stackoverflow.com/>

<http://php.net/manual/en/functions.arguments.php>

<https://www.tutorialrepublic.com/php-tutorial/php-functions.php>

<http://speakingjs.com/es5/ch15.html>

<https://chortle.ccsu.edu/java5/Notes/chap34A/ch34A_3.html>

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array>

<https://www.python-course.eu/passing_arguments.php>

<http://effbot.org/zone/default-values.htm>

https://docs.quantifiedcode.com/python-anti-patterns/correctness/mutable\_default\_value\_as\_argument.html