BILKENT UNIVERSITY

DEPARTMENT OF COMPUTER ENGINEERING

CS315

HOMEWORK 2

BERAT BİÇER

21503050

11.12.2017

ANKARA, TURKEY

JavaScript

Parameter Correspondence

```
pContent += "----- Parameter Correspondance -----\n";

var a = 10,

	b = "Berat",

	c = "123" + 4,

	d = "812" - 57,

	e = "Bicer";

correspondance(a, b, c, d, e);

correspondance(b, c, d, e, a);

correspondance(c, d, e, a, b);

correspondance(d, e, a, b, c);

correspondance(e, a, b, c, d);

function correspondance(a, b, c, d, e){

	pContent += "correspondance: a = " + a + "\n";

	pContent += "correspondance: b = " + b + "\n";

	pContent += "correspondance: c = " + c + "\n";

	pContent += "correspondance: d = " + d + "\n";

	pContent += "correspondance: e = " + e + "\n";

	pContent += "\n";

}

pContent += "\n";
```

This JavaScript code piece shows the order of arguments is the key to determine parameter correspondence. JS does not allow correspondence with keywords, so this is the only way to pass arguments to a function. As we change the order of parameters given into the function, we observe parameters' values change.

Here's the output:

```
correspondance: a = 10
correspondance: b = Berat
correspondance: c = 1234
correspondance: d = 755
correspondance: e = Bicer

correspondance: a = Berat
correspondance: b = 1234
correspondance: c = 755
```

```
correspondance: d = Bicer
correspondance: e = 10

correspondance: a = 1234
correspondance: b = 755
correspondance: c = Bicer
correspondance: d = 10
correspondance: e = Berat

correspondance: a = 755
correspondance: b = Bicer
correspondance: c = 10
correspondance: d = Berat
correspondance: e = 1234

correspondance: a = Bicer
correspondance: b = 10
correspondance: c = Berat
correspondance: d = 1234
correspondance: e = 755
```

Default Values of Formals

In JS, one can define default value of a function argument by assigning a value to it in the declaration of the function. Then, if no parameters are given for that variable, its default value is used.

```
function defaults(a = 10, b = "Berat", c = "123" + 4, d = "812" - 57, e = "Bicer") {

	pContent += "defaults: a = " + a + "\n";

	pContent += "defaults: b = " + b + "\n";

	pContent += "defaults: c = " + c + "\n";

	pContent += "defaults: d = " + d + "\n";

	pContent += "defaults: e = " + e + "\n";

	pContent += "\n";

}

a = "18983", b = "masjdm", c = "bekjhr" + "hjakld", d = '7123' + 894, e = "kdmahs";

defaults(a, b, c, d, e);

defaults(a, b, c, d);

defaults(a, b, c);

defaults(a, b);

defaults(a);

defaults();
```

If the user neglects to enter a variable, the system uses the default version. That's why in the last call, there is no user-defined value for any of the parameters. Here's the output:

```
defaults: a = 18983

defaults: b = masjdm

defaults: c = bekjhrhjakld

defaults: d = 7123894

defaults: e = kdmahs


defaults: a = 18983

defaults: b = masjdm

defaults: c = bekjhrhjakld

defaults: d = 7123894

defaults: e = Bicer


defaults: a = 18983

defaults: b = masjdm

defaults: c = bekjhrhjakld

defaults: d = 755

defaults: e = Bicer


defaults: a = 18983

defaults: b = masjdm

defaults: c = 1234

defaults: d = 755

defaults: e = Bicer


defaults: a = 18983

defaults: b = Berat

defaults: c = 1234

defaults: d = 755

defaults: e = Bicer
```

```
defaults: a = 10

defaults: b = Berat

defaults: c = 1234

defaults: d = 755

defaults: e = Bicer
```

Variable Number of Actuals

JS defines an array named “arguments” for any function after invocation. This array includes the parameters given to the function. This implies any number of arguments can be provided to the function, and they are both accessible inside the function with “arguments”. This is the formal way of providing variable number of actuals to a function.

```
a = 237, b = [12,2545,53002,66454], c = 173*8975 - 8023/32 + 8292, d = "berat", e = false;

function vParameter(a){

    for (i = 0; i < arguments.length; i++) {

        if (["number", "string", "boolean", "object"].includes(typeof arguments[i]))

            pContent += "arguments[i]: " + typeof arguments[i] + "; value: " + arguments[i] + "\n";

    }

    pContent += "\n";

}

vParameter(a);

vParameter(a, b);

vParameter(a, b, c);

vParameter(a, b, c, d);

vParameter(a, b, c, d, e);
```

By accessing parameters with “arguments”, we are not concerned about the number of parameters or how to access them directly using their names. Instead, we use “arguments” array.

Here’s the output:

```
arguments[i]: number; value: 237
```

```
arguments[i]: number; value: 237
arguments[i]: object; value: 12,2545,53002,66454

arguments[i]: number; value: 237
arguments[i]: object; value: 12,2545,53002,66454
arguments[i]: number; value: 1560716.28125

arguments[i]: number; value: 237
arguments[i]: object; value: 12,2545,53002,66454
arguments[i]: number; value: 1560716.28125
arguments[i]: string; value: berat

arguments[i]: number; value: 237
arguments[i]: object; value: 12,2545,53002,66454
arguments[i]: number; value: 1560716.28125
arguments[i]: string; value: berat
arguments[i]: boolean; value: false
```

Parameter Passing Methods

JS uses pass-by-value when passing parameters, and important scenarios are as follows:

```
a = {foo: "Tnjd", goo: "AbGwesad"},
b = {foo: "e5g32f", goo: 456},
c = {foo: "9FNjW", goo: false},
d = {foo: 21503050, goo: 05030512},
e = {foo: "78d=msnf1JP1", goo: "751djbn89MsthQlfHF"},
paraPass(a, b, c, d, e); // Due to hoisting, this is placed before the declaration
function paraPass(a, b, c, d, e) {
      a.foo = "bicerberat";
      b.foo = 123456;
      c.goo = 987654;
```

```
        d = {foo: 564, goo: 1562};
        e = {foo: "yjQknIkne3", goo: "KMnmsdOPIms-nb29"};
        pContent += "paraPass: a.foo = " + a.foo + "\n";
        pContent += "paraPass: a.goo = " + a.goo + "\n";
        pContent += "paraPass: b.foo = " + b.foo + "\n";
        pContent += "paraPass: b.goo = " + b.goo + "\n";
        pContent += "paraPass: c.foo = " + c.foo + "\n";
        pContent += "paraPass: c.goo = " + c.goo + "\n";
        pContent += "paraPass: d.foo = " + d.foo + "\n";
        pContent += "paraPass: d.goo = " + d.goo + "\n";
        pContent += "paraPass: e.foo = " + e.foo + "\n";
        pContent += "paraPass: e.goo = " + e.goo + "\n";
        pContent += "\n";
}
pContent += "main of paraPass: a.foo = " + a.foo + "\n";
pContent += "main of paraPass: a.goo = " + a.goo + "\n";
pContent += "main of paraPass: b.foo = " + b.foo + "\n";
pContent += "main of paraPass: b.goo = " + b.goo + "\n";
pContent += "main of paraPass: c.foo = " + c.foo + "\n";
pContent += "main of paraPass: c.goo = " + c.goo + "\n";
pContent += "main of paraPass: d.foo = " + d.foo + "\n";
pContent += "main of paraPass: d.goo = " + d.goo + "\n";
pContent += "main of paraPass: e.foo = " + e.foo + "\n";
pContent += "main of paraPass: e.goo = " + e.goo + "\n";
```

When accessing the content of an object, JS follows the pointer logic: it is similar to pass by reference, or in other words the change inside the function made by accessing class attributes is visible outside the function scope, as if we're dereferencing a pointer. However, pass-by-value logic still holds: if we try to create a new object, and just like with the pointers, try to assign the pointer a new address to point, this change happens locally and is not visible outside the function. Therefore, "`c.goo = 987654;`" is visible from the outside whereas "`d = {foo: 564, goo: 1562};`" is not. Other than this issue with objects, regular functions accepting parameters do not affect the outer, actual parameters because of pass-by-value logic.

Here's the output:

```
paraPass: a.foo = bicerberat
paraPass: a.goo = AbGwesad
paraPass: b.foo = 123456
paraPass: b.goo = 456
paraPass: c.foo = 9FNjW
paraPass: c.goo = 987654
paraPass: d.foo = 564
paraPass: d.goo = 1562
paraPass: e.foo = yjQknIkne3
paraPass: e.goo = KMnmsdOPIms-nb29

main of paraPass: a.foo = bicerberat
main of paraPass: a.goo = AbGwesad
main of paraPass: b.foo = 123456
main of paraPass: b.goo = 456
main of paraPass: c.foo = 9FNjW
main of paraPass: c.goo = 987654
main of paraPass: d.foo = 21503050
main of paraPass: d.goo = 1323338
main of paraPass: e.foo = 78d=msnf1JP1
main of paraPass: e.goo = 751djbn89MsthQlfHF
```

PHP

Parameter Correspondence

In PHP, actuals are assigned to formals with order, and this is the only way of parameter correspondence.

```
$var1 = 679;

$var2 = "bErAt";

$var3 = "278kjndDSGUJmsdbkq-dsajb28kj";

$var4 = 882*4-2456;

$var5 = 673.52;

function paraCorr($a, $b, $c, $d, $e){

      echo "paraCorr: \$a = $a, \$b = $b, \$c = $c, \$d = $d, \$e = $e" .
"<br/>";

}

paraCorr($var1, $var2, $var3, $var4, $var5);

paraCorr($var2, $var3, $var4, $var5, $var1);

paraCorr($var3, $var4, $var5, $var1, $var2);

paraCorr($var4, $var5, $var1, $var2, $var3);

paraCorr($var5, $var1, $var2, $var3, $var4);
```

Here's the output:

```
paraCorr: $a = 679, $b = bErAt, $c = 278kjndDSGUJmsdbkq-dsajb28kj, $d = 1072, $e = 673.52

paraCorr: $a = bErAt, $b = 278kjndDSGUJmsdbkq-dsajb28kj, $c = 1072, $d = 673.52, $e = 679

paraCorr: $a = 278kjndDSGUJmsdbkq-dsajb28kj, $b = 1072, $c = 673.52, $d = 679, $e = bErAt

paraCorr: $a = 1072, $b = 673.52, $c = 679, $d = bErAt, $e = 278kjndDSGUJmsdbkq-dsajb28kj

paraCorr: $a = 673.52, $b = 679, $c = bErAt, $d = 278kjndDSGUJmsdbkq-dsajb28kj, $e = 1072
```

When the order of actuals is changed, we see the values of formals change too.

Default Values of Formals

As with JS, we can provide the default value of a parameter in the function declaration:

```
$var1 = 679;

$var2 = "bErAt";

$var3 = "278kjndDSGUJmsdbkq-dsajb28kj";

$var4 = 882*4-2456;

$var5 = 673.52;

function defPara($a ="berat", $b = "bicer", $c = "21503050", $d = "section3", $e = "cs315"){

      echo "defPara: \$a = $a, \$b = $b, \$c = $c, \$d = $d, \$e = $e" . "<br/>";

}

defPara();

defPara($var1);

defPara($var1, $var2);

defPara($var1, $var2, $var3);

defPara($var1, $var2, $var3, $var4);

defPara($var1, $var2, $var3, $var4, $var5);
```

If the system does not see a custom actual from the call corresponding to a default argument, then the default value is used.

Here's the output:

```
defPara: $a = berat, $b = bicer, $c = 21503050, $d = section3, $e = cs315

defPara: $a = 679, $b = bicer, $c = 21503050, $d = section3, $e = cs315

defPara: $a = 679, $b = bErAt, $c = 21503050, $d = section3, $e = cs315

defPara: $a = 679, $b = bErAt, $c = 278kjndDSGUJmsdbkq-dsajb28kj, $d =
section3, $e = cs315

defPara: $a = 679, $b = bErAt, $c = 278kjndDSGUJmsdbkq-dsajb28kj, $d = 1072,
$e = cs315

defPara: $a = 679, $b = bErAt, $c = 278kjndDSGUJmsdbkq-dsajb28kj, $d = 1072,
$e = 673.52
```

Variable Number of Actuals

There are 2 major ways to do this in PHP: either wrap all of the arguments into an array and pass the array as the argument or use built-in functions to access parameters.

1)

```
$var1 = 679;

$var2 = "bErAt";

$var3 = "278kjndDSGUJmsdbkq-dsajb28kj";

$var4 = 882*4-2456;

$var5 = 673.52;

function varArgArray($var){

      for($i = 0; $i < sizeof($var); $i++){

            echo "varArgArray: \$var$i = " . $var[$i] . "<br/>";

      }

}

# size = 1

$arr = [$var1];

varArgArray($arr);

# size = 2

$arr = [$var1, $var2];

varArgArray($arr);

# size = 3
```

```
$arr = [$var1, $var2, $var3];

varArgArray($arr);

# size = 4

$arr = [$var1, $var2, $var3, $var4];

varArgArray($arr);

# size = 5

$arr = [$var1, $var2, $var3, $var4, $var5];

varArgArray($arr);
```

Here, we create an array each time we want to pass multiple parameters to the function. Then, using a for loop, we access individual parameters if necessary. This method is useful if the parameter array is not large. Also, it is easier to write than the other method.

Here's the output:

```
varArgArray: $var0 = 679


varArgArray: $var0 = 679

varArgArray: $var1 = bErAt


varArgArray: $var0 = 679

varArgArray: $var1 = bErAt

varArgArray: $var2 = 278kjndDSGUJmsdbkq-dsajb28kj


varArgArray: $var0 = 679

varArgArray: $var1 = bErAt

varArgArray: $var2 = 278kjndDSGUJmsdbkq-dsajb28kj

varArgArray: $var3 = 1072


varArgArray: $var0 = 679

varArgArray: $var1 = bErAt

varArgArray: $var2 = 278kjndDSGUJmsdbkq-dsajb28kj

varArgArray: $var3 = 1072

varArgArray: $var4 = 673.52
```

2)

```
function varArgWithBuiltIn(){
      for($i = 0; $i < func_num_args(); $i++){
            echo "varArgWithBuiltIn: \$var$i = " . func_get_arg($i) . "<br/>";
      }
}
# empty arguments
varArgWithBuiltIn();
# size = 1
$arr = [$var1];
varArgWithBuiltIn($var1);
# size = 2
varArgWithBuiltIn($var1, $var2);
# size = 3
varArgWithBuiltIn($var1, $var2, $var3);
# size = 4
varArgWithBuiltIn($var1, $var2, $var3, $var4);
# size = 5
varArgWithBuiltIn($var1, $var2, $var3, $var4, $var5);
```

There are 3 important built-in functions used to have variable number of actuals: `func_num_args` which returns the amount of parameters the function is called with, `func_get_arg($i)` accesses the ith parameter and `func_get_args` returns an array of parameters of the function. This method works similar to the first one, however it is more difficult to write.

Here's the output, identical to the previous one:

```
varArgWithBuiltIn: $var0 = 679


varArgWithBuiltIn: $var0 = 679
varArgWithBuiltIn: $var1 = bErAt


varArgWithBuiltIn: $var0 = 679
```

```
varArgWithBuiltIn: $var1 = bErAt
varArgWithBuiltIn: $var2 = 278kjndDSGUJmsdbkq-dsajb28kj

varArgWithBuiltIn: $var0 = 679
varArgWithBuiltIn: $var1 = bErAt
varArgWithBuiltIn: $var2 = 278kjndDSGUJmsdbkq-dsajb28kj
varArgWithBuiltIn: $var3 = 1072

varArgWithBuiltIn: $var0 = 679
varArgWithBuiltIn: $var1 = bErAt
varArgWithBuiltIn: $var2 = 278kjndDSGUJmsdbkq-dsajb28kj
varArgWithBuiltIn: $var3 = 1072
varArgWithBuiltIn: $var4 = 673.52
```

Parameter Passing

PHP is by default a pass-by-value language. However, there are some cases where the logic might be confusing, for example:

```
# Pass by Value
function paraPassDefault($a) {
      $a = 123;
      echo "paraPassDefault: \$a = $a";
      echo "<br/>";
}
# Pass by Reference
function paraPassReference(&$a) {
      $a = 456;
      echo "paraPassReference: \$a = $a";
      echo "<br/>";
}
# Dereference pointer
function paraPassDeref($a){
```

```
	$a->foo = "berat";

	$a->goo = "bicer";

	echo "paraPassDeref: ";

	$a->printData();

	echo "<br/>";

}

# New local object

function paraPassNewObj($a){

	$a = new myObj();

	$a->setPara("berat", "bicer");

	echo "paraPassNewObj: ";

	$a->printData();

	echo "<br/>";

}
```

First function is a regular pass-by-value function: changes are not visible outside the function's scope. Second function uses "&" in front of its parameter, enforcing it to be a pass-by-reference. Therefore, any changes on its parameter is visible outside its scope. Third function is similar to dereferencing a pointer: we access the inner attributes of an object. Therefore, these changes will be visible outside the function scope. Lastly, creating a new location for its parameter to "point" to is done locally, and does not affect the outer actual.

Here's the output:

```
main of paraPassDefault before: $var1 = 3663bdPl

paraPassDefault: $a = 123

main of paraPassDefault after: $var1 = 3663bdPl


main of paraPassReference before: $var2 = 67hahUk

paraPassReference: $a = 456

main of paraPassReference after: $var2 = 456


main of paraPassDeref before: $var3->foo = yI5/5kjdU7; $var3->goo = 428

paraPassDeref: printData: foo = berat; goo = bicer

main of paraPassDeref after: $var3->foo = berat; $var3->goo = bicer
```

```
main of paraPassNewObj before: $var4->foo = Dsa47j-is5M; $var4->goo = 9371

paraPassNewObj: printData: foo = berat; goo = bicer

main of paraPassNewObj after: $var4->foo = Dsa47j-is5M; $var4->goo = 9371
```

Python

Parameter Correspondence and Default Parameter Values

Python allows keyword based and positional parameter correspondence:

```
def paraCorrDef(a, b, c = 67):
      print("paraCorrDef: a = {0}, b = {1}, c = {2}".format(a, b, c))
      return;

paraCorrDef(17, c = "5:}", b = 5)

paraCorrDef(9, "yH53-b", 88)

paraCorrDef("7-4Bjs7", b = 46)

paraCorrDef("berat", "bicer")

paraCorrDef(b = 5282, a = "iHyhN7/6%", c = 97)

paraCorrDef(b = 6465, a = 3314)
```

First case assigns a to 17, c to “5:}” and b to 5 using keywords and positions together.

Second case assigns using positions: a to 9, b to “yH53-b” and c to 88.

Third case shows how default values are useful: by its position a is “7-4Bjs7” and by its keyword b is 46. Since value of c is not defined, c becomes the default value which is 67.

Fourth case is similar to third case, except both a and b are positionally assigned.

In the fifth case, we use keywords for all three parameters and assign b to 5282, a to “iHyhN7/6%” and c to 97, ignoring their positions.

In the last case, we use keywords to assign a and b, but c is assigned automatically to its default value.

Here’s the output:

```
paraCorrDef: a = 17, b = 5, c = 5:}

paraCorrDef: a = 9, b = yH53-b, c = 88

paraCorrDef: a = 7-4Bjs7, b = 46, c = 67

paraCorrDef: a = berat, b = bicer, c = 67

paraCorrDef: a = iHyhN7/6%, b = 5282, c = 97
```

```
paraCorrDef: a = 3314, b = 6465, c = 67
```

Variable Number of Actuals

Python uses tuples to pass multiple arguments to functions:

```
def varArgs(*args):
      index = 0
      for var in args:
            print ("varArgs: var", index, " = ", var)
            index = index + 1
      return;
def varArgsDict(**args):
      for var in args.keys():
            print( "varArgsDict: ", var, " = ", args[var])
      return;
varArgs(1, 2, 3, 4, 5, 6)
varArgs(1, 2, 3)
print ("\n")
varArgsDict(a="1")
varArgsDict(a="1", b="2")
varArgsDict(a="1", b="2", c="3")
varArgsDict(a="1", b="2", c="3", d="4")
```

Tuples can be in array form or in dictionary form. We can access the parameters using the tuple in for loops or by direct indexing.

Here's the output:

```
varArgs: var 0  =  1
varArgs: var 1  =  2
varArgs: var 2  =  3
varArgs: var 3  =  4
varArgs: var 4  =  5
varArgs: var 5  =  6
```

```
varArgs: var 0  =  1

varArgs: var 1  =  2

varArgs: var 2  =  3


varArgsDict:  a  =  1

varArgsDict:  b  =  2

varArgsDict:  a  =  1

varArgsDict:  c  =  3

varArgsDict:  b  =  2

varArgsDict:  a  =  1

varArgsDict:  c  =  3

varArgsDict:  b  =  2

varArgsDict:  a  =  1

varArgsDict:  d  =  4
```

Parameter Passing

If we pass a mutable reference to a function, then the program behaves as if its pass by reference, and the changes on that mutable object is visible outside the function's scope. If we change the reference, refer it to a new object, this change is local and invisible from the outside. Also, if we pass an immutable reference to a function and change it, this change too is local and invisible from the outside. Other than these cases, Python uses pass-by-value.

```
def passMutable(x):

      x.append(5)

      x.append(7)

      x.append(87)

      print("passMutable: x = ", x)

      return;


def passChangeMutableRef(x):

      x = ["berat", "bicer", 21503050, "sec3"]

      print("passChangeMutableRef: x = ", x)
```

```
	return;


def passImmutableRef(x):
	x = x, ":)$83883 syje euejke 17728"
	print("passImmutableRef: x = : ", x)
	return;


var = ["126","26$2","2$2$2772"]
print("main of passMutable before: var = ", var)
passMutable(var)
print("main of passMutable after: var = ", var)
print ("\n")


var = ["126", "26$2", "2$2$2772"]
print("main of passChangeMutableRef before: var = ", var)
passChangeMutableRef(var)
print("main of passChangeMutableRef after: var = ", var)
print ("\n")


var = "beratbicer"
print("main of passImmutableRef before: var = ", var)
passImmutableRef(var)
print("main of passImmutableRef after: var = ", var)
print ("\n")
```

Here's the output:

```
main of passMutable before: var =  ['126', '26$2', '2$2$2772']
passMutable: x =  ['126', '26$2', '2$2$2772', 5, 7, 87]
main of passMutable after: var =  ['126', '26$2', '2$2$2772', 5, 7, 87]


main of passChangeMutableRef before: var =  ['126', '26$2', '2$2$2772']
```

```
passChangeMutableRef: x =  ['berat', 'bicer', 21503050, 'sec3']
main of passChangeMutableRef after: var =  ['126', '26$2', '2$2$2772']


main of passImmutableRef before: var =  beratbicer
passImmutableRef: x = :  ('beratbicer', ':)$83883 syje euejke 17728')
main of passImmutableRef after: var =  beratbicer
```

Sources:

- http://docs.php.net/manual/da/functions.arguments.php#functions.arguments.by-reference
- https://www.w3schools.com/js/js_function_parameters.asp
- https://www.w3schools.com/js/js_function_invocation.asp
- https://www.w3schools.com/js/js_function_call.asp
- https://www.w3schools.com/js/js_function_apply.asp
- https://stackoverflow.com/questions/894860/set-a-default-parameter-value-for-a-javascript-function
- https://stackoverflow.com/questions/986006/how-do-i-pass-a-variable-by-reference