CS 315

Programming Languages

HW2

(Subprogram Parameters)

Balaj Saleem

21701041

Section 2

Description:

During this homework 5 languages: Dart, Python, Perl, Javascript and PHP, were investigated to determine how parameters are passed to subprograms. The following items were investigated

- Parameter Correspondence
    - positional
    - Keyword
    - Combination
- Formal Parameter[1] Default Values
- Variable Number of Actual Parameters
- Parameter passing methods (checked for the following, where applicable)
    - Primitive Types
    - Arrays/Lists
    - Objects of classes

Dart:

For parameter correspondence dart has the following:

- Positional correspondence
    - Parameters can be entered in the order they were positioned as formal parameters in function definition. Can be seen at line 12 and line 49. These are compulsory.
- Keyword correspondence
    - If the formal parameters are optional and name addressable i.e. they appear inside {} , they can be addressed by name in any order. For instance the function fun3(line 25) given below has optional name addressable formal parameters, and on line 58, where the third parameter is addressed by name. These are optional.
- Combinational correspondence
    - As given in line 58 combinations can be used to address variables, also optional.

Formal parameters in dart that are optional i.e. they are included in [] or {}. They can be assigned default values, however if they are not optional they can not have default values.

In a very specific way dart can have a somewhat variable number of actual parameters(e.g line 52 and 53) by including optional parameters such as those on lines 12 and 25. However this is still a limited number of parameters and this can not be a list or array of parameter unlike other languages, provided below.

All primitive types in dart are passed by value[2] i.e only in mode.In place operations for arrays/lists change the original array, and maybe considered to be passed by reference[2] i.e in-out mode. However, assignments do not change the reference Objects of classes are passed as references i.e. in-out mode. Functions can also be passed to functions(line 43 and 77).

*saleem_balaj_dart.dart | saleem_balaj_javascript.html | saleem_balaj_python.py | saleem_balaj_

```dart
//only optional parameters can have default values, if no default value is assigned, it is null
//classes passed by reference
class Tester{
   int testNumber = 0;
}
//functions can be passed as parameters. This is used in fun5
cuber(int i){
   return i*i*i;
}
//simple positional parameter correspondense
//fun1(int a=1, int b=2, int c=3){ //error since non optional variables can not have default values
fun1(int a, int b, int c){
   print("---FUN1---");
   print("a is: $a b is: $b c is: $c");
}
//[]allows for optional parameters can not be accessed by name,
//positional parameters can not be added after this
fun2(int a, [int b, int c = 3] ){
   print("---FUN2---");
   print("a is: $a b is: $b c is: $c");
}
//variable lenght arguments not possible
//{} allows for optional parameters that must be addressed by name,
//however positional parameters can not be added after this
fun3(int a, {int b = 2, int c = 3} ){
   print("---FUN3---");
   print("a is: $a b is: $b c is: $c");
}
//test for passing methods, only in mode - pass by value - possible for primitive types,
//objects are passed by reference - in out mode -
//fomral variables DO NOT need to have types
fun4( a, array, obj){
   print("---FUN4---");
   a = a*a;
   //array += [99,99,99]; //does not affect the orignial array
   array.add(99);
   obj.testNumber = 9999;
   print("a inside fun4: $a");
   print("array inside fun4: $array");
   print("object inside fun4: ${obj.testNumber}");
}
//functions can be passed as arguments in dart
fun5(f, i){
   print("---FUN5---");
   print(f(i));
}
void main() {

   fun1(1,2,3); //simple positional correspondence
   //fun1(a=1,c=2,d=3); error, cannot address variables by names unless these variables are enclosed by {}

   fun2(1);
   fun2(1,99); // b and c are optional, here only b is provided
   //fun2() //error since the first argument is needed, it is not optional
   //fun2(1, b:99); //error since the variables are optional but not name addressable by {}


   fun3(1, c:99 ); //addresses the variable c by name, but a by position, since b is optional and not provided, it is null
   fun3(1); //WORKS, as the parameters b,c are optional, just as demonstrated above with "fun2(1);"
   //fun3(1,2,3); //error, the parameters enclosed by {} can not be provided positionally
   //fun3(1, d:99); //error, since no parameter with name d exists
   //check if the it is pass by value/or pass by result(which should give error, since a is not declared)

   //declaring different data types to check their values after a function, i.e whether the function uses, in , out , or inout methods
   int a = 3;
   var array = [0,0,0];
   Tester obj = new Tester();
   print("----------");
   print("a before fun4: $a");
   print("array before fun4: $array");
   print("object variable before fun4: ${obj.testNumber}");
   fun4(a,array,obj);
   print("a after fun4: $a");
   print("array after fun4: $array");
   print("object variable after fun4: ${obj.testNumber}");

   fun5(cuber, 2); //passing a function(cuber) to another function as a parameter
}
```

Output:

```
[balaj.saleem@dijkstra saleem_balaj]$ dart saleem_balaj_dart.dart
---FUN1---
a is: 1 b is: 2 c is: 3
---FUN2---
a is: 1 b is: null c is: 3
---FUN2---
a is: 1 b is: 99 c is: 3
---FUN3---
a is: 1 b is: 2 c is: 99
---FUN3---
a is: 1 b is: 2 c is: 3
----------
a before fun4: 3
array before fun4: [0, 0, 0]
object variable before fun4: 0
---FUN4---
a inside fun4: 9
array inside fun4: [0, 0, 0, 99]
object inside fun4: 9999
a after fun4: 3
array after fun4: [0, 0, 0, 99]
object variable after fun4: 9999
---FUN5---
8
```

Javascript:

Actual parameters in javascript are not compulsory, i.e. the function runs without them with their values undefined(see output).

For parameter correspondence javascript has the following:

- Positional correspondence
    - Parameters can be entered in the order they were positioned as formal parameters in function definition. Can be seen at line 14 and line 60.
- Keyword correspondence
    - There is no native support for keyword/named correspondence of parameters.
- Combinational correspondence
    - Since keyword correspondence is not possible, neither is combinational one.

Formal parameters can be assigned default values[3] as done in function definition of fun2 (line 20). If no default value is assigned by the programmer they have a default value of ‘undefined’.

Actual parameters can be assigned an undefined value, and in this case the default value is used.

Variable number of parameters are supported(line 25, 67 and 68). The parameters are passed in an array called ‘arguments’.

All primitive types in Javascript are passed by value i.e only in mode.In place operations for arrays/lists change the original array, and maybe considered to be passed by reference i.e in-out mode. However, assignments do not change the reference. Objects of classes are passed as references i.e. in-out mode. Functions can also be provided as arguments.

```
 5 //class objects passed by reference
 6 class Tester {
 7      testVar = 0;
 8 }
 9 //functions can be passed as parameters. This is used in fun5
10 function cuber(i){
11    return i*i*i;
12 }
13 //simple positional parameter correspondense
14 function fun1( a,  b,  c){
15    console.log ("---FUN1---");
16    console.log ("a is: " + a + " b is: " + b + " c is: " + c);
17 }
18 //default values can be assigned
19 //can have default values in any order
20 function fun2( a,  b=3,  c=3, d ){
21    console.log ("---FUN2---");
22    console.log ("a is: " + a + " b is: " + b + " c is: " + c + " d is: " + d);
23 }
24 //variable number of parameters can be provided
25 function fun3(){
26    console.log ("---FUN3---");
27    var parameters = ""; //a list to store all given parameters
28    console.log("total parameters provided: " + arguments.length);
29    for (var i = 0; i < arguments.length; i++){
30          parameters += arguments[i] + " | ";
31    }
32    console.log(parameters);
33 }
34 //test for passing methods, only in mode - pass by value - possible for primitive types,
35 //objects are passed by reference - in out mode -
36 //fomral variables DO NOT need to have types
37 function fun4( a, array, obj){
38    console.log ("---FUN4---");
39    a = a*a;
40    //array += [99,99,99]; //does not change the actual array provided in argument
41    array.push("99");
42    obj.testVar = 9999;
43    console.log ("a inside fun4: " + a);
44    console.log ("array inside fun4: " + array);
45    console.log ("object inside fun4: " + obj.testVar);
46 }
```

```
function fun5(f, i){
   console.log ("---FUN5---");
   console.log (f(i));
}
//rest parameter(must be last formal parameters)
function fun6(a,b, ...others){
   console.log ("---FUN6---");
   console.log ("a is: " + a);
   console.log ("b is: " + b);
   console.log ("rest of the arguments: " + others);
}
fun1(1,2,3); //simple positional correspondence
fun1(1); //only first variable assigned(positionally) rest are undefined

fun2(1); //default value for b and c used, while d is undefined
fun2(1,undefined,undefined,2) //works, for arguments that are undefined the default value is used
//fun2(a=1,c=3,b=2); //does not work as expected(works with positional correspondence), No native support for name

fun3(1,2,3,4,5,6);//variable number of arguments passed
fun3(1,2,3,4,5,6,7,8,9,10) //also works

//declaring different data types to check their values after a function, i.e whether the function uses, in , out ,
var a = 3;
var array = [0,0,0];
var obj = new Tester();
console.log ("----------");
console.log ("a before fun4: " + a);
console.log ("array before fun4: " + array);
console.log ("object before fun4: " + obj.testVar);
fun4(a,array,obj);
console.log ("a after fun4: " + a);
console.log ("array after fun4: " + array);
console.log ("object after fun4: " + obj.testVar);

fun5(cuber, 2);

fun6(1,2, 99,99,99,99,99)//positional and variable number of arguments(rest parameters)
</script>
</HEAD>
```

Output:

```
---FUN1---                                  saleem_balaj_javascript.html:16
a is: 1 b is: 2 c is: 3                     saleem_balaj_javascript.html:17
---FUN1---                                  saleem_balaj_javascript.html:16
a is: 1 b is: undefined c is: undefined     saleem_balaj_javascript.html:17
---FUN2---                                  saleem_balaj_javascript.html:22
a is: 1 b is: 3 c is: 3 d is: undefined     saleem_balaj_javascript.html:23
---FUN2---                                  saleem_balaj_javascript.html:22
a is: 1 b is: 3 c is: 3 d is: 2             saleem_balaj_javascript.html:23
---FUN3---                                  saleem_balaj_javascript.html:27
total parameters provided: 6                saleem_balaj_javascript.html:29
1 | 2 | 3 | 4 | 5 | 6 |                     saleem_balaj_javascript.html:33
---FUN3---                                  saleem_balaj_javascript.html:27
total parameters provided: 10               saleem_balaj_javascript.html:29
1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |  saleem_balaj_javascript.html:33
----------                                  saleem_balaj_javascript.html:74
a before fun4: 3                            saleem_balaj_javascript.html:75
array before fun4: 0,0,0                    saleem_balaj_javascript.html:76
object before fun4: 0                       saleem_balaj_javascript.html:77
---FUN4---                                  saleem_balaj_javascript.html:39
a inside fun4: 9                            saleem_balaj_javascript.html:44
array inside fun4: 0,0,0,99                 saleem_balaj_javascript.html:45
object inside fun4: 9999                    saleem_balaj_javascript.html:46
a after fun4: 3                             saleem_balaj_javascript.html:79
array after fun4: 0,0,0,99                  saleem_balaj_javascript.html:80
object after fun4: 9999                     saleem_balaj_javascript.html:81
---FUN5---                                  saleem_balaj_javascript.html:50
8                                           saleem_balaj_javascript.html:51
---FUN6---                                  saleem_balaj_javascript.html:55
a is: 1                                     saleem_balaj_javascript.html:56
b is: 2                                     saleem_balaj_javascript.html:57
rest of the arguments: 99,99,99,99,99       saleem_balaj_javascript.html:58
```

Python:

For parameter correspondence python has the following:

- Positional correspondence
    - Parameters can be entered in the order they were positioned as formal parameters in function definition. Can be seen at line 12 and line 45.

- Keyword correspondence
    - All formal parameters can be addressed by name/keyword when the function is being called this can be seen in line 46.
- Combinational correspondence
    - Variables can be called using both positional and keyword correspondence. All actual parameters with positional correspondence should be entered before the ones with keyword correspondence(line 53). Keyword correspondence does not need to follow any order(line 46), however positional does(line 45)

Formal parameters can be assigned default values(line 17). These variables are optional, however the ones that have not been assigned default values are compulsory.

Variable number of actual parameters are supported. Lists of variables and dictionaries can be provided in actual parameters. They must be declared using *x for list and **x for dictionary(line 22 and line 57).

All data values are objects in python and the method used is pass by assignment[4], which is an in-out method. Unless a new variable with the same name is declared in the scope. The value of the actual parameter is changed. In place operations on lists and other mutable types can change their actual values i.e. outer instance. All class objects also have their actual value changed in a function. Functions can also be provided as arguments.

```
class Tester:
    testNumber = 0

#functions can be passed as parameters. This is used in fun5
def cuber(i):
    return i*i*i

#simple positional parameter correspondense
def fun1(a, b, c):
    print("---FUN1---")
    print("a is:" , a , "b is:" , b , "c is:" , c)

#def fun2(a, b=3, c ): #error, non defualt argument cannot follow default one
def fun2(a, b=2, c=3 ):
    print("---FUN2---")
    print("a is:" , a , "b is:" , b , "c is:" , c)
#variable lenght of provided parameters, * for list and ** for dictionary,
#other arguments can not follow lists/dictionaries
def fun3(a, *b, **c):
    print("---FUN3---")
    print("a is:" , a , "b is:" , b , "c is:" , c)




#pass by assignment, all immutables changed, unless new ones are declared
def fun4(a, array, obj):
    print("---FUN4---")
    a = a*a
    array += [99,99,99]
    #array = [99,99,99] #will not change the list, instead it will create a new one, the outer instance of array will not be af
    obj.testNumber = 9999
    print("a inside fun4: " , a)
    print("array inside fun4: " , array)
    print("object variable inside fun4: " , obj.testNumber)

#functions can be passed as arguments in python
def fun5(f, i):
    print("---FUN5---")
    print(f(i))
```

```
def fun5(f, i):
    print("---FUN5---")
    print(f(i))

fun1(1,2,3) #simple positional correspondence
fun1(a=1,c=3,b=2) #works, since all required parameters are given, however if any of the three are missing we get an error
#fun1(a=1,b=2,c=2,d=3) #error, d is not declared as a formal parameter
#fun1(a=1,b=2) #error, c is not provided as an argument, and it does not have a default value

fun2(a=1, c=99)
fun2(1, c=999, b=99) # b and c are optional, can be provided in any order
fun2(1,2,3) #works since varibles have positional corespondence
#fun2(a=1, b=99, 99) #error, can not have positional corespondence after name corespondence
#fun2() #error since the first argument is needed, it is not optional
#fun2(1, b=99, 3) #error since the variables are optional but not name addressable by :

fun3(1, 2,3,4,5, c=99,d=999,f=9999) #takes a single parameter, a list and a dictionary(key-value pairs)

#declaring different data types to check their values after a function, i.e whether the function uses, in , out , or inout methods
a = 3
array = [0,0,0]
obj =  Tester
print("----------")
print("a before fun4: " , a)
print("array before fun4: " , array)
print("object variable before after fun4: " , obj.testNumber)
fun4(a,array,obj)
print("a after fun4: " , a)
print("array after fun4: " , array)
print("object variable after fun4: " , obj.testNumber)

fun5(cuber, 2) #passing a function(cuber) to another function as a parameter
```

Output:

```
[balaj.saleem@dijkstra saleem_balaj]$ python saleem_balaj_python.py
---FUN1---
('a is:', 1, 'b is:', 2, 'c is:', 3)
---FUN1---
('a is:', 1, 'b is:', 2, 'c is:', 3)
---FUN2---
('a is:', 1, 'b is:', 2, 'c is:', 99)
---FUN2---
('a is:', 1, 'b is:', 99, 'c is:', 999)
---FUN2---
('a is:', 1, 'b is:', 2, 'c is:', 3)
---FUN3---
('a is:', 1, 'b is:', (2, 3, 4, 5), 'c is:', {'c': 99, 'd': 999, 'f': 9999})
----------
('a before fun4: ', 3)
('array before fun4: ', [0, 0, 0])
('object variable before after fun4: ', 0)
---FUN4---
('a inside fun4: ', 9)
('array inside fun4: ', [0, 0, 0, 99, 99, 99])
('object variable inside fun4: ', 9999)
('a after fun4: ', 3)
('array after fun4: ', [0, 0, 0, 99, 99, 99])
('object variable after fun4: ', 9999)
---FUN5---
8
```

Perl

Perl has no formal parameters in the definition of the function

For parameter correspondence perl has the following:

- Positional correspondence
    - Parameters are provided in a list of arguments '$_' and this list stores every actual parameter in the order of its position. Parameters can be accessed using $_[position_of_actual_variable].
- Keyword correspondence
    - There is no native support for keyword correspondence.
- Combinational correspondence
    - Since there is no keyword correspondence there can be no combination of positional and keyword.

Since there are no formal parameters they cannot have default values.

Variable number of actual parameters are supported, and are all available in is an array '@_'.

All actual variables are passed using in-out methodology since aliases to the actual variables are available in the arguments array. This can be considered as pass by reference. However it should be noted that pass by value can be simulated using local variables(line 21).

```
1  #parameter list is passed to functions in $_
2  sub fun1{
3     print("---FUN1---\n");
4     print("arguments passed: \n");
5     foreach $item(@_){ #all actual parameters available in $_
6        print("$item ");
7     }
8     print("\n");
9
10 }
11 #all variables are passed by reference(inout), also called passed by alias
12 sub fun2{
13    print("---FUN2---\n");
14    foreach $item(@_){
15       $item++;
16    }
17    print("a in fun2: $_[0]      "); #first parameter passed is called a, it is first in param list
18    print("b in fun2: $_[1]\n"); #second one is b
19 }
20 #variables can be simulated to be passed by value(in)
21 sub fun3{
22    print("---FUN3---\n");
23    my $a = shift; #shift can be used to shift the argument array pointer ahead
24    my $b = shift;
25    $a++;
26    $b++;
27    print("a in fun3: $a      "); #first parameter passed is called a, it is first in param list
28    print("b in fun3: $b\n"); #second one is b
29 }
30 #no INBUILT mechanism to set default values \\ no formal parameters
31 #no INBUILT mechanism to assign actual parameters by name
32 fun1(1,2);          #simple positional correspondence
33 fun1(1,2,3,4,5,6); #variable number of actual parameters
34 my $a = 0;
35 my $b = 1;
36 print("------\n");
37 print("a before fun2: $a  ");
38 print("b before fun2: $b\n");
39 fun2($a,$b);
40 print("------\n");
41 print("a after fun2: $a   ");
42 print("b after fun2: $b\n");
43 print("------\n");
44 print("a before fun3: $a  ");
45 print("b before fun3: $b\n");
46 fun3($a,$b);
47 print("------\n");
48 print("a after fun3: $a   ");
49 print("b after fun3: $b\n");
```

Output:

```
[balaj.saleem@dijkstra saleem_balaj]$ perl saleem_balaj_perl_dynamic.pl
---FUN1---
arguments passed:
1 2
---FUN1---
arguments passed:
1 2 3 4 5 6
------
a before fun2: 0  b before fun2: 1
---FUN2---
a in fun2: 1      b in fun2: 2
------
a after fun2: 1   b after fun2: 2
------
a before fun3: 1  b before fun3: 2
---FUN3---
a in fun3: 2      b in fun3: 3
------
a after fun3: 1   b after fun3: 2
```

PHP

For parameter correspondence perl has the following:

- Positional correspondence
    - Parameters can be entered in the order they were positioned as formal parameters in function definition. Can be seen at line 11 and line 69.
- Keyword correspondence
    - There is no native support for keyword correspondence.
- Combinational correspondence
    - Since there is no keyword correspondence there can be no combination of positional and keyword.

Default values can be assigned to formal parameters however they should not be followed by non default values. These values are used when they are not provided in function call, for instance in line 73.

Variable number of actual parameters are supported there are two ways these can be incorporated into the formal parameters.

- Firstly, all arguments, even if they exceed the number of formal arguments (line 70) are available in an arguments array that can be received via the function: func_get_args(). They are available according to their position in this array.
- Secondly '...x]' format can be used to access certain formal parameters by name and others as a list of arguments(line 58 and 105). However, this should be the last formal parameter in the function definition.

Actual variables can be passed by value or by reference. By default all non object parameters are passed by value i.e. in method. However if the '&' operator preceded the formal parameter the actual value provided by reference i.e. in-out method. All objects are provided by reference. Functions can also be provided as arguments.

```php
<?php
//class objects passed by reference
class Tester {
    public $testVar=0;
}
//functions can be passed as parameters. This is used in fun5
function cuber($i){
   return $i*$i*$i;
}
//simple positional parameter correspondense
function fun1( $a,  $b,  $c){
   echo("---FUN1---\n");
   echo("a is: $a ");
   echo("b is: $b ");
   echo("c is: $c \n");
}
//default values can be assigned, but they must be last
function fun2( $a,  $b=99,  $c=99 ){
   echo("---FUN2---\n");
   echo("a is: $a ");
   echo("b is: $b ");
   echo("c is: $c \n");
}
//variable number of parameters can be provided
function fun3(){
   echo("---FUN3---\n");
   $totalArg = func_num_args();
   $arg_list = func_get_args();
   echo("total parameters provided: $totalArg \n");
   for ($i = 0; $i < $totalArg ; $i++) {
       echo("$arg_list[$i] ");
   }
   echo("\n");
}
//test for passing methods, a passed by value, b passed by reference
//array is also passed by reference(by choice), objects always passed by reference
function fun4( $a, &$b, &$array, $obj){
   echo("---FUN4--- \n");
   $a = $a*$a;
   $b = $b*$b;
   array_push($array,"99");
   $obj->testVar = 9999;
   $value = $obj->testVar;
   echo("a inside: $a ");
   echo("b inside: $b \n");
   echo("object inside: $value \n");
   echo("array inside: ");
   foreach($array as $item){ echo("$item "); } echo("\n");

}
//functions can be passed as arguments in php
function fun5($f, $i){
   echo("---FUN5---\n");
   echo($f($i));
   echo("\n");
}
```

```php
//rest parameter(must be last formal parameters)
function fun6($a,$b, ...$others){
   echo("---FUN6---\n");
   echo("a is: $a ");
   echo("b is: $b \n");
   echo("rest of the parameters: ");
   foreach($others as $item){
   echo("$item ");
   }
   echo("\n");
}

fun1(1,2,3); //simple positional correspondence
//fun1(1,2,3,4); //also works, 4 is available in $arg_list(tested ahead)
//fun1(1); //error, not enough arguments provided

fun2(1); //default value for b and c used
fun2(1,2,3); //works, arguments possistionally assisgned
//fun2($a=1,$c=2,$b=3); //does not work as expeced, still uses positional correspondence

fun3(1,2,3,4,5,6);//variable number of arguments passed
//fun3(1,2,3,4,5,6,7,8,9,10) //also works
//
//declaring different data types to check their values after a function, i.e whether the function uses, in , out , or inout methods
$a = 3;
$b = 5;
$array = [0,0,0];
$obj = new Tester();
$value = $obj->testVar;
echo("----------\n");
echo("initial a: $a ");
echo("initial b: $b \n");
echo("initial object: $value \n");
echo("initial array:");
foreach($array as $item){ echo("$item "); } echo("\n");

fun4($a,$b,$array,$obj);

echo("----------\n");
$value = $obj->testVar;
echo("a after: $a ");
echo("b after: $b \n");
echo("object after: $value\n");
echo("array after:");
foreach($array as $item){ echo("$item "); } echo("\n");

fun5('cuber', 2);

fun6(1,2, 99,99,99,99,99); //mixed positional and variable number of arguments
?>
```

Output:

```
[balaj.saleem@dijkstra saleem_balaj]$ php saleem_balaj_php.php
---FUN1---
a is: 1 b is: 2 c is: 3
---FUN2---
a is: 1 b is: 99 c is: 99
---FUN2---
a is: 1 b is: 2 c is: 3
---FUN3---
total parameters provided: 6
1 2 3 4 5 6
----------
initial a: 3 initial b: 5
initial object: 0
initial array:0 0 0
---FUN4---
a inside: 9 b inside: 25
object inside: 9999
array inside: 0 0 0 99
----------
a after: 3 b after: 25
object after: 9999
array after:0 0 0 99
---FUN5---
8
---FUN6---
a is: 1 b is: 2
rest of the parameters: 99 99 99 99 99
```

Conclusion:

Each of these languages uses a unique methodology for handling subprogram parameters. Each methodology has its strengths and weaknesses.

Firstly, correspondence methods varied from simple positional correspondence, to keyword, to list or arguments, to a combination of all of these. Each of these methods except arguments list, improves readability[5]. However the code is more concise when using argument lists such as those for perl and php.

Secondly, variable length of arguments allows for greater flexibility and makes the programs less error prone for example in PHP, since invalid number of provided arguments does not affect the program at runtime.

Default values also make the program less error prone and more readable, for example in python. Since all variables do not have to be entered again. Javascript's approach to making default value of every formal parameter to be 'undefined' also accomplishes a similar purpose.

Finally the passing methods mainly revolved through a variety of approaches. Pass by value, which is a safer approach, since the outer instance of the variable is not affected is used(one way or the other) by all the languages tested except perl, where this approach can only be simulated. Pass by reference, is a more robust and functional approach since the actual variables can be changed inside the function, this can allow for a great deal of freedom however it is also more error prone. Python uses the pass by assignment/object approach since all values are considered objects and unless redeclared, mutable object can have their values changed.

References:

[1]https://chortle.ccsu.edu/Java5/Notes/chap34A/ch34A_3.html

[2]https://www.mathwarehouse.com/programming/passing-by-value-vs-by-reference-visual-explanation.php

[3]https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Default_parameters

[4]https://medium.com/school-of-code/passing-by-assignment-in-python-7c829a2df10a

[5]http://www.informit.com/articles/article.aspx?p=661370&seqNum=4