Deniz Yüksel
21600880
Section 1

PL HOMEWORK 2

In this homework, the design issues below for Python, Javascript, Perl and PHP will be addressed and explained with properly commented example codes:

* What are the types of loop control variables?

* What are the scopes of loop control variables?
* Is it legal for the loop control variable or loop parameters to be changed in the loop, and if so, does the change affect loop control?
* Are the loop parameters evaluated only once, or once for every iteration?

Python:

In Python, the counter controlled loop can be used with the keyword for. The for loop can also have an else statement inside for control. For the range() function, the only legal control variable is integer. The function can also take 1,2 or 3 parameters. If function is given no parameters, an error is generated. In a for loop, once the interpreter reads the line, the counter variable can be modified in the first iteration, but the new manipulated value does not affect the number of upcoming iterations. It is legal to manipulate the loop control variables but since loop parameters are evaluated only for once, the change only affects the value inside the body for once and manipulation does not affect the number of iterations. A counter variable can be initialized before the for loop. However, using the same variable name in the for loop statement as “for variable in range(*int*)” redefines the variable. After the loop is terminated, the value of the counter variable is not deallocated. The value of the variable can still be used. So, they can be reached outside their scope after the loop.

I mentioned the counter variable as the variable that is named in the for loop statement, but I also think that the variable that I am talking about is the counted variable. The counter variable is initialized when the number of counts is detected and can never be reached. This is the other but more probable possibility that I think of.

```
print "Considering the range function, 3rd and 4th question..."

interval = 5
for x in range(interval):
  print x
  x = x + 9999          # x gets incremented by 9999 in the first iteration, but gets incremented 1 by
1 in each iteration. It is legal to manipulate the loop control variables but since loop parameters are
evaluated only for once, the change only affects the value inside the body for once and
manipulation does not affect the number of iterations.
  print x
  interval = interval + 1        #interval gets incremented but does not affect the number of iterations.
print interval   #interval was incremented 5 times.
print x           # Even though the counter variable was not defined before the loop, its value is
retained.
```

```
print "Range funciton question 1 and 2..."

#for x in range(4.6):
#  print x
'''
   Prints the error: Traceback (most recent call last):
   File "yuksel_deniz_python.py", line 22, in <module>
   for x in range(4.6):
   TypeError: range() integer end argument expected, got float.
'''

#for x in range('apple'):
#   print 'pear'
'''
   Traceback (most recent call last):
   File "yuksel_deniz_python.py", line 31, in <module>
   for x in range('apple'):
   TypeError: range() integer end argument expected, got str.

'''
# Only integer argument is expected in range function, others are not legal.
for y in range(2):
   y = 999                    #This causes no change.
   for y in range(5):
      print y
print y             #The value of y is retained.

#Each time a new loop counter variable is declared in the #for statement, the value gets destroyed.
So, the counter #control variable cannot be predefined.
print '\n'
#STEP SIZE FORM...
steps = 4
end = 17
start = 3
for x in range(start, end, steps):
   steps = 2
   print x

# step size cannot be modified.
# Demonstration of for loop_variable in object:

print "Considering for loop_variable in object form..."

iteratedList = [1,2,4,5,10,15,209,42]
for element in iteratedList:
    iteratedList[2] = 99999
    # The element in a location can be modified.
    print element
print '\n'
```

```
x = 10
for i in [x,x,x,x,x,x,x,x,x,x]:
    print i

print '\n'
```

Javascript:

In Javascript, loop control variables and counter variables accepts double values. In the console, the values are evaluated without their values getting truncated. The loop control variables and counter variables also accept string and array inputs. But because in Javascript the variables don't have a type specified when declared, when an array is initialized to a loop control variable, the value is not defined and the executions written inside the loop does not occur. For the string case, when both the loop control and counter variables are Strings, the loop only evaluates the initial counter variable once, and then exits. In Javascript, values of control counter variables can be modified, and thus affects the number of iterations accordingly. This implies that loop control and counter variables are checked for every evaluation, except both string case. The design issues discussed here are demonstrated with a code below:

```
<!DOCTYPE html>

<html>
<head>
        <title> CS 315 JAVASCRIPT LOOP FUN </title>

</head>

<body>

<p>PL HW2 Javascript LOOP CHECK. PRESS F12 TO SEE THE CONSOLE.</p>

<script>

var myJSArray = ["word", "WELCOME", "zero", 4, 55];
var intArray = [ 5, 3, 4, 19, 20, -2, -36];

for( let i = 0; i <10; i++){
        console.log( "Current value is " + i);
}

//console.log(i); Generates an error because i is not defined. Therefore the scope of the loop control variable is its block. Its value is not retained like it occurs in Python or PHP.
for( let i = 1.4; i < 19.8; i++){
        console.log( "Current value is " + i);
}
//Loop control variable and counter variables accepts double values. In the console, the values are printed without their values getting truncated.

let control = myJSArray.length;
console.log(control);
for( let i = 0; i < control; i++){
	i = i + 5;
	console.log( "Current value is " + i);
	control = control + 3;
}

// The value of the control and counter variable can be modified, and thus affects the number of
iterations accordingly. This implies that loop control and counter variables are checked for every
evaluation.
let control2 = 50;
for( let i = intArray; i < control2; i++){
	i = 2;
	console.log( "Current value is " + i);
	i = 5;
}
//In Javascript the variables don't have a type specified when declared. Therefore when an array is
initialized to a loop control variable, the value is not defined and the executions written inside the
loop does not occur.

for( let i = "Deniz"; i < "Javascript"; i++){
	console.log( "Current value is " + i);
}
//When both the loop control and counter variables are Strings, the loop only evaluates the initial
counter variable once, and then exits.

for( let i = "Deniz"; i < intArray; i++){
	console.log( "Current value is " + i);
}
</script>
</body>
</html>
```

Perl:

In Perl language, the control variable can be manipulated inside the body of the loop. This affects the number of iterations. Also unlike Python and PHP, the value of the counter variable is unavailable whence the block of the declaration is over. This implies that scopes of the loop counter variable is valid in the very block. Moreover, the loop control variable can be manipulated, and thus changes the number of iterations. Therefore, the loop variables are not evaluated only once likewise in Python, but checked for every iteration. The for loop in Perl accepts a variety of parameters both for the control and the counter variable. If the value of the counter variable is a string, the value of the string is evaluated as the first value of the counter variable, and then other values are printed according to the control variable. When an array is assigned to an integer register as a loop control variable, the control variable's value becomes the size of that array. The foreach loop in Perl accepts double values and does not generate an error. However the values of the doubles both included. The double values are truncated and their integer forms are printed. Also, it appears to be no difference between typing foreach and for, in the case of my code's last two methods. The print-outs are exactly the same. The below code illustrates the design issues that were discussed:

```
#!/usr/bin/perl
use strict;
use warnings;
```

```
my @myArray = (52,23,4,74,"Deniz",892,755,6,7, 50, 23, 55);

for( my $i = 0; $i < 10; $i++){
	print "Current value: $i \n";
	$i = $i + 2;
##The control variable can be manipulated inside the body of the loop. This affects the number of
iterations.

#print "The value of i is " . "$i ";
}
print "\n";
#print "The value of i is " . "$i ";
# This print statement generated errors because the variable $i is not defined. This shows unlike
PHP or Python, in Perl, the value of the loop counter variable is deallocated whence the loop
terminates. The scope of the variable is only in the declared block.

my $size = scalar @myArray;
print "$size";

for( my $i = 0; $i < $size; $i++){
	print "Current value: $i \n";
	$size = $size - 2;
	# The loop control variable can be manipulated, and thus changes the number of iterations.
Therefore, the loop variables are not evaluated only once likewise in Python, but checked for every
iteration.
}

my $control = 10.4;
#The for loop accepts double parameters for control and counter variables.
for( my $i = 0.8; $i <= $control; $i = $i + 0.3){
	print "Current value: $i \n";
}

my $word = "PerlFun";
#If the value of the counter variable is a string, the value of the string is printed as the first print-out
value, and then other values are printed according to the control variable.
for( my $i = "ThisIsTheZerothPrintOut"; $i < 10; $i++){
	print "Current value: $i \n";
}

$size = scalar @myArray;
#When an array is assigned to an integer register, the register's value is the size of that array. This is
why this statement prints 12 13 14
for( my $i = @myArray; $i < 15; $i++){
	print "$i ";
}
print "\n";

#This little code prints out integers from 0 to 12, both included. The double values are truncated and
their integer forms are printed.
foreach my $element ( 0.4..12.5){
        print "$element ";
}
print "\n";

foreach my $element (@myArray){
        print "$element ";
}
print "\n";

#It appears to be no difference between typing foreach and for, for this case...
for my $element (@myArray){
        print "$element ";
}
```

PHP:

In PHP, there are for and foreach loops for counter- controlled loops. For the case of for loops, the loop control variable and counter variable can be integers, floating point numbers, strings and arrays. If the counter and control variables are floating values, this will not cause a problem even at the run time. The values that are in between interval will be printed if there is a print statement inside the body. Also if a counter variable is initialized within the for statement, its final value is retained after the loop and can be reused. For this reason, if the same variable name for the counter will be used again, the programmer should initialize it to the desired number. Thus, the range of counter and control variables are after their declaration. The value of counter variable can be manipulated. When iterating through an array with a counter variable, if the control variable is set to a value that is more than the size of the array, the program generates PHP Notices that declared “Undefined offset...” However, if attempted, the console prints a space characters. Therefore, control and counter variables can be manipulated in the loop body and it affects the number of iterations. The foreach loop in PHP can only take array parameter and a temporary variable in which the value in the element is held. When a parameter other than array is inscribed in the foreach loop, the program gives an error and does not generate any outputs. The commented code below illustrates all the points that are mentioned:

```
<!DOCTYPE html>
<html>
  <body>
    <?php

      $arr = array(1,2,"Arif",3,"Deniz",5);
      $length = count($arr);

      for( $i = 0; $i < $length; $i++ ) {
        echo $arr[$i] . " is at location " . $i . "\n";
      }
     echo $i;
     /* If attempted to be printed, i's value is retained and it can be printed. Although the counter
varible was initialized and used in the loop, its value is not deallocated but is still retained. After the
loop, the counter variable can be used.*/
     echo "\n\n";
```

```
        for( $i = 0; $i < $length; $i++){
            /* Attempting to change counter and loop control variable respectively below.*/
            /* INFINITE LOOP IF THIS HAPPENS--><!--$i = 3;*/
            /* $length = "Random string"; */ /* If this initialization occurs, the print statement occurs
only for the first element.*/
            $length = 10; /* PHP Notice, but no errors.*/
            /* AGAIN, AN INFINITE LOOP, BUT IS ALLOWED. GIVES PHP Notice:  Undefined
offset */ /*$length = array(2,3,4);*/
            echo $arr[$i] . " is at location " . $i . "\n";

            /* In PHP, loop control variable and loop parameters can be changed in the loop, and it
affects the loop control. If the counter variable is changed in the loop, the number of iterations alter
accordingly. Also, the loop control variable can be changed to other values. */
        }

        /* In PHP, loop control variable and loop counter can be arrays. */
        $myStr = "hatice";
        $count = count($myStr);
        echo $myStr[2] . " is the 3rd letter of this word: " . $myStr . "\n";
        echo "Count of \$myStr is " . $count . " known that it is a string. " . "\n";
        echo "The word will therefore be printed " . $count . " times in the next loop: " . "\n";
        for($x = 0; $x < $count; $x++){
            echo $myStr;
        }

        /* Control variable and counter variable can be float values in PHP: */
        /* The below code prints "2 is at location 1.2
a is at location 2.2
3 is at location 3.2
4 is at location 4.2
5 is at location 5.2" */

        $doubleVal = 5.6;
        for( $i = 1.2; $i < $doubleVal; $i++ ) {
            echo $arr[$i] . " is at location " . $i . "\n";
         }

         /* The below code prints all the fruit names.*/
         $fruits = array("apple", "pear", "orange", "grapefruit", "watermelon", "plum");
        foreach ($fruits as $value) {
            echo "the corresponding fruit is $value " . "\n";
            echo $fruits[4] . "\n";
            $value = 2; /* Although there is an assignment to value here, it neither changes anything,
nor generates warnings.*/
            $fruits[4] = "BANANA!!!!"; /* Fourth fruit is not watermelon anymore, it's BANANA! */
        }

    ?>
   </body>
</html>
```

EVALUATION FOR THE MOST PREFERRED LANGUAGE FOR COUNTER CONTROLLED LOOPS

In my opinion, a programming language’s priority must be reliability. In Python language, although the loop counter variable can be modified, the number of iterations do not change. This, in a sense is reliable. On the other hand, in Javascript, Perl, and PHP, the loop counter variables can be modified. This can also be powerful because there might be consequences that programmer may want to manipulate loop control variables in order to achieve a more practical solution. However, I prefer reliability. There is also the reusability issue of loop counter variables in Python, but the solution can come with tracing the uses of the variable. However, it is also risky. In PHP, values are kept in registers and scope of variables are the own blocks. This is why the program does not let users to use variables that are defined inside the loop blocks outside. This occasion also provides reliability. I have experienced in this homework that printing double values in a loop does not cause any problems, at least for this occasion. PHP and Javascript in a way resembled me Matlab because of the loops accepting and evaluating double values. I find that Javascript is more reliable than PHP because the values because where Javascript generates errors, PHP lets the programmer go on and generates warnings. I find Javascript to be the language that is compatible with the loops.