CS 315

Homework Assignment 2

User-Located Loop Control Mechanisms in Dart, Javascript, Lua, PHP, Python, Ruby, and Rust

Zeynep Selcen Öztunç

21902941

Table of Contents

1.Dart

1.1 Should the conditional mechanism be an integral part of the exit (conditional or unconditional exit)?

In the Dart language, the **break** statement can be used to break the flow of a loop and **continue** is used to jump to the next iteration of the loop [1]. So it can be said that the conditional mechanism is not an integral part of the exit. An example of the use of a break statement is given below. In the example, it can be seen that the loop only iterates until break, after the break statement it exits the loop.

```dart
void main() {
   var loopVar = 10;

   while(loopVar >=1) {
      loopVar--;
      break;
   }
   print("The value of loopVar after the loop is: ${loopVar}");
   print("\n");

}
```

```
The value of loopVar after the loop is: 9
```

The **continue** statement in the while loop just continues to the next iteration of the loop, so in this case1 is incremented from loopVar in each iteration, which in the end becomes 0.

```dart

   print("\n Example  of continue");
    loopVar = 10;

   while(loopVar >=1) {
      loopVar--;
      continue;
   }
   print("The value of loopVar after the loop is: ${loopVar}");

```

```
 Example  of continue
The value of loopVar after the loop is: 0
```

1.2 Should only one loop body be exited or can enclosing loops also be exited (labeled or unlabeled exit)?

In the Dart language, when there are two enclosing loops, the outer loop can be exited when we write labels for the loops. The **break** statement, followed by the name of the outer label can be used to break the outer loop [1]. So the Dart language supports labeled exits. In the example below, there are two labels called outer and inner and inside of the inner label, the outer loop is breaked by the **break** outer statement. It can be seen that the outer loop is exited after that statement.

```
13  //outer label
14  outer:
15     for (var i = 0; i < 5; i++) {
16        print("Inside the outer loop, i: ${i}");
17        print("------------------------------");
18
19        //inner label
20        inner:
21          for (var j = 6; j < 9; j++) {
22
23             //this break statement quits the outer loop
24             if (i == 2) {
25               print("Exiting from outer");
26               break outer;
27             }
28             print("Inside the inner loop, j:  ${j}");
29          }
30          print("\n");
31
32     }
```

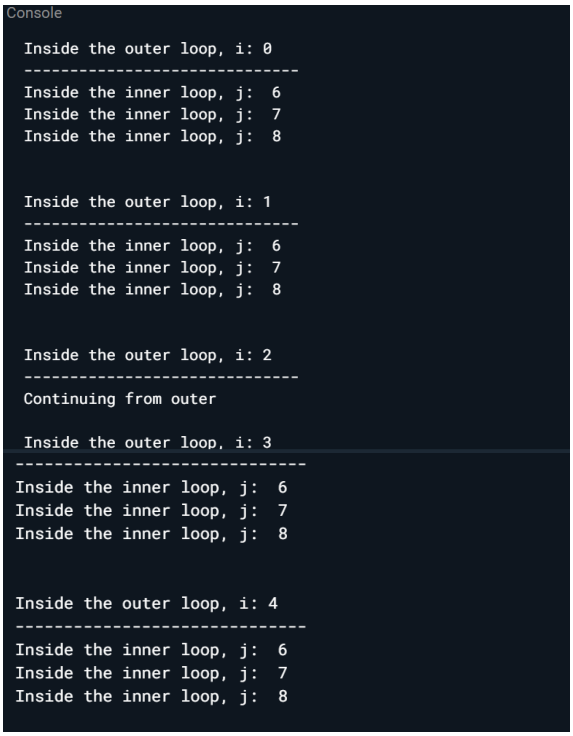
```
Inside the outer loop, i: 0
------------------------------
Inside the inner loop, j:  6
Inside the inner loop, j:  7
Inside the inner loop, j:  8


Inside the outer loop, i: 1
------------------------------
Inside the inner loop, j:  6
Inside the inner loop, j:  7
Inside the inner loop, j:  8


Inside the outer loop, i: 2
------------------------------
Exiting from outer
```

When we use the **continue** statement following with the name of the outer label inside the inner loop, it stops the execution of the inner loop and then continues with the next iteration of the outer loop [1].

```
35    print("Example  of continue ");
36    outer:
37     for (var i = 0; i < 5; i++) {
38        print("Inside the outer loop, i: ${i}");
39        print("------------------------------");
40
41        //inner label
42        inner:
43          for (var j = 6; j < 9; j++) {
44
45
46             //this break statement quits the outer l
47             if (i == 2) {
48               print("Continuing from outer \n");
49               continue outer;
50             }
51             print("Inside the inner loop, j:  ${j}")
52
53          }
54          print("\n");
55     }
56
57  }
```



2.Javascript

2.1 Should the conditional mechanism be an integral part of the exit (conditional or unconditional exit)?

In Javascript language, the **break** statement can be used to break the flow of a loop and **continue** is used to jump to the next iteration of the loop [2]. So it can be said that a conditional mechanism is not an integral part of the exit. An example of the use of a break statement is given below. In the example, it can be seen that the loop only iterates until break, after the break statement it exits the loop.

```
1  //example of an unconditional exit using break
2  var loopVar=10
3  while (loopVar >=1) {
4    loopVar--;
5    break;
6  }
7  console.log("The value of loopVar after the loop is " + loopVar
       );
8
```

```
node /tmp/1tDV5jWaOO.js
The value of loopVar aftert the loop is 9
```

The **continue** statement in the while loop just continues to the next iteration of the loop, so in this case1 is incremented from loopVar in each iteration, which in the end becomes 0.

```
8
9  //example of using continue
10 loopVar=10
11 while (loopVar >=1) {
12   loopVar--;
13   continue;
14 }
15 console.log("The value of loopVar after the loop is " + loopVar
       );
16
```

```
The value of loopVar aftert the loop is 0
```

2.2 Should only one loop body be exited or can enclosing loops also be exited (labeled or unlabeled exit)?

In the JavaScript language, when there are two enclosing loops, the outer loop can be exited when we write labels for the loops. The **break** operator, followed by the name of the outer label can be used to break the outer loop. So the Javascript language supports labeled exits. In the example below, there are two labels called outer and inner and inside of the inner label, the outer loop is breaked by the **break** outer statement. It can be seen from the output that the outer loop is exited after that statement.

```
10 //outer label
11 outer:
12   for (var i = 0; i < 5; i++) {
13     console.log("Inside the outer loop, i: " + i);
14     console.log("------------------------------")
15
16     //inner label
17     inner:
18       for (var j = 6; j < 9; j++) {
19
20         //this break statement quits the outer loop
21         if (i == 2) {
22           console.log("Exiting from outer")
23           break outer;
24         }
25         console.log("Inside the inner loop, j: " + j);
26       }
27       console.log("\n")
28   }
29
30 //example of using continue inside enclosing loops
```

```
Inside the outer loop, i: 0
------------------------------
Inside the inner loop, j: 6
Inside the inner loop, j: 7
Inside the inner loop, j: 8


Inside the outer loop, i: 1
------------------------------
Inside the inner loop, j: 6
Inside the inner loop, j: 7
Inside the inner loop, j: 8


Inside the outer loop, i: 2
------------------------------
Exiting from outer
```

When we use the **continue** statement following with the name of the outer label inside the inner loop, it stops the execution of the inner loop and then continues with the next iteration of the outer loop [2].

```
41  //outer label
42  outer:
43     for (var i = 0; i < 5; i++) {
44        console.log("Inside the outer loop, i: " + i);
45        console.log("------------------------------")
46
47        //inner label
48        inner:
49          for (var j = 6; j < 9; j++) {
50
51            console.log("Inside the inner loop, j: " + j);
52            //this continue statement continues from the outer
                  loop
53            if (i == 2) {
54              console.log("Continuing from outer \n")
55              continue outer;
56            }
57          }
58          console.log("\n")
59     }
```

```
Inside the outer loop, i: 0
------------------------------
Inside the inner loop, j: 6
Inside the inner loop, j: 7
Inside the inner loop, j: 8


Inside the outer loop, i: 1
------------------------------
Inside the inner loop, j: 6
Inside the inner loop, j: 7
Inside the inner loop, j: 8


Inside the outer loop, i: 2
------------------------------
Inside the inner loop, j: 6
Continuing from outer

Inside the outer loop, i: 3
```

```
----------------------------
Inside the inner loop, j: 6
Inside the inner loop, j: 7
Inside the inner loop, j: 8


Inside the outer loop, i: 4
----------------------------
Inside the inner loop, j: 6
Inside the inner loop, j: 7
Inside the inner loop, j: 8
```

3.Lua

3.1 Should the conditional mechanism be an integral part of the exit (conditional or unconditional exit)?

In the Lua language, the **break** statement can be used to break the flow of a loop. So it can be said that the conditional mechanism is not an integral part of the exit. There is no **continue** statement in Lua.[3]. An example of the use of a break statement is given below. In the example, it can be seen that the loop only iterates until break, after the break statement it exits the loop.

```lua
--example of an unconditional exit
local loopVar=10
while( loopVar>=10 )
do
   loopVar=loopVar-1
   break
end

print("The value of loopVar after loop is: ".. loopVar)

```

```
The value of loopVar after loop is: 9
Inside the outer loop, i:0
```

3.2 Should only one loop body be exited or can enclosing loops also be exited (labeled or unlabeled exit)?

In Lua, the break operator only exits the inner loop, so it can be said that Lua has an unlabeled exit. In the example code below, this behavior is represented by having nested loops and writing an if statement inside the inner loop. When the condition inside the if statement is true, a statement indicating that the loop is being exited is printed and break is executed. As it can be seen from the output of the example code, only the inner loop is

breaked

```
--outer loop
for i=0,5 do
  print("Inside the outer loop, i:"..i);
  print("------------------------------");

    --inner loop
    for j=6,8 do
         --this break statement quits inner loop
         if i == 2 then
            print("Exiting from the inner for loop")
            break
          end

        print("Inside the inner loop, j ".. j)
    end


    print("\n")
end
```

```
Inside the outer loop, i:0
------------------------------
Inside the inner loop, j 6
Inside the inner loop, j 7
Inside the inner loop, j 8


Inside the outer loop, i:1
------------------------------
Inside the inner loop, j 6
Inside the inner loop, j 7
Inside the inner loop, j 8


Inside the outer loop, i:2
------------------------------
Exiting from the inner for loop


Inside the outer loop, i:3
------------------------------
Inside the inner loop, j 6
Inside the inner loop, j 7
Inside the inner loop, j 8


Inside the outer loop, i:4
------------------------------
Inside the inner loop, j 6
Inside the inner loop, j 7
Inside the inner loop, j 8


Inside the outer loop, i:5
------------------------------
Inside the inner loop, j 6
Inside the inner loop, j 7
Inside the inner loop, j 8
```

A way of exiting the outer loop is using **goto**. When followed by a label name, **goto** jumps to that label. If the label is put outside of the nested for loops, both of the loops are exited. It should be noted that this is just a workaround.

```
26
27      print("Using goto to exit")
28      for i=0,5 do
29      print("Inside the outer loop, i:"..i);
30      print("------------------------------");
31
32          for j=6,8 do
33          print("Inside the inner loop, j ".. j)
34
35              --this goto statement jumps to Exit label
36              if i == 2 then
37                  print("Exiting from the inner for loop")
38                  goto Exit
39              end
40        end
41        print("\n")
42      end
43      :: Exit ::
44
```

```
Using goto to exit
Inside the outer loop, i:0
------------------------------
Inside the inner loop, j 6
Inside the inner loop, j 7
Inside the inner loop, j 8


Inside the outer loop, i:1
------------------------------
Inside the inner loop, j 6
Inside the inner loop, j 7
Inside the inner loop, j 8


Inside the outer loop, i:2
------------------------------
Inside the inner loop, j 6
Exiting from the inner for loop
```

4.PHP

4.1 Should the conditional mechanism be an integral part of the exit (conditional or unconditional exit)?

In PHP, the **break** statement can be used to break the flow of a loop and **continue** is used to jump to the next iteration of the loop [4]. So it can be said that a conditional mechanism is not an integral part of the exit. An example of the use of a break statement is given below. In the example, it can be seen that the loop only iterates until break, after the break statement it exits the loop.

```php
//example of an unconditional exit
$loopVar = 10;

while($loopVar >= 1) {
  $loopVar--;
    break;
}
echo "The value of lopoVar after the loop is: $loopVar \n";
```

```
The value of lopoVar after the loop is: 9
```

The **continue** statement in the while loop just continues to the next iteration of the loop, so in this case1 is incremented from loopVar in each iteration, which in the end becomes 0.

```php
//example of using continue
$loopVar = 10;

while($loopVar >= 1) {
  $loopVar--;
   continue;
}
echo "The value of lopoVar after the loop is: $loopVar \n";
```

```
The value of lopoVar after the loop is: 0
```

4.2 Should only one loop body be exited or can enclosing loops also be exited (labeled or unlabeled exit)?

Even though there are no labels in PHP, enclosing loops can be exited using **break**. The **break** statement is followed by a numeric argument, which defines how many loops will be terminated. The argument is 1 by default, so when only break is used (without arguments), then only the loop that is being executed will be exited [4]. In the code below, the inner and the outer loops are exited since I used **break** 2.

```php
//example of breaking the outer loop using break
//the outer for loop
for ($i = 0; $i < 5; $i++) {
   echo "\n Inside the outer loop, i:  $i \n" ;
   print("---------------------------- \n");

   //the inner for loop
   for ($j = 6; $j < 9; $j++) {
      if ($i == 2) {
        echo "Exiting \n";
        break 2; //terminates the 2nd outer loop
      }
      echo "Inside the inner loop, j:  $j \n" ;

   }
   echo "\n";
}
```

```
The value of lopoVar after the loop is: 9

 Inside the outer loop, i:  0
-----------------------------
Inside the inner loop, j:  6
Inside the inner loop, j:  7
Inside the inner loop, j:  8


 Inside the outer loop, i:  1
-----------------------------
Inside the inner loop, j:  6
Inside the inner loop, j:  7
Inside the inner loop, j:  8


 Inside the outer loop, i:  2
-----------------------------
Exiting
```

When we use the **continue** it stops the execution of the inner loop and then continues with the next iteration of the outer loop.

```php

//the outer for loop
for ($i = 0; $i < 5; $i++) {
   echo "\nInside the outer loop, i:  $i \n" ;
   print("----------------------------- \n");

   //the inner for loop
   for ($j = 6; $j < 9; $j++) {

      if ($i == 2) {
        echo "Continuing from outer \n";
        continue;  //continues from the outer loop
      }
          echo "Inside the inner loop, j:  $j \n" ;

   }
   echo "\n";
}
?>
```

```
 Using continue

Inside the outer loop, i:  0
-----------------------------
Inside the inner loop, j:  6
Inside the inner loop, j:  7
Inside the inner loop, j:  8


Inside the outer loop, i:  1
-----------------------------
Inside the inner loop, j:  6
Inside the inner loop, j:  7
Inside the inner loop, j:  8


Inside the outer loop, i:  2
-----------------------------
Continuing from outer
Continuing from outer
Continuing from outer


Inside the outer loop, i:  3
-----------------------------
Inside the inner loop, j:  6
Inside the inner loop, j:  7
Inside the inner loop, j:  8


Inside the outer loop, i:  4
-----------------------------
Inside the inner loop, j:  6
Inside the inner loop, j:  7
Inside the inner loop, j:  8
```

5.Python

5.1 Should the conditional mechanism be an integral part of the exit (conditional or unconditional exit)?

In Python, the **break** statement can be used to break the flow of a loop and **continue** is used to jump to the next iteration of the loop [5]. So it can be said that the conditional mechanism is not an integral part of the exit. An example of the use of a break statement is given below. In the example, it can be seen that the loop only iterates until break, after the break statement it exits the loop.

```
#example of an unconditional exit
loopVar = 10
while loopVar >= 1:
  loopVar=loopVar-1
  break
print"Value of the loopVar after the loop is: ",loopVar
```

```
Value of the loopVar after the loop is:  9
Inside the outer loop, i: 0
```

The **continue** statement in the while loop just continues to the next iteration of the loop, so in this case1 is incremented from loopVar in each iteration, which in the end becomes 0.

```
#example of an unconditional exit using continue
loopVar = 10
while loopVar >= 1:
  loopVar=loopVar-1
  continue
print"Value of the loopVar after the loop is: ",loopVar
```

```
Value of the loopVar after the loop is:  0
```

5.2 Should only one loop body be exited or can enclosing loops also be exited (labeled or unlabeled exit)?

Python does not have labels. The break operator only exits the inner loop [5]. So it can be said that Python has an unlabeled exit. In the example code below, there is an if condition inside the inner loop which checks whether the value of i (loop variable) is equal to 2. If the condition is satisfied, it uses the break. As it can be seen from the output of the code, only the inner loop is exited.

```
#outer loop
for i in range(0,6):
    print"Inside the outer loop, i:", i
    print"------------------------------"

    #inner loop
    for j  in range(6,9):

        #this break statement quits the inner loop
        if i == 2:
            print "Exiting from inner"
            break
        print"Inside the inner loop, j:", j

    print"\n"
```

```
Inside the outer loop, i: 0
------------------------------
Inside the inner loop, j: 6
Inside the inner loop, j: 7
Inside the inner loop, j: 8


Inside the outer loop, i: 1
------------------------------
Inside the inner loop, j: 6
Inside the inner loop, j: 7
Inside the inner loop, j: 8


Inside the outer loop, i: 2
------------------------------
Exiting from inner


Inside the outer loop, i: 3
------------------------------
Inside the inner loop, j: 6
Inside the inner loop, j: 7
Inside the inner loop, j: 8


Inside the outer loop, i: 4
------------------------------
Inside the inner loop, j: 6
Inside the inner loop, j: 7
Inside the inner loop, j: 8


Inside the outer loop, i: 5
------------------------------
Inside the inner loop, j: 6
Inside the inner loop, j: 7
Inside the inner loop, j: 8
```

The **continue** statement behaves in a similar manner as break when used in enclosing scopes. As mentioned before, it stops the current iteration, which in this case, is inside the inner loop and continues from the next iteration. So it never executes the print statement which says “Inside the inner loop” since it keeps jumping to the next iteration.

```
#example of continue statement in enclosing loops
#outer loop
for i in range(0,6):
    print"Inside the outer loop, i:", i
    print"------------------------------"

    #inner loop
    for j  in range(6,9):

        #this continue statement continues to the next iteration
        if i == 2:
            print "Continuing to the next iteration"
            continue
        print"Inside the inner loop, j:", j

    print"\n"
```

```
Inside the outer loop, i: 0
------------------------------
Inside the inner loop, j: 6
Inside the inner loop, j: 7
Inside the inner loop, j: 8


Inside the outer loop, i: 1
------------------------------
Inside the inner loop, j: 6
Inside the inner loop, j: 7
Inside the inner loop, j: 8


Inside the outer loop, i: 2
------------------------------
Continuing to the next iteration
Continuing to the next iteration
Continuing to the next iteration


Inside the outer loop, i: 3
------------------------------
Inside the inner loop, j: 6
Inside the inner loop, j: 7
Inside the inner loop, j: 8


Inside the outer loop, i: 4
------------------------------
Inside the inner loop, j: 6
Inside the inner loop, j: 7
Inside the inner loop, j: 8


Inside the outer loop, i: 5
------------------------------
Inside the inner loop, j: 6
Inside the inner loop, j: 7
Inside the inner loop, j: 8
```

6. Ruby

6.1 Should the conditional mechanism be an integral part of the exit (conditional or unconditional exit)?

In the Ruby language, the **break** statement can be used to break the flow of a loop. So it can be said that a conditional mechanism is not an integral part of the exit. There is no **continue** statement in Ruby, however, there is a **next** statement which is really similar to the **continue** statement in other languages.[6]. An example of the use of a break statement is given below. In the example, it can be seen that the loop only iterates until break, after the break statement it exits the loop.

```ruby
#example of unconditional exit using break
loopVar=10

while loopVar >= 1
   loopVar = loopVar - 1
   break
end
puts "The value of the loop variable after the loop is: #{loopVar}"
```

```
The value of the loop variable after the loop is: 9
```

The **next** statement in the while loop just continues to the next iteration of the loop, so in this case1 is incremented from loopVar in each iteration, which in the end becomes 0.

```ruby
#example of unconditional exit using next
loopVar=10

while loopVar >= 1
    loopVar = loopVar - 1
    next
end
puts "The value of the loop variable after the loop is: #{loopVar}"
```

```
The value of the loop variable after the loop is: 0
```

6.2 Should only one loop body be exited or can enclosing loops also be exited (labeled or unlabeled exit)?

In Ruby language, enclosing loops can be exited using the break operator. In the code example below, there is an if statement inside of the inner for loop which checks a boolean value, in this case it whether i (variable of the outer loop) is equal to 2. When the condition is satisfied, it prints out a message stating that the loop is being exited (“Exiting”) and exits the loop using break. It can be seen from the output statements that after the break operator is used, only the inner loop is exited.

```ruby
#example of using break in enclosing loops
#outer loop
for i in 0..4 do

    puts "Inside outer loop, i is: #{i}"
    puts "------------------------------"

    #inner loop
    for j in 6..8 do
        if i==2
            puts "Exiting \n"
            break
        end

        puts "Inside inner loop, j is: #{j}"
    end
    puts "\n"

end
```

As mentioned before, **next** skips the rest of the current iteration.The sample code demonstrates the use of **next** in enclosing loops:

```ruby

puts "\n Using next \n"
#example of using next in enclosing loops
#outer loop
for i in 0..4 do

    puts "Inside outer loop, i is: #{i}"
    puts "------------------------------"

    #inner loop
    for j in 6..8 do
        if i==2
            puts "Continuing \n"
            next
        end
        puts "Inside inner loop, j is: #{j}"
    end


    puts "\n"
end
```

```
 Using next
Inside outer loop, i is: 0
------------------------------
Inside inner loop, j is: 6
Inside inner loop, j is: 7
Inside inner loop, j is: 8

Inside outer loop, i is: 1
------------------------------
Inside inner loop, j is: 6
Inside inner loop, j is: 7
Inside inner loop, j is: 8

Inside outer loop, i is: 2
------------------------------
Continuing
Continuing
Continuing

Inside outer loop, i is: 3
------------------------------
Inside inner loop, j is: 6
Inside inner loop, j is: 7
Inside inner loop, j is: 8

Inside outer loop, i is: 4
------------------------------
Inside inner loop, j is: 6
Inside inner loop, j is: 7
Inside inner loop, j is: 8
```

7.Rust

7.1 Should the conditional mechanism be an integral part of the exit (conditional or unconditional exit)?

In the Rust language, the **break** statement can be used to break the flow of a loop and **continue** is used to jump to the next iteration of the loop [7]. So it can be said that the conditional mechanism is not an integral part of the exit. An example of the use of a break statement is given below. In the example, it can be seen that the loop only iterates until break, after the break statement it exits the loop.

```rust
  //example of unconditional exit using break
 let mut loopVar = 10;
 while loopVar >= 1{
    loopVar-=1;
    break;
 }
 println!("The value of loopVar after the loop is {}",loopVar);
```

```
The value of loopVar after the loop is 9
```

The **continue** statement in the while loop just continues to the next iteration of the loop, so in this case1 is incremented from loopVar in each iteration, which in the end becomes 0.

```
//example of using continue
let mut loopVar = 10;
while loopVar >= 1{
    loopVar-=1;
    continue;
}
println!("The value of loopVar after the loop is {}",loopVar);
```

```
The value of loopVar after the loop is 0
```

7.2 Should only one loop body be exited or can enclosing loops also be exited (labeled or unlabeled exit)?

In the Rust language, when there are two enclosing loops, the outer loop can be exited when we write labels for the loops. The **break** statement, followed by the name of the outer label can be used to break the outer loop [7]. So the Rust language supports labeled exits. In the example below, there are two labels called outer and inner and inside of the inner label, the outer loop is breaked by the **break** outer statement. It can be seen that the outer loop is exited after that statement.

```
//example of break in enclosing loops
//outer label
'outer:
    for i in 0..5 {
    println!("Inside the outer loop, i : {}", i);
    println!( "------------------------------");

    //inner label
    'inner:
    for j in 6..9 {

        //condition for breaking
        if(i==2){
           println!("Exiting");
           break 'outer;
        }
        println!("Inside the inner loop, j : {}", j);

    }
    println!( "\n");
}
```

```
Inside the outer loop, i : 0
------------------------------
Inside the inner loop, j : 6
Inside the inner loop, j : 7
Inside the inner loop, j : 8


Inside the outer loop, i : 1
------------------------------
Inside the inner loop, j : 6
Inside the inner loop, j : 7
Inside the inner loop, j : 8


Inside the outer loop, i : 2
------------------------------
Exiting
```

When we use the **continue** statement following with the name of the outer label inside the inner loop, it stops the execution of the inner loop and then continues with the next iteration of the outer loop [7]. A sample code below demonstrates this use of **continue**:

```
//example of continue in enclosing loops
//outer label
'outer:
    for i in 0..5 {
    println!("Inside the outer loop, i : {}", i);
    println!( "------------------------------");

    //inner label
    'inner:
    for j in 6..9 {

        //condition for continuing
        if(i==2){
           println!("Continuing");
           continue 'outer;
        }
        println!("Inside the inner loop, j : {}", j);

    }
    println!( "\n");
}
```

```
Example of continue

Inside the outer loop, i : 0
------------------------------
Inside the inner loop, j : 6
Inside the inner loop, j : 7
Inside the inner loop, j : 8


Inside the outer loop, i : 1
------------------------------
Inside the inner loop, j : 6
Inside the inner loop, j : 7
Inside the inner loop, j : 8


Inside the outer loop, i : 2
------------------------------
Continuing
Inside the outer loop, i : 3
------------------------------
Inside the inner loop, j : 6
Inside the inner loop, j : 7
Inside the inner loop, j : 8


Inside the outer loop, i : 4
------------------------------
Inside the inner loop, j : 6
Inside the inner loop, j : 7
Inside the inner loop, j : 8
```

8.Evaluation of the Languages

8.1 Dart

I think the user located loop variables were both easy to read and write in the Dart language, especially because it supported labels. With labels, the programmer can easily exit enclosing loops using the **break** statement followed by the label name. This made it very easy to write, and it was not a problem in terms of readability either.

8.2. JavaScript

I think the JavaScript was easy to read since the language is close to the English language with its specifiers and keywords and function names. The user located mechanisms, **break** and **continue**, and the loops were straightforward and easy to read. I think JavaScript was also easy to write as well. The existence of labels gave the opportunity to exit enclosing loops, and this made the language writable in my opinion.

8.3 Lua

I think Lua was not really readable or writable compared to the other languages, mainly because it does not have labels. So when the programmer wants to exit enclosing loops, they

have to find workarounds, which means they have to write more lines of code which makes the language hard to both read and write. Also, there is no **continue** statement in Lua, and I think this makes the language both hard to read and write.

8.4 PHP

I think PHP was moderately readable and writable. It does not have labels, so at first I thought breaking enclosing loops was going to make the language hard to write, however, it can exit enclosing loops because the **break** statement accepts a numeric argument next to it which indicates how many loops will be terminated. So I believe PHP is okay in terms of readability and writability.

8.5 Python

I think the readability and the writability of Python were moderate. The user located loop mechanisms were pretty simple. The loops were easy to read and write as well. This might be due to the reason that Python is similar to the English language. However, Python also does not have labels and this makes it hard to exit enclosing loops and it is a hindrance to readability and writability.

8.6 Ruby

I think Ruby was partly readable and writable, it also does not have labels so it makes exiting enclosing loops a bit trickier. The syntax of the loops and the user located loop mechanisms were easy, however, as opposed to many other languages which use **continue**, Ruby language uses the **next** statement. This was not something I have herald of before so I had to look up to the documentations

8.7 Rust

In my opinion, Rust was partly readable and writable in terms of user located loop variables. It supports labels so it was easy to exit enclosing loops which made it readable and writable. However, writing the labels was a bit tricky, since intuitively I would just write the name of the label followed by colons, however, the label names actually start with a single quotation mark. So I had to look up documentation for small things such as this, which made the language less writable.

8.8 Conclusion

I believe out of all these languages, Lua was the hardest in terms of the readability and writability of associative arrays because it neither had labels or a **continue** statement. I also think that the easiest language to implement the user controlled loop variables was Dart. The

labels made it easy to write and read, and the syntax was easy to catch up with. Even though I was not familiar with the Dart language, I had no difficulties writing it.

9.Learning Strategy

To learn the user-located loop variables in all of these languages, I first checked the official documentation of the languages. When I had an error, I checked the internet for the actual syntax that should have been used. When I faced a specific error, I solved it by looking at stackoverflow and other forums. In each loop and each iteration, I printed the array variable so that it can be tracked by the readers of the code. Also, I printed a statement which indicates that the inner/outer loop is being exited. Even though I was not familiar with the languages except for JavaScript, I eventually figured out how to implement user-located loop control mechanisms in these languages in a short amount of time.

Some of the compilers that I have used are:
For Dart: https://dartpad.dev/?
For Javascript: https://www.programiz.com/javascript/online-compiler/
For Lua: https://www.tutorialspoint.com/execute_lua_online.php
For other languages: https://rextester.com/l/lua_online_compiler

Here are some of the other websites that I have used, the ones I referenced in the document can be found in the References section.
https://www.educative.io/answers/what-are-loop-control-statements-in-dart
https://www.tutorialspoint.com/dart_programming/dart_programming_break_statement.htm
https://www.tutorialspoint.com/why-does-lua-have-no-continue-statement
https://www.lua.org/pil/4.3.4.html
https://www.geeksforgeeks.org/how-to-break-an-outer-loop-with-php/
https://www.programiz.com/python-programming/break-continue
https://doc.rust-lang.org/reference/expressions/loop-expr.html
https://doc.rust-lang.org/rust-by-example/flow_control/loop/nested.html

10. References

[1] "*Dart – Loop Control Statements (Break and Continue).*" geeksforgeeks.org https://www.geeksforgeeks.org/dart-loop-control-statements-break-and-continue/ (accessed December 2, 2022).
[2] "*JavaScript Break and Continue .*" w3schools.com https://www.w3schools.com/js/js_break.asp (accessed December 1, 2022).
[3] "*Lua-break Statement.*" tutorialpoints.com https://www.tutorialspoint.com/lua/lua_break_statement.htm (accessed December 1, 2022).
[4] "*PHP Break and Continue.*" w3schools.com https://www.w3schools.com/php/php_looping_break.asp (accessed December 1, 2022).
[5] "*Python break and continue.*" programiz.com

https://www.programiz.com/python-programming/break-continue (accessed December 2, 2022).
[6] "*Ruby Break and Next Statement*."
https://www.geeksforgeeks.org/ruby-break-and-next-statement/ (accessed December 1, 2022).
[7] "*Rust break and continue.*"
programiz.comhttps://www.programiz.com/rust/break-continue (accessed December 3, 2022).