Bilkent University

CS 315 Programming Languages Homework #2

User-Controlled Loop Control Mechanisms in Dart, JavaScript, Lua, PHP, Python, Ruby and Rust

Barış Tan Ünal | 22003617

03.12.2022

Table of Contents

1.0 CODE SEGMENTS AND THE RESULTS OF EXECUTION

1.1 DART

1.1.1 CODE SEGMENT

```
void main() {

  // PART 1: Should the conditional mechanism be an integral part of
the exit (conditional or unconditional exit)?

  // Both conditional and unconditional exits are possible to
implement in Dart. A simple conditional exit is as follows.
  print( "\nCONDITIONAL EXIT \nCountdown from 10:" );
  var x = 9;
  while( x >= 0 ){
    print(x);
    x--;
  }

  // However, conditional mechanism is NOT an integral part of the
exit in Dart. One can use "break" statement. There are two types of
break statements. First one is the unlabeled one.
   print( "\nUNCONDITIONAL UNLABELED EXIT \nCounting up to 10:" );
  for( var i = 0; i < 10; i++ ) {
    if( i == 5 ) {
      break;
    }
    print(i);
  }
  // The loop is exited once i becomes 5, which causes to stop
counting at after 4.

  // PART 2: Should only one loop body be exited or can enclosing
loops also be exited (labeled or unlabeled exit)?

  // The second option to use the "break" statement is to state a
label which points a line of code to continue executing.
  print( "\nUNCONDITIONAL LABELED EXIT \nThree nested loops with break
statement: " );
  for( var i = 0; i <= 2; i++ ) {
    print( "Inside first loop i = " + i.toString() );
    loop1:
    for( var j = 0; j <= 2; j++ ) {
      print( "Inside second loop j = " + j.toString() );
```

```
      for( var k = 0; k <= 2; k++ ) {
        if( j > 1 ) {
          break loop1;
          // exits the second loop and continues execution of the
first loop's body
        }
        print( "Inside third loop k = " + k.toString());
      }
    }
  }
  // When we reach the innermost loop when j==2 i.e, for the third
time, the "break" statement is executed and the block which label
"loop1" points to i.e, the second loop, is exited.


    // The "continue" keyword can be used with or without labels. The
one without a label is as follows.
  print( "\nUNLABELED CONTINUE \nTwo nested loops with unlabeled
continue statement:" );
  for( var i = 0; i <= 3; i++ ) {
    print( "Inside first loop i = " + i.toString() );
    for( var j = 0; j <= 3; j++ ) {
      if( j == 1 ){
        continue;
      }
      print( "Inside second loop j = " + j.toString() );
    }
  }
  // When j==1, continue statement is executed and the program
continues to check the condition for the innermost loop of the body
that "continue" statement is in and executes the body if the condition
holds.

  // A "continue" statement with a label is as follows.
  print( "\nLABELED CONTINUE \nTwo nested loops with labeled continue
statement:" );
  loopOuter:
  for( var i = 0; i <= 3; i++ ) {
    print( "Inside first loop i = " + i.toString() );
    for( var j = 0; j <= 3; j++ ) {
      if( j == 1 ){
        continue loopOuter;
      }
      print( "Inside second loop j = " + j.toString() );
    }
  }
```

```
  // When j==1, continue statement is executed and the program
continues to check the condition for the outer loop since the label
"loopOuter" points to that loop. Therefore, we never see j become 1, 2
or 3.
}
```

1.1.2 EXPLANATION

In Dart programming language, one can use either conditional or unconditional statements to exit a loop. In my program, I have shown the classical conditional while loop at the beginning. That program counts down from 10 to 0. When x becomes less than 0, the condition becomes false, and the program exits the loop. However, conditional mechanism is not an integral part of the exit in Dart. One can use "break" statements to unconditionally exit the loop while executing the loop body without checking the condition of the loop itself. However, it does not make sense to just put a break statement in the body of the loop since it would always exit the loop no matter what happens before the "break" statement. In that case, implementing a loop structure would be useless. Therefore, I have put the "break" in an if statement. It exits the loop when the "break" statement is executed which happens when variable "i" becomes 5.

In the case for nested loops, unlabeled "break" statement exits the innermost loop. In order to choose which loop to exit, one can use labels. In my program, I have put a label to the body of the second (middle) loop. So, when the "break loop1" statement is executed, the program exits the current execution of the middle loop and continues with the execution of the outer loop. Therefore, we do not see the values of k when j becomes 2 which is greater than 1.

Also, there is "continue" statement in Dart. This statement can be labeled and unlabeled just as the "break". In the example of an unlabeled one, the program skips the execution of the inner loop when j is 1. Therefore, we do not see the program printing "j = 2" ever. But it does not terminate the loop completely unlike the "break" statement. Instead, it continues with the next iteration. The program printing "j = 2" and "j = 3" is a proof of that.

Labeled "continue" has exactly the same logic as labeled "break". It allows us to choose which loop's body to skip. In my program, I have shown that one can also skip the execution of the outer loop when a "continue" is encountered. Unlike "break", it continues with the next iteration, therefore we see "i" becoming 2 and 3 respectively.

1.2 JAVASCRIPT

1.2.1 CODE SEGMENT

```
// PART 1: Should the conditional mechanism be an integral part of the
exit (conditional or unconditional exit)?

// Both conditional and unconditional exits are possible to implement
in JavaScript. A simple conditional exit is as follows.
console.log( "CONDITIONAL EXIT \nCountdown from 10:" );
x = 9;
while( x >= 0 ){
```

```
  console.log(x);
  x--;
}

// However, conditional mechanism is NOT an integral part of the exit
in JavaScript. One can use "break" statement. There are two types of
break statements. First one is the unlabeled one.
console.log( "\nUNCONDITIONAL UNLABELED EXIT \nCounting up to 10:" );
for( let i = 0; i < 10; i++ ) {
  if( i == 5 ) {
    break;
  }
  console.log(i);
}

// PART 2: Should only one loop body be exited or can enclosing loops
also be exited (labeled or unlabeled exit)?

// The second option to use the "break" statement is to state a label
which points a line of code to continue executing.
console.log( "\nUNCONDITIONAL LABELED EXIT \nThree nested loops with
break statement: " );
for( let i = 0; i <= 2; i++ ) {
  console.log( "Inside first loop i = " + i );
  loop1:
  for( let j = 0; j <= 2; j++ ) {
    console.log( "Inside second loop j = " + j );
    for( let k = 0; k <= 2; k++ ) {
      if( j > 1 ) {
        break loop1;
        // exits the second loop and continues execution of the first
loop's body
      }
      console.log( "Inside third loop k = " + k);
    }
  }
}
// When we reach the innermost loop when j==2 i.e, for the third time,
the "break" statement is executed and the block which label "loop1"
points to i.e, the second loop, is exited.

// The "continue" keyword can be used with or without labels. The one
without a label is as follows.
console.log( "\nUNLABELED CONTINUE \nTwo nested loops with unlabeled
continue statement:" );
for( let i = 0; i <= 3; i++ ) {
```

```
  console.log( "Inside first loop i = " + i );
  for( let j = 0; j <= 3; j++ ) {
    if( j == 1 ){
      continue;
    }
    console.log( "Inside second loop j = " + j );
  }
}
// When j==1, continue statement is executed and the program continues
to check the condition for the innermost loop of the body that
"continue" statement is in and executes the body if the condition
holds.

// A "continue" statement with a label is as follows.
console.log( "\nLABELED CONTINUE \nTwo nested loops with labeled
continue statement:" );
loopOuter:
for( let i = 0; i <= 3; i++ ) {
  console.log( "Inside first loop i = " + i );
  for( let j = 0; j <= 3; j++ ) {
    if( j == 1 ){
      continue loopOuter;
    }
    console.log( "Inside second loop j = " + j );
  }
}
// When j==1, continue statement is executed and the program continues
to check the condition for the outer loop since the label "loopOuter"
points to that loop. Therefore, we never see j become 1, 2 or 3.
```

1.2.2 EXPLANATION

Just as the Dart programming language, it is possible to exit to loop with both conditional and unconditional statements in JavaScript. The same countdown example is given in the sample program above. Again, as in Dart, conditional mechanism is not an integral part of the exit in JavaScript. It has “break” statements which breaks the innermost loop. Same “break” example is given in which the program exits the loop when i becomes 5.

JavaScript has a labeled version of the “break”, too. Just as in Dart, one can specify which loop body to stop executing and terminate. In my example I have shown that the middle loop can be exited with a labeled “break” in the case for three nested for loops. The program does not print the values of k after encountering the “break” statement and continues with the next value of i.

JavaScript has two kinds of “continue” statement as it was in Dart. The unlabeled one’s behavior is the same as Dart which skips the current execution of the innermost loop, not the outer one. However, one can use labeled “continue” to specify which enclosing loop to skip executing by putting a label just before that for loop statement. In my program, I have put a label

to the outer loop and put a “continue” statement in the body of the inner loop which skips executing the current body of the whole outer loop. Therefore, we do not see the values of j ever become 1, 2 or 3 even though we were expecting j to count up to 3 if there was not a “continue” statement.

1.3 LUA

1.3.1 CODE SEGMENT

```
-- PART 1: Should the conditional mechanism be an integral part of the
exit (conditional or unconditional exit)?

-- Both conditional and unconditional exits are possible to implement
in Lua. A simple conditional exit is as follows.
print( "\nCONDITIONAL EXIT \nCountdown from 10:" )
x = 9;
while( x >= 0 )
do
  print( x );
  x = x - 1;
end

-- However, conditional mechanism is NOT an integral part of the exit
in Lua. One can use "break" statement for unconditional unlabeled
exit.
print( "\nUNCONDITIONAL UNLABELED EXIT BREAK \nCounting up to 10
interrupted by break:" );
for i = 0, 10, 1
do
  if i == 5 then break end
  print(i);
end
-- The loop is exited once i becomes 5, which causes to stop counting
at after 4.

-- PART 2: Should only one loop body be exited or can enclosing loops
also be exited (labeled or unlabeled exit)?
-- The "break" statement only exits the innermost loop in Lua.
print( "\nUNCONDITIONAL UNLABELED EXIT \nThree nested loops with break
statement: " );
for i = 0, 2, 1
do
  print( "\tIn the first loop i = ", i );
  for j = 0, 2, 1
  do
```

```
    print( "\t\tIn the second loop j = ", j );
    for k = 0, 2, 1
    do
      print( "\t\t\tIn the third loop k = ", k );
      if i == 1 then break end
    end
  end
end
-- Only the third loop was exited when i equals 1.

print( "\nUNCONDITIONAL LABELED EXIT \nNested for loops with a goto
statement to illustrate which statement is executed, i.e., which for
loop is being exited: " );
x = 0;
for i = 0, 10, 1 do
  for j = 0, 10, 1 do
    goto out
  end
  x = 1;
end
::out::
print( "x = ", x);
-- In the inner loop, the program jumps to the line of the label 'out'
and therefore x never becomes 1.

-- In Lua, one can only exit the innermost loop body with "break"
statement. There is no way to exit outer loops with "break" statement
because there is no labeled "break" statements unlike JavaScript and
Dart.
-- "Goto" statements does not work neither since a label is only
visible in the block that it is defined. In the case for nested loops,
the 'label' would be invisible from the line goto 'label'.
print( "\nUNCONDITIONAL LABELED EXIT WITH GOTO \nThree nested loops
with goto statement: " );
i = 0;
while i <= 2
do
  i = i + 1;
  print( "In the first loop i = ", i );
  j = 0;
  ::exit::
  while j <= 2
  do
    j = j + 1;
    print( "\tIn the second loop j = ", j );
    k = 0;
```

```
    while k <= 2
    do
      k = k + 1;
      print( "\t\tIn the third loop k = ", k );
      if i == 2 then
          goto exit
        end
    end
  end
end
```

1.3.2 EXPLANATION

In Lua, it is possible to exit a loop with both a conditional and unconditional statement. So, one can easily say that conditional mechanism is not an integral part of the exit in Lua. I have the exact same example with Dart and JavaScript for conditional exit. My program counts down from 10 to 0 and stops printing values of x when it becomes negative. The keyword for unconditional exit statement is again the same with Dart and JavaScript which is “break”.

Since there is no labeled version of “break” in Lua, it always exits the innermost loop when it is executed. There is a great difference between Lua and JavaScript in the sense of the possibility of exiting enclosing loops with a “break” statement. Even though we cannot use “break” to exit an enclosing loop in Lua, there is another unconditional statement which is called “goto”. It has the exact same logic with the “jump” statement in MIPS Assembly. Unlike Dart and JavaScript, there is no obstacle that was set by the language to write a “spaghetti code”. Even though “goto” statements may lead to unstructured programs, it is possible to use it as a labeled unconditional exit in the loops. In my program I have shown an example of a “goto” statement which was used as a labeled unconditional exit. When the program reaches the body of the inner loop, it jumps to outside of the outer loop which results in x remaining 0 and never becoming 1.

One can use “goto” statements to choose which loop to exit in the case for multiple nested loops. I have used “goto” to exit the middle loop in my last example. When my inner loop body detects that variable i has become 2, it jumps to the beginning of the middle loop’s condition which then continues with its execution. This is the same as breaking the innermost loop. One can change the position of the label to choose which loop to break.

There is no “continue” statement in Lua, but one can use “goto” statement which acts like a “continue” by putting the label just before the condition of the loop that they would like to skip. For example, my last example is the same as a labeled “continue” statement for the middle loop.

1.4 PHP

1.4.1 CODE SEGMENT

```
<?php
// PART 1: Should the conditional mechanism be an integral part of the
exit (conditional or unconditional exit)?
```

```
  // Both conditional and unconditional exits are possible to
implement in PHP. A simple conditional exit is as follows.
print( "\nCONDITIONAL EXIT \nCountdown from 10:" );
  $x = 9;
  while( $x >= 0 ){
    print_r("\n");
    print_r($x);
    $x--;
  }

// However, conditional mechanism is NOT an integral part of the exit
in PHP. One can use "break" statement. There are two types of break
statements. First one is the unlabeled one.
print( "\n\nUNCONDITIONAL UNLABELED EXIT \nCounting up to 10:" );
for( $i = 0; $i < 10; $i++ ) {
  if( $i == 5 ) {
    break;
  }
  print_r("\n");
  print_r($i);
}
  // The loop is exited once i becomes 5, which causes to stop
counting at after 4.

// In the case for nested for loops, "break" statement exits the
innermost for loop.
print( "\n\nUNCONDITIONAL UNLABELED EXIT \nNested for loops and brake
usage: " );
$x = 0;
for ( $i = 0; $i < 10; $i++ ) {
  for ( $j = 0; $j < 10; $j++ ) {
    $x = 1;
    break;
  }
  $x = 2;
}
print( "\nx = " );
print( $x );
// x is 2 at the end

// PART 2: Should only one loop body be exited or can enclosing loops
also be exited (labeled or unlabeled exit)?

// We can specify the number of layers to exit, too, with "break"
statement. In the following example, the program goes two layers up
```

```
when j equals 1. This results in continuing with the first loop when
we encounter the "break" statement.
print( "\n\nUNCONDITIONAL LABELED EXIT WITH BREAK \nNested for loops
and brake usage with specifying layers (2): " );
for( $i = 0; $i < 3; $i++ ) {
  print( "\n\tIn the first loop i = " );
  print( $i );
  for( $j = 0; $j < 3; $j++ ) {
    print( "\n\t\tIn the second loop j = " );
    print( $j );
    for( $k = 0; $k < 3; $k++ ) {
      if( $j == 1 ) {
        break 2;
      }
      print( "\n\t\t\tIn the third loop k = " );
      print( $k );
    }
  }
}

// But in this case break statement goes only one layer up, which
causes the program to only skip the print statements of third loop
when j equals 1.
print( "\n\nUNCONDITIONAL LABELED EXIT WITH BREAK \nNested for loops
and brake usage with specifying layers (1): " );
for( $i = 0; $i < 3; $i++ ) {
  print( "\n\tIn the first loop i = " );
  print( $i );
  for( $j = 0; $j < 3; $j++ ) {
    print( "\n\t\tIn the second loop j = " );
    print( $j );
    for( $k = 0; $k < 3; $k++ ) {
      if( $j == 1 ) {
        break 1;
      }
      print( "\n\t\t\tIn the third loop k = " );
      print( $k );
    }
  }
}

// There is also the "goto" statement in PHP.
print( "\n\nUNCONDITIONAL LABELED EXIT WITH GOTO \nNested for loops
and goto usage: " );
$x = 0;
for( $i = 0; $i < 3; $i++ ) {
```

```
  $x = 1;
  for( $j = 0; $j < 3; $j++ ) {
    goto exitLabel;
  }
  $x = 2
}
exitLabel:
print( "\nx = " );
print($x);
?>
```

1.4.2 EXPLANATION

In PHP, conditional expression is not an integral part of the exit since one can use both conditional and unconditional statements for exiting a loop. The first three examples are exactly the same with Dart and JavaScript. However, one can specify the number of layers to exit in the case for nested for loops. Using a “break” statement without specifying number of layers is the same as writing “break 1” which only exits the innermost loop. The explanations of what the code does is given in comments and since they are the same with the examples of JavaScript and Dart, their explanations are omitted here.

The fourth example which uses the layered version of “break” shows how one can use “break” in PHP just like specifying a label in Dart and JavaScript. In my example, the “break” statement inner loop’s body exits the execution of the middle loop and does not print the values of j and k when i is 1. In this case there are no labels, but it acts as a labeled unconditional exit. Similarly, breaking 1 layer means that it will exit the inner loop and will not print the values of k when j becomes 1, as it was shown in the next example.

There is a “goto” statement in PHP, too. It could be used as a “continue” statement again as it was explained in 1.3.2. In my example, I have used it to skip the execution of the outer loop. It has a disadvantage of causing a possible spaghetti code but also it gives the programmer a certain flexibility. But in many cases, there is no need to use “goto” statement in the loops.

1.5 PYTHON

1.5.1 CODE SEGMENT

```
# PART 1: Should the conditional mechanism be an integral part of the
exit (conditional or unconditional exit)?

# Both conditional and unconditional exits are possible to implement
in Python. A simple conditional exit is as follows.
print("\nPART 1 \nCONDITIONAL EXIT \nPowers of two that have two
digits at maximum:")
x = 1
# Loop body is executed until the condition does not hold which
happens when x equals 128 for the first time.
```

```
while x < 100:
  print(x)
  x = x * 2

# However, conditional mechanism is NOT an integral part of the exit
in Python. One can use "break" statement which exits the innermost
loop body.
print( "\nUNCONDITIONAL EXIT" )
x = 1
print( "Before entering the loop, x = " + str(x) )
while x < 50:
  x = x + 1
  print( "Inside the loop after incrementing, x = " + str(x) )
  break
  x = x * 3
  print( "Inside the loop after multiplying by 3, x = " + x )
print( "After exiting the loop, x = " + str(x) )
# The loop body would be executed for 4 times if there was no "break"
statement but it exits the loop immediately without any conditions
when the line with "break" is executed.

# PART 2: Should only one loop body be exited or can enclosing loops
also be exited (labeled or unlabeled exit)?

# There is no "goto" statement in Python. Instead, there is a "break"
statement as a unconditional unlabeled exit which exits the innermost
loop in nested loop structures. Unlike Java, one CANNOT directly
perform a unconditional labeled exit with "break". An example of the
usage of "break" is given below.
print( "\nPART 2 \nThe positive composite numbers between 1 and 30
exclusive: " )
for i in range (2,10,1):
  for j in range( 2, 10, 1 ):
    if( ( i*j > 30 ) or ( i*j == 2 ) ):
      break
    print( str(i) + " * " + str(j) + " = " + str(i*j) )
# As one can see, when i*j value exceeds 30, the inner loop is exited
and the outer loop continues to iterate and execute its body. If the
outer loop was meant to be exited, then the multiplications after
4*7=28 would not have been printed at all. But it continues with
5*2=10 which proves that "break" only exits one loop body but not the
enclosing loops.

# There is also a "continue" statement which stops the current
execution of the body of the loop and continues to the next iteration
if the condition of the loop still holds.
```

```
print( "\nOne digit odd numbers except 5 (with continue): " )
oddNumbers = range( 1, 10, 2)
for i in oddNumbers:
  if( i == 5 ):
    continue
  print(i)

# But "break" statement would have terminated the loop entirely
without continuing with the next iterations, which would not give the
promised result in the print statement.
print( "\nOne digit odd numbers except 5 (with break): " )
oddNumbers = range( 1, 10, 2)
for i in oddNumbers:
  if( i == 5 ):
    break
  print(i)
```

1.5.2 EXPLANATION

Just as the other languages, conditional mechanism is not an integral part of the exit in Python. Both conditional and unconditional exits are possible. The explanation of the code is given with comments. Unlike Dart and JavaScript, the “break” and “continue” statements have only one version in Python which is unlabeled. In my examples, one can easily see that only the innermost loop is exited. If the outer loop was meant to be exited, then the multiplications after 4*7=28 would not have been printed at all. But it continues with 5*2=10 which proves that "break" only exits one loop body but not the enclosing loops.

The “continue” has the same logic since it skips the current execution of the body that it was called in. The difference between “continue” and “break” is given in the last two examples. In conclusion, it is not possible to exit the enclosing loops directly with a single “break” statement in nested loop structures in Python.

1.6 RUBY

1.6.1 CODE SEGMENT

```
puts 'Hello, Ruby!'

# PART 1: Should the conditional mechanism be an integral part of the
exit (conditional or unconditional exit)?

# Both conditional and unconditional exits are possible to implement
in Ruby. A simple conditional exit is as follows.
puts( "\nCONDITIONAL EXIT \nCountdown from 10:" );
x = 9
while( x >= 0 ) do
```

```
  puts( x )
  x = x - 1
end

# However, conditional mechanism is NOT an integral part of the exit
in Dart. One can use "break" statement. There are two types of break
statements. First one is the unlabeled one.
puts( "\nUNCONDITIONAL UNLABELED EXIT WITH BREAK \nCounting up to 10:"
)
for i in 0..10 do
  if i == 5 then
    break
  end
  puts( i );
end
# The loop is exited once i becomes 5, which causes to stop counting
at after 4.

# PART 2: Should only one loop body be exited or can enclosing loops
also be exited (labeled or unlabeled exit)?

# There is no labelled version of "break", but one can use exceptions
to exit the enclosing loops in nested loops.
puts "\nUNCONDITIONAL LABELED EXIT WITH EXCEPTION \nInteger couples in
a two layer nested loop."
catch (:exit) do
  4.times do |i|
    4.times do |j|
      throw :exit if i + j >= 5
      puts "i = #{i}, j = #{j}"
    end
  end
end
# The program throws an exception in the inner loop when i+j equals 5
where i = 2 and j = 3.

# Additionally, Ruby has a "next" statement instead of "continue" in
JavaScript and others.
puts "\nUNCONDITIONAL UNLABELED NEXT \nCounting even numbers up to 10
with 'next' statement."
x = 0
while x <= 10
  if x % 2 == 1 then
    x = x + 1
    next
  end
```

```
  puts "x = #{x}"
  x = x + 1
end
# This loop does not print odd numbers since it will skip the current
iteration when x is odd.
```

1.6.2 EXPLANATION

Just as other languages, conditional mechanism is not an integral part of exit in Ruby. The exact same examples that were given in Dart are also given in Ruby in the first two examples of countdown and breaking the count.

There is no labeled version of “break” in Ruby. Just as the others, “break” statement exits the inner loop, and it is not possible to exit an enclosing loop with a “break” statement since one cannot declare labels in Ruby. Instead, exceptions could be used to break an outer loop as in the example in the code segment. It throws an exception when i + j becomes 5 where i = 2 and j = 3.

Instead of “continue”, Ruby has “next” statement which has the same functionality. It affects the inner loop again. In my example, the program skips the executions of the body of the loop whenever x is odd, therefore we only see even numbers being printed.

1.7 RUST

1.7.1 CODE SEGMENT

```
fn main() {

  // PART 1: Should the conditional mechanism be an integral part of
the exit (conditional or unconditional exit)?

  // Both conditional and unconditional exits are possible to
implement in Rust. A simple conditional exit is as follows.
  println!( "\nCONDITIONAL EXIT \nCountdown from 10:" );
  let mut x = 9;
  while x >= 0 {
    println!( "x = {}", x );
    x = x - 1;
  }

  // However, conditional mechanism is NOT an integral part of the
exit in Rust. One can use "break" statement. There are two types of
break statements. First one is the unlabeled one.
  println!( "\nUNCONDITIONAL UNLABELED EXIT \nCounting up to 10:" );
  for x in 1..10 {
    if x > 4 {
        break;
    }
```

```
    println!( "x = {}", x );
  }
  // The loop is exited once x becomes 5, which causes to stop
counting at after 4.

  // In the case for nested for loops, "break" statement exits the
innermost for loop.
  println!( "\n\nUNCONDITIONAL UNLABELED EXIT \nNested for loops and
brake usage: " );
  let mut x = 0;
  for _i in 1..10 {
    for _j in 1..10 {
      x = 1;
      break;
    }
    x = 2;
  }
  println!( "x = {}", x );

// x is 2 at the end

// PART 2: Should only one loop body be exited or can enclosing loops
also be exited (labeled or unlabeled exit)?

// The second option to use the "break" statement is to state a label
which points a line of code to continue executing.
println!( "\nUNCONDITIONAL LABELED EXIT \nThree nested loops with
break statement: " );
  for i in 0..3 {
    println!( "Inside first loop i = {}", i );
    'loop1:
    for j in 0..3 {
      println!( "\tInside second loop j = {}", j );
      for k in 0..3 {
        if j > 1 {
          break 'loop1;
          // exits the second loop and continues execution of the
first loop's body
        }
        println!( "\t\tInside third loop k = {}", k );
      }
    }
  }

// Also there is an unlabeled 'continue' statement in Rust. It skips
the current execution of the loop body and continues with the next
```

```
iteration. Unlabeled version of 'continue' skips the current execution
of the innermost loop.
println!( "\nUNCONDITIONAL UNLABELED CONTINUE \nTwo nested loops: " );
  for x in 0..5 {
    for y in 0..5 {
        if y % 2 != 0 {
          continue;
        } // skips the execution of the inner loop
        println!("x: {}, y: {}", x, y);
    }
  } // this loop prints all of the x's but only even y's

  // The last one is the labeled continue statement where one can
manipulate which loop to skip the iteration of.
  println!( "\nUNCONDITIONAL LABELED CONTINUE \nTwo nested loops where
the program prints all of the even number combinations for two numbers
up to 10: " );
  'loopOuter: for x in 0..10 {
    'loopInner: for y in 0..10 {
        if x % 2 != 0 {
          continue 'loopOuter;
        } // skips the execution of the outer loop
        if y % 2 != 0 {
          continue 'loopInner;
        } // skips the execution of the inner loop
        println!("x: {}, y: {}", x, y);
    }
  } // this loop prints only the even x's and event y's

}
```

1.7.2 EXPLANATION

All possible scenarios are available in Rust except “goto” statement, which is not a crucial functionality in loops. Conditional mechanism is not an integral part of the exit in Rust. The exact same examples with Dart are given in the first two examples which shows very basic examples of conditional and unconditional exits. The third example in the first part proves that “break” statements without a label only exits the inner loop in the case for nested loops. If it was meant to exit the enclosing loops, x would be 1 at the end.

The labeling logic is the same with JavaScript where “break” followed by a label exits the loop that is pointed by that label. Therefore, we do not see the values of k being printed when j becomes 2.

Additionally, there is both unlabeled and labeled version of “continue”. The explanations of the code were given with comments. It has the same logic with JavaScript where “continue” followed by a label skips the current execution of the loop body that is pointed by that label.

2.0 EVALUATION

These seven languages vary in their design in a few aspects. The conditional mechanism is not an integral part of the exit in none of them and both conditional and unconditional exits are possible, so that does not make a difference at all. Also, all of these languages have an unlabeled version of "break" statement which exits the body of the innermost loop.

There is a difference between these languages when we try to exit an enclosing loop. In Python, it is impossible to exit the enclosing loop directly with a single "break" statement. Since there is only one possibility, it increases the readability, but it drastically decreases writability since one should implement a function for such a simple objective or exit the enclosing loops indirectly by combining "break" and "continue" statements, which might even decrease readability too.

It is possible to point to a loop to specify which loop to be exited with a labeled version of "break" statement in Dart, JavaScript and Rust. This design choice increases both the readability and the writability in my opinion since it is both easy to find and specify the loop that we want to exit with the pointer logic. PHP's logic of specifying layers could raise some readability problems since in a complicated nested loop, it could be difficult to count the layers. Ruby's inability to exit enclosing loops with a "break" statement affects its readability and writability negatively.

Finally, "goto" statements are risky since it could lead to unreadable and badly structured code. Therefore, Lua and PHP are not preferable.

In conclusion, Rust is the most preferable language to implement for loops in terms of readability and writability.

3.0 LEARNING STRATEGY

Each time I have started to implement associative arrays in a new programming language for the report, first of all I have looked up for documentation of comments since I have used comments in all of my code segments to separate sections. Then, I have learned the structure of a basic while and for loop in that language. Some of those languages had a very different documentation of for loops such as Ruby. After that, I have checked if there is a possibility of exiting the loops unconditionally which each seven language supports. The complication was with the existence of labeled unconditional exit statements in most cases. Additionally, not every language had "continue" statements or some of them had it but with a different name. The documentation of the languages was sufficient to solve these problems. However, after I was not able to find a way to break a loop with a label while writing a program in Ruby, I have used someone's entry on "www.stackoverflow.com" website which suggested to use exceptions for labeled unconditional exits.

If I had any unexpected errors that I cannot solve, I have generally checked "www.stackoverflow.com" website to find the similar errors and their solutions. I have used "www.w3schools.com" and "www.tutorialspoint.com" website for further documentation information with examples in some cases.

I have used "www.replit.com" as my online compiler except for Lua. For Lua, I have used https://www.tutorialspoint.com.

I have had a little personal communication with some of my friends on the evaluation of the languages on readability and writability to make sure that I have not missed a data structure that is more efficient and usable for any of the programming languages. Below, one can access all of the online sources that I have made use of.

RUST:
https://doc.rust-lang.org/std/keyword.break.html
https://loige.co/how-to-to-string-in-rust/
https://web.mit.edu/rust-lang_v1.25/arch/amd64_ubuntu1404/share/doc/rust/html/book/first-edition/loops.html

RUBY:
https://docs.ruby-lang.org/en/2.4.0/syntax/control_expressions_rdoc.html
https://ruby-doc.org/docs/ruby-doc-bundle/Tutorial/part_02/loops.html

LUA:
https://stackoverflow.com/questions/70523546/lua-goto-back-to-for-loop
https://stackoverflow.com/questions/23090836/how-to-jump-out-of-the-outer-loop-if-inner-for-loop-is-executed-in-lua
https://www.lua.org/pil/4.4.html
https://www.tutorialspoint.com/lua/lua_variables.htm
https://www.tutorialspoint.com/lua/lua_for_loop.htm

PYTHON:
https://docs.python.org/3/tutorial/controlflow.html
https://wiki.python.org/moin/ForLoop

DART:
https://dart.dev/guides/language/language-tour#for-loops
https://dart.dev/guides/language/language-tour#break-and-continue

PHP:
https://www.php.net/manual/en/control-structures.for.php
https://www.w3schools.com/php/php_looping_for.asp

ONLINE COMPILERS:
https://replit.com/languages/python3
https://replit.com/languages/nodejs
https://replit.com/languages/dart
https://replit.com/languages/php_cli
https://replit.com/languages/ruby
https://replit.com/languages/rust
https://www.tutorialspoint.com/execute_lua_online.php