**2022-2023 Fall**

**CS 315 - Programming Languages**

**Homework 2 – Report**

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

Berk Çakar - 22003021 - Section 1

# Table of Contents

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

*a) Dart* ..... *3*

*b) Javascript* ..... *5*

*c) Lua* ..... *7*

*d) PHP* ..... *9*

*e) Python* ..... *11*

*f) Ruby* ..... *12*

*g) Rust* ..... *14*

**Evaluation** **16**

**Learning Strategy** **17**

**References** **17**

## a) Dart

```dart
void main() {
  print("INTRODUCTION TO USER-LOCATED LOOP CONTROL MECHANISMS IN DART");

  while (true) {
    print("iterating once");
    break;
  }

  for (int i = 0; i < 10; i++) {
    continue; // continue to the next iteration without executing the rest of the loop body (print statement)
    print("$i");
  }

  // Labels can also be used to break out of nested loops (see the examples below)

  print("\nSAMPLE 1: BREAK ------------------------------------------------------------");

  // An example for break
  for (int i = 0; i < 44; i++) {
    if (i == 2) {
      /*
        When i is 2 in the outer loop, break statement will be executed
        and the outer loop will be terminated
      */
      print(
          "  in outer loop i is 2, now break out of the outer loop, otherwise i would go as 0, 1, 2, ..., 43");
      break;
    }

    print("in outer loop i is $i");

    for (int j = 0; j < 12; j++) {
      /*
        When j is 5 in the inner loop, break statement will be executed
        However, the outer loop will continue to run since the break statement
        only breaks out of the inner loop
      */
      if (j == 3) {
        print(
            "  in inner loop j is 3, now break out of the inner loop, otherwise j would go as 0, 1, 2, ..., 11");
        break;
      }
      print("  in inner loop j is $j");
    }
  }

  print("\nSAMPLE 2: CONTINUE ---------------------------------------------------------");

  // An example for continue
  var i = 0;
  while (i < 10) {
    i++;
    if (i % 2 == 0) {
      /*
        When i is even, continue statement will be executed
        and the while loop will continue to the next iteration
        without executing the print statement for even numbers
      */
      continue;
    }
    print("in while loop, i is odd, now printing: i = $i");
  }

  print("\nSAMPLE 3: LABELED BREAK ----------------------------------------------------");

  // An example for labeled break
  outerLoop:
  for (int j = 0; j < 10; j++) {
    print("in outer loop j is $j");

    int i = 0;
    // In Dart, labels can be used to break or continue a specific loop
    innerLoop:
    do {
      print("  in inner loop, i is $i");
      /*
        When j is 5, break statement will be executed in the inner loop
        and the outer loop will be terminated. Otherwise, the outer loop
        would continue to run until j is 9
      */
      if (j == 5) {
        print("j is 5, break out the outer loop, normally 'j' would go as 0, 1, 2, ..., 9");
        break outerLoop;
      }
      /*
        When i is 3, break statement will terminate the inner loop and
        the outer loop will continue to run. Otherwise, the inner loop
        would continue to run until i is 2
      */
      if (i == 1) {
        print("  i is 1, break out the inner loop, normally 'i' would go as 0, 1, 2");
        break innerLoop;
      }
      i++;
    } while (i < 3);
  }

  print("\nSAMPLE 4: LABELED CONTINUE -------------------------------------------------");

  // An example for labeled continue
  outerLoop:
  for (var i = 0; i <= 3; i++) {
    print("in outer loop i is $i");
    innerLoop:
    for (var j = 0; j <= 2; j++) {
      /*
        When j is 1, continue statement will be executed in the inner loop
        which means the inner loop will continue to the next iteration
        without executing the print statement when j is 1
      */
      if (j == 1) {
        continue innerLoop;
      }
      /*
        When i is 2, continue statement will be executed in the outer loop
        which means the outer loop will continue to the next iteration
        with skipping the inner loop
      */
      if (i == 2) {
        continue outerLoop; // same as break innerloop;
      }
      print("  in inner loop j is $j");
    }
  }
}
```

Figure 1: Code samples from the Dart programming language

```
friberk@Berk-MacBook-Pro  hw2 ❯ dart "cakar_berk.dart"
INTRODUCTION TO USER-LOCATED LOOP CONTROL MECHANISMS IN DART
iterating once

SAMPLE 1: BREAK ------------------------------------------------------------
in outer loop i is 0
  in inner loop j is 0
  in inner loop j is 1
  in inner loop j is 2
  in inner loop j is 3, now break out of the inner loop, otherwise j would go as 0, 1, 2, ..., 11
in outer loop i is 1
  in inner loop j is 0
  in inner loop j is 1
  in inner loop j is 2
  in inner loop j is 3, now break out of the inner loop, otherwise j would go as 0, 1, 2, ..., 11
  in outer loop i is 2, now break out of the outer loop, otherwise i would go as 0, 1, 2, ..., 43

SAMPLE 2: CONTINUE ---------------------------------------------------------
in while loop, i is odd, now printing: i = 1
in while loop, i is odd, now printing: i = 3
in while loop, i is odd, now printing: i = 5
in while loop, i is odd, now printing: i = 7
in while loop, i is odd, now printing: i = 9

SAMPLE 3: LABELED BREAK ----------------------------------------------------
in outer loop j is 0
  in inner loop, i is 0
  in inner loop, i is 1
  i is 1, break out the inner loop, normally 'i' would go as 0, 1, 2
in outer loop j is 1
  in inner loop, i is 0
  in inner loop, i is 1
  i is 1, break out the inner loop, normally 'i' would go as 0, 1, 2
in outer loop j is 2
  in inner loop, i is 0
  in inner loop, i is 1
  i is 1, break out the inner loop, normally 'i' would go as 0, 1, 2
in outer loop j is 3
  in inner loop, i is 0
  in inner loop, i is 1
  i is 1, break out the inner loop, normally 'i' would go as 0, 1, 2
in outer loop j is 4
  in inner loop, i is 0
  in inner loop, i is 1
  i is 1, break out the inner loop, normally 'i' would go as 0, 1, 2
in outer loop j is 5
  in inner loop, i is 0
j is 5, break out the outer loop, normally 'j' would go as 0, 1, 2, ..., 9

SAMPLE 4: LABELED CONTINUE -------------------------------------------------
in outer loop i is 0
  in inner loop j is 0
  in inner loop j is 2
in outer loop i is 1
  in inner loop j is 0
  in inner loop j is 2
in outer loop i is 2
in outer loop i is 3
  in inner loop j is 0
  in inner loop j is 2
```

Figure 2: Output of the code samples given in Figure 1

A user can use break, continue, and labels in Dart programming language to control a loop [1], [2]. Break and continue can only appear in a loop body, and they can be used both conditionally and unconditionally. When used without a label, a break statement terminates only the loop body in which it is contained. Similarly, a continue statement skips only the current iteration of the loop body in which it is contained. However, Dart supports the usage of labels together with break and continue [3]. When used with labels, break and continue statements become effective on the labeled loop. However, Dart only supports this kind of branching with loops. In other words, there is not any goto statement. Because of this, labels should only be used before loop bodies, although it is allowed to put labels anywhere in the code. Otherwise, specified labels would be useless since the break and continue statements cannot be used outside of a loop statement. Lastly, referring to the labels follow a similar principle to static scoping. For example, think of a nested loop with labels outerLoop and innerLoop. One cannot break or continue the innerLoop inside the outerLoop body because the innerLoop is not “defined” there. However, the outerLoop can be continued or broken inside the innerLoop body.

Regarding the subject, detailed code samples in Dart programming language are given in Figure 1 above. These code samples are important because they demonstrate all user-located loop control mechanisms in Dart. Explanations of the code segments can be found in the corresponding comment lines. Outputs of the code samples are also given in Figure 2.

## b) Javascript

```javascript
console.log("INTRODUCTION TO USER-LOCATED LOOP CONTROL MECHANISMS IN JAVASCRIPT");

while (true) {
    console.log("iterating once");
    break;
}

for (let i = 0; i < 10; i++) {
    continue; // continue to the next iteration without executing the rest of the loop body (print statement)
    console.log("$i");
}

// Labels can also be used to break out of nested loops (see the examples below)

console.log("\nSAMPLE 1: BREAK ------------------------------------------------------------");

// An example for break
for (let i = 1; i <= 6; i++) {
    for (let j = 1; j <= 6; j++) {
        /*
           When both i and j are 2, break out the inner loop
           i.e., '2 2' '2 3' '2 4' '2 5' and '2 6' will not be printed
        */
        if (i == 2 && j == 2) {
            console.log("i j are 2 2, break out the inner loop");
            break;
        }

        console.log(i + " " + j);
    }
    /*
       When i is 4, break out the outer loop, otherwise the outer loop will
       continue to print '4 1' '4 2' ... '6 6'
    */
    if (i == 4) {
        console.log("Fourth iteration of outer loop is completed, now break out the outer loop");
        break;
    }
}

console.log("\nSAMPLE 2: CONTINUE ---------------------------------------------------------");

// An example for continue
var days = {
    1: "monday",
    2: "tuesday",
    3: "wednesday",
    4: "thursday",
    5: "friday",
    6: "saturday",
    7: "sunday",
};
for (let day in days) {
    /*
       When the for loop encounters 6th and 7th day of the week, continue to the next iteration
       i.e., 'saturday' and 'sunday' will not be printed
    */
    if (day % 7 == 6 || day % 7 == 0) {
        continue;
    }

    console.log(days[day] + " is a week day");
}

console.log("\nSAMPLE 3: LABELED BREAK ----------------------------------------------------");

// An example for labeled break
var i = 4;
outerLoop: while (i > 0) {
    console.log("in outer loop i is: " + i);
    innerLoop: for (let j = 6; j > 0; j--) {
        /*
           When i is 2 break out the outer loop
           which means that the outer loop will be terminated
        */
        if (i == 2) {
            console.log("i is 2, break out the outer loop, otherwise the outer loop will continue to print until i reaches 1");
            break outerLoop;
        }
        console.log("  in inner loop j is : " + j);
        /*
           When j is 5 break out the inner loop
           which means that the inner loop will only print
           6 and 5 instead of 6, 5, ..., 1
        */
        if (j == 5) {
            console.log("j is 5, break out the inner loop, otherwise the inner loop will continue to print until j reaches 1");
            break innerLoop;
        }
    }
    i--;
}

console.log("\nSAMPLE 4: LABELED CONTINUE -------------------------------------------------");

// An example for labeled continue
var x = 0;
outerLoop: do {
    console.log("in outer loop x is : " + x);
    x++;
    innerLoop: for (let k = 0; k < 6; k++) {
        /*
           When k is 3 continue to the next iteration of the outer loop
           which means that the inner loop will only print
           0, 1, 2 instead of 0 to 5
        */
        if (k == 3) {
            console.log("k is 3, continue to the next iteration of the outer loop, the inner loop otherwise would print 0 to 5");
            continue outerLoop; // same as break innerLoop
        }
        console.log("  in inner loop k is : " + k);
    }
} while (x < 4);
```

Figure 3: Code samples from the Javascript programming language

```
friberk@Berk-MacBook-Pro  hw2 ❯ node "cakar_berk.js"



INTRODUCTION TO USER-LOCATED LOOP CONTROL MECHANISMS IN JAVASCRIPT
iterating once

SAMPLE 1: BREAK ------------------------------------------------------------
1 1
1 2
1 3
1 4
1 5
1 6
2 1
i j are 2 2, break out the inner loop
3 1
3 2
3 3
3 4
3 5
3 6
4 1
4 2
4 3
4 4
4 5
4 6
Fourth iteration of outer loop is completed, now break out the outer loop

SAMPLE 2: CONTINUE ---------------------------------------------------------
monday is a week day
tuesday is a week day
wednesday is a week day
thursday is a week day
friday is a week day

SAMPLE 3: LABELED BREAK ----------------------------------------------------
in outer loop i is: 4
  in inner loop j is : 6
  in inner loop j is : 5
j is 5, break out the inner loop, otherwise the inner loop will continue to print until j reaches 1
in outer loop i is: 3
  in inner loop j is : 6
  in inner loop j is : 5
j is 5, break out the inner loop, otherwise the inner loop will continue to print until j reaches 1
in outer loop i is: 2
i is 2, break out the outer loop, otherwise the outer loop will continue to print until i reaches 1

SAMPLE 4: LABELED CONTINUE -------------------------------------------------
in outer loop x is : 0
  in inner loop k is : 0
  in inner loop k is : 1
  in inner loop k is : 2
k is 3, continue to the next iteration of the outer loop, the inner loop otherwise would print 0 to 5
in outer loop x is : 1
  in inner loop k is : 0
  in inner loop k is : 1
  in inner loop k is : 2
k is 3, continue to the next iteration of the outer loop, the inner loop otherwise would print 0 to 5
in outer loop x is : 2
  in inner loop k is : 0
  in inner loop k is : 1
  in inner loop k is : 2
k is 3, continue to the next iteration of the outer loop, the inner loop otherwise would print 0 to 5
in outer loop x is : 3
  in inner loop k is : 0
  in inner loop k is : 1
  in inner loop k is : 2
k is 3, continue to the next iteration of the outer loop, the inner loop otherwise would print 0 to 5
```

Figure 4: Output of the code samples given in Figure 3

In the Javascript programming language, user-controlled loop mechanisms are similar to the ones in Dart, if not identical. Labels are available, as well as break and continue statements [4], [5]. As usual, the break statement jumps out of a loop (terminates it) [6], whereas the continue statement skips the current iteration of the loop [7]. On their own, break and continue statements become effective on the loop body in which they are used. Like in Dart, it is possible to use the continue and break conditionally or unconditionally (i.e., outside of an if statement). However, labels have to be used to break or continue nested loops or a specific loop [7]. Similar to Dart programming language, labels can be located in any line of the code piece. Still, break and continue statements can only be used inside a loop body; therefore, labels will be effective and useful when they are located just before a loop statement. Again, like Dart, a break or a continue statement can refer to a label within the label's enclosing body (i.e., in outerLoop, innerLoop cannot be continued or broken).

In Figure 3 above, Javascript code samples about this topic can be found. In the first sample, all possible outcomes of rolling two dice are simulated. But we are manipulating the procedure with break statements so that when the first die and second die become two simultaneously, we discard any outcomes where the first die is two and the second die is 3, 4, 5, or 6. Similarly, we also discard the outcomes where the first die is 5 or 6. In the second example, we are traversing through an array that contains seven days of a week. Only the weekdays (i.e., Monday to Friday) are printed using a continue statement. Lastly, in the third and fourth samples, the usage of labels for controlling loops is investigated. For all code samples, detailed explanations can be found in corresponding comment lines. These code samples are important because they demonstrate all relevant concepts for user-located loop control mechanisms in the Javascript programming language. Outputs of the code samples are also given in Figure 4.

## c) Lua

```lua
print("INTRODUCTION TO USER-LOCATED LOOP CONTROL MECHANISMS IN LUA")

while true do
    print("iterating once")
    break
end

for i = 1, 10 do
    -- continue is not directly supported in Lua
    goto continue -- continue to the next iteration without executing the rest of the loop body (print statement)
    print("iterating once")
    ::continue::
end

print("\nSAMPLE 1: BREAK ------------------------------------------------------------");

-- An example for break
-- In this for loop, a modified Floyd's triangle is printed
for i = 1, 10 do
    for j = 1, i do
        io.write("*")
        if j == 5 then
            -- When the number of stars printed (columns) is 5, break out of the inner loop
            io.write(" The inner loop has completed its fifth iteration (column), now break out the inner loop (stop adding new columns)")
            break
        end
    end
    if i == 8 then
        -- When the number of rows is 8, break out of the outer loop
        io.write("\nThe outer loop has completed its eight iteration (row), now break out the outer loop (stop adding new rows)")
        break
    end
    print();
end
-- In the end, it is expected to see a trapazoid with 8 rows and 5 columns

-- Lua does not have a continue statement, but you can use the goto statement to jump to the next iteration of a loop.

io.write("\n\nSAMPLE 2: LABELED BREAK (VIA GOTO) --------------------------------------\n");

-- An example for labeled break
local array = {
    { 1,  2,  3,  4,  5},
    { 6,  7,  8,  9, 10},
    {11, 12, 13, 14, 15},
    {16, 17, 18, 19, 20},
    {21, 22, 23, 24, 25}
}
for i = 1, #array do
    for j = 1, #array[i] do
        if i == 4 then
            -- When the fourth row of the 2D array is reached, break out of the outer loop
            goto outerLoopDone -- break out of the outer loop
        end
        if j == 4 then
            -- When the fourth column of the 2D array is reached, break out of the inner loop
            goto innerLoopDone -- break out of the inner loop
        end
        io.write(array[i][j], "\t")
    end
    ::innerLoopDone::
    io.write("\n")
end
::outerLoopDone::
-- In the end, it is expected to see the left-top 3x3 sub-matrix of the 5x5 2D array

io.write("\nSAMPLE 3: LABELED CONTINUE (VIA GOTO) --------------------------------------");

-- An example for labeled continue
for i = 1, 4 do
    io.write("\n")
    for j = 1, 3 do
        if i == j then
            -- When the row number equals the column number (i.e., a square number is encountered),
            -- continue the outer loop, so that no number is printed after the square number
            io.write(">", "\t")
            goto continue -- same as continue outerLoop, or break innerLoop
        end
        io.write(i * j, "\t")
    end
    ::continue::
end
print("\n> means i == j, which indicates a square number has been reached, so we continue to the next iteration of the outer loop")
```

Figure 5: Code samples from the Lua programming language

```
friberk@Berk-MacBook-Pro  hw2 ❯ lua "cakar_berk.lua"
INTRODUCTION TO USER-LOCATED LOOP CONTROL MECHANISMS IN LUA
iterating once

SAMPLE 1: BREAK ------------------------------------------------------------
*
**
***
****
***** The inner loop has completed its fifth iteration (column), now break out the inner loop (stop adding new columns)
***** The inner loop has completed its fifth iteration (column), now break out the inner loop (stop adding new columns)
***** The inner loop has completed its fifth iteration (column), now break out the inner loop (stop adding new columns)
***** The inner loop has completed its fifth iteration (column), now break out the inner loop (stop adding new columns)
The outer loop has completed its eight iteration (row), now break out the outer loop (stop adding new rows)

SAMPLE 2: LABELED BREAK (VIA GOTO) ---------------------------------------
1       2       3
6       7       8
11      12      13

SAMPLE 3: LABELED CONTINUE (VIA GOTO) --------------------------------------
>
2       >
3       6       >
4       8       12
> means i == j, which indicates a square number has been reached, so we continue to the next iteration of the outer loop
```

Figure 6: Output of the code samples given in Figure 5

Lua programming language differs from Dart and Javascript in terms of the user-located loop control mechanisms [8], [9]. First of all, user-located loop control mechanisms in Lua consist of break and goto statements [9]. As can be seen, there is no continue statement in Lua [10]. However, the same functionality can be achieved with the usage of goto's [11]. As always, break statements only terminate the loop body in which it is located (i.e., nested loops cannot be broken with a single break statement). Similar to Dart and Javascript, break statements can only be found in loops, and the exit can occur conditionally or unconditionally (i.e., a break statement can be placed outside of an if statement). It is previously mentioned that the continue functionality can be provided with the usage of labels. Similarly, in order to break nested loops or a specific loop, a break statement can be combined with a goto statement. A goto statement jumps to the given label during the code execution. This feature is not available in Dart and Javascript and is much more powerful than labeled break and continue statements. Because goto statements can be used to jump anywhere in the code (i.e., not restricted with loops). Moreover, labels can be located anywhere in the code, and regardless of the position, a label would still be effective and useful. However, similar to Javascript and Dart's labeled break and continue, the given label has to be visible to the scope where the goto statement is located.

In Figure 5 above, Lua code samples about this topic can be found. In the first example, we use plain break statements to convert a Floyd's triangle into a trapezoid. According to that, when the inner loop, which prints the columns, completes its fifth iteration, we break out the inner loop so that the maximum number of columns will be five. Similarly, when the outer loop completes its eighth iteration, we break out the outer loop so that no more rows will be printed. Furthermore, in the second sample, we combine goto with labels to achieve a labeled break functionality. As a result, we obtain the 3x3 top-left submatrix of a 5x5 matrix. Lastly, in the third sample, we mimic the continue functionality with a goto statement. Note that the labels are declared using the "::labelName::" syntax. For all code samples, detailed explanations can be found in corresponding comment lines. These code samples are important because they highlight all mechanisms related to user-located loop control in the Lua programming language. Outputs of the code samples are also given in Figure 6.

## d) PHP

```php
<?php

echo "INTRODUCTION TO USER-LOCATED LOOP CONTROL MECHANISMS IN PHP", PHP_EOL;

while (true) {
    echo "Iterating once", PHP_EOL;
    break;
}

for ($i = 0; $i < 10; $i += 1) {
    continue; // continue to the next iteration without executing the rest of the loop body (echo statement)
    echo $i;
}


echo PHP_EOL, "SAMPLE 1: BREAK ----------------------------------------------------------", PHP_EOL;

// An example for break
for ($i = 0; $i < 8; $i++) {
    /*
        When $i is 3, the loop will break and the rest of the loop will not be executed.
        Otherwise, the loop will continue to execute until $i becomes 8.
    */
    if ($i == 3) {
        echo "in outer loop \$i is 3, now break out of the outer loop, otherwise \$i would be 4 in the next iteration";
        break;
    }

    echo "in outer loop \$i is $i", PHP_EOL;
    $j = 0;
    while ($j < 7) {
        /*
            If $j is 3, break out the inner loop and continue to the next iteration of the outer loop.
            So that, $j values can only be 0, 1, 2, 3 instead of 0, 1, 2, 3, 4, 5, 6.
        */
        if ($j == 3) {
            echo "  in inner loop \$j is 3, now break out of the inner loop, otherwise \$j would go [0, 6]", PHP_EOL;
            break;
        }
        echo "  in inner loop \$j is $j", PHP_EOL;
        $j++;
    }

}
echo PHP_EOL, PHP_EOL;

echo "SAMPLE 2: CONTINUE ----------------------------------------------------------", PHP_EOL;

// An example for continue
// Find the right angled triangle combinations with maximum side length of 10
for ($i = 1; $i <= 10; $i++) {
    for ($j = 1; $j <= 10; $j++) {
        /*
            If j is 3 or 4, then continue to the next iteration of the inner loop.
            This way we would not get the 3-4-5 or 4-3-5 triangle.
        */
        if ($j == 3 || $j == 4) {
            continue;
        }

        for ($z = 1; $z <= 10; $z++) {
            if ($z == 3 || ($z == 4)) {
                /*
                    Again, we basically skip the iteration of the most inner
                    loop when $z is 3 or 4, such that, we would never get the
                    3-4-5 or 4-3-5 triangle
                */
                continue;
            }

            if (pow($z, 2) + pow($j, 2) == pow($i, 2)) {
                echo "Found a Pythagorean triple: $j, $z, $i", PHP_EOL;
                break;
            }
        }
    }
}

echo PHP_EOL, "SAMPLE 3: LEVELED BREAK ----------------------------------------------------", PHP_EOL;

// An example for leveled break
$i = 0;
while ($i < 10) {
    echo "in outer loop \$i is $i", PHP_EOL;
    $j = 0;
    do {
        if ($j == 3) {
            /*
                When $j is 3 in the inner loop, break out 2 levels of the loop.
                So that, outer loop will be broken out.
            */
            echo "  in inner loop \$j is $j, now break out 2 enclosing loops", PHP_EOL;
            break 2;
        }
        echo "  in inner loop \$j is $j", PHP_EOL;
        $j++;
    } while ($j < 5);
    $i++;
}

echo PHP_EOL, "SAMPLE 4: LEVELED CONTINUE ----------------------------------------------------", PHP_EOL;

// An example for leveled continue
for ($i = 0; $i < 3; $i++) {
    echo "in outer loop \$i is $i", PHP_EOL;
    for ($j = 0; $j < 4; $j++) {
        /*
            Similar to the previous sample, when $j is 3 in the inner loop,
            continue the loop which is 2 levels above. So that, the outer
            loop will continue to the next iteration.
        */
        if ($j == 2) {
            echo "  in inner loop \$j is $j, now continue to the next iteration of the outer loop", PHP_EOL;
            continue 2;
        }
        else {
            echo "  in inner loop \$j is $j", PHP_EOL;
        }
    }
    echo "This line will not be executed", PHP_EOL;
}

echo PHP_EOL, "SAMPLE 5: LABELED BREAK (VIA GOTO) ----------------------------------------", PHP_EOL;

// An example for labeled break (via goto)
for ($i = 0; $i < 5; $i++) {
    echo "in outer loop \$i is $i", PHP_EOL;
    $j = 10;
    while ($j > 0) {
        /*
            When $i is 3, break out of the outer loop.
            This is possible by using goto and a label.
            (i.e., we can break out of the outerLoop from the innerLoop)
        */
        if ($i == 3) {
            echo "since \$i is 3, now break out of the outer loop", PHP_EOL;
            goto outerLoopDone;
        }

        /*
           When $j is 8, break out of the inner loop, such that
           $j values can only be 10, 9, 8 instead of 10, 9, ..., 1
        */
        if ($j == 8) {
            echo "  in inner loop \$j is $j, now break out of the inner loop", PHP_EOL;
            goto innerLoopDone;
        }

        else {
            echo "  in inner loop \$j is $j", PHP_EOL;
        }
        $j--;
    }
    innerLoopDone:
}
outerLoopDone:

echo PHP_EOL, "SAMPLE 6: LABELED CONTINUE (VIA GOTO) ------------------------------------", PHP_EOL;

// An example for labeled continue (via goto)
$months = array('January', 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'October', 'November', 'December');
for ($i = 0; $i < count($months); $i++) {
    // Another loop for processing summer months
    while ($i % 12 >= 5 && $i % 12 <= 7) {
        if ($months[$i] == 'July') {
            // When the month is July, continue to the next iteration of the outer loop
            // So that, we would not get the month of July in the output
            goto innerLoopContinue;
        }
        else {
            echo "  in inner loop month is $months[$i]", PHP_EOL;
        }
        innerLoopContinue:
        $i++; // Increment $i here as well, otherwise the loop will be infinite
    }
    echo "in outer loop month is $months[$i]", PHP_EOL;
}

?>
```

Figure 7: Code samples from the PHP programming language

```
friberk@Berk-MacBook-Pro  hw2 ❯ php "cakar_berk.php"



INTRODUCTION TO USER-LOCATED LOOP CONTROL MECHANISMS IN PHP
iterating once

SAMPLE 1: BREAK ------------------------------------------------------------
in outer loop $i is 0
  in inner loop $j is 0
  in inner loop $j is 1
  in inner loop $j is 2
  in inner loop $j is 3, now break out of the inner loop, otherwise $j would go [0, 6]
in outer loop $i is 1
  in inner loop $j is 0
  in inner loop $j is 1
  in inner loop $j is 2
  in inner loop $j is 3, now break out of the inner loop, otherwise $j would go [0, 6]
in outer loop $i is 2
  in inner loop $j is 0
  in inner loop $j is 1
  in inner loop $j is 2
  in inner loop $j is 3, now break out of the inner loop, otherwise $j would go [0, 6]
in outer loop $i is 3, now break out of the outer loop, otherwise $i would be 4 in the next iteration

SAMPLE 2: CONTINUE ---------------------------------------------------------
Found a Pythagorean triple: 6, 8, 10
Found a Pythagorean triple: 8, 6, 10

SAMPLE 3: LEVELED BREAK ----------------------------------------------------
in outer loop $i is 0
  in inner loop $j is 0
  in inner loop $j is 1
  in inner loop $j is 2
  in inner loop $j is 3, now break out 2 enclosing loops

SAMPLE 4: LEVELED CONTINUE -------------------------------------------------
in outer loop $i is 0
  in inner loop $j is 0
  in inner loop $j is 1
  in inner loop $j is 2, now continue to the next iteration of the outer loop
in outer loop $i is 1
  in inner loop $j is 0
  in inner loop $j is 1
  in inner loop $j is 2, now continue to the next iteration of the outer loop
in outer loop $i is 2
  in inner loop $j is 0
  in inner loop $j is 1
  in inner loop $j is 2, now continue to the next iteration of the outer loop

SAMPLE 5: LABELED BREAK (VIA GOTO) -----------------------------------------
in outer loop $i is 0
  in inner loop $j is 10
  in inner loop $j is 9
  in inner loop $j is 8, now break out of the inner loop
in outer loop $i is 1
  in inner loop $j is 10
  in inner loop $j is 9
  in inner loop $j is 8, now break out of the inner loop
in outer loop $i is 2
  in inner loop $j is 10
  in inner loop $j is 9
  in inner loop $j is 8, now break out of the inner loop
in outer loop $i is 3
since $i is 3, now break out of the outer loop

SAMPLE 6: LABELED CONTINUE (VIA GOTO) --------------------------------------
in outer loop month is January
in outer loop month is February
in outer loop month is March
in outer loop month is April
in outer loop month is May
  in inner loop month is June
  in inner loop month is August
in outer loop month is September
in outer loop month is October
in outer loop month is November
in outer loop month is December
```

Figure 8: Output of the code samples given in Figure 7

The PHP programming language may be the most comprehensive of the languages studied so far in terms of user-located loop control mechanisms. First of all, break and continue statements are available in this language [12]. In addition, parameterized versions of break and continue, as well as with goto, are also provided [13]. As in Dart, Javascript, and Lua, when used alone, the break statement terminates the execution of the current loop structure [14]. Similarly, the continue statement in PHP, like in Dart and Javascript, skips the rest of the current loop iteration and continues execution at the next iteration of the loop body [15]. However, when used with an integer (i.e., break 2; or continue 3;), break and continue statements become effective on the loop body, which is located "n" number of levels above [14], [15]. For example, in a nested loop structure consisting of two loops, namely innerLoop and outerLoop, using "break 2;" inside the innerLoop would break the outerLoop since the outerLoop is located two levels above the innerLoop (in this PHP syntax, innerLoop is considered as the level 1). Overall, this notation is quite helpful as it gives users the power to control nested loops without dealing with labels. Lastly, break and continue statements can only be found in loops, and the exit can occur conditionally or unconditionally (i.e., a break or a continue statement can be placed outside of an if statement). Unlike Dart and Javascript programming languages, PHP does not support labeled breaks directly. However, we can use goto statements to achieve the same functionality [16]. The way goto statements work is similar to Lua's goto statements. According to that, we can branch almost anywhere in the code using goto statements. Still, some restrictions apply. For example, we can "goto" outside of a loop when in a loop, but we cannot "goto" inside a loop when we are outside of that particular loop [16]. Moreover, the label to be used has to be visible to the goto statement (i.e., they need to be located in the same static scope).

Regarding the subject, detailed code samples in the PHP programming language are given in Figure 7. For example, in the second code sample, we use the continue statement to obtain right-angled triangles except for 3-4-5 or 4-3-5 triangles. Moreover, in the fifth code sample, goto statements are used with labels to get a labeled break functionality. These code samples are important because they demonstrate PHP's user-located loop control mechanisms in all aspects. Explanations of the code segments can be found in the corresponding comment lines. Outputs of the code samples are also given in Figure 8.

### e) Python

```python
print("INTRODUCTION TO USER-LOCATED LOOP CONTROL MECHANISMS IN PYTHON")

while (True):
    print("iterating once\n")
    break

for i in range(0, 30):
    continue # continue to the next iteration without executing the rest of the loop body (print statement)
    print(i)

print("SAMPLE 1: BREAK ------------------------------------------------------------")

# An example for break
odd = [1, 3, 5, 7, 9]
even = [2, 4, 6, 8]
result = []
# This nested loop form odd-even pairs and stores them in the result list
for i in odd:
    # When i is 7 in the outer loop, break out of the outer loop,
    # so that no pair will include 7 and 9
    if (i == 7):
        print("in outer loop i is 7, now break out of the outer loop, such that no pair in result will contain 7 and 9")
        break
    for j in even:
        # When j is 6 in the inner loop, break out of the inner loop,
        # so that no pair will include 6 and 8
        if (j == 6):
            print("  in inner loop j is 6, now break out of the inner loop, such that no pair in result will contain 6 and 8")
            break
        result.append('{}-{}'.format(i, j))
print("Resulting list: " + str(result))

print("\nSAMPLE 2: CONTINUE ---------------------------------------------------------")

# An example for continue
text = "Hello!World"
for i in range(0, 3):
    # Skip the second iteration so that HelloWorld will be printed twice
    if i == 1: continue
    print("i is {}: ".format(i), end='')
    for char in text:
        # Skip the exclamation mark
        if (char == '!'):
            continue
        print(char, end='')
    print("")
```

Figure 9: Code samples from the Python programming language

```
friberk@Berk-MacBook-Pro  hw2 ❯ python3 "cakar_berk.py"
INTRODUCTION TO USER-LOCATED LOOP CONTROL MECHANISMS IN PYTHON
iterating once

SAMPLE 1: BREAK ------------------------------------------------------------
  in inner loop j is 6, now break out of the inner loop, such that no pair in result will contain 6 and 8
  in inner loop j is 6, now break out of the inner loop, such that no pair in result will contain 6 and 8
  in inner loop j is 6, now break out of the inner loop, such that no pair in result will contain 6 and 8
in outer loop i is 7, now break out of the outer loop, such that no pair in result will contain 7 and 9
Resulting list: ['1-2', '1-4', '3-2', '3-4', '5-2', '5-4']

SAMPLE 2: CONTINUE ---------------------------------------------------------
i is 0: HelloWorld
i is 2: HelloWorld
```

Figure 10: Output of the code samples given in the Figure 9

Regarding the user-located loop control mechanisms, Python might be the least versatile programming language among Dart, Javascript, Lua, and PHP. The language only provides the fundamental break and continue functionality [17], [18]. There are no extended features such as goto, labeled breaks or labeled continues. A plain break statement has the same functionality as a break statement in other languages. According to that, a break statement is used to end (terminate) a loop statement in which it is present [18]. Similarly, a continue statement forces the loop to execute the next iteration of the loop [18]. However, as can be seen, there is no built-in way to take control of a specific loop or a nested loop [19]. The missing functionalities can be partially gained using workarounds such as using for…else or embedding the loop in a function and then using return statements, but it can be suggested that all of these solutions are inconvenient to use [19]. Finally, break and continue statements can only be found in loops, and the exit can occur conditionally or unconditionally (i.e., a break or a continue statement can be placed outside of an if statement).

In Figure 10 above, Python code samples about this topic can be found. In the first sample, we combine odd and even numbers to obtain pairs. However, we use break statements to filter out the results such that we will not get any pair containing 6, 7, 8, and 9. Similarly, in the second sample, we benefit from the continue statement to discard an unwanted character in a string ('!' in our case). Also, we again use a continue statement to decrease the number of prints to 2 from 3. These code samples are important because they show how the Python programming language is limited in terms of loop control mechanisms. Explanations of the code segments can be found in the corresponding comment lines. Outputs of the code samples are also given in Figure 10.

### f) Ruby

```ruby
puts "INTRODUCTION TO USER-LOCATED LOOP CONTROL MECHANISMS IN RUBY"

while true
    puts "iterating once", ""
    break
end

for i in 1..10
    # next is the continue equivalent in Ruby
    next # continue to the next iteration without executing the rest of the loop body (puts statement)
    puts i
end

puts "SAMPLE 1: BREAK ------------------------------------------------------------"

# An example for break
for i in 0..10
    # When i is 3 in outer loop, break the outer loop
    # so that i become 0, 1, 2, and 3 instead of 0, 1, ..., 10
    if i == 3
        puts "in outer loop i is 3, now break out of the outer loop, such that i will not go beyond 3"
        break
    end

    puts "in outer loop, i is #{i}"

    j = 12;
    loop do
        # When j is 9 in inner loop, break the inner loop
        # so that j become 12, 11, 10, 9 instead of 12, 11, ..., -infinity
        if j == 9
            puts "  in inner loop, j is 9, now break out of the inner loop, such that j will not go minus infinity"
            break
        end
        puts "  in inner loop, j is #{j}"
        j -= 1
    end
end

puts "", "SAMPLE 2: NEXT ----------------------------------------------------------"

# An example for next
# Print a simple multiplication table
for i in 1..4
    # When the order of the row is divisible by 2, continue to the next iteration
    if i.even?
        puts "in outer loop i is even, now continue to the next iteration of the outer loop"
        next
    end
    for j in 1..4
        # When the order of the column is divisible by 3, continue to the next iteration
        # For example instead of 3 6 9 12 this will print 3 6 12
        if j == 3
            next
        end
        # Otherwise print the multiplication result
        print i*j, " "
    end
    puts
end
```

Figure 11: Code samples from the Ruby programming language

```
friberk@Berk-MacBook-Pro  hw2 ❯ ruby "cakar_berk.rb"
INTRODUCTION TO USER-LOCATED LOOP CONTROL MECHANISMS IN RUBY
iterating once

SAMPLE 1: BREAK ------------------------------------------------------------
in outer loop, i is 0
  in inner loop, j is 12
  in inner loop, j is 11
  in inner loop, j is 10
  in inner loop, j is 9, now break out of the inner loop, such that j will not go minus infinity
in outer loop, i is 1
  in inner loop, j is 12
  in inner loop, j is 11
  in inner loop, j is 10
  in inner loop, j is 9, now break out of the inner loop, such that j will not go minus infinity
in outer loop, i is 2
  in inner loop, j is 12
  in inner loop, j is 11
  in inner loop, j is 10
  in inner loop, j is 9, now break out of the inner loop, such that j will not go minus infinity
in outer loop i is 3, now break out of the outer loop, such that i will not go beyond 3

SAMPLE 2: NEXT -------------------------------------------------------------
1 2 4
in outer loop i is even, now continue to the next iteration of the outer loop
3 6 12
in outer loop i is even, now continue to the next iteration of the outer loop
```

Figure 12: Output of the code samples given in the Figure 11

Like Python programming language, Ruby is very limited in terms of the user-located loop control mechanisms. Ruby programming only provides simple statements such as break and next [20]. Richer statements such as labeled break, labeled continue, and goto do not exist in the Ruby programming language. As usual, a break statement terminates the execution of the loop in which it is located [21]. On the other hand, next is the equivalent of the continue statement in other languages [21]. In that sense, the next statement skips the rest of the current loop iteration and continues execution at the next iteration of the loop body [21]. However, like Python, there is no built-in way to take control of a specific loop or a nested loop [22]. Still, similar functionality can be gained by using throw/catch statements or wrapping an internal method around the loops using return statements [22]. Yet, those are just unofficial workarounds to mimic the functionality of labeled break or labeled continue statements in order to break or continue a specific loop or a nested loop structure [22]. Lastly, break and continue statements can only be found in loops, and the exit can occur conditionally or unconditionally (i.e., a break or a continue statement can be placed outside of an if statement).

In Figure 12 above, code samples in the Ruby programming language are given regarding the subject. In the first sample, we use break statements to limit the outer loop's boundary and block an infinity loop in the inner loop. In the second example, we are manipulating the generation process of a multiplication table by using the next statements. These code samples are important because they indicate how the Ruby programming language is restricted regarding loop control mechanisms. Explanations of the code segments can be found in the corresponding comment lines. Outputs of the code samples are also given in Figure 12.

## g) Rust

```rust
fn main() {
    println!("INTRODUCTION TO USER-LOCATED LOOP CONTROL MECHANISMS IN RUST");

    while true {
        println!("iterating once\n");
        break;
    }

    for i in 0..5 {
        continue; // continue to the next iteration without executing the rest of the loop body (println statement)
        println!("{}", i);
    }

    println!("SAMPLE 1: BREAK ------------------------------------------------------------");

    // An example for break
    let mut i = 0;
    loop {
        println!("in the outer loop i is {}", i);

        for j in 0..5 {
            println!("  in the inner loop j is {}", j);

            // When j is 2 in the inner loop, break out of the inner loop
            // and continue the outer loop
            if j == 2 {
                break;
            }
        }


        // When i is 2 in the outer loop, break out of the outer loop
        // Such that i will not go beyond 2
        if i == 2 {
            break;
        }

        i += 1;
    }

    println!("\nSAMPLE 2: CONTINUE ---------------------------------------------------------");

    // An example for continue
    // This loop finds some relatively prime numbers
    for i in 1..6 {
        // If i is 1 in the outer loop, continue to the next iteration of the outer loop
        // Since 1 can divide any number
        if i == 1 {
            println!("1 can divide any number, now continue to the next iteration of the outer loop");
            continue;
        }

        println!("in the outer loop i is {}", i);


        for j in 1..6 {
            // If i divides j or j divides i, continue to the next iteration of the inner loop
            if (i % j == 0) || (j % i == 0) {
                continue;
            }

            println!("  in the inner loop j is {}", j);
        }
    }

    println!("\nSAMPLE 3: LABELED BREAK ----------------------------------------------------");

    // An example for labeled break
    'outerLoop: for x in 0..8 {
        'innerLoop: for y in 0..15 {
            // When y is 4, break out of the inner loop
            if y == 4 {
                break 'innerLoop;
            }

            // When x is 3, break out of the outer loop
            if x == 3 {
                break 'outerLoop;
            }

            println!("{}, {}", x, y);
        }
    }

    println!("\nSAMPLE 4: LABELED CONTINUE -------------------------------------------------");

    // An example for labeled continue
    // In this loop, we print a square of asterisks
    // Although it is specified as 5x5, it will be 4x4 because of the labeled continue
    'outer: for i in 0..5 {
        'inner: for j in 0..5 {
            // When i is 4, continue to the next iteration of the outer loop
            if i == 4 {
                continue 'outer;
            }

            // When j is 4, continue to the next iteration of the inner loop
            if j == 4 {
                continue 'inner;
            }

            print!("* ");
        }
            println!();
    }
}
```

Figure 13: Code samples from the Rust programming language

```
friberk@Berk-MacBook-Pro  hw2 > rustc "cakar_berk.rs" && ./cakar_berk
INTRODUCTION TO USER-LOCATED LOOP CONTROL MECHANISMS IN RUST
iterating once

SAMPLE 1: BREAK ------------------------------------------------------------
in the outer loop i is 0
  in the inner loop j is 0
  in the inner loop j is 1
  in the inner loop j is 2
in the outer loop i is 1
  in the inner loop j is 0
  in the inner loop j is 1
  in the inner loop j is 2
in the outer loop i is 2
  in the inner loop j is 0
  in the inner loop j is 1
  in the inner loop j is 2

SAMPLE 2: CONTINUE ---------------------------------------------------------
1 can divide any number, now continue to the next iteration of the outer loop
in the outer loop i is 2
  in the inner loop j is 3
  in the inner loop j is 5
in the outer loop i is 3
  in the inner loop j is 2
  in the inner loop j is 4
  in the inner loop j is 5
in the outer loop i is 4
  in the inner loop j is 3
  in the inner loop j is 5
in the outer loop i is 5
  in the inner loop j is 2
  in the inner loop j is 3
  in the inner loop j is 4

SAMPLE 3: LABELED BREAK ----------------------------------------------------
0, 0
0, 1
0, 2
0, 3
1, 0
1, 1
1, 2
1, 3
2, 0
2, 1
2, 2
2, 3

SAMPLE 4: LABELED CONTINUE -------------------------------------------------
* * * *
* * * *
* * * *
* * * *
```

Figure 14: Output of the code samples given in the Figure 13

In terms of the user-located loop control mechanisms, the Rust programming language is very similar to Dart and Javascript. In Rust, break, continue, labeled break, and labeled continue statements are available [23], [24]. When used without a label, a break statement terminates only the loop body in which it is contained [25]. Similarly, a continue statement skips only the current iteration of the loop body in which it is contained [25]. However, when used with labels, continue and break statements can be effective on a specific loop [24], [26]. Like Dart, Rust only supports branching via labels in loops such that there is no goto-like statement. Yet, different from the languages that support labels, Rust allows placing labels just before loop statements; otherwise, the compiler will raise an error. Moreover, referring to the labels follow a similar principle to static scoping. For example, think of a nested loop with labels outerLoop and innerLoop. One cannot break or continue the innerLoop inside the outerLoop body because the innerLoop is not “declared” there. However, the outerLoop can be continued or broken inside the innerLoop body. Finally, break and continue statements can only be found in loops, and the exit can occur conditionally or unconditionally (i.e., a break or a continue statement can be placed outside of an if statement).

About the topic, detailed code samples in Rust programming language are given in Figure 13. These code samples are important because they demonstrate all user-located loop control mechanisms in Rust. Explanations of the code segments can be found in the corresponding comment lines. Outputs of the code samples are also given in Figure 14.

**Evaluation**

After comprehending the user-located loop control mechanisms in seven different programming languages, in short, I think that Dart is the best in terms of the readability and writability. However, considering the overall language design, there is a common point across these seven languages. Unfortunately, every language we covered allows unconditional break and continue statements. I think that this is pointless because breaking or continuing a loop unconditionally means that loop will be terminated or skipped whenever the break or continue statement is encountered. In that sense, typically, loops will disappear or be skipped after only one iteration when the unconditional exit is allowed. This aspect reduces the readability of the written code because the reader will most likely spend time understanding the point of performing an "emergency loop manipulation" since the condition is not specified properly, even if that is done intentionally. Moreover, it can also be argued that the same problem reduces the reliability since the compiler will not warn the developer about the unwanted consequences of an unconditional exit. However, as it is mentioned before, that is a common problem among all seven languages; thus, it does not constitute an advantage or disadvantage for any language in this evaluation (i.e., they are all equal regarding this issue). Other than that, some languages, namely Python and Ruby, do not include a proper way to handle exiting multiple loop bodies or enclosing loops. I think offering labeled break and labeled continue statements is important in that sense because developers will try to handle these issues manually by writing more code; therefore, both readability and writability will decrease. Because since that is the case in both Python and Ruby, for me, it is not possible to pick these languages as the best programming languages in terms of the user-located loop control mechanisms. Furthermore, some languages require the usage of goto statements to perform break or continue operations on certain loops. Although goto statements are very powerful statements that can easily change the execution flow of the programs, by today's standards, they are considered as a reason which leads to "spaghetti code". Obviously, regarding the language evaluation criteria, it might not be possible to classify these languages which contain goto, even if the purpose is limited to controlling loops, as readable and writable since developers still have a chance to create unmaintainable and understandable code pieces using goto statements. Hence, in my opinion, it is not possible to pick Lua and PHP as the best programming languages regarding the subject either. As a result, the remaining three candidates are Dart, Javascript, and Rust. From my observation and research for this homework, in terms of the user-located loop control mechanisms, these three supply the most simple but efficient structures such as break, continue, labeled break, and labeled continue. However, I want to separate Rust because although the Rust programming language has the same functionality as Dart and Javascript regarding the topic, the language has a complex syntax which reduces the overall readability and writability. At this point, I think that both Dart and Javascript have the same level of writability and readability in terms of the user-located loop control mechanisms, and, actually in general that is the case since they both a member of the C-family programming languages. However, since Dart is more reliable (includes type-checking) and has more overall features, it is the best pick regarding the user-located loop control mechanisms. To sum up, my ranking for these seven languages in terms of the user-located loop control mechanisms is the following:

1. Dart
2. Javascript
3. Rust
4. PHP (leveled breaks can be quite useful)
5. Lua (no continue statement)
6-7. Python or Ruby (no built-in mechanism for handling exiting multiple loop bodies or enclosing loops)

**Learning Strategy**

Throughout the course of completing this assignment, I generally referred to the official documentations of the seven languages provided. I searched through these documentations' loops and loop control statements sections to see which alternatives are available for each language. I carried out detailed online searches whenever I could not find any comprehensive information in the official documentations or I encountered unclear explanations, again, in the official documentations. After that, in order to find trustworthy sources, I filtered the information from these searches. In other words, in addition to official documentations, I also consulted reliable tech blogs and Q&A websites like StackOverflow. Since I already had all the necessary compilers and interpreters installed on my computer, I did not use any online compilers or interpreters for the experiments I conducted regarding the user-controlled loop mechanisms. But whenever I could, I also ran my programs on the Dijkstra server to make sure there were not any platform-related problems.

Also, whenever I felt that the information on the internet is ambiguous or inefficient, I performed the experiments on user-controlled loop mechanisms by using trial-error methodology. Therefore, I was able to explore the concepts better. In fact, mostly, that was the case in Lua and Rust.

**References**


*Dart*

[1] https://dart.dev/guides/language/language-tour#control-flow-statements
[2] https://www.geeksforgeeks.org/dart-loops/
[3] https://www.geeksforgeeks.org/labels-in-dart/

*Javascript*

[4] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration
[5] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/
[6] https://www.w3schools.com/js/js_break.asp
[7] https://www.freecodecamp.org/news/javascript-loops-label-statement-continue-statement-and-break-statement-explained/

*Lua*

[8] https://www.lua.org/pil/4.3.html
[9] https://www.lua.org/pil/4.4.html
[10] https://stackoverflow.com/questions/3524970/why-does-lua-have-no-continue-statement
[11] http://lua-users.org/wiki/GotoStatement

*PHP*

[12] https://www.geeksforgeeks.org/php-loops/
[13] https://www.w3schools.com/php/php_looping_break.asp
[14] https://www.php.net/manual/en/control-structures.break.php
[15] https://www.php.net/manual/en/control-structures.continue.php

[16] https://www.php.net/manual/en/control-structures.goto.php

*Python*

[17] https://docs.python.org/3/tutorial/controlflow.html#
[18] https://www.programiz.com/python-programming/break-continue
[19] https://stackoverflow.com/questions/653509/breaking-out-of-nested-loops

*Ruby*

[20] https://docs.ruby-lang.org/en/2.4.0/syntax/control_expressions_rdoc.html
[21] https://www.geeksforgeeks.org/ruby-break-and-next-statement/
[22] https://stackoverflow.com/questions/1352120/how-to-break-outer-cycle-in-ruby

*Rust*

[23] https://doc.rust-lang.org/reference/expressions/loop-expr.html
[24] https://doc.rust-lang.org/rust-by-example/flow_control/loop/nested.html
[25] https://www.programiz.com/rust/break-continue
[26] https://blog.boot.dev/rust/loops-in-rust-breaking-from-nested-loops/