CS315 PROGRAMMING LANGUAGES HW-2

Mahmut Furkan Gön
21902582
Section 3

1. Sample Code, Output, and Explanation

I used the same purpose code as much as possible for this assignment. Some languages don't support labeling, and some don't support continue statements. In such cases, if possible, I simulate those statements; if not, I state them as comments in code and the explanation parts for these languages.

The general logic of the code is the following:

- A labeled outer loop
- A loop for the unlabeled exit, break, statement
- A loop for the labeled exit, break label, statement
- A loop for the unlabeled continue statement
- A loop for the labeled continue statement
- A loop for the labeled exit for more than one loop exiting from the labeled outer loop

Throughout the report, I used loop counters between 0 to 5 (exclusive.) For some languages, loop variables are inclusive, so I made those variables 0 to 4.

1.1. Dart

1.1.1. Code

```
void main() {

 outerLoop:// label

   for (var i = 0; i < 5; i++) {

     print("------------------------------");

     // shows unlabeled exit
     // break statement exits only one loop
     for (var j = 0; j < 5; j++){
       if(j==1) break;
       print("innerLoopBreakUnlabeled: $j");
     }// end of for

     print("------------------------------");

     innerLoopBreakLabeled:// label

     // shows labeled exit
     // break statement exits only one loop
     for (var j = 0; j < 5; j++){
       if(j==1) break innerLoopBreakLabeled;
       print("innerLoopBreakLabeled: $j");
     }// end of for

     print("------------------------------");

     // shows unlabeled continue
```

```
      // continue statement affects only one loop
      for (var j = 0; j < 5; j++){
        if(j==2) continue;
        print("innerLoopContinueUnlabeled: $j");
      }// end of for

      print("------------------------------");

      innerLoopContinueLabeled:// label

      // shows labeled continue
      // continue statement affects only one loop
      for (var j = 0; j < 5; j++){
        if(j==2) continue innerLoopContinueLabeled;
        print("innerLoopContinueLabeled: $j");
      }// end of for

      print("------------------------------");

      // shows labeled exit for more than one loop
      // break statement exits from outerLoop and ends the program
      for (var j = 0; j < 5; j++){
        if(j==3) break outerLoop;
        print("outerLoopBreakLabeled: $j");
      }// end of for

    }// end of for

}
```

1.1.2. Output

```
------------------------------
innerLoopBreakUnlabeled: 0
------------------------------
innerLoopBreakLabeled: 0
------------------------------
innerLoopContinueUnlabeled: 0
innerLoopContinueUnlabeled: 1
innerLoopContinueUnlabeled: 3
innerLoopContinueUnlabeled: 4
------------------------------
innerLoopContinueLabeled: 0
innerLoopContinueLabeled: 1
innerLoopContinueLabeled: 3
innerLoopContinueLabeled: 4
------------------------------
outerLoopBreakLabeled: 0
outerLoopBreakLabeled: 1
outerLoopBreakLabeled: 2
```

1.1.3. Explanation

Dart language supports labeling. So, the first statement puts a label for the outer loop, **outerLoop**.

After that, the first inner loop checks if the counter is 1. If this check is successful, it breaks the loop using the unlabeled break statement **break**. If not, it prints the counter with the section name. This part prints the section name and 0.

After the first loop, there is another label, **innerLoopBreakLabeled**. Following loop checks, again, if the counter is 1. If this check is successful, it breaks the loop using the labeled break statement **break innerLoopBreakLabeled**. If not, it prints the counter with the section name. This code section aims to show that the labeled break statements can work with one loop without any problem. This part of the program prints the section name and 0, the same as the previous section, with only the section name differences.

After this loop, the continue statements come with the same logic. In the following two loops, there is an if checking whether the loop counter is 2. If it is 2, it continues without a label in the first, using **continue**, with the label, using **continue innerLoopContinueLabeled**, in the latter loop. Otherwise, it prints the counter with the section name in both loops. These two sections print section names and 0-1-3-4 in different lines on the console.

The last loop checks if the counter is 3. If it is 3, it breaks from the outer loop using the label **outerLoop**. Otherwise, it prints the counter with the section name. This code section aims to show that the labeled break statements can work with more than one loop without any problem. This part prints the section name and 0-1-2 in different lines on the console.

1.2. Javascript

1.2.1. Code

```
outerLoop:// label

  for (var i = 0; i < 5; i++) {

    console.log("------------------------------");

    // shows unlabeled exit
    // break statement exits only one loop
    for (var j = 0; j < 5; j++){
      if(j==1) break;
      console.log("innerLoopBreakUnlabeled: ", j);
    }// end of for

    console.log("------------------------------");

    innerLoopBreakLabeled:// label

    // shows labeled exit
    // break statement exits only one loop
    for (var j = 0; j < 5; j++){
      if(j==1) break innerLoopBreakLabeled;
      console.log("innerLoopBreakLabeled: ", j);
```

```
    }// end of for

    console.log("------------------------------");

    // shows unlabeled continue
    // continue statement affects only one loop
    for (var j = 0; j < 5; j++){
      if(j==2) continue;
      console.log("innerLoopContinueUnlabeled: ", j);
    }// end of for

    console.log("------------------------------");

    innerLoopContinueLabeled:// label

    // shows labeled continue
    // continue statement affects only one loop
    for (var j = 0; j < 5; j++){
      if(j==2) continue innerLoopContinueLabeled;
      console.log("innerLoopContinueLabeled: ", j);
    }// end of for

    console.log("------------------------------");

    // shows labeled exit for more than one loop
    // break statement exits from outerLoop and ends the program
    for (var j = 0; j < 5; j++){
      if(j==3) break outerLoop;
      console.log("outerLoopBreakLabeled: ", j);
    }// end of for

  }// end of for
```

1.2.2. Output

```
------------------------------
innerLoopBreakUnlabeled:  0
------------------------------
innerLoopBreakLabeled:  0
------------------------------
innerLoopContinueUnlabeled:  0
innerLoopContinueUnlabeled:  1
innerLoopContinueUnlabeled:  3
innerLoopContinueUnlabeled:  4
------------------------------
innerLoopContinueLabeled:  0
innerLoopContinueLabeled:  1
innerLoopContinueLabeled:  3
innerLoopContinueLabeled:  4
------------------------------
outerLoopBreakLabeled:  0
outerLoopBreakLabeled:  1
outerLoopBreakLabeled:  2
```

1.2.3. Explanation

Javascript language supports labeling. So, the first statement puts a label for the outer loop, **outerLoop**.

After that, the first inner loop checks if the counter is 1. If this check is successful, it breaks the loop using the unlabeled break statement **break**. If not, it prints the counter with the section name. This part prints the section name and 0.

After the first loop, there is another label, **innerLoopBreakLabeled**. Following loop checks, again, if the counter is 1. If this check is successful, it breaks the loop using the labeled break statement **break innerLoopBreakLabeled**. If not, it prints the counter with the section name. This code section aims to show that the labeled break statements can work with one loop without any problem. This part of the program prints the section name and 0, the same as the previous section, with only the section name differences.

After this loop, the continue statements come with the same logic. In the following two loops, there is an if checking whether the loop counter is 2. If it is 2, it continues without a label in the first, using **continue**, with the label, using **continue innerLoopContinueLabeled**, in the latter loop. Otherwise, it prints the counter with the section name in both loops. These two sections print section names and 0-1-3-4 in different lines on the console.

The last loop checks if the counter is 3. If it is 3, it breaks from the outer loop using the label **outerLoop**. Otherwise, it prints the counter with the section name. This code section aims to show that the labeled break statements can work with more than one loop without any problem. This part prints the section name and 0-1-2 in different lines on the console.

1.3. Lua

1.3.1. Code

```
for i = 0, 5, 1
do
    print("-------------------------------")

    -- shows unlabeled exit
    -- break statement exits only one loop
    for j = 0, 4, 1
    do
        if j==1 then break end
        print("innerLoopBreakUnlabeled: ", j)
    end


    -- no labeled exit in Lua

    print("-------------------------------")
```

```
    -- no continue in Lua
    -- no labeled exit in Lua
end
```

1.3.2. Output

```
------------------------------
innerLoopBreakUnlabeled:   0
------------------------------
------------------------------
innerLoopBreakUnlabeled:   0
------------------------------
------------------------------
innerLoopBreakUnlabeled:   0
------------------------------
------------------------------
innerLoopBreakUnlabeled:   0
------------------------------
------------------------------
innerLoopBreakUnlabeled:   0
------------------------------
------------------------------
innerLoopBreakUnlabeled:   0
------------------------------
```

1.3.3. Explanation

Lua language doesn't support labeling. This means we don't have any label in any part of the code.

In the code there is an outer loop. After that, the first inner loop checks if the counter is 1. If this check is successful, it breaks the loop using the unlabeled break statement **break**. If not, it prints the counter with the section name. This part prints the section name and 0, five times.

Since Lua language doesn't support continue statements, there is neither any code nor output for continue statements[2]. However, Lua supports goto statements. Continue statements can be simulated using this goto statements. However, this is out of scope of ths assignment.

Labels don't exist in the Lua language. However, they can be simulated using flags. However, this is out of scope of this homework.

1.4. PHP

1.4.1. Code

```
<?php
```

```
    for ($i = 0; $i < 5; $i++) {

     echo "-------------------------------\n";


     // shows unlabeled exit
     // break statement exits only one loop
    for ($j = 0; $j < 5; $j++) {
       if ($j==1) break;
       echo("innerLoopBreakUnlabeled: $j \n");
    }// end of for


    echo("-------------------------------\n");


     // shows labeled exit
     // break statement exits only one loop
    for ($j = 0; $j < 5; $j++){
       if ($j==1) break 1;
       echo("innerLoopBreakLabeled: $j \n");
    }// end of for

    echo("-------------------------------\n");


     // shows unlabeled continue
     // continue statement affects only one loop
    for ($j = 0; $j < 5; $j++){
       if($j==2) continue;
       print("innerLoopContinueUnlabeled: $j \n");
    }// end of for

    echo("-------------------------------\n");

    // shows labeled continue
    // continue statement affects only one loop
    for ($j = 0; $j < 5; $j++){
       if($j==2) continue 1;
       print("innerLoopContinueLabeled: $j \n");
    }// end of for

    echo("-------------------------------\n");


    // shows labeled exit for more than one loop
    // break statement exits from outerLoop and ends the program
    for ($j = 0; $j < 5; $j++){
       if($j==3) break 2;
       echo("outerLoopBreakLabeled: $j \n");
    }// end of for

   }// end of for
?>
```

1.4.2. Output

```
-------------------------------
```

```
innerLoopBreakUnlabeled: 0
------------------------------
innerLoopBreakLabeled: 0
------------------------------
innerLoopContinueUnlabeled: 0
innerLoopContinueUnlabeled: 1
innerLoopContinueUnlabeled: 3
innerLoopContinueUnlabeled: 4
------------------------------
innerLoopContinueLabeled: 0
innerLoopContinueLabeled: 1
innerLoopContinueLabeled: 3
innerLoopContinueLabeled: 4
------------------------------
outerLoopBreakLabeled: 0
outerLoopBreakLabeled: 1
outerLoopBreakLabeled: 2
```

1.4.3. Explanation

PHP language supports labeling in a different way from other labeling-supportive languages. In PHP, the current loop is labeled 1, and outer loops are labeled 2-3, etc., implicitly[4]. That is, developers don't need to specify a label name for loops if they want to use break or continue statements with labels. They just need to specify which enclosing loop to break or continue using numbers.

In the code, there is an outer loop. After that, the first inner loop checks if the counter is 1. If this check is successful, it breaks the loop using the unlabeled break statement **break**. If not, it prints the counter with the section name. This part prints the section name and 0.

After the first loop, the following loop checks again if the counter is 1. If this check is successful, it breaks the loop using the labeled break statement **break 1**. (In the manual, it says **break (1)** would also work.) If not, it prints the counter with the section name. This code section aims to show that the labeled break statements can work with one loop without any problem. This part of the program prints the section name and 0, the same as the previous section, with only the section name differences.

After this loop, the continue statements come with the same logic. In the following two loops, there is an if checking whether the loop counter is 2. If it is 2, it continues without a label in the first, using **continue**, with the label, using **continue 1**, in the latter loop. Otherwise, it prints the counter with the section name in both loops. These two sections print section names and 0-1-3-4 in different lines on the console.

The last loop checks if the counter is 3. If it is 3, it breaks from the outer loop using the label **2**. Otherwise, it prints the counter with the section name. This code section aims to show that the labeled break statements can work with more than one loop without any problem. This part prints the section name and 0-1-2 in different lines on the console.

1.5. Python

1.5.1. Code

```
for i in range(5):
    print("------------------------------")
    # shows unlabeled exit
    # break statement exits only one loop
    for j in range(5):
        if j==1:
            break
        print("innerLoopBreakUnlabeled: ", j)

    print("------------------------------")


    # no labeled exit in python

    # shows unlabeled continue
    # continue statement affects only one loop
    for j in range(5):
        if j==2:
            continue
        print("innerLoopContinueUnlabeled: ", j)

    print("------------------------------")

    # no labeled continue in python

    # no labeled exit in python
```

1.5.2. Output

```
------------------------------
innerLoopBreakUnlabeled:  0
------------------------------
innerLoopContinueUnlabeled:  0
innerLoopContinueUnlabeled:  1
innerLoopContinueUnlabeled:  3
innerLoopContinueUnlabeled:  4
------------------------------
------------------------------
innerLoopBreakUnlabeled:  0
------------------------------
innerLoopContinueUnlabeled:  0
innerLoopContinueUnlabeled:  1
innerLoopContinueUnlabeled:  3
innerLoopContinueUnlabeled:  4
------------------------------
------------------------------
innerLoopBreakUnlabeled:  0
------------------------------
innerLoopContinueUnlabeled:  0
innerLoopContinueUnlabeled:  1
innerLoopContinueUnlabeled:  3
innerLoopContinueUnlabeled:  4
------------------------------
```

```
------------------------------
innerLoopBreakUnlabeled:  0
------------------------------
innerLoopContinueUnlabeled:  0
innerLoopContinueUnlabeled:  1
innerLoopContinueUnlabeled:  3
innerLoopContinueUnlabeled:  4
------------------------------
------------------------------
innerLoopBreakUnlabeled:  0
------------------------------
innerLoopContinueUnlabeled:  0
innerLoopContinueUnlabeled:  1
innerLoopContinueUnlabeled:  3
innerLoopContinueUnlabeled:  4
------------------------------
```

1.5.3. Explanation

Python language doesn't support labeling. This means we don't have any label in any part of the code.

In the code there is an outer loop. After that, the first inner loop checks if the counter is 1. If this check is successful, it breaks the loop using the unlabeled break statement **break**. If not, it prints the counter with the section name. This part prints the section name and 0 for five times.

After this loop, the continue statement come with the same logic. In the following loop, there is an if checking whether the loop counter is 2. If it is 2, it continues without a label, using **continue.** Otherwise, it prints the counter with the section name. These two sections print section names and 0-1-3-4 in different lines on the console for five times.

Labels don't exist in the Python language. However, they can be simulated using flags[5,6]. However, this is out of scope of this homework.

1.6. Ruby

1.6.1. Code

```
for i in 0..4 do
    puts("------------------------------")
    # shows unlabeled exit
    # break statement exits only one loop
    for j in 0..4 do
        break if j==1
        puts("innerLoopBreakUnlabeled: ", j)
    end


  puts("------------------------------")

    # no labeled exit in ruby
```

```
    # shows unlabeled continue
    # continue statement affects only one loop
    for j in 0..4 do
        next if j==2
        puts("innerLoopContinueUnlabeled: ", j)
    end
    puts("------------------------------")

    # no labeled continue in ruby

    # no labeled exit in ruby
end
```

1.6.2. Output

```
------------------------------
innerLoopBreakUnlabeled:
0
------------------------------
innerLoopContinueUnlabeled:
0
innerLoopContinueUnlabeled:
1
innerLoopContinueUnlabeled:
3
innerLoopContinueUnlabeled:
4
------------------------------
------------------------------
innerLoopBreakUnlabeled:
0
------------------------------
innerLoopContinueUnlabeled:
0
innerLoopContinueUnlabeled:
1
innerLoopContinueUnlabeled:
3
innerLoopContinueUnlabeled:
4
------------------------------
------------------------------
innerLoopBreakUnlabeled:
0
------------------------------
innerLoopContinueUnlabeled:
0
innerLoopContinueUnlabeled:
1
innerLoopContinueUnlabeled:
3
innerLoopContinueUnlabeled:
4
------------------------------
------------------------------
innerLoopBreakUnlabeled:
0
```

```
------------------------------
innerLoopContinueUnlabeled:
0
innerLoopContinueUnlabeled:
1
innerLoopContinueUnlabeled:
3
innerLoopContinueUnlabeled:
4
------------------------------
------------------------------
innerLoopBreakUnlabeled:
0
------------------------------
innerLoopContinueUnlabeled:
0
innerLoopContinueUnlabeled:
1
innerLoopContinueUnlabeled:
3
innerLoopContinueUnlabeled:
4
------------------------------
```

1.6.3. Explanation

Ruby language doesn't support labeling. This means we don't have any label in any part of the code.

In the code there is an outer loop. After that, the first inner loop checks if the counter is 1. If this check is successful, it breaks the loop using the unlabeled break statement **break**[7]. If not, it prints the counter with the section name. This part prints the section name and 0 for five times.

After this loop, the continue statement come with the same logic. In the following loop, there is an if checking whether the loop counter is 2. If it is 2, it continues without a label, using **next.** Otherwise, it prints the counter with the section name. These two sections print section names and 0-1-3-4 in different lines on the console for five times.

Labels don't exist in the Ruby language. However, they can be simulated using flags[8]. However, this is out of scope of this homework.

1.7. Rust

1.7.1. Code

```
fn main() {
  'outerLoop:  // label
    for _i in 0..5 {
      println!("----------------------");

      // shows unlabeled exit
```

```
        // break statement exits only one loop
        for j in 0..5{
          if j == 1{
            break;
          }
          println!("innerLoopBreakUnlabeled: {}", j);
        }

        println!("---------------------");

        'innerLoopBreakLabeled: // label
        // shows labeled exit
        // break statement exits only one loop
        for j in 0..5{
           if j == 1{
            break 'innerLoopBreakLabeled;
          }
          println!("innerLoopBreakLabeled: {}", j);
        }

        println!("---------------------");

        // shows unlabeled continue
        // continue statement affects only one loop
        for j in 0..5{
          if j == 2{
            continue;
          }
          println!("innerLoopContinueUnlabeled: {}", j);
        }

        println!("---------------------");

        'innerLoopContinueLabeled:// label

        // shows labeled continue
        // continue statement affects only one loop
        for j in 0..5{
           if j == 2{
            continue 'innerLoopContinueLabeled;
          }
          println!("innerLoopContinueLabeled: {}", j);
        }

        println!("---------------------");

        // shows labeled exit for more than one loop
        // break statement exits from outerLoop and ends
        // the program
        for j in 0..5{
          if j == 3{
            break 'outerLoop;
          }
          println!("outerLoopBreakLabeled: {}", j);
        }
}
}
```

1.7.2. Output

```
--------------------
innerLoopBreakUnlabeled: 0
--------------------
innerLoopBreakLabeled: 0
--------------------
innerLoopContinueUnlabeled: 0
innerLoopContinueUnlabeled: 1
innerLoopContinueUnlabeled: 3
innerLoopContinueUnlabeled: 4
--------------------
innerLoopContinueLabeled: 0
innerLoopContinueLabeled: 1
innerLoopContinueLabeled: 3
innerLoopContinueLabeled: 4
--------------------
outerLoopBreakLabeled: 0
outerLoopBreakLabeled: 1
outerLoopBreakLabeled: 2
```

1.7.3. Explanation

Rust language supports labeling. So, the first statement puts a label for the outer loop, **'outerLoop**. In Rust, labels start with (')[9].

After that, the first inner loop checks if the counter is 1. If this check is successful, it breaks the loop using the unlabeled break statement **break**. If not, it prints the counter with the section name. This part prints the section name and 0.

After the first loop, there is another label, **'innerLoopBreakLabeled**. Following loop checks, again, if the counter is 1. If this check is successful, it breaks the loop using the labeled break statement **break 'innerLoopBreakLabeled**. If not, it prints the counter with the section name. This code section aims to show that the labeled break statements can work with one loop without any problem. This part of the program prints the section name and 0, the same as the previous section, with only the section name differences.

After this loop, the continue statements come with the same logic. In the following two loops, there is an if checking whether the loop counter is 2. If it is 2, it continues without a label in the first, using **continue**, with the label, using **continue** '**innerLoopContinueLabeled**, in the latter loop. Otherwise, it prints the counter with the section name in both loops. These two sections print section names and 0-1-3-4 in different lines on the console.

The last loop checks if the counter is 3. If it is 3, it breaks from the outer loop using the label **'outerLoop**. Otherwise, it prints the counter with the section name. This code section aims to show that the labeled break statements can work with more than one loop without any problem. This part prints the section name and 0-1-2 in different lines on the console.

2. Evaluation of The Languages

The evaluation of the languages in this homework will be done according to their readability,
writeability and also their comprehensiveness.

In my opinion, Dart, Javascript, and Rust are the best languages for readability of User-Located Loop Control Mechanisms. Since they are easy to follow with their instructions like break, and continue, and their labeling mechanism is intuitive. PHP follows these languages as its labeling mechanism which contains numbers might confuse some readers. Python and Ruby follow PHP. They don't contain labels. When coders simulate labels using flags they are hard to read. Also Ruby has next instruction for the continue purpose. People are accustomed to read continue statements might have some problems there. Lua is the worst in terms of readability because it doesn't contain continue statement and developers simulate them using goto statements. This makes Lua the worst readable language.

I think, Dart, Javascript, and Rust are the best languages for writability of User-Located Loop Control Mechanisms. Since they are easy to write with their instructions like break, and continue, and their labeling mechanism is intuitive. They provide an opportunity to developers to give easy-to-understand label names for the loops. PHP follows these languages as its labeling mechanism which contains numbers might confuse developers. Python and Ruby follow PHP. They don't contain labels. When coders simulate labels using flags they are hard to write and requires some more steps to achieve the same goal. Also next statement in Ruby might be confusing for some developers. Lua is the worst in terms of readability because it doesn't contain continue statement and developers simulate them using goto statements. This makes Lua the worst writable language.

Since Dart, Javascript, PHP and Rust contains labeling and supports both continue and break statements, they are the most comprehensive languages. Python and Ruby supports break and continue statements but doesn't support labeling. This makes them the second most comprehensive languages. Lua doesn't support continue statements, and labeling. Thus it is the least comprehensive language.

After all, I chose Dart as the best language for User-Located Loop Control Mechanisms. It is easy to write and read, and also contains every instruction with intuitive approaches. Hence, Dart is the best language for me. Javascript and Rust follows it.

3. Learning Strategy

Before starting writing code I studied from the slides and the book to the topic. I realized that labeling should be similar process in languages that contains labeling. Indeed I was right. Labeling is the same in every language in this homework that contains labeling but PHP.

When I stuck, I googled the points I. At these times, I took the advantage of some well-known programming web site and forums such as StackOverflow, GeeksForGeeks, etc.

I used online compilers except for Python since I am already using it on my local machine.

There are many online compilers, but after all the ones that I used seem fair enough for this
assignment.

The list of the links for each online compilers and useful resources I used is given in below.

Dart

Online Compiler: https://dartpad.dev/?id=57b5bb0b8e8eea41d662f2aad36893d7

Additional: I haven't used any other resource for Dart.

Javascript

Online Compiler: https://www.programiz.com/javascript/online-compiler/

Additional: I haven't used any other resource for Javascript.

Lua

Online Compiler: https://www.tutorialspoint.com/execute_lua_online.php

Additional:
[1], [2]

PHP

Online Compiler: https://paiza.io/projects/VwgjBAq-hmMsuv3N6w08ow

Additional: [3], [4]

Python

Online Compiler: I used my local interpreter to run python code

Additional:[5], [6]

Ruby

Online Compiler: https://replit.com/languages/ruby

Additional: [7],[8]

Rust

Online Compiler: https://replit.com/languages/rust

Additional: [9],[10]

References

[1]“Lua - loops,” *Tutorials Point*. [Online]. Available: https://www.tutorialspoint.com/lua/lua_loops.htm. [Accessed: 03-Dec-2022].

[2]“Why does Lua have no ‘continue’ statement?,” *Tutorials Point*. [Online]. Available: https://www.tutorialspoint.com/why-does-lua-have-no-continue-statement. [Accessed: 03-Dec-2022].

[3]“PHP for loop,” *PHP for loops*. [Online]. Available: https://www.w3schools.com/php/php_looping_for.asp. [Accessed: 03-Dec-2022].

[4]Marty, “How can I break an outer loop with php?,” *Stack Overflow*, 01-Jun-1958. [Online]. Available: https://stackoverflow.com/questions/5880442/how-can-i-break-an-outer-loop-with-php. [Accessed: 03-Dec-2022].

[5]Y. Zhou, “5 ways to break out of nested loops in Python,” *Medium*, 09-Jan-2022. [Online]. Available: https://medium.com/techtofreedom/5-ways-to-break-out-of-nested-loops-in-python-4c505d34ace7. [Accessed: 03-Dec-2022].

[6]Sahas, “Python: Continuing to next iteration in Outer Loop,” *Stack Overflow*, 01-Feb-1957. [Online]. Available: https://stackoverflow.com/questions/1859072/python-continuing-to-next-iteration-in-outer-loop. [Accessed: 03-Dec-2022].

[7]“Ruby: Loops (for, while, do..while, until),” *GeeksforGeeks*, 05-Jul-2021. [Online]. Available: https://www.geeksforgeeks.org/ruby-loops-for-while-do-while-until/. [Accessed: 03-Dec-2022].

[8]Fluffy, “How to break outer cycle in ruby?,” *Stack Overflow*, 01-Oct-1956. [Online]. Available:

https://stackoverflow.com/questions/1352120/how-to-break-outer-cycle-in-ruby#:~:text=Your%20options%20are%3A,break%20out%20of%20the%20loop
https://stackoverflow.com/questions/4010039/equivalent-of-continue-in-ruby. [Accessed: 03-Dec-2022].

[9]“The rust reference,” *Loop expressions - The Rust Reference*. [Online]. Available: https://doc.rust-lang.org/reference/expressions/loop-expr.html. [Accessed: 03-Dec-2022].

[10]“Rust by example,” *if/else - Rust By Example*. [Online]. Available: https://doc.rust-lang.org/rust-by-example/flow_control/if_else.html. [Accessed: 03-Dec-2022].