Programming Languages

CS 315

Homework 2

Kutay Tire

22001787

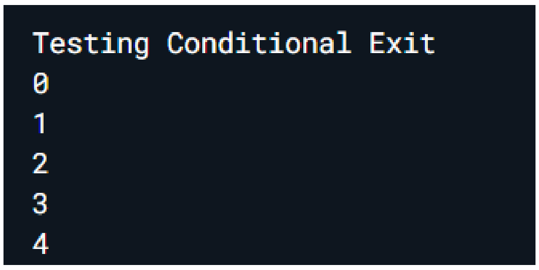
Section 2

1. Dart

Testing Conditional Exit

```
print("Testing Conditional Exit");
 for (int index = 0; index < 5; index++)
   print(index);
```

Output



```
Testing Conditional Exit
0
1
2
3
4
```

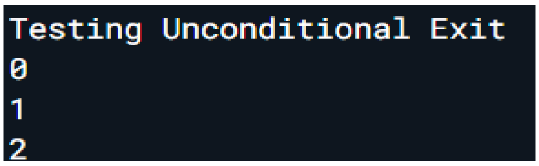
Explanation

This code segment shows the conditional exit from a loop in Dart. When the index becomes 5, the condition of the for-loop is no longer satisfied and the loop is terminated.

Testing Unconditional Exit

```
print("Testing Unconditional Exit");
 for (int index = 0; index < 5; index++) {
   print(index);
   if (index == 2)
     break;}
```

Output



```
Testing Unconditional Exit
0
1
2
```

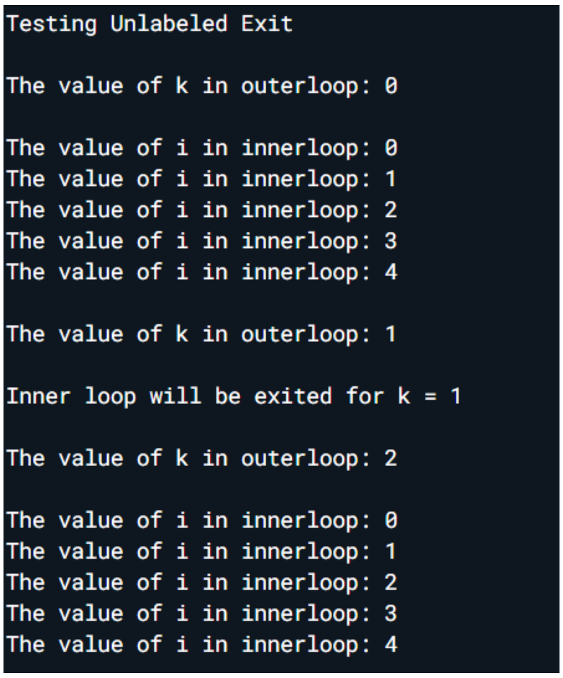
Explanation

This code segment shows the unconditional exit from a loop in Dart. When the break statement is executed, the loop is automatically terminated hence the program only prints 0, 1 and 2.

Testing Unlabeled Exit

```
print("Testing Unlabeled Exit");
 outerLoop2:
  for (int k = 0; k < 3; k++) {
   print("\nThe value of k in outer loop: $k \n");
  innerLoop2:
   for (int i = 0; i < 5; i++) {
    if(k == 1) {
      print("Inner loop will be exited for k = $k");
      break;}
    print("The value of i in innerloop: $i ");}}
```

Output



```
Testing Unlabeled Exit

The value of k in outerloop: 0

The value of i in innerloop: 0
The value of i in innerloop: 1
The value of i in innerloop: 2
The value of i in innerloop: 3
The value of i in innerloop: 4

The value of k in outerloop: 1

Inner loop will be exited for k = 1

The value of k in outerloop: 2

The value of i in innerloop: 0
The value of i in innerloop: 1
The value of i in innerloop: 2
The value of i in innerloop: 3
The value of i in innerloop: 4
```

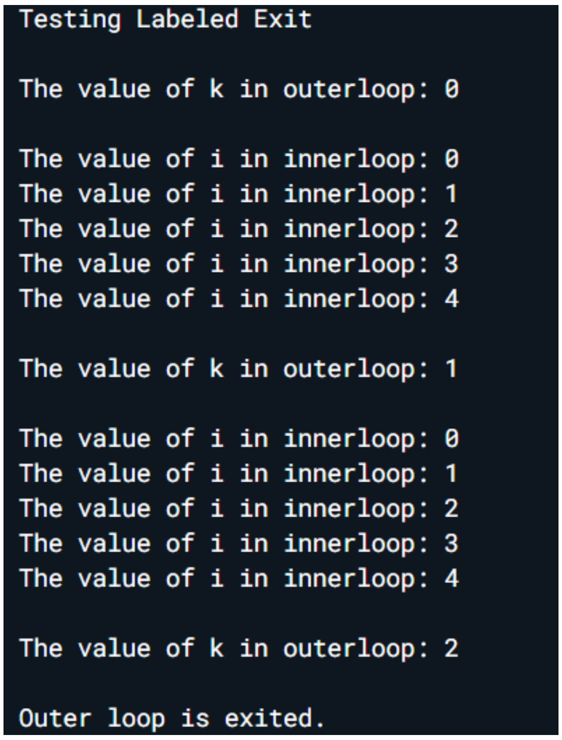
Explanation

This code segment is a sample for unlabeled exit in Dart. Dart supports labeling the loops like JavaScript or Ruby. However, when the break statement is used with no labels, the first inner loop where break is used is terminated. As a result, the “i” values are not printed when k is 1. Program automatically skips to the case when k is 2.

Testing Labeled Exit

```
print("Testing Labeled Exit");
 outerLoop:
  for (int k = 0; k < 10; k++) {
   print("\nThe value of k in outerloop: $k \n");
  innerLoop:
   for (int i = 0; i < 5; i++) {
     if(k == 2)
       break outerLoop;
   print("The value of i in innerloop: $i "); }}
 print("Outer loop is exited.");
```

Output



```
Testing Labeled Exit

The value of k in outerloop: 0

The value of i in innerloop: 0
The value of i in innerloop: 1
The value of i in innerloop: 2
The value of i in innerloop: 3
The value of i in innerloop: 4

The value of k in outerloop: 1

The value of i in innerloop: 0
The value of i in innerloop: 1
The value of i in innerloop: 2
The value of i in innerloop: 3
The value of i in innerloop: 4

The value of k in outerloop: 2

Outer loop is exited.
```

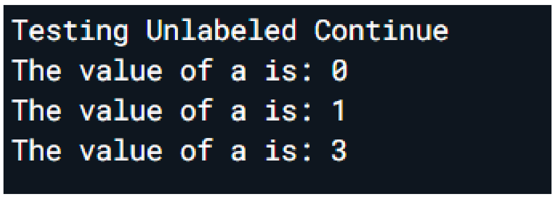
Explanation

When labels are used, specific loops can be exited with the help of break statements. In this code segment, the outer loop can be directly exited with **break outerLoop** statement. As a result, when k is 2, the execution of the whole loop is terminated. Note that without any labels, only the inner loop will be terminated similar to the previous example.

Testing Unlabeled Continue

```
print("\nTesting Unlabeled Continue");
 int a = 0;
 while(a < 4){
    if (a == 2) {
     a++;
     continue;}
   print("The value of a is: $a");
   a++;}
```

Output



```
Testing Unlabeled Continue
The value of a is: 0
The value of a is: 1
The value of a is: 3
```

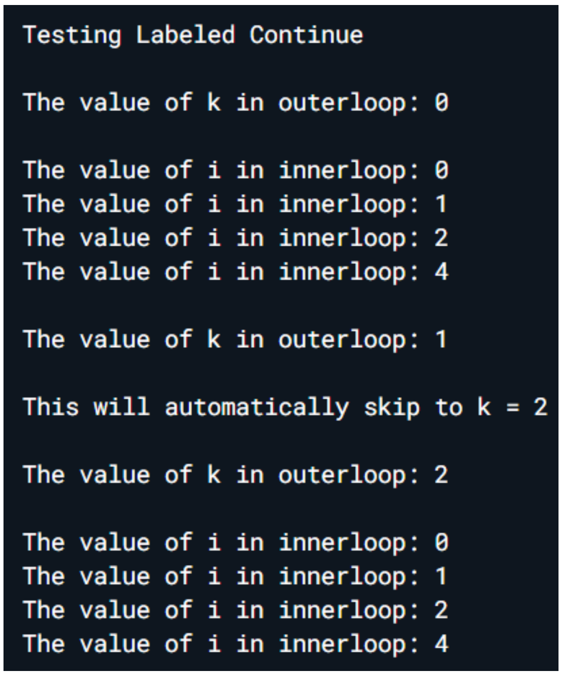
Explanation

Using the **continue** statement is an alternative way to exit a loop. Similar to break, continue statements can also be labelled. When no label is used, the inner loop will automatically be considered in nested loops. In this example, the program skips the case when "a" is 2 and it is not printed as it can be seen from the output.

Testing Labeled Continue

```
print("\nTesting Labeled Continue");
 outerLoop3:
  for (int k = 0; k < 3; k++) {
    print("\nThe value of k in outerloop: $k \n");
  innerLoop3:
    for (int i = 0; i < 5; i++) {
      if(k == 1) {
        print("This will automatically skip to k = 2");
        continue outerLoop3; }
      if(i == 3) {
        continue innerLoop3;}
      print("The value of i in innerloop: $i ");}}
```

Output



```
Testing Labeled Continue

The value of k in outerloop: 0

The value of i in innerloop: 0
The value of i in innerloop: 1
The value of i in innerloop: 2
The value of i in innerloop: 4

The value of k in outerloop: 1

This will automatically skip to k = 2

The value of k in outerloop: 2

The value of i in innerloop: 0
The value of i in innerloop: 1
The value of i in innerloop: 2
The value of i in innerloop: 4
```

Explanation

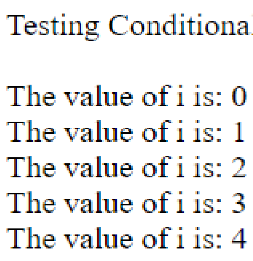
Using the **continue** statement with labels is also supported in Dart. In this example, when k is 2, the outer loop automatically skips to the case when k is 2 with **continue outerLoop3** statement. The same is also possible with inner loops. To demonstrate this, the code segments also executes another continue statement when every time “i” is 3. As a result, 3 is never printed as a value of i.

2. JavaScript

Testing Conditional Exit

```
document.write("Testing Conditional Exit<br>");

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

   document.write("<br>The value of i is: ", i,);}

document.write("<br>");
```

Output



```
Testing Conditional Exit

The value of i is: 0
The value of i is: 1
The value of i is: 2
The value of i is: 3
The value of i is: 4
```

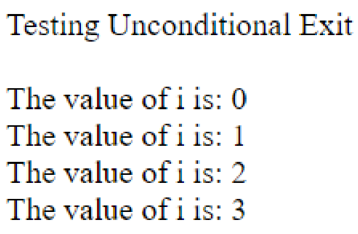
Explanation

This code segment shows the conditional exit from a loop in JavaScript. When the “i” value becomes 5, the condition of the for-loop is no longer satisfied and the loop is terminated. This is true for other loops like a **while** or a **do-while** loop. So, it is not separately examined.

Testing Unconditional Exit

```
document.write("<br>Testing Unconditional
Exit<br>");

for(let i = 0; i < 6; i++) {

  document.write("<br> The value of i is: ", i);

   if (i == 3){

            break;}}

document.write("<br>");
```

Output



```
Testing Unconditional Exit

The value of i is: 0
The value of i is: 1
The value of i is: 2
The value of i is: 3
```

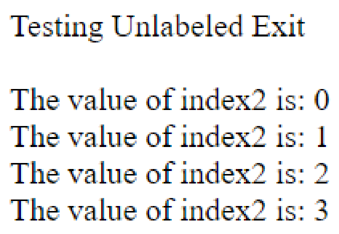
Explanation

This code segment shows the unconditional exit from a loop in JavaScript. When the break statement is executed, the loop is automatically terminated hence the program only prints 0, 1, 2 and 3. Normally, 4 and 5 would also be printed if the loop was not exited with a break statement.

Testing Unlabeled Exit

```
document.write("<br>Testing Unlabeled
Exit<br>");

let index2 = 0;

while(index2 < 7) {

  document.write("<br>The value of index2 is: ",
  index2);

   if(index2 == 3) {
          break;}
   index2++;}
document.write("<br>");
```

Output



```
Testing Unlabeled Exit

The value of index2 is: 0
The value of index2 is: 1
The value of index2 is: 2
The value of index2 is: 3
```

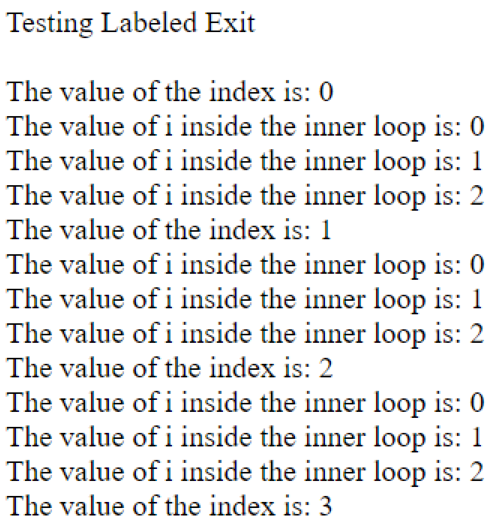
Explanation

This code segment is a sample for unlabeled exit in JavaScript. In JavaScript, labeling loops is supported, but if a **break** or a **continue** is used without any labels, the first inner loop is considered. In this sample, the loop is automatically exited when index is 4.

Testing Labeled Exit

```
document.write("<br>Testing Labeled Exit<br>");

let index = 0;

outerLoop:

  while(index < 6) {

          document.write("<br>The value of the
          index     is:   ", index);

          innerLoop:

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

                    if(i == 3) {

                            break innerLoop;}

                    if(index == 3){

                            break outerLoop;}

                    document.write("<br>The   value
                    of i inside the inner loop is: ", i);}

          index++;}
```

Output



```
Testing Labeled Exit

The value of the index is: 0
The value of i inside the inner loop is: 0
The value of i inside the inner loop is: 1
The value of i inside the inner loop is: 2
The value of the index is: 1
The value of i inside the inner loop is: 0
The value of i inside the inner loop is: 1
The value of i inside the inner loop is: 2
The value of the index is: 2
The value of i inside the inner loop is: 0
The value of i inside the inner loop is: 1
The value of i inside the inner loop is: 2
The value of the index is: 3
```

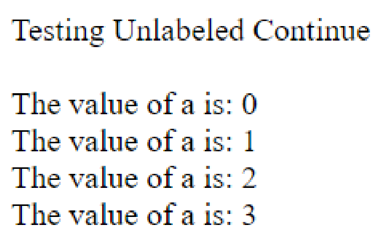
Explanation

The mechanism for labeled exit is very similar to Dart. The loops can be exited with a break or a continue statement by using a label. In this code segment, the outer loop is exited when index is 3. As a result, the program is terminated after this happens. However, inner loops can also be exited with labels. This is demonstrated with **break innerLoop** statement as the value of “i” is not printed after it becomes 3.

Testing Unlabeled Continue

```
document.write("<br>Testing Unlabeled
Continue<br>");

let a = 0;

while(a < 4) {

   document.write("<br>The value of a is: ", a);

   a++;

   continue;

   document.write("This statement is skipped.");}
```

Output



```
Testing Unlabeled Continue

The value of a is: 0
The value of a is: 1
The value of a is: 2
The value of a is: 3
```

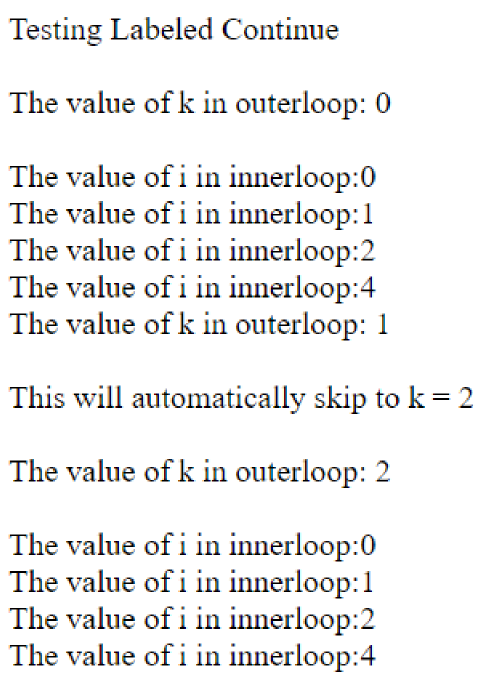
Explanation

Using the **continue** statement is an alternative way to exit a loop. Similar to break, continue statements can also be labelled. When no label is used, the inner loop will automatically be considered in nested loops in JavaScript. In this example, the print statement after the **continue** is never executed. As a result, “This statement is skipped” sentence cannot be seen at the output.

Testing Labeled Continue

```
document.write("Testing Labeled Continue<br>");

outerLoop2:

  for (let k = 0; k < 3; k++) {

          document.write("<br>The value of k in
          outerloop: ", k, "<br>");

          innerLoop2:

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

              if(k == 1) {

                    document.write("<br>This will
                    automatically skip to k = 2<br>");

                    continue outerLoop2;}

             if(i == 3) {

                    continue innerLoop2;}

             document.write("<br>The value of i in
             innerloop:" , i);}}
```

Output



```
Testing Labeled Continue

The value of k in outerloop: 0

The value of i in innerloop:0
The value of i in innerloop:1
The value of i in innerloop:2
The value of i in innerloop:4
The value of k in outerloop: 1

This will automatically skip to k = 2

The value of k in outerloop: 2

The value of i in innerloop:0
The value of i in innerloop:1
The value of i in innerloop:2
The value of i in innerloop:4
```

Explanation

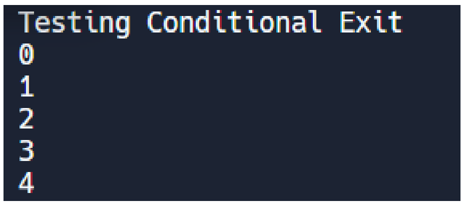
Using the **continue** statement with labels is also supported in JavaScript similar to Dart. In this example, when k is 1, the outer loop automatically skips to the case when k is 2 as it is printed. The same is also possible with inner loops. To demonstrate this, the code segments also executes another continue statement when every time “i” is 3. As a result, 3 is never printed as a value of i.

3. Lua

Testing Conditional Exit

```
print("Testing Conditional Exit")

local index = 0

while(index < 5)

do

  print(index)

  index = index + 1

end
```

Output



```
Testing Conditional Exit
0
1
2
3
4
```

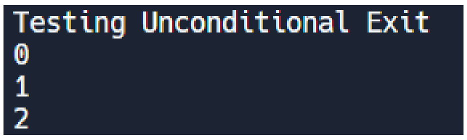
Explanation

This code segment shows the conditional exit from a loop in Lua. When the index becomes 5, the condition of the while-loop is no longer satisfied and the loop is terminated. Note that the loops in Lua are ended with the **end** key word.

Testing Unconditional Exit

```
print("Testing Unconditional Exit")

local index2 = 0

while(index2 < 6)

do

  print(index2)

  index2 = index2 + 1

  if(index2 == 3) then

    break

  end

end
```

Output



```
Testing Unconditional Exit
0
1
2
```

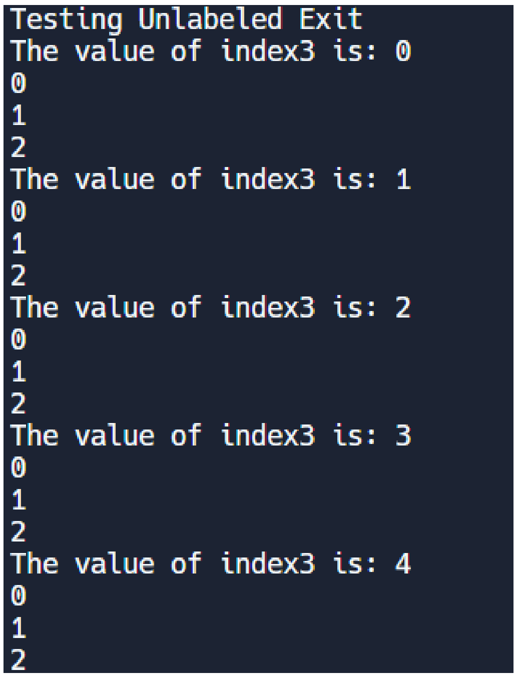
Explanation

This code segment shows the unconditional exit from a loop in Lua. When the break statement is executed, the loop is automatically terminated hence the program only prints 0, 1, and 2. Normally, 3, 4 and 5 would also be printed if the loop was not exited with a break statement.

Testing Unlabeled Exit

```
print("Testing Unlabeled Exit")

local index3 = 0

while(index3 < 5)

do

  print("The value of index3 is:", index3)

  index3 = index3 + 1

  for index4 = 0,10,1

  do

    if(index4 == 3) then

      break

    end

    print(index4)

  end

end
```

Output



```
Testing Unlabeled Exit
The value of index3 is: 0
0
1
2
The value of index3 is: 1
0
1
2
The value of index3 is: 2
0
1
2
The value of index3 is: 3
0
1
2
The value of index3 is: 4
0
1
2
```

Explanation

There no labelled loops in Lua. As a result, **break** statements automatically break the inner loop as shown in the sample. Also, for-loops do not follow a usual convention. The first entry after the identifier specifies the start index, the second one specifies the end index and the third one specifics the step size. In the code segment, when index4 becomes 3, the inner loop is exited. As a result, the value of index4 is not printed after 2.

Testing Labeled Exit

Explanation

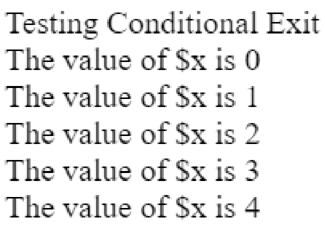
In Lua, there are no labeled exit mechanisms. Also, **continue** statements are not supported to exit a loop. Instead of these, Lua uses **go-to** statements. A go-to statement can also be used to exit a loop. However, as there are no labels, users cannot specify which loop to break. Furthermore, using go-to statements can make the language a little bit hard to understand. Yet, conditional/unconditional exit mechanisms are similar to other languages.

4. PHP

Testing Conditional Exit

```
echo "Testing Conditional Exit<br>";

for($x = 0; $x < 5; $x++) {

        echo "The value of \$x is $x <br>";}
```

Output



```
Testing Conditional Exit
The value of $x is 0
The value of $x is 1
The value of $x is 2
The value of $x is 3
The value of $x is 4
```

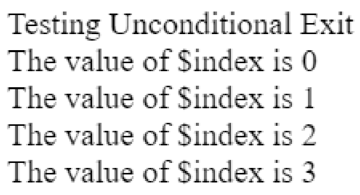
Explanation

This code segment shows the conditional exit from a loop in PHP. When the index becomes 5, the condition of the for-loop is no longer satisfied and the loop is terminated. Also, the syntax for the for-loops is similar to languages like Java or Dart.

Testing Unconditional Exit

```
echo "<br>Testing Unconditional Exit<br>";

$index = 0;

while($index < 8) {

        echo "The value of \$index is $index <br>";

        $index++;

         if($index == 4) {

                break;}}
```

Output



```
Testing Unconditional Exit
The value of $index is 0
The value of $index is 1
The value of $index is 2
The value of $index is 3
```

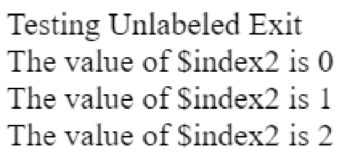
Explanation

This code segment shows the unconditional exit from a loop in PHP. When the break statement is executed, the loop is automatically terminated hence the program only prints 0, 1, 2 and 3. Normally, 4, 5, 6 and 7 would also be printed if the loop was not exited with a break statement.

Testing Unlabeled Exit

```
echo "Testing Unlabeled Exit<br>";

$index2 = 0;

while($index2 < 5) {

        echo "The value of \$index2 is $index2
        <br>";

  $index2++;

  if($index2 == 3) {

        break;}}
```

Output



```
Testing Unlabeled Exit
The value of $index2 is 0
The value of $index2 is 1
The value of $index2 is 2
```

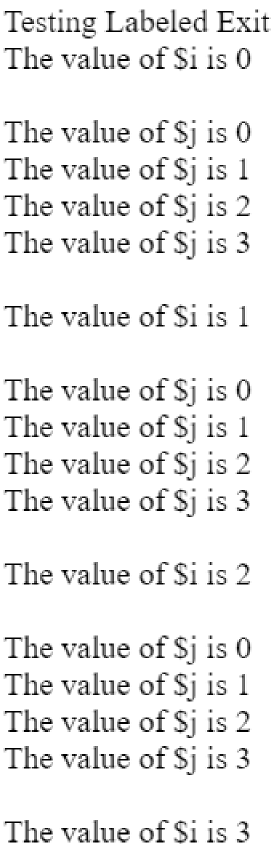
Explanation

Like in Dart or JavaScript, unlabeled exit mechanisms are supported with the help of **break** or **continue** statements in PHP. In this code segment, use of unlabeled break is demonstrated and use of continue is shown in the following samples. When the value of index2 becomes 3, the loop is exited as expected.

Testing Labeled Exit

```
echo "<br> Testing Labeled Exit";

for($i = 0; $i < 8; $i++) {

        echo "<br>The value of \$i is $i <br><br>";

        $j = 0;

        while($j < 10) {

        if ($i == 3) {break 2;}

        if($j == 4) {

                break 1;}

        echo "The value of \$j is $j <br>";

        $j++;}}
```

Output



```
Testing Labeled Exit
The value of $i is 0

The value of $j is 0
The value of $j is 1
The value of $j is 2
The value of $j is 3

The value of $i is 1

The value of $j is 0
The value of $j is 1
The value of $j is 2
The value of $j is 3

The value of $i is 2

The value of $j is 0
The value of $j is 1
The value of $j is 2
The value of $j is 3

The value of $i is 3
```

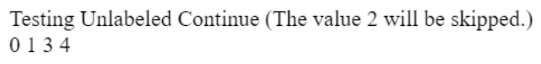
Explanation

In PHP, loops can be exited with the help of labels. However, unlike other languages like Dart or JavaScript, this is done by using integers instead of using the label of the loop. The outer loops are represented with bigger integers. For instance, if a nested loop contains three loops, the most outer one can be exited by using **break 3** statement. In this code sample, the outer loop is exited by using **break 2** when the value of $i is 3. Also, the inner loop is exited with **break 1** when the value of $j is 4.

Testing Unlabeled Continue

```
echo "<br>Testing Unlabeled Continue (The value
2 will be skipped.) <br>";

for ($a = 0; $a < 5; $a++) {

        if ($a == 2) {

                continue; }

        echo "$a ";}
```

Output



```
Testing Unlabeled Continue (The value 2 will be skipped.)
0 1 3 4
```

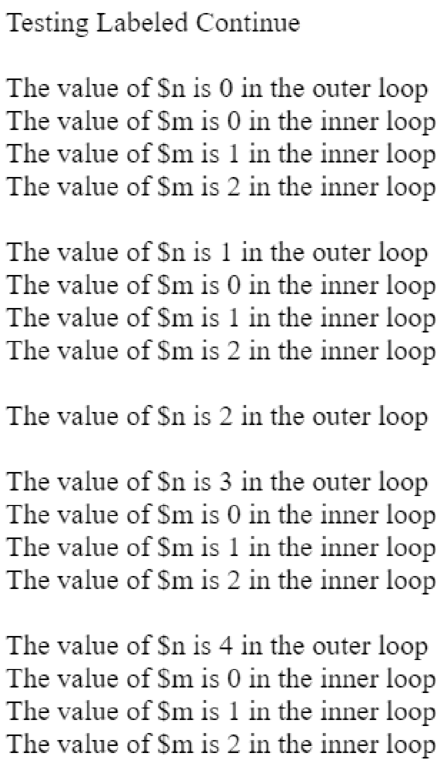
Explanation

Using the **continue** statement is an alternative way to exit a loop. When the value of $a is 2, the code segment automatically continues with $a = 3.

Testing Labeled Continue

```
echo "<br>Testing Labeled Continue <br>";

for ($n = 0; $n < 5; $n++) {

        echo "<br>The value of \$n is $n in the
        outer loop<br>";

        $m = 0;

         while($m < 3) {

                if ($n == 2) {

                        continue 2;}

                echo "The value of \$m is $m in
                the inner loop<br>";

                $m++;}}
```

Output



```
Testing Labeled Continue

The value of $n is 0 in the outer loop
The value of $m is 0 in the inner loop
The value of $m is 1 in the inner loop
The value of $m is 2 in the inner loop

The value of $n is 1 in the outer loop
The value of $m is 0 in the inner loop
The value of $m is 1 in the inner loop
The value of $m is 2 in the inner loop

The value of $n is 2 in the outer loop

The value of $n is 3 in the outer loop
The value of $m is 0 in the inner loop
The value of $m is 1 in the inner loop
The value of $m is 2 in the inner loop

The value of $n is 4 in the outer loop
The value of $m is 0 in the inner loop
The value of $m is 1 in the inner loop
The value of $m is 2 in the inner loop
```

Explanation

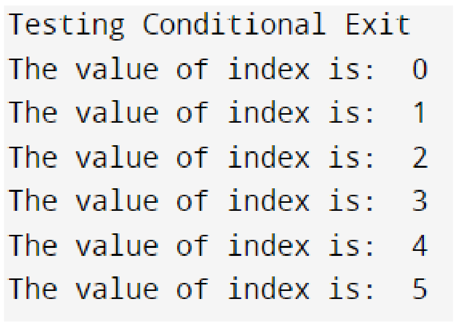
Using the **continue** statement with labels is also supported in PHP. Similar to break, labeling the loops is done with the help of integers. In this example, when $n is 2, the program skips to the case when $n is 3 and this is achieved by **continue 2** statement. Without labels, the continue statement would modify the inner while-loop whereas the users can specify which loop to modify with continue.

5. Python

Testing Conditional Exit

```
print("Testing Conditional Exit")

index = 0

while(index < 6):

    print("The value of index is: ", index)

    index += 1
```

Output



```
Testing Conditional Exit
The value of index is:  0
The value of index is:  1
The value of index is:  2
The value of index is:  3
The value of index is:  4
The value of index is:  5
```

Explanation

This code segment shows the conditional exit from a loop in Python. When the index becomes 6, the condition of the while-loop is no longer satisfied and the loop is terminated. Naturally, to avoid infinite loops, the value of index is incremented by one after the execution of the while-loop every time.

Testing Unconditional Exit

```
print("\nTesting Unconditional Exit")

for x in range(10):

    print("The value of x is: ", x)
```

```
  if (x == 4):
    break
print("\nTesting Unlabeled Exit")
```

Output

```
Testing Unconditional Exit
The value of x is:  0
The value of x is:  1
The value of x is:  2
The value of x is:  3
The value of x is:  4
```

Explanation

This code segment shows the unconditional exit from a loop in Python. When the break statement is executed, the loop is automatically terminated hence the program only prints 0, 1, 2, 3 and 4. Normally, all integer up to 10 would be printed without break.

Testing Unlabeled Exit

```
print("\nTesting Unlabeled Exit")
index2 = 0
while(index2 < 3):
  print("\nThe value of index2 is: ", index2, "\n")
  index3 = 0
  while(index3 < 7):
    print("The value of index3 is: ", index3)
    if(index3 == 3):
      break
    index3 += 1
  index2 += 1
```

Output

```
Testing Unlabeled Exit

The value of index2 is:  0

The value of index3 is:  0
The value of index3 is:  1
The value of index3 is:  2
The value of index3 is:  3

The value of index2 is:  1

The value of index3 is:  0
The value of index3 is:  1
The value of index3 is:  2
The value of index3 is:  3

The value of index2 is:  2

The value of index3 is:  0
The value of index3 is:  1
The value of index3 is:  2
The value of index3 is:  3
```

Explanation

Like in Dart or JavaScript, unlabeled exit mechanisms are supported with the help of **break** or **continue** statements in Python. In this code segment, use of unlabeled break is demonstrated. As it can be seen, when no label is used, break statement terminates the inner loop. So, index3 is not printed after it is 3 because the loop is exited.

Testing Labeled Exit

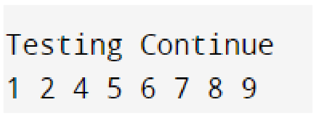
Explanation

Similar to Lua, there are no labeled exit mechanisms in Python. However, **continue** statements are supported to exit a loop which was not the case for Lua. As there are no labels, users cannot specify which loop to exit.

Testing Unlabeled Continue

```
print("\nTesting Unlabeled Continue")

a = 0

while (a < 9):

  a = a + 1

  if a == 3:

    continue

  print(a, end = " ")

print()
```

Output



```
Testing Continue
1 2 4 5 6 7 8 9
```

Explanation

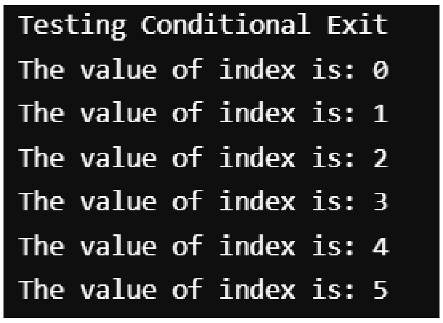
Using the **continue** statement is an alternative way to exit a loop. Similar to break, continue statements cannot be labelled in Python. When **continue** is used, the inner loop will automatically be considered in nested loops in Python. In this example, the value of "a" when it is 3 is not printed as the program skips to the case when it is 4 thanks to the **continue** statement.

6. Ruby

Testing Conditional Exit

```
puts "Testing Conditional Exit"

index = 0

while(index < 6)

  print "The value of index is: ", index

  puts nil

  index = index + 1

end

puts nil
```

Output



```
Testing Conditional Exit
The value of index is: 0
The value of index is: 1
The value of index is: 2
The value of index is: 3
The value of index is: 4
The value of index is: 5
```

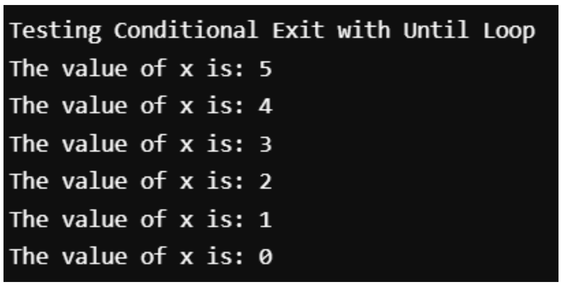
Explanation

This code segment shows the conditional exit from a loop in Ruby. When the index becomes 6, the condition of the while-loop is no longer satisfied and the loop is terminated. Naturally, to avoid infinite loops, the value of index is incremented by one after the execution of the while-loop every time.

Testing Conditional Exit with Until

```
puts "Testing Conditional Exit with Until Loop"

x = 5

until (x < 0)

  print "The value of x is: ", x

  x -= 1

  puts nil

end

puts nil
```

Output



```
Testing Conditional Exit with Until Loop
The value of x is: 5
The value of x is: 4
The value of x is: 3
The value of x is: 2
The value of x is: 1
The value of x is: 0
```

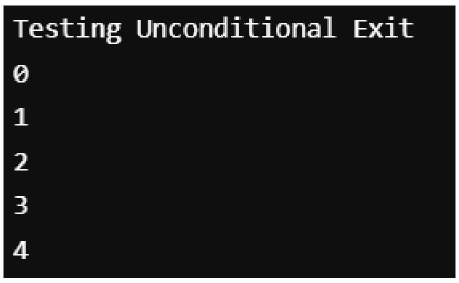
Explanation

This code segment shows another conditional exit mechanism in Ruby using **until**. The mechanism is very similar to **while** or **for**. In this code, as long as x is bigger than zero, the loop is executed and when it becomes 0, it is exited.

Testing Unconditional Exit

```
puts "Testing Unconditional Exit"

for i in 0..8 do

  puts i

  if(i == 4)

    break

  end

end
```

Output



```
Testing Unconditional Exit
0
1
2
3
4
```

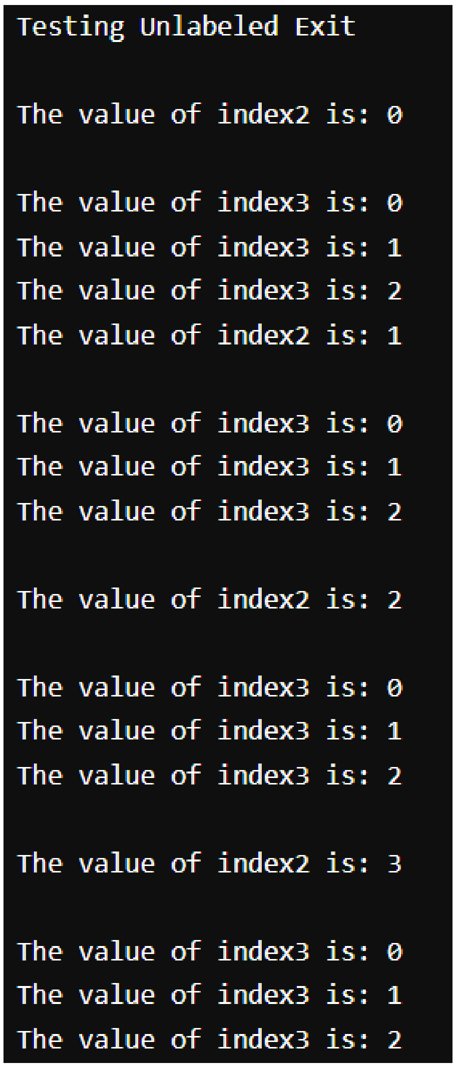
Explanation

This code segment shows the unconditional exit from a loop in Ruby. When the break statement is executed, the loop is automatically terminated hence the program only prints 0, 1, 2, 3 and 4. Normally, all integer up to 8 would be printed without break.

Testing Unlabeled Exit

```
puts "Testing Unlabeled Exit"

index2 = 0

while(index2 < 4)

  puts nil

  print "The value of index2 is: ", index2

  puts nil, nil

  index2 = index2 + 1

  index3 = 0

  while(index3 < 8)

    print "The value of index3 is: ", index3

    puts nil

    index3 = index3 + 1

    if(index3 == 3)

      break

    end

  end

end

puts nil
```

Output



```
Testing Unlabeled Exit

The value of index2 is: 0

The value of index3 is: 0
The value of index3 is: 1
The value of index3 is: 2
The value of index2 is: 1

The value of index3 is: 0
The value of index3 is: 1
The value of index3 is: 2

The value of index2 is: 2

The value of index3 is: 0
The value of index3 is: 1
The value of index3 is: 2

The value of index2 is: 3

The value of index3 is: 0
The value of index3 is: 1
The value of index3 is: 2
```

Explanation

Like in Dart or Python, unlabeled exit mechanisms are supported with the help of **break** or **next** (instead of continue) statements in Ruby. In this code segment, use of unlabeled break is demonstrated. As it can be seen, when no label is used, break statement terminates the inner loop. So, index3 is not printed after it is 3 because the loop is exited. However, the outer loop terminates conditionally as break does not affect it.

Testing Labeled Exit

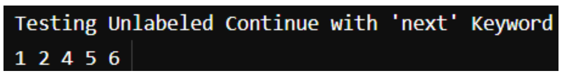
Explanation

Like Lua or Python, there are no labeled exit mechanisms in Ruby. However, **next** statements are supported to exit a loop which was not the case for Lua. As there are no labels, users cannot specify which loop to exit. So, labeled exits are not part of the language.

Testing Unlabeled Continue

```
k = 0
while k < 6
  k = k + 1
  if k == 3
    next
  end
  print k, " "
end
```

Output



```
Testing Unlabeled Continue with 'next' Keyword
1 2 4 5 6
```

Explanation

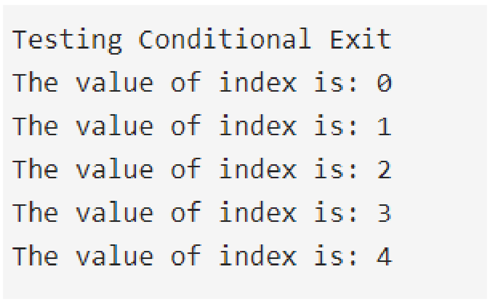
Using the **next** statement is an alternative way to exit a loop. Unlike other languages, Ruby uses the **next** keyword instead of continue but it has the same functionality with the continue statement in Python or Dart. In this example, the case when k is 3 is skipped with **next** and it is not printed as can be seen.

7. Rust

Testing Conditional Exit

```
println!("Testing Conditional Exit");
  let mut index = 0;
  while index < 5 {
    println!("The value of index is: {}", index);
    index = index + 1;}
```

Output



```
Testing Conditional Exit
The value of index is: 0
The value of index is: 1
The value of index is: 2
The value of index is: 3
The value of index is: 4
```

Explanation

This code segment shows the conditional exit from a loop in Rust. When the index becomes 5, the condition of the while-loop is no longer satisfied and the loop is terminated. Naturally, to avoid infinite loops, the value of index is incremented by one after the execution of the while-loop every time.

Testing Unconditional Exit

```
println!("\nTesting Unconditional Exit");
  let mut index2 = 0;
  while index2 < 7  {
    if index2 == 3 {
      break; }
    println!("The value of index2 is: {} ", index2);
    index2 = index2 + 1;}
```

Output

```
Testing Unconditional Exit
The value of index2 is: 0
The value of index2 is: 1
The value of index2 is: 2
```

Explanation

This code segment shows the unconditional exit from a loop in Rust. When the break statement is executed, the loop is automatically terminated hence the program only prints 0, 1, and 2. Normally, all integer up to 7 would be printed without break.

Testing Unlabeled Exit

```
println!("\nTesting Unlabeled Exit");
  'outer2: for i in 0..2 {
    println!("\nThe value of i in outer loop is: {}
\n", i, );
    'inner2: for j in 0..10 {
        println!("The value of j in inner loop is:
{}", j);
        if j == 3 {
          break;}}}
```

Output

```
Testing Unlabeled Exit

The value of i in outer loop is: 0
The value of j in inner loop is: 0
The value of j in inner loop is: 1
The value of j in inner loop is: 2
The value of j in inner loop is: 3

The value of i in outer loop is: 1
The value of j in inner loop is: 0
The value of j in inner loop is: 1
The value of j in inner loop is: 2
The value of j in inner loop is: 3
```

Explanation

Like in Dart or Python, unlabeled exit mechanisms are supported with the help of **break** or **continue** statements in Rust. In this code segment, use of unlabeled break is demonstrated. As it can be seen, when no label is used, break statement terminates the inner loop when j is 3. However, labeling loops is also possible in Rust.

Testing Labeled Exit

```
println!("\nTesting Labeled Exit");
  'outer: for x in 0..5 {
    println!("The value of x in outer loop is: {} \n",
x, );
    'inner: for y in 0..10 {
        if x == 2 {
          break 'outer;}
        if y == 3 {
          break 'inner;}
        println!("The value of y in inner loop is:
{}", y);}}
```

Output

```
Testing Labeled Exit
The value of x in outer loop is: 0

The value of y in inner loop is: 0
The value of y in inner loop is: 1
The value of y in inner loop is: 2
The value of x in outer loop is: 1

The value of y in inner loop is: 0
The value of y in inner loop is: 1
The value of y in inner loop is: 2
The value of x in outer loop is: 2
```

Explanation

In Rust, loops can be exited with the help of labels. In this example, loops are first labeled as inner and outer. The inner loop goes from 0 to 10 whereas the outer loop goes from 0 to 5. However, when x is 2, the outer loop is automatically terminated with **break ‘outer** statement. Similarly, the inner loop is terminated after y becomes 3. Then, the inner loop is terminated with **break ‘inner** statement.

Testing Unlabeled Continue

```
println!("\nTesting Unlabeled Continue");

  let mut index3 = 0;

  while index3 < 6 {

    index3 = index3 + 1;

    if index3 == 2 {

      continue; }

    print!("{} ", index3);}
```

Output

```
Testing Unlabeled Continue
1 3 4 5 6
```

Explanation

Using the **continue** statement is an alternative way to exit a loop. In this example, the case when index3 is 2 is skipped with **continue** and it is not printed as can be seen. Like Dart, labeling a continue statement is possible in Rust.

Testing Labeled Continue

```
println!("\nTesting Labeled Continue");

  'outer3: for x in 0..3 {

    println!("The value of x in outer loop is: {} \n",
x, );

    'inner3: for y in 0..3 {

        if x == 1 {

          continue 'outer3;}

        if y == 2 {

          continue 'inner3;}

        println!("The value of y in inner loop is: {}",
y);}}
```

Output

```
Testing Labeled Continue
The value of x in outer loop is: 0

The value of y in inner loop is: 0
The value of y in inner loop is: 1
The value of x in outer loop is: 1

The value of x in outer loop is: 2

The value of y in inner loop is: 0
The value of y in inner loop is: 1
```

Explanation

Using the **continue** statement with labels is also supported in Rust. In this example, when x is 1, the program skips to the case when x is 2 and this is achieved by **continue ‘outer3** statement. Likewise, the continue statement is executed when y is 2 with **continue ‘inner3** statement.

Language Evaluation

Dart

Dart was a suitable language for user-located loop control mechanisms. It supported both break and continue statements which is very useful. It was also readable and writable. The use of for-loops or while-loops was easy and similar to known languages like Java. Furthermore, writability was also really good as it did not require any extra symbols like a ":" in Python. Exiting encapsulating loops was also present in Dart making it more flexible for users. Therefore, Dart was my favorite language for user-located loop control mechanisms.

JavaScript

JavaScript also supported unconditional/conditional or labeled/unlabeled exits like Dart. For this matter, the functionality of JavaScript was good. Moreover, it was a readable and a writable language in terms of loop control mechanisms as understanding for/while loops was very easy. Labeling loops was also straightforward which makes the language writable.

Lua

Lua does not support any continue statements or labeling loops. As a result, the language has limited functionality and the users could not exit encapsulating loops which makes the language difficult to use. The alternative go-to statements also make the language less readable as it becomes harder to track where go-to statements direct the program. Also, adding the end key word after a loop makes the language less writable than Dart or JavaScript.

PHP

Although labeled exits are supported in PHP, it is difficult to understand and read as it is done by using numbers. For this matter, PHP was not very readable. Also, the necessity of putting dollar signs before variables in for/while loops decreased the writability of the language. Nevertheless, conditional/unconditional exits and labeled/unlabeled exits with break and continue statements were present in the language which are good attriubutes.

Python

Python was both very readable and writable except for control loop mechanisms except the necessity of putting ":" after a while or a for statement. However, it did not support any labeling of loops like Lua which decreases the flexibility of the language for control loop mechanisms.

Ruby

The extra futures like an **until** loop was really nice for Ruby. Furthermore, the use of for loops was also very easy in Ruby which makes the language extremely writable for control loop mechanisms. The only downside was Ruby also lacked labeled control mechanisms similar to Python or Lua. Nevertheless, it was easy to understand and test loop control mechanisms in Ruby.

Rust

Rust was readable and writable for loop control mechanisms. The use of labels improved the readability and also made possible exiting encapsulating loops. The use of for loops was also really easy which makes it writable. The only problem was the extra **'** sign before loop labels which makes the language a little bit harder to use.

Learning Strategy

For this homework, I started by learning the basic principles of while/for loops of languages like Ruby or Rust as I was more unfamiliar with them. I checked out the type of loops and their implementations. I usually gained these information from the official documentations of the languages. I also learnt the basics of labeled and unlabeled exits and which languages support these type of exits. Furthermore, I researched whether each language has a unique attribute for loops like the use of **next** and **until** in Ruby. Finally, I wrote and compiled the codes with the help of online compilers. When I stuck on a problem, I consulted my friends on whether they faced a similar problem. From time to time, I ran small test programs to understand a concept better or see possible results of a loop.

URLs of Online Compilers

Dart: https://dartpad.dev/?

JavaScript: https://js.do/

Lua: https://www.tutorialspoint.com/execute_lua_online.php

PHP: https://www.w3schools.com/php/phptryit.asp?filename=tryphp_compiler

Python: https://www.programiz.com/python-programming/online-compiler/

Ruby: https://replit.com/languages/ruby

Rust: https://play.rust-lang.or