

Dart

In Dart, I examined both conditional/unconditional and labeled/unlabeled exits. While doing this examination is done for both nested and normal loops.

Code Segment

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

print("-----");

// break
for(var i = 0; i < 5; i++)
{
  print(i);
  break;
}

print("-----");

goto1:
for(var i = 0; i < 5; i++)
{
  print(i);
  break goto1;
}

print("-----");

goto2:
for(var i = 0; i < 5; i++)
{
  for(var j = 0; j < 5; j++)
  {
    print("i = " + i.toString() + " j = " + j.toString());
    if(i == 2)
    {
      break goto2;
    }
  }
}
```

```
i = 0
i = 1
i = 2
i = 3
i = 4
-----
i = 0
-----
i = 0
-----
i = 0 j = 0
i = 0 j = 1
i = 0 j = 2
i = 0 j = 3
i = 0 j = 4
i = 1 j = 0
i = 1 j = 1
i = 1 j = 2
i = 1 j = 3
i = 1 j = 4
```

This code segment shows four different loops. The first one is normal for loop without a break or continue statement. The second loop shows how we exit the loop with the break statement. The third loop shows a labeled exit with one normal loop, and the last loop shows how we exit the nested loop with the labeled exit.

```
// continue
for(var i = 0; i < 5; i++)
{
    print(i);
    continue;
}

print("-----");

goto3:
for(var i = 0; i < 5; i++)
{
    if(i == 2)
    {
        continue goto3;
    }
    print(i);
}

print("-----");
goto4:
for(var i = 0; i < 5; i++)
{
    for(var j = 0; j < 5; j++)
    {
        if(i == 2)
        {
            continue goto4;
        }
        print("i = " + i.toString() + " j = " + j.toString());
    }
}
```

```
i = 0
i = 1
i = 2
i = 3
i = 4
-----
i = 0
i = 1
i = 3
i = 4
-----
i = 0 j = 0
i = 0 j = 1
i = 0 j = 2
i = 0 j = 3
i = 0 j = 4
i = 1 j = 0
i = 1 j = 1
i = 1 j = 2
i = 1 j = 3
i = 1 j = 4
i = 3 j = 0
i = 3 j = 1
i = 3 j = 2
i = 3 j = 3
i = 3 j = 4
i = 4 j = 0
i = 4 j = 1
i = 4 j = 2
i = 4 j = 3
i = 4 j = 4
```

This code segment's functionality is the same as the previous code segment, except this code segment does this with continue statement. In addition, when we used the break statement in nested loop it fully exited and ended the loop, but when we do it with the continue statement it just skips the case in the if statement, and continues after that.

Javascript

In Javascript, I examined both conditional/unconditional and labeled/unlabeled exits. While doing this examination is done for both nested and normal loops.

Code Segment

<pre>for(let i = 0; i < 5; i++) { console.log("i = " + i); }</pre>	i = 0
	i = 1
	i = 2
	i = 3
	i = 4
<pre>console.log("-----")</pre>	-----
<pre>for(let i = 0; i < 5; i++) { console.log("i = " + i); break; }</pre>	i = 0

	i = 0
<pre>console.log("-----")</pre>	-----
<pre>goto1: for(let i = 0; i < 5; i++) { console.log("i = " + i); break goto1; }</pre>	i = 0 j = 0
	i = 0 j = 1
	i = 0 j = 2
	i = 0 j = 3
	i = 0 j = 4
	i = 1 j = 0
	i = 1 j = 1
<pre>console.log("-----")</pre>	i = 1 j = 2
<pre>goto2: for(let i = 0; i < 5; i++) { for(let j = 0; j < 5; j++) { if(i == 3) { break goto2; } console.log("i = " + i + " j = " + j); } }</pre>	i = 1 j = 3
	i = 1 j = 4
	i = 2 j = 0
	i = 2 j = 1
	i = 2 j = 2
	i = 2 j = 3
	i = 2 j = 4

This code segment shows four different loops. First one is normal for loop without break or continue statement. The second loop shows how we exit the loop with the break statement. The third loop shows a labeled exit with one normal loop and the last loop shows how we exit the nested loop with the labeled exit.

```

for(let i = 0; i < 5; i++)
{
    console.log("i = " + i);
    continue;
}

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

goto3:
for(let i = 0; i < 5; i++)
{
    if(i == 2)
    {
        continue goto3;
    }
    console.log("i = " + i);
}

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

goto4:
for(let i = 0; i < 5; i++)
{
    for(let j = 0; j < 5; j++)
    {
        if( i == 3)
        {
            continue goto4;
        }
        console.log("i = " + i + " j = " + j);
    }
}

```

```

i = 0
i = 1
i = 2
i = 3
i = 4
-----
i = 0
i = 1
i = 3
i = 4
-----
i = 0 j = 0
i = 0 j = 1
i = 0 j = 2
i = 0 j = 3
i = 0 j = 4
i = 1 j = 0
i = 1 j = 1
i = 1 j = 2
i = 1 j = 3
i = 1 j = 4
i = 2 j = 0
i = 2 j = 1
i = 2 j = 2
i = 2 j = 3
i = 2 j = 4
i = 4 j = 0
i = 4 j = 1
i = 4 j = 2
i = 4 j = 3
i = 4 j = 4

```

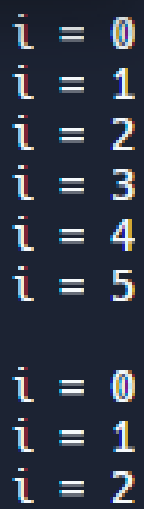
This code segment's functionality is the same with the previous code segment except this code segment does this with continue statement. In addition, when we used break statement in nested loop it fully exited and ended the loop, but when we do it with the continue statement, it just skips the case in the if statement, and continues after that.

Lua

In Lua, we get exit loop by using break statement, but we can only break one loop, not two loop at once, because we do not have labels in Lua. In addition, we do not use the continue statement in Lua, since it does not exist. There is goto statement, but we choose not to use it. In terms of readability and writability I think Lua is good, but using the keyword do and end decreases the writability. If I would use User-Located Loop Control Mechanisms in future, I would not use Lua.

Code Segment

```
1  for i = 0, 5, 1
2  do
3      print("i =", i)
4  end
5
6  print()
7
8  for i = 0, 5, 1
9  do
10     if i == 3 then
11         break
12     end
13     print("i =", i)
14 end
```

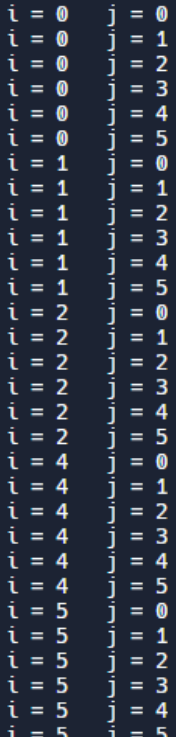


```
i = 0
i = 1
i = 2
i = 3
i = 4
i = 5

i = 0
i = 1
i = 2
```

This code basically does printing in normal for loop to see how loops work, then executes new for loop with break statement to show how the break statement works for exiting the loop.

```
18  for i = 0, 5, 1
19  do
20      for j = 0, 5, 1
21      do
22          if i == 3 then
23              break
24          end
25          print("i =", i, "j =", j)
26      end
27  end
```



```
i = 0 j = 0
i = 0 j = 1
i = 0 j = 2
i = 0 j = 3
i = 0 j = 4
i = 0 j = 5
i = 1 j = 0
i = 1 j = 1
i = 1 j = 2
i = 1 j = 3
i = 1 j = 4
i = 1 j = 5
i = 2 j = 0
i = 2 j = 1
i = 2 j = 2
i = 2 j = 3
i = 2 j = 4
i = 2 j = 5
i = 4 j = 0
i = 4 j = 1
i = 4 j = 2
i = 4 j = 3
i = 4 j = 4
i = 4 j = 5
i = 5 j = 0
i = 5 j = 1
i = 5 j = 2
i = 5 j = 3
i = 5 j = 4
i = 5 j = 5
```

This code segment demonstrates how break statement works with the nested loops in Lua. When i is equal to the 3 the loop breaks and it continues with the i = 4.

PHP

In PHP, I examined both ways of exiting the loop conditional and unconditional. I used break, continue, and labels. In terms of readability and writability I think PHP is good, but using keyword echo and printing new line decreases writability. If I would use User-Located Loop Control Mechanisms in future, I would not use PHP.

Code Segment

<pre>1 <?php 2 for(\$i = 0; \$i < 5; \$i++) 3 { 4 echo ("i = \$i"); 5 echo("\r\n"); 6 } 7 8 echo("-----"); 9 echo("\r\n"); 10 11 for(\$i = 0; \$i < 5; \$i++) 12 { 13 if(\$i == 3) 14 { 15 break; 16 } 17 echo ("i = \$i"); 18 echo("\r\n"); 19 }</pre>	<pre>i = 0 i = 1 i = 2 i = 3 i = 4 ----- i = 0 i = 1 i = 2</pre>
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------

Here I examined normal for loop without break and other for loop is showing how to use break statement in PHP.

<pre>for(\$i = 0; \$i < 5; \$i++) { for(\$j = 0; \$j < 5; \$j++) { if(\$i == 3) { break 2; } echo ("i = \$i, j = \$j "); echo("\r\n"); } }</pre>	<pre>i = 0, j = 0 i = 0, j = 1 i = 0, j = 2 i = 0, j = 3 i = 0, j = 4 i = 1, j = 0 i = 1, j = 1 i = 1, j = 2 i = 1, j = 3 i = 1, j = 4 i = 2, j = 0 i = 2, j = 1 i = 2, j = 2 i = 2, j = 3 i = 2, j = 4</pre>
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Since we do not have labels in PHP, we use level break statement. The integer after break statement shows how many loops will be exited.

```

for($i = 0; $i < 5; $i++)
{
    if($i == 3)
    {
        continue;
    }
    echo "i = $i";
    echo "\r\n";
}

```

```

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

```

```

for($i = 0; $i < 5; $i++)
{
    for($j = 0; $j < 5; $j++)
    {
        if($i == 3)
        {
            continue;
        }
        echo ("i = $i, j = $j ");
        echo("\r\n");
    }
}

```

```

i = 0
i = 1
i = 2
i = 4

```

```

-----
i = 0, j = 0
i = 0, j = 1
i = 0, j = 2
i = 0, j = 3
i = 0, j = 4
i = 1, j = 0
i = 1, j = 1
i = 1, j = 2
i = 1, j = 3
i = 1, j = 4
i = 2, j = 0
i = 2, j = 1
i = 2, j = 2
i = 2, j = 3
i = 2, j = 4
i = 4, j = 0
i = 4, j = 1
i = 4, j = 2
i = 4, j = 3
i = 4, j = 4

```

This code segment shows how continue statement exits the loop.

Python

In Python we use break and continue statements for exiting loop. On the other hand, Python does not have labels or level break statement. In terms of writability and readability Python is one of the best languages. If I would to program loop-based program Python won't be my first choice, because it does not have label.

Code Segment

```
for x in range (10):
    print("x = " , x)

print("//////////")

for x in range (10):
    print("x = " , x)
    break

print("////")

for x in range (10):
    if(x == 5):
        break
    print("x = " , x)
```

```
x = 0
x = 1
x = 2
x = 3
x = 4
x = 5
x = 6
x = 7
x = 8
x = 9
//////////
x = 0
//////////
x = 0
x = 1
x = 2
x = 3
x = 4
```

This code basically shows how for loops executed in Python and how break statement causes code to exit loop.

```
for x in range (5):
    for y in range (5):
        if(x == 3):
            break
        print("x = " + str(x) + ", y = " + str(y) )
```

```
x = 0, y = 0
x = 0, y = 1
x = 0, y = 2
x = 0, y = 3
x = 0, y = 4
x = 1, y = 0
x = 1, y = 1
x = 1, y = 2
x = 1, y = 3
x = 1, y = 4
x = 2, y = 0
x = 2, y = 1
x = 2, y = 2
x = 2, y = 3
x = 2, y = 4
x = 3, y = 0
x = 3, y = 1
x = 3, y = 2
x = 3, y = 3
x = 3, y = 4
```

This code shows how break statement causes to exit the loop.


```

for x in range (10):
    print("x = ", x)
    continue

print("////")

for x in range (10):
    if(x == 2):
        continue
    print("x = ", x)

print("////////////////")

for x in range(10):
    for y in range(10):
        if(x == 4):
            continue
        print("x = " + str(x) + " y = " + str(y) )

```

```

x = 0
x = 1
x = 2
x = 3
x = 4
x = 5
x = 6
x = 7
x = 8
x = 9
////////////////////////////////
x = 0
x = 1
x = 3
x = 4
x = 5
x = 6
x = 7
x = 8
x = 9
////////////////////////////////
x = 0 y = 9
x = 1 y = 9
x = 2 y = 9
x = 3 y = 9
x = 4 y = 9
x = 5 y = 9
x = 6 y = 9
x = 7 y = 9
x = 8 y = 9
x = 9 y = 9

```

This code segment shows how continue statement causes to exit the loop.

Ruby

In Ruby, we do not have labels such as in Python. In addition, in Ruby, there is no continue statement, but there is next statement. In terms of readability and writability Ruby is good. In addition, if I would use User-Located Loop Control Mechanisms in future Ruby won't be my choice, because there is no label.

Code Segment

```
for i in 1..5 do
  puts i
end

puts "-----"

for i in 1..5 do
  puts i
  break
end

puts "-----"

for i in 1..5 do
  if i == 3 then
    break
  end
  puts i
end

puts "-----"

for i in 1..5 do
  for j in 1..5 do
    if i == 3 then
      break
    end
    puts i
  end
end
```

Execution trace:

```
i = 1
i = 2
i = 3
i = 4
i = 5
-----
i = 1
-----
i = 1
i = 2
-----
i = 1, j = 1
i = 1, j = 2
i = 1, j = 3
i = 1, j = 4
i = 1, j = 5
i = 2, j = 1
i = 2, j = 2
i = 2, j = 3
i = 2, j = 4
i = 2, j = 5
i = 4, j = 1
i = 4, j = 2
i = 4, j = 3
i = 4, j = 4
i = 4, j = 5
i = 5, j = 1
i = 5, j = 2
i = 5, j = 3
i = 5, j = 4
i = 5, j = 5
```

This code segment shows how normal for loop works in Ruby and after that it shows how break statement exits the loop. As we can see in nested loop break statement exits both loops.

```

for i in 1..5 do
  if i == 2 then
    next
  end
  puts i
end

puts "-----"

for i in 1..5 do
  for j in 1..5 do
    if i == 3 then
      next
    end
    puts "i = #{i}, j = #{j}"
  end
end

```

```

i = 1
i = 3
i = 4
i = 5
-----
i = 1, j = 1
i = 1, j = 2
i = 1, j = 3
i = 1, j = 4
i = 1, j = 5
i = 2, j = 1
i = 2, j = 2
i = 2, j = 3
i = 2, j = 4
i = 2, j = 5
i = 4, j = 1
i = 4, j = 2
i = 4, j = 3
i = 4, j = 4
i = 4, j = 5
i = 5, j = 1
i = 5, j = 2
i = 5, j = 3
i = 5, j = 4
i = 5, j = 5

```

This code segment shows how next (continue) statement works in Ruby.

Rust

In Rust, we have all break, continue, and labels. In this homework, conditional/unconditional and labeled/unlabeled exit is examined for Ruby. In terms of readability and writability, Rust is not a good language, because printing string with integers are not good. If I would use User-Located Loop Control Mechanisms in the future, Rust can be my choice.

Code Segment

```
for i in 1..10{
    print!("i = {}", i);
}

println!("\n");

for i in 1..10{
    print!("i = {}", i);
    if i == 6
    {
        break;
    }
}

println!("\n");

for i in 1..5{
    for j in 1..5{
        if i == 3
        {
            break;
        }
        println!("i = {}, j = {}", i, j);
    }
}
```

i = 1, i = 2, i = 3, i = 4, i = 5, i = 6, i = 7, i = 8, i = 9,

i = 1, i = 2, i = 3, i = 4, i = 5, i = 6,

i = 1, j = 1
i = 1, j = 2
i = 1, j = 3
i = 1, j = 4
i = 2, j = 1
i = 2, j = 2
i = 2, j = 3
i = 2, j = 4
i = 4, j = 1
i = 4, j = 2
i = 4, j = 3
i = 4, j = 4

This code segment shows how normal for loop prints and how break statement exits the normal loop and nested loop. As we can see from the second loop when i is equal to the 6, it exits the loop, and when we use it with the nested loop it exists the inner and outer loop.

```

'goto1: loop
{
    for i in 1..5{
        if i == 3{
            break 'goto1;
        }
        print!("i = {}", i);
    }

    println!("\n");

'goto2: loop
{
    for i in 1..5{
        for j in 1..5{
            if i == 3{
                break 'goto2;
            }
            println!("i = {}, j = {}", i, j);
        }
    }
}

```

i = 1, i = 2,
i = 1, j = 1
i = 1, j = 2
i = 1, j = 3
i = 1, j = 4
i = 2, j = 1
i = 2, j = 2
i = 2, j = 3
i = 2, j = 4

This code segment shows how labeled exit works in Rust. It shows that when we use labels with one loop and nested loops it exits fully from the loop(s).

```

for i in 1..5{
    if i == 3{
        continue;
    }
    print!("i = {}", i);
}

println!("\n");

'goto3:
for i in 1..5{
    if i == 3{
        continue 'goto3;
    }
    print!("i = {}", i);
}

println!("\n");

'goto4:
for i in 1..5{
    for j in 1..5{
        if i == 3{
            continue 'goto4;
        }
        println!("i = {}, j = {}", i, j);
    }
}

```

i = 1, i = 2, i = 4,
i = 1, i = 2, i = 4,
i = 1, j = 1
i = 1, j = 2
i = 1, j = 3
i = 1, j = 4
i = 2, j = 1
i = 2, j = 2
i = 2, j = 3
i = 2, j = 4
i = 4, j = 1
i = 4, j = 2
i = 4, j = 3
i = 4, j = 4

This code segment has the same functionality with previous code segment except this code segment does this with continue statement.

Evaluation of Languages

As I did the evaluation for all languages in their part, in this part, why Javascript is the best language for User-Located Loop Control Mechanisms. Java is our mother language, and Javascript's loops and printing statements are similar to Java, so it increases the writability and readability. In addition to these, since we have labels, it is very easy to use User-Located Loop Control Mechanisms.

Learning Strategy

For this homework, I mainly used online sources such as W3Schools, Tutorialspoint, Javatpoint, GeeksForGeeks, and Youtube. In addition to these, I used official documentation of the languages. My learning strategy was first to understand the assignment and understand the requirements of the assignment. Firstly, I looked for languages' loop mechanisms, and then I tried to do normal printing in the for loop. Then I searched for how to use break statement and its functionality. After that, I did the same thing for continue statement. When I figured out all these, I tested these with labels and nested loops. When I've done with this as well, the homework was done. The online compilers that I've used:

Dart: <https://replit.com/languages/dart>

JavaScript: I wrote the code in VSCode and tested it in HTML file.

Lua: <https://replit.com/languages/lua>

PHP: <https://paiza.io/en/projects/new>

Pyhton: I wrote the code in VSCode and tested it with Python compiler.

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

Rust: <https://play.rust-lang.org/>