CS315

HW2

Elifsena ÖZ

22002245

Section 1

1. Conditional and Unconditional Exit

This section includes code segments of conditionally and unconditionally exited loops for Dart, JavaScript, PHP, Python, Ruby, and Rust. For each language the code is designed to give the same output. In the first loop I takes numbers from 0 to 4 inclusive and all values of i are printed. The second loop is the same except for the break statemen in it. When i is 3 the loop exits unconditionally.

1.1 Dart

In this code segment break is used to exit the loop independent of the loops exit condition.

```dart
  // This aims to print numbers from 0 to 4 inclusive
  print("\nConditional");
  for (int i = 0; i <=4; i++) {
    print("i: $i");
  }

  // This aims to print numbers from 0 to 4 inclusive
  print("\nUnconditional");
  for (int i = 0; i <= 4; i++) {
    print("i: $i");

    if (i == 3) {
      print("Exit loop");
      break;  // When i is 3 the loops is exited
    }
  }
```

Output:

```
Conditional
i: 0
i: 1
i: 2
i: 3
i: 4

Unconditional
i: 0
i: 1
i: 2
i: 3
Exit loop
```

1.2 JavaScript

```javascript
console.log("\nConditional");
    for (let i = 0; i <=4; i++) {
        console.log("i: " + i);
```

```
    }

    console.log("\nUnconditional");
    for (let i = 0; i <= 4; i++) {
        console.log("i: " + i);

        if (i == 3) {
            console.log("Exit loop");
            break;
        }
    }
```

Output:

```
Conditional
i: 0
i: 1
i: 2
i: 3
i: 4

Unconditional
i: 0
i: 1
i: 2
i: 3
Exit loop
```

1.3 Lua

```
print("\nConditional")
for i = 0,4,1 do
    print("i: ", i)
end

print("\nUnconditional")
i = 0
repeat
    print("i: ", i)
    i = i + 1
    if (i == 4) then
        break
    end
until Unconditional
```

Output:

```
Conditional
```

```
i:      0
i:      1
i:      2
i:      3
i:      4

Unconditional
i:      0
i:      1
i:      2
i:      3
```

1.4 Python

```python
print("\nConditional")
for i in range(0,5):
    print("i: " + str(i))


print("\nUnconditional")
for i in range(0,5):
    if i == 4:
        print("Exit Loop")
        break
    print("i: " + str(i))
```

Output:

```
Conditional
i: 0
i: 1
i: 2
i: 3
i: 4

Unconditional
i: 0
i: 1
i: 2
i: 3
Exit Loop
```

1.5 Ruby

```ruby
puts("\nConditional")
for i in 0..4 do
```

```
    print("i: ")
    puts(i)
end

puts("\nUnconditional")
for i in 0..4 do
    print("i: ")
    puts(i)

    if i == 3 then
      puts("Exit loop")
      break;
    end
end
```

Output:

```
Conditional
i: 0
i: 1
i: 2
i: 3
i: 4

Unconditional
i: 0
i: 1
i: 2
i: 3
Exit loop
```

1.6 Rust

```
    println!("\nConditional");
    for i in 0..5 {
        println!("i: { }", i);
    }

    println!("\nUnconditional");
    for i in 0..5 {
        println!("i: { }", i);
        if i == 3 {
            println!("Exit Loop");
            break
        }
    }
```

Output:

```
Conditional
i: 0
i: 1
i: 2
i: 3
i: 4

Unconditional
i: 0
i: 1
i: 2
i: 3
Exit Loop
```

2. Labeled and Unlabeled Exit

This section includes code segments of labeled and unlabeled exited loops for Dart, JavaScript, PHP, Python, Ruby, and Rust. For each language the code is designed to give the same output. First, 2 nested loops are given with a break statement in the inner loop. Outer loop aims to print i from 0 to 2 inclusive. The inner loop aims to print j from 3 to 4 inclusive. The inner loop exits with a unconditional break statement when j is 4, so inner loop can only print 3 as a value of j. Secondly, there are two nested loops that work in the same way as the first example. If the language contains labeling both loops are exited using the label.

2.1 Dart

Dart allows labeling loops. Therefore, labeledLoop (the outer loop) is exited by using break with label.

```
  print("\nUnlabeled");
  for (int i = 0; i < 3; i++) { // This aims to print numbers from 0 to 2
inclusive
    print("i: $i");
    for (int j = 3; j <= 4; j++) { // This aims to print numbers from 3 to 4
inclusive
      if (j == 4) {
        break; // j is 4  // When j is 4 the inner loop is exited
      }
      print("j: $j");
    }
  }

  print("\nLabeled");
  labeledLoop: for (int i = 0; i < 3; i++) { // This aims to print numbers
from 0 to 2 inclusive
    print("i: $i");
```

```
    for (int j = 3; j <= 4; j++) { // This aims to print numbers from 3 to 4
inclusive
      if (j == 4) {
        break labeledLoop; // j is 4 // When j is 4 the outer loop
(labeledLoop) is exited
    }
    print("j: $j");
    }
  }
```

Output:

```
Unlabeled
i: 0
j: 3
i: 1
j: 3
i: 2
j: 3

Labeled
i: 0
j: 3
```

2.2 JavaScript

Javascript allows labeling loops. Therefore, labeledLoop (the outer loop) is exited by using break with label.

```
console.log("\nUnlabeled");
    for (let i = 0; i < 3; i++) {
        console.log("i: " + i)
        for (let j = 3; j <= 4; j++) {
            if (j == 4) {
                break; // j is 4
            }
            console.log("j: " + j);
        }
    }

    console.log("\nLabeled");
    labeledLoop: for (let i = 0; i < 3; i++) {
        console.log("i: " + i)
        for (let j = 3; j <= 4; j++) {
            if (j == 4) {
                break labeledLoop; // j is 4
            }
            console.log("j: " + j);
```

```
        }
    }
```

Output:

```
Unlabeled
i: 0
j: 3
i: 1
j: 3
i: 2
j: 3

Labeled
i: 0
j: 3
```

2.3 Lua

Lua allows labeling. Therefore, endLabel is used with goto keyword to exit loops and execute the code that comes after.

```
for i = 0,3,1 do
    print("i: ", i)
    for j = 3,4,1 do
        if (j == 4) then
            break
        end
        print("j: ", j)
    end
end

print("\nLabeled")
for i = 0,3,1 do
    print("i: ", i)
    for j = 3,4,1 do
        if (j == 4) then
            goto endLabel
        end
        print("j: ", j)
    end
end
:: endLabel ::
```

Output:

```
Unlabeled
i:      0
```

```
j:      3
i:      1
j:      3
i:      2
j:      3
i:      3
j:      3

Labeled
i:      0
j:      3
```

2.4 Python
Python does not have labeled loops.

2.5 Ruby
Ruby does not have labeled loops.

2.6 Rust
Rust allows labeling loops. Therefore, ‘labeledLoop (the outer loop) is exited by using break with label.

```
    println!("\nUnlabeled");
    for i in 0..3 {
        println!("i: { }", i);
        for j in 3..5 {
            if j == 4 {
                break;
            }
            println!("j: { }", j);
        }
    }

    println!("\nLabeled");
    'labeledLoop: for i in 0..3 {
        println!("i: { }", i);
        for j in 3..5 {
            if j == 4 {
                break 'labeledLoop;
            }
            println!("j: { }", j);
        }
    }
```

Output:

```
Unlabeled
```

```
i: 0
j: 3
i: 1
j: 3
i: 2
j: 3

Labeled
i: 0
j: 3
```

2. Language Evaluation

2.1 Dart

Unconditional exit statements are easy to use in Dart. The break statement is a commonly used version of unconditional exits which makes the language easy to understand increasing is readability. Although they should not be used unnecessarily break statements can be life savers in certain situations, increasing Dart's writability. In dart loops can be labeled. It is similar to naming something and having a definition that comes after the name. It is an easily used control mechanism. Overall, Dart is a readable and writeable language in terms of loop control mechanisms.

2.2 JavaScript

JavaScript allows breaks and labeled loops in the same way as Dart. As mentioned before, I believe that these are commonly used and usual ways to do it. Therefore, they are easy in terms of readability and writability.

2.3 Lua

In Lua break statements exist and are easy to use just like Dart and JavaScript. But labeling loops is not an option in this language. However general line labels can be used with goto exit a loop by labeling the line after the loop. I do not believe that this way is conventional. In terms of readability, one would have to read couple or maybe more lines ahead to find the label. In terms of writability loop label relations can be confused and may result in bugs.

2.4 PHP

PHP is a readable and writeable language in terms of conditional and unconditional exits. The break statement is conventional and easy to use, similar to the three languages described above. However, labeled exits are the same es they are in Lua. As mentioned above these conventions can cause many problems in terms of understanding and debugging the code. Therefore, PHP is not very readable and writeable in labeled loop exits.

2.5 Pyhton

Unconditionally exiting a loop with break statement in Python is standard, and easy. Again, it is the same with all languages described above. Python does not have labeled loops or lines

therefore loop control mechanism is deficient compared to other languages. I believe this does not create any issues in terms of writability. Overall Python is an easily readable and writeable language.

2.6 Ruby

Unconditional exit is the same as it is in all other languages mentioned above in this report. Break statement is used to exit the loop unconditionally. It is very convenient to use. Labeled loops do not exist on Ruby.

2.7 Rust

Rust has break statements for unconditional exits. As mentioned above they are nice and straightforward just like other languages. Labeled loops exist on rust and loop labels need to have a single quotation mark in front of them. I believe this convention makes them easy to realize and distinguishable. I summary, it is a highly readable and writeable language.

I believe rust is the best language in terms of the User-Located Loop Control Mechanisms. Because the loops labels can be identified from other declarations. The single quotation mark highly increases its readability and writability compared to other good languages like Dart and JavaScript.

3. Learning Mechanism

I have researched the loop mechanisms, labels and break statements as well as printing and if-else statements for each language. For most languages I have checked the articles in GeeksForGeeks instead of the documentations. I found their explanation easier to understand and follow. Seeing code examples really help me understand the syntax of language. However, I have also used documentation when needed. The learning process involved a lot of trial and error until I found the right ways. I used online compilers for Dart, PHP, Rust and Ruby. I had compilers for Python and Lua on my own computer. For JavaScript code, I have created a separate JavaScript file in my own computer and compiled it using node. When I was satisfied with the output, I have pasted the JavaScript code into the html file.

The online compilers I have used:

- https://www.tutorialspoint.com/compile_rust_online.php
- https://www.tutorialspoint.com/execute_ruby_online.php
- https://www.tutorialspoint.com/execute_php_online.php
- https://dartpad.dev/?

My sources for learning syntax and other details:

- https://www.geeksforgeeks.org/
- https://www.ruby-lang.org/en/documentation/
- https://www.rust-lang.org/learn