CS315

Homework Assignment 2

Fall 2022

Zeynep Doğa Dellal 22002572 Section 3

Instructor: Karani Kardaş

Teaching Assistants:

Dilruba Sultan Haliloğlu, Hamza İslam, Onur Yıldırım

First picture is the input code segment and the second picture is the output in the samples.

1. DART

a. Should the conditional mechanism be an integral part of the exit (conditional or unconditional exit)?

Continue function in DART does not do anything without a condition. **Break** function terminates the function in the first execution. Without condition, exit mechanisms are not very necessary. Conditional exit example is given in the first example of part b. Continue operation skips a specific step or steps and break exits the loop depending on the operation.

```
print("--UNCONDITIONAL CONTINUE--");
for (var i = 0; i < 5; i++) {
  print(i);
  continue;
}

print("--UNCONDITIONAL EXIT--");
for (var i = 1; i < 5; i++) {
  print(i);
  break;
}
```

```
--UNCONDITIONAL CONTINUE--
0
1
2
3
4
--UNCONDITIONAL EXIT--
1
```

b. Should only one loop body be exited or can enclosing loops also be exited (labeled or unlabeled exit)?

Example below both shows **continue** and **break** for unlabeled exit. It is also an example of conditional exit for part a. Without label, **continue** and **break** functions affect the loop right before that, therefore if user wanted to exit outer loop inside the inner loop, they could not.

```
print("--UNLABELED CONTINUE--");
for (var i = 0; i < 5; i++) {
  if (i == 3) {
    continue;
  }
  print(i);
}

print("--UNLABELED BREAK--");
var i = 0;
while (i < 5) {
  i++;
  print(i);
  if (i == 3) break;
}
```

```
--UNLABELED CONTINUE--
0
1
2
4
--UNLABELED BREAK--
1
2
3
```

Example below shows **continue** and **break** for labeled exit. Labeled break in DART allows user to exit or jump from a specific loop in nested loops. Labeled continue allows user to skip steps of a specific loop in nested loops.

```
print("--LABELED CONTINUE--");
labelforcontinue:
for (var i = 0; i <= 5; i++) {
  for (var j = 0; j < 3; j++) {
    if (i == j) {
      continue labelforcontinue;
    }
  }
  print(i);
}

print("--LABELED BREAK--");
var a = 1;
exitouterloop:
while (a < 5) {
  while (a > 0) {
    print(a);
    if (a == 3) break exitouterloop;
    a++;
  }
}
```

```
--LABELED CONTINUE--
3
4
5
--LABELED BREAK--
1
2
3
```

c. Evaluation of DART Language

In terms of readability DART language is sufficient enough because it requires "{}" characters to determine the scope. It also has English characters for break and continue which are understandable. DART language is a writable language because it doesn't require unnecessary declarations and it has the necessary functions to use loops in an efficient and easy way.

2. JAVASCRIPT

a. Should the conditional mechanism be an integral part of the exit (conditional or unconditional exit)?

Continue function in Javascript does not do anything without a condition. **Break** function terminates the function in the first execution. Like in DART language, without condition exit mechanisms are not very necessary. In the first example of part b conditional exit is shown.

```
console.log("--UNCONDITIONAL CONTINUE--");
for (var i = 1; i < 5; i++) {
  console.log(i);
  continue;
}

console.log("--UNCONDITIONAL BREAK--");
for (var i = 1; i < 5; i++) {
  console.log(i);
  break;
}
```

```
--UNCONDITIONAL CONTINUE--
1
2
3
4
--UNCONDITIONAL BREAK--
1
```

b. Should only one loop body be exited or can enclosing loops also be exited (labeled or unlabeled exit)?

Example below both shows **continue** and **break** for unlabeled exit. It is also an example of conditional exit for part a. Unlabeled exit mechanisms only affect the first loop they are in and do not affect the ones before them.

```
console.log("--UNLABELED CONTINUE--");
for (var i = 0; i < 5; i++) {
  if (i == 3) {
    continue;
  }
  console.log(i);
}

console.log("--UNLABELED BREAK--");
var i = 0;
while (i < 5) {
  i++;
  console.log(i);
  if (i == 3) break;
}
```

```
--UNLABELED BREAK--
1
2
3
--LABELED CONTINUE--
3
4
5
```

Example below shows **continue** and **break** for labeled exit. Labeled break in Javascript allows user to exit or jump from a specific loop in nested loops. Labeled continue allows user to skip steps of a specific loop in nested loops.

```
console.log("--LABELED CONTINUE--");
labelforcontinue:
for (var i = 0; i <= 5; i++) {
  for (var j = 0; j < 3; j++) {
    if (i == j) {
      continue labelforcontinue;
    }
  }
  console.log(i);
}

console.log("--LABELED BREAK--");
var a = 1;
exitouterloop:
while (a < 5) {
  while (a > 0) {
      console.log(a);
    if (a == 3) break exitouterloop;
    a++;
  }
```

```
--LABELED CONTINUE--
3
4
5
--LABELED BREAK--
1
2
3
```

c. Evaluation of Javascript language

In terms of readability Javascript is sufficient enough because it requires "{}" characters to determine the scope. It also has English characters for break and continue which are understandable. Javascript language is a writable language because it doesn't require unnecessary declarations and it has the necessary functions to use loops in an efficient and easy way. It also supports incrementing variables using "++".

3. Lua

a. Should the conditional mechanism be an integral part of the exit (conditional or unconditional exit)?

Lua does not have **continue** function directly but it has **goto** function that could be used to work around it. **Break** function terminates the function in the first execution.

```
print("--UNCONDITIONAL CONTINUE--")
for i=1,5 do
  print(i)
  goto continuegoto
  ::continuegoto::
end
print("--UNCONDITIONAL BREAK--")
for i=1,5 do
  print(i)
  break
end
```

```
--UNCONDITIONAL CONTINUE--
1
2
3
4
5
--UNCONDITIONAL BREAK--
1
```

b. Should only one loop body be exited or can enclosing loops also be exited (labeled or unlabeled exit)?

```
local i  = 1;
local a = 1;
print("--UNLABELED BREAK--")
while i < 5 do
    if(i == 4) then break end
    print( i )
    i = i+1
end

print("--GOTO CONTINUE--");
for i=1,5 do
  if i == 3 then goto continuegoto end
  print(i)
  ::continuegoto::
end
```

```
--UNLABELED BREAK--
1
2
3--GOTO CONTINUE--
1
2
4
5
```

Example above shows unlabeled **break** and **continue** with goto in Lua. These two examples can also be an example for conditional exit for part a.

Lua supports goto break which can be used as labeled exit. In the example shown below break operation terminates the outer loop.

```
print("--GOTO BREAK--")
while a < 5 do
    while a > 0 do
        if(a == 4) then goto breakgoto end
        print( a )
        a = a+1
    end
end
::breakgoto::
```

```
--GOTO BREAK--
1
2
3
```

c. Evaluation of Lua Language

In terms of readability Lua does not contain characters such as “()” and “{}” which makes it harder to read. It also requires the keyword goto for some operations and usage and the name of goto is confusing. In terms of writability Lua does not require extra symbols other than the declaration of goto and usage of loops are simpler than most of the programming languages. That is why Lua is an easily writable language but it is not a easily readable language.

4. PHP

a. Should the conditional mechanism be an integral part of the exit (conditional or unconditional exit)?

Continue function in PHP language does not do anything without a condition. **Break** function terminates the function in the first execution. In conditional exit operations continue operation skips a specific step or steps and break exits the loop depending on the operation.

```
echo "--UNCONDITIONAL BREAK-- <br>";
for($i = 1; $i < 5; $i++) {
  echo "i: $i <br>";
  break;
}

echo "--UNCONDITIONAL BREAK-- <br>";
for($i = 1; $i < 5; $i++) {
  echo "i: $i <br>";
  continue;
}
```

```
--UNCONDITIONAL BREAK--
i: 1
--UNCONDITIONAL BREAK--
i: 1
i: 2
i: 3
i: 4
```

b. Should only one loop body be exited or can enclosing loops also be exited (labeled or unlabeled exit)?

Example below both shows **continue** and **break** for unlabeled exit. It is also an example of conditional exit for part a. Unlabeled loop does not allow user to exit the chosen loop in enclosing loops in PHP.

```
$i = 1;
$a = 1;
echo "--UNLABELED BREAK-- <br>";
while($i < 5) {
  echo "i: $i <br>";
  $i++;
  if ($i == 4) break;
}
$i = 0;
echo "--UNLABELED CONTINUE-- <br>";
while($i < 5) {
  $i++;
  if ($i == 3) continue;
  echo "i: $i <br>";
}
```

--UNLABELED BREAK--
i: 1
i: 2
i: 3
--UNLABELED CONTINUE--
i: 1
i: 2
i: 4
i: 5

In PHP there are no labels but it is possible to exit chosen loop by using numbers. This system is efficient and enables users to escape or skip loops in enclosing loops but number system could be confusing if there are many loops enclosed.

```
$i = 0;
$a = 0;
echo "--BREAK FROM OUTER LOOP-- <br>";
while($i < 5) {
 echo "";
    while($a < 5){
      $a++;
      $i++;
      if ($a == 3) {
          break 2;
      }
      echo "i in outer loop: $i , a in inner loop: $a <br>";
    }
  }

$i = 0;
$a = 0;
echo "--CONTINUE FROM OUTER LOOP-- <br>";
while($i < 5) {
    while($a < 5){
      $a++;
      $i++;
      if ($a == 3) {
          continue 2;
      }
      echo "i in outer loop: $i , a in inner loop: $a <br>";
    }
  }
```

--BREAK FROM OUTER LOOP--
i in outer loop: 1 , a in inner loop: 1
i in outer loop: 2 , a in inner loop: 2
--CONTINUE FROM OUTER LOOP--
i in outer loop: 1 , a in inner loop: 1
i in outer loop: 2 , a in inner loop: 2
i in outer loop: 4 , a in inner loop: 4
i in outer loop: 5 , a in inner loop: 5

c. Evaluation of PHP Language

In terms of readability PHP is sufficient enough. It does not require extra symbols but uses curly brackets that make indicating the scope of the loops easier. In terms of writability, PHP is the best programming language. Usage of numbers instead of labels make it easy to write. Only part where PHP is not writable enough is declaring variables. Using dollar sign makes this language harder to write. The number system instead of labels makes PHP loop control systems confusing.

5. PYTHON

a. Should the conditional mechanism be an integral part of the exit (conditional or unconditional exit)?

Unconditional **continue** in Python does not do anything. **Break** exits the loop after the first execution without the condition.

```
print("--UNCONDITIONAL CONTINUE--")
for i in range(1, 5):
    print(i)
    continue

print("--UNCONDITIONAL BREAK--")
for i in range(1, 5):
    print(i)
    break
```

```
--UNCONDITIONAL CONTINUE--
1
2
3
4
--UNCONDITIONAL BREAK--
1
```

Conditional **continue** in Python skips a step or steps of the loop. **Break** exits the loop depending on the condition. In the below example for break, execution stops when there is a “ ” and for continue, it skips the “ ” parts of the sentence. The below examples are a way to show that Python supports looping through a sentence letter by letter.

```
print("--CONDITIONAL BREAK--")
word = "CS315 HW 2"
for letter in word:
    if letter == " ":
        break
    print(letter)

print("--CONDITIONAL CONTINUE--")
word = "CS315 HW 2"
for letter in word:
    if letter == " ":
        continue
    print(letter)
```

```
--CONDITIONAL BREAK--
C
S
3
1
5
--CONDITIONAL CONTINUE--
C
S
3
1
5
H
W
2
```

b. Should only one loop body be exited or can enclosing loops also be exited (labeled or unlabeled exit)?

Python does not support labeled exit. Above examples in the part a are unlabeled examples. In order to exit a certain loop, break or continue could be used inside that loop but outside any loops it might contain.

c. Evaluation of Python Language

In terms of readability Python is clear and understandable with the usage of words such as range. It is also sufficient because scope of the loops can be determined by the index of the loops. In terms of writability Python is very easy to understand and write. It lifts the requirements such as "()" and "{}" characters but still makes it clear where the loop starts with the usage of ":".

6. Ruby

a. Should the conditional mechanism be an integral part of the exit (conditional or unconditional exit)?

Continue is called with the next keyword in Ruby. Unconditional continue in Ruby does not do anything. **Break** exits the loop after the first execution without the condition.

```
puts "--UNCONDITIONAL BREAK--";
i  = 1;
for i in 1..5 do
  puts i;
  i = i + 1;
  break;
end

puts "--UNCONDITIONAL CONTINUE--";
i  = 1;
for i in 1..5 do
  puts i;
  i = i + 1;
  next;
end
```

```
--UNCONDITIONAL BREAK--
1
--UNCONDITIONAL CONTINUE--
1
2
3
4
5
```

b. Should only one loop body be exited or can enclosing loops also be exited (labeled or unlabeled exit)?

Below examples show conditional exit for part a. Unlabeled **break** and **continue** exits the loop that it is in. The first loop before the line where break exist is the loop break and continue affects.

```
puts "--UNLABELED BREAK--";
i  = 1;
a = 1;
while i < 5 do
    if i == 4 then break end
    puts i;
    i = i+1;
  end

puts "--UNLABELED CONTINUE--";
i  = 0;
a = 0;
while i < 5 do
    i = i+1;
    if i == 3 then next end
    puts i;
  end
```

```
--UNLABELED BREAK--
1
2
3
--UNLABELED CONTINUE--
1
2
4
5
```

Ruby does not support labeled exit. Above examples in the part a are unlabeled examples. In order to exit a certain loop, break or continue could be used inside that loop but outside any loops it might contain.

c. Evaluation of Ruby Language

Ruby language is readable because rules of declaring loops is clear, showing range by using two dots makes the language easy to read. It does not require symbols for range but it has do and end keywords for indicating scope which makes it more readable compared to languages that use symbols such as “{}”. In terms of writability Ruby lifts variable declaration keywords, extra symbols and makes the language mostly consist of words which makes it easier to write. In Ruby, we can observe that syntax of loops is similar to primary languages but it was changed in a way that it is easy to write.

7. Rust

a. Should the conditional mechanism be an integral part of the exit (conditional or unconditional exit)?

Break operation without a condition terminates the loop in its first execution. **Continue** without a condition does not do anything.

```
println!("UNCONDITIONAL BREAK");
let mut i = 1;
while i < 5 {
    println!("{}", i);
    i = i + 1;
    break;
}

println!("UNCONDITIONAL CONTINUE");
i = 1;
while i < 5 {
    println!("{}", i);
    i = i + 1;
    continue;
}
```

```
UNCONDITIONAL BREAK
1
UNCONDITIONAL CONTINUE
1
2
3
4
```

Continue operation skips a step or steps depending on the condition but does not terminate the loop. **Break** immediately exits the loop.

```
println!("CONDITIONAL CONTINUE");
i = 1;
while i < 5 {
    i = i + 1;
    if i == 3 {continue;}
    println!("{}", i);
}

println!("CONDITIONAL BREAK");
i = 1;
while i < 5 {
    i = i + 1;
    if i == 3 {break;}
    println!("{}", i);
}
```

```
CONDITIONAL CONTINUE
2
4
5
CONDITIONAL BREAK
2
```

b. Should only one loop body be exited or can enclosing loops also be exited (labeled or unlabeled exit)?

Unlabeled exit operations operate on the loop right before they are written. Therefore they are not useful for escaping outer loop inside the inner loop.

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

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

```
UNLABELED BREAK
i is 1
i is 2
i is 3
i is 4
UNLABELED CONTINUE
i is 1
i is 2
i is 3
i is 4
```

Rust language supports usage of labels. Labels enable user to jump steps or terminate a certain loop in nested loops. In the below example, for continue, label makes it possible to skip a step in outer loop inside the inner loop. For break example, outer loop terminates when the loop variables are equal.

```
println!("LABELED BREAK");
'label:for i in 1..8{
    println!("i is {}",i);
    for j in 3..8{
      if i==j {
        break 'label;
      }
  }
}

println!("LABELED CONTINUE");
'labelforcontinue:for i in 1..5{
    for j in 1..5{
      if i==3 {
        continue 'labelforcontinue;
      }
  }
  println!("i is {}",i);
}
```

```
LABELED BREAK
i is 1
i is 2
i is 3
LABELED CONTINUE
i is 1
i is 2
i is 4
```

c. Evaluation of Rust Language

In terms of readability Rust language provides clarity with symbols that determine scope, range declaration using “..” and English names for operations. In terms of writability, Rust includes unnecessary

symbols such as " ' " in front of the label, "{}" for showing a variable while printing and it does not support incrementation using "++". That is why Rust isn't a easily writable language.

8. A General Evaluation of All Languages

DART, Javascript and Rust supports usage of labels, Lua does not support labels but it has a goto function. PHP has number system that enables escaping enclosed loops. Python and Ruby does not support label system. DART is the best language for using User-Located Loop Control Mechanisms. It is easily readable and writable, supports labels, does not require unnecessary symbols or extra words but makes the scope of the loops clear. In my opinion worst language for User-Located Loop Control Mechanisms is Lua. Goto function is very confusing. Even though it could provide what labels could, it is so confusing to use that this extra function is not understandable.

To give a general answer to the questions in the homework, conditional mechanism should be an integral part of loops if the usage of break and continue is mandatory. Because unconditional usage makes the whole break-continue system unnecessary. On the other hand, labeled usage is not necessary. Most languages make scope of the loops clear. The break and continue statements can be put between the loops in a way that only affects the loop that is wanted to be affected.

9. Learning Strategy

I started from DART language and looked into how to use for and while loops, how to declare variables and print out the outputs. After learning these I searched whether it allows usage of labels. I applied this strategy to all of the languages one by one. I checked whether they support usages similar to labels. Learning some of the languages was harder. For example, Rust language is useful than most languages in terms of loops but it requires a lot of extra symbols and words therefore it was harder to learn.

Here are the list of sources I used to learn and test the given languages:

1- Official site of DART: https://api.dart.dev/stable/2.10.4/index.html

2- Official site of Javascript: https://developer.mozilla.org/en-US/docs/Web/JavaScript

3- Official site of Lua: https://www.lua.org/

4- Official site of PHP: https://www.php.net/docs.php

5- Official site of Python: https://docs.python.org/3/

6- Official site of Ruby: https://www.ruby-lang.org/en/documentation/

7- Official site of Rust: https://doc.rust-lang.org/

8- Rust compiler: https://play.rust-lang.org/

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

10- Lua compiler: https://www.tutorialspoint.com/execute_lua_online.php

11- Helpful tutorial site I used: https://www.tutorialspoint.com/index.htm

12- Helpful site I used to clear errors: https://stackoverflow.com/

13- Another helpful site I used while learning: https://www.geeksforgeeks.org/

14- Coding environment I used for DART, Javascript, Ruby: https://code.visualstudio.com/

15- Coding environment I used for Python: https://www.jetbrains.com/pycharm/

10. Communication

I consulted my friends about the homework assignment because the given documentation was ambiguous. We tried to understand the assignment and decided on how to do it together but I did the whole homework by myself. Slack was a big communication channel that was very helpful in terms of understanding the assignment. I also consulted StackOverflow and similar sites for help.