Bilkent University
Fall 2022-2023

CS315 Programming Languages

Homework 2
User-Located Loop Control Mechanisms

Section 2

Deniz Tuna Onguner
22001788

December 3, Saturday

CONTENTS

1 User-Located Loop Control Mechanisms

1.1 Python (3.9)

In Python, there are two loop control mechanisms which are **break** and **continue.** Both them have unlabeled design, which means they cannot be used to control an entire block of nested loops. Both of them are also unconditional mechanisms, no kind of a condition is required to control a loop with them. So that, break and continue can be used to control conditional loops without conditioning. Moreover, break can also be used to exit an infinite loop.

Example 1:

```
var = 0
while True:
    var += 1
    if var >= 10:
        break
```

[Output]: 10

In the code block left, an infinite loop is given, but the loop will exit when break (line 5) is executed.

Example 2:

```
var = 0
while True:
    var += 1
    break
```

[Output]: 1

It is not a necessity to use break with a conditional block like in Example 1. The code right is also valid.

Example 3:

```
var = 0
while True:
    while True:
        var += 1
        if var >= 20:
            break
    if var >= 20:
        break
```

[Output]: 20

Loop control mechanism of Python are unlabeled. Which means the one must apply multiple control statements in order to control a nested loop like in the given example left. Such an implementation, on the other hand, requires self-repeating code. This design choice can be considered as an issue.

Example 4:

```
var = 0
for i in range(10):
    if i == 5:
        continue
    var += i
```

[Output]: 40

continue has also exact same design with break in terms of the way they should be used. Again, no condition is required and labels are not allowed.

Example 5:

```
var = 0
for i in range(5):
    if i >= 3:
        continue
    for j in range(10):
        if j == 5:
            continue
        var += j
```

[Output]: 120

Similar to Example 3, multiple continue statements are required to control an entire block of nested loops since labels are not allowed, as stated earlier.

Example 6:

```
var = 0
for i in range(5):
    for j in range(5):
        var += 1
        if var == 3:
            break
    else:
        var = -10
```

[Output]: -10

Uniqe to Python, an else block can be used right after a loop. If loop exits with its condition, else block will be passed; else if loop exits with break, else block will be executed. Since the loop in the given example right will exit with break, var will be equal to -10.

NOTE: There is another flow control mechanism in Python: **pass**, which is not included in this report because it simply does nothing.

1.2 JavaScript (ES6+)

In JavaScript, similar to Python, there are two loop control mechanisms which are **break** and **continue**. However, unlike Python, these control mechanisms of JavaScript support both labeled and unlabeled usage. So that, a nested loop can exit by only one break expression which is in an inner loop. Again, these statements can used without any conditioning as well.

Example 1:

```
while (true) {
    temp += 1
    if (temp === 5) break
}
```

break can be used to exit infinite loops similar with Python. An example is provided on the left. For this example, and the followings assume that variable temp is initially equal to zero.

[Output]: 5

Example 2:

```
temp = 0
while (temp < 10) {
    temp += 1
    if (temp < 5)
        continue
    break
}
```

continue can also used to control loops, it will ignore the codes after itself and start a new loop iteration. In the example left, break and continue are used together in the same loop. Until if condition become true, break will be passed, then loop will exit when break is executed.

[Output]: 5

Example 3:

```
temp = 0
for (let i = 0; i < 10; i++) {
    if (i >= 5) continue
    for (let j = 0; j < 20; j++) {
        temp += j
        if (j >= 5) break
    }
}
```

In the example left, continue and break are used together in a nested loop block without labels. So that, they will only control the loop nearest. continue will start a new iteration of the outer loop, and break is going to exit the inner loop.

[Output]: 75

Example 4:

```
temp = 0
outerLoop:
    for (let i = 0; i < 10; i++) {
        if (i >= 5)
            continue
        // innerLoop:
        for (let j = 0; j < 20; j++) {
            temp += j
            if (j >= 5)
                break outerLoop
        }
        temp += i
    }
```

JavaScript supports labels for user-located loop control mechanisms. A nested loop can exit from an inner loop. In the example left, the outer loop will exit when break expression which is in the inner loop is run, since a label is added to the initial definition of the outer loop and that particular break exists to this label.

```
[Output]: 15
```

Example 5:

```
outerLoop:
    for (let i = 0; i < 10; i++) {
        if (i >= 5)
            break
        // innerLoop:
        for (let j = 0; j < 20; j++) {
            temp += j
            if (j >= 5)
                continue outerLoop
        }
        temp += i
    }
```

Similarly, a continue expression can also be used with a label as well. In such a case, the next iteration of the loop associated with the given label is going to be started. Example 5 on the left is simply doing that. The label innerLoop is commented in both Example 4 and 5 since it is not applied in the program.

```
[Output]: 75
```

NOTE: Even though, there are two labels named “outerLoop” in the program, it will still run because the control statements will return to the nearest one.

1.3 Dart (2.18.5)

In Dart, just like Python and JavaScript, there are **break** and **continue** as unconditional loop control mechanisms. Similar with JavaScript, in Dart, these control statements can be used with or without labels. So that, nested loops can exit simply.

Example 1:

```
while (true) {
    ex = ex + 1;
    if (ex >= 5)
        break;
}
```

In Dart, break and continue can be used without any conditioning. In the given code example left, break is used to exit an infinite loop.

[Output]: 5

Example 2:

```
while (ex < 10) {
    while (true) {
        ex += 1;
        if (ex == 10)
            break;
    }
}
```

break can be used either with labels or without labels. In case there is no label declared, break is going to exit the loop that it is in…

[Output]: 10

Example 3:

```
outerLoop: for (var i = 0; i < 10; i++) {
    innerLoop: for (var j = 0; j < 5; j++) {
        ex += j;
        if (i == 5)
            break outerLoop;
    }
    ex += i;
}
```

…if break is used with a label then, it will exit the loop that is associated with the provided label. In Example 3 left, when break is executed, the entire nested loop block will exit.

[Output]: 60

Example 4:

```
while (ex < 10) {
    while (ex < 20) {
        ex += 1;
        if (ex % 2 == 0)
            continue;
        ex += 10;
    }
}
```

[Output]: 23

continue is also available in Dart with the exact same behavior of the former programming languages covered. Just like break, continue can also be used either with or without a label. In the example left, a usage without a label is provided.

Example 5:

```
outerLoop: for (var i = 0; i < 10; i++) {
    innerLoop: for (var j = 0; j < 5; j++) {
        ex += j;
        if (i == 5)
            continue outerLoop;
    }
    ex += i;
}
```

[Output]: 130

In the example code left, a usage of a continue expression with a label is given. If continue is removed from the code, the output would be equal to 145 rather than 130.

NOTE: Just like in JS, there are two different labels with the exact same naming in the program. However, that will not cause any error nor warning because control statements with labels will simply affect the one that they are in.

1.4 PHP (8.1.13)

In PHP, **break** and **continue** are also available as user-located loop control mechanisms. Unlike some programming languages covered in this report, these control mechanism do not support labels. However, there are still several ways to exit a nested loop. **goto** expression can be used to jump to the end of a loop, or break and continue can be used with an integer value, which corresponds to the level of the nested block.

Example 1:

```
for (;;) {
     $var = $var + 1;
     if ($var >= 7)
         break;
}
```

break, in PHP, can be used to exit an infinite loop. An example is provided on the left.

```
[Output]: 7
```

Example 2:

```
for (; $var < 100; $var++) {
    if ($var % 2 != 0)
        continue;
    $var *= 10;
}
```

continue can be used to start a new loop iteration and pass the following codes.

```
[Output]: 221
```

Example 3:

```
while (true) {
    $var++;
    if ($var >= 7)
        break;
}
```

while loop is also available in PHP, and can be used with break and continue as well. An example code which has the exact same behavior with the code in Example 1 is provided on the left.

```
[Output]: 7
```

Example 4:

```
while (true) {
    $var++;
    if ($var == 13)
        goto exitWhileLoop;
} exitWhileLoop:
```

goto can be used to exit a loop or a nested block since break and continue cannot run with labels. goto will simply jump to the desired label and start to execute codes that are after the label.

```
[Output]: 13
```

NOTE: The exact label naming cannot be used in PHP, unlike Dart and JS. Because goto is going to jump to that label, independently from the loop. If there are more than one label with the exact same naming, that will cause an error since goto will not be able to which one to jump.

Example 5:

```
for (;;) {
    echo "continue is ran\n”;

    while (true) {
        $var++;
        if ($var == 5)
            continue 2;
        else if ($var == 9)
            break 2;
    }
}
```

continue and break can used with an integer value that is the level of the nested block to control, even though labels are not allowed. On the example left, both break and continue statements are used with an integer of 2, so that when they are executed, the for loop will be controlled. break is going to exit the for loop and continue will start a new iteration of the for loop.

```
[Output]: {
      continue is ran
      continue is ran
      Output[ Example 5 ]: 9 }
```

NOTE: When break and continue are used with an integer value to control nested blocks, if the value is not matched with the level of the block that will cause an error. For instance, in Example 5 above, there is a for loop and a while loop as a nested loop block, the level is 2. So that such statements **break 3**, **continue 4** etc. will cause an error.

1.5 Rust (1.65.0)

In Rust, **break** and **continue** are available as unconditional loop control mechanisms. Moreover, they can be used either with or without labels. Additionally, different from the other programming languages covered, these control mechanisms can also be used with a variable since loops can return to a value in Rust.

Example 1:

```
while var < 10 {
    var = var + 1;
    if var % 5 == 0 && var != 0 {
        break;
    }
}
```

break can be used to terminate a loop execution unconditionally.

```
[Output]: 5
```

Example 2:

```
for i in set {
    if i % 2 == 0 {
        continue;
    }
    var += i;
}
```

continue can be used to start the next loop iteration before the ongoing one is finished.

```
[Output]: 9
```

Example 3:

```
'loopEx3: loop {
    var += 1;
    if var == 7 {
        break 'loopEx3;
    }
}
```

Labels are allowed in Rust after break and continue to control loops.

```
[Output]: 7
```

Example 4:

```
'outerLoop: loop {
    var = var + 1;
    loop {
        var = var * 10;
        if var > 200 {
            break 'outerLoop;
        } else if var % 10 == 0 {
            continue 'outerLoop;
        }
        break;
    }
}
```

[Output]: 1110

Labels can be useful to exit nested blocks. On the example left, a nested loop block is given and it exits with the break expression at line 6. continue can also be used with a label. If no label is provided after these control statements, then they affect the enclosing loop.

Example 5:

```
let (mut temp1, mut temp2) = (0, 0);

let calc = loop {
    if temp1 > 20 {
        break temp2;
    } else {
        temp1 += 1;
        temp2 = temp1 * 2;
    }
};
```

[Output]: 42

In Rust, break expressions can be used with variables; on Example 5 left, such a usage is provided. Loop is used to make an arithmetic calculation, and the result is kept in a variable. Since this loop is expected to return a value, break is used with a variable which will be the return value of the loop. Because it is an infinite loop, there is no other way to exit it.

1.6 Ruby (2.6.10)

In Ruby, **break** is available as a control expression to terminate loops immediately. There is also another loop control expression: **next**, which is simply the equivalent of continue of some other languages. However, unlike several languages, Ruby does not support labels after these loop control expressions; and unlike PHP, there is no other way to exit a nested block. Just like in Python, the one must apply its own solutions to exit a nested block.

Example 1:

```
while true
    var += 1
    if var == 8
        break
    end
end
```

break expression is to exit a loop immediately. An example is provided on the left.

```
[Output]: 8
```

Example 2:

```
for i in 0..10
    if i % 3 == 0
        next
    end
    var = var + i
end
```

next is basically the equivalent of continue expression.

```
[Output]: 37
```

Example 3:

```
while true do
    while true do
        if var == 10
            break end
        var += 1 end
    break end
```

As stated, there is no easy way to exit a nested loop. The break expression at line 4 on the example left is going to just exit the inner loop. Because of that, another break loop is provided right after the inner one, which is going to exit the outer loop.

```
[Output]: 10
```

1.7 Lua (5.4.4)

In Lua, **break** is available to exit loops, but there is no expression for continue. Labels after break is not allowed, however **goto** is also available to exit nested loops.

Example 1:

```
while (true) do
    var = var + 1
    if var >= 11 then
        break end end
```

break expression can be used to exit an infinite loop.

[Output]: 11

Example 2:

```
repeat
    var = var + 1
    while (true) do
        var = var * 10
        if var >= 100 then break end
    end until (var >= 200)
```

If a break expression is used in a nested loop block, then it only will exit the enclosing one.

[Output]: 1010

Example 3:

```
while (true) do
    while (true) do
        var = var + 1
        if var > 10 then
            goto exitOuterLoop
        elseif var > 5 then
            goto exitInnerLoop end
    end ::exitInnerLoop::
    var = var * 10
end ::exitOuterLoop::
```

goto expression can be used to exit an entire nested block since labels after break expression is not allowed. In the example left, two goto expressions are used to control the entire block.

[Output]: 61

1.8 *Summary*

Programming Language	**Conditional/ Unconditional control**	**Labeled/Unlabeled control**
Python	break, continue, pass	Unlabeled
JavaScript	break, continue	Both
Dart	break, continue	Both
PHP	break, continue, goto	Unlabeled (Except for goto)
Rust	break, continue	Both
Ruby	break, next	Unlabeled
Lua	break, goto	Unlabeled (Except for goto)

2 Analysis of the Programming Languages in terms of User-Located Loop Control Mechanisms

In this report 7 different programming languages are analyzed in terms of user-located loop control mechanisms. It can be said that these languages provide different behaviors for their loop control designs. When their readability and writability are considered separately, Rust is seem to be having the best design choices for loops and user-located control.

First of all, Rust provides both break and continue as its control mechanisms. Which is a good design choice because these two expressions are common in many other languages; this common behavior makes the language easy to read and write. That is also the reason why Lua and Ruby are eliminated since they do not support either these two expressions.

Secondly, Rust also allows labels after break and continue to exit nested loops, which is also a good design choice because it does not require to keep additional variables or self-repeating code blocks, unlike Python.

Moreover, Rust does not support and also not require goto expression, contrary to PHP. Which is also a good design since goto is considered to be a bad programming practice widely.

Finally, Rust, different from JavaScript and Dart, allows break expression to receive a variable after itself, so that loop expression can return a variable like a function, which might be very useful in some cases for programmers.

Because of all these reasons, it is decided that Rust is the best programming language among others in terms of user-located loop control mechanism, even though it is a statically typed language.

3 Personal Learning Strategies and Tools Used

After used online compiler *Replit* in the very first homework assignment, I decided not to use it this time because it is very time consuming to create a project for each programming and run them online. So that, I setup each languages compilers into my own device and run them in this way.

Programming Language/Editing	**Tools Used (All local compilers/interpreters)**
Python	PyCharm, IDE (JetBrains)
JavaScript	Webstrom, IDE (JetBrains)
Dart	VSCode, Text Editor (Microsoft)
PHP	
Rust	
Ruby	
Lua	
PDF	Pages, Word Processor (Apple)

To complete this assignment, I simply applied to online documents and websites for each and every programing language covered. For Rust and PHP, I read the official documentations' related parts (Please, see references for further details).

REFERENCES

"4. More Control Flow Tools¶." *4. More Control Flow Tools - Python 3.11.0 Documentation*, https://docs.python.org/3/tutorial/controlflow.html.

"Break - Manual." *Php*, https://www.php.net/manual/en/control-structures.break.php.

"Continue - Manual." *Php*, https://www.php.net/manual/en/control-structures.continue.php.

"Dart - Loop Control Statements (Break and Continue)." *GeeksforGeeks*, 18 June 2020, https://www.geeksforgeeks.org/dart-loop-control-statements-break-and-continue/.

"Dart Programming - Loops." *Tutorials Point*, https://www.tutorialspoint.com/dart_programming/dart_programming_loops.htm.

"JavaScript - Loop Control." *Tutorials Point*, https://www.tutorialspoint.com/javascript/javascript_loop_control.htm.

"Labels in Dart." *GeeksforGeeks*, 28 Jan. 2022, https://www.geeksforgeeks.org/labels-in-dart/.

"Lua - Loops." *Tutorials Point*, https://www.tutorialspoint.com/lua/lua_loops.htm.

PHP Break and Continue, https://www.w3schools.com/php/php_looping_break.asp.

Programming in Lua : 4.3.4, https://www.lua.org/pil/4.3.4.html.

"Ruby Break and next Statement." *GeeksforGeeks*, 13 Aug. 2019, https://www.geeksforgeeks.org/ruby-break-and-next-statement/.

"Ruby: Loops (for, While, Do..while, until)." *GeeksforGeeks*, 5 July 2021, https://www.geeksforgeeks.org/ruby-loops-for-while-do-while-until/.

"The Rust Programming Language." *Control Flow - The Rust Programming Language*, https://doc.rust-lang.org/book/ch03-05-control-flow.html.

"The Rust Reference." *Loop Expressions - The Rust Reference*, https://doc.rust-lang.org/reference/expressions/loop-expr.html.