Bilkent University
Department of Computer Engineering

CS 315

Homework Assignment 2

User-Located Loop Control Mechanisms in Dart, Javascript, Lua, PHP, Python, Ruby, and Rust

Alper Bozkurt
21802766
CS 315 - 003

0 - Questions

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

In language of the Dart, javascript, Lua, PHP, Python, Ruby and Rust, conditional mechanism is not the integral part of the exit. When unconditional mechanism is applied in conditional mechanism such as for loop. The unconditional mechanism prevails and the iteration over. It should be the integral part to possess more control of the code. Morever, it should be like this for easier access and change of the code. For functionality and controlability, unconditional mechanism should be the integral.

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

In languages of Dart, javascript, Rust are labeled languages. After attending a label to the loops. It can be exited from specific loops with label. In languages of Lua and PHP, there is no label but functions like goto provides the same functionality. It breaks the Loop and direct the iteration to another loop or exit directly from loop. However, in the language of Python and Ruby, there is no such functionality. For more control about the code the label's are quite important. It provides wider outcome and much more diversity to function as it demonstrated in the example codes.. Therefore, enclosing loops should also be exited.

1 - Dart

First two part of the code are to understand the hierarchy between conditional and unconditional exit from the unlabelled loops. The first code is a watchable condition of the football according to how much score is made. If the score is above 4, football match becomes fun to watch. There is if statement inside the while loop to understand if there are 3 score which is necessary to get of the loop.. In the second example there is also a conditional loop which is for loop. However inside the for loop there is if statement the cut loop because the parameter is year and it goes to zero. However, the necessary part is to reach the year of 2005. The code is to understand tennis is watchable according to years. And before the years of 2005 it is not watchable. Therefore, it is cut from 2005 and the loop stop iterating thanks to the unconditional exit which is break in this example.

```dart
void main() {

  //conditional exit
  int totalScore = 0;
  bool footballIsWatchable = false;
  while(footballIsWatchable == false){
    print("Football is not watchable if score is ${totalScore}");
    if(totalScore == 3){
      footballIsWatchable = true;
      print("If score is  ${totalScore + 1} or higher footbal is cool to watch");
    }
    totalScore++;
  }

   //unconditional exit
  bool tennisIsWatchable;
  int year = 2015;
  for(year; year > 0; year = year - 3){
    if(year < 2005){
      tennisIsWatchable = false;
      print("Tennis is not watchable and pathetically funny in ${year} and before ${year}");
      break;
    }
    print("Tennis is watchable in ${year}");
  }
}
```

```
Console

Football is not watchable if score is 0
Football is not watchable if score is 1
Football is not watchable if score is 2
Football is not watchable if score is 3
If score is  4 or higher footbal is cool to watch
Tennis is watchable in 2015
Tennis is watchable in 2012
Tennis is watchable in 2009
Tennis is watchable in 2006
Tennis is not watchable and pathetically funny in 2003 and before 2003
```

After dealing with unlabeled loops, this example deals with the labeled example It is for understand also the difference between labeled and unlabelled exit. Also, the capability of what could be doing in the labeled exit. The code is a tennis match score displayer with the overboring npc named Gokcebot. The outerLoop and innerLoop is for having a control to continue or break the loop. That is pretty functional to make tennis match score which is hard to write without labeled loop. To illustrate more control of the code, the inner and outer loop is both used.

```
  //labeled exit
  //Tennis match with npc named gökçe
  var game = 0;
  var point = 0;

  outerloop:
  for (game = 0; game <= 6; game++ ) {
      print("In game: ${game}");

      innerloop:
      for (point = 0; point <= 60; point = point + 15) {
        if (game == 2) {
          print ("Alper is bored with npc named GökçeBot and quit the game");
          break outerloop;
        }
         if (point == 60) {
          break innerloop;
        }
        if(point != 45 ){
          print ("Alper: ${point} GökçeBot: 0 ");
        }
        if (point == 45) {
          print ("Alper: ${point - 5} GökçeBot: 0");
        }
      }
   }
}
```

```
In game: 0
Alper: 0 GökçeBot: 0
Alper: 15 GökçeBot: 0
Alper: 30 GökçeBot: 0
Alper: 40 GökçeBot: 0
In game: 1
Alper: 0 GökçeBot: 0
Alper: 15 GökçeBot: 0
Alper: 30 GökçeBot: 0
Alper: 40 GökçeBot: 0
In game: 2
Alper is bored with npc named GökçeBot and quit the game
```

2 - javascript

First two part of the code are to understand the hierarchy between conditional and unconditional exit from the unlabelled loops. The first code is an tavuk doner capacity illustration that how many tavuk doner I can eat. After the capacity is satisfied, the function is finished. The condition is held by while loop. After reaching capacity, the code is finished. In the second example, it is for et doner. There is also a conditional loop which is while loop. However, inside the while loop there is if statement the cut loop because there exist a external problem which is money. So loop is supposed to cut when it does not come to capacity. Therefore it proofs the hierarchy conditional loop is below the unconditional loop and unconditional loop is more controlable.

```
//conditional exit
var tavukDoner = 1;
var donerCounter = 1;
var capacity = 0;

while(capacity < 4){
  capacity += tavukDoner;
  console.log("I eat " + donerCounter + " tavuk döner and I am okay!");
  donerCounter ++;
}
console.log("I eat "+donerCounter+
" tavuk döner and I'm never gonna eat tavuk döner again.");

//unconditional exit
var etDoner = 1;
var donerCounter = 1;
var capacity = 0;

while(capacity < 5){
  capacity += etDoner;
  console.log("I eat " + donerCounter + " et döner and I am okay!");
  donerCounter ++;
  if(capacity == 3){
    console.log("I eat " + donerCounter + " et döner but I'm out of money");
    break;
  }
}
```

Output:

```
I eat 1 tavuk döner and I am okay!
I eat 2 tavuk döner and I am okay!
I eat 3 tavuk döner and I am okay!
I eat 4 tavuk döner and I am okay!
I eat 5 tavuk döner and I'm never gonna eat tavuk döner again.
I eat 1 et döner and I am okay!
I eat 2 et döner and I am okay!
I eat 3 et döner and I am okay!
I eat 4 et döner but I'm out of money
```

After dealing with unlabeled loops, this example deals with the labeled example It is for understand also the difference between labeled and unlabeled exit. Also, the capability of what could be doing in the labeled exit. The code is a calculator and instant report about which day how many fix menu eaten. The outerLoop and innerLoop is for having a control to continue or break the loop. That is pretty functional to make unstable calculator which is hard to write without labeled loop. To illustrate more control of the code, the inner and outer loop is both used.

```
//labeled exit

//daily fixed menu eating report

let day,eaten;
var time = 1;

outerLoop:
for (day = 1; day <= 5;day++) {
  innerLoop:
  for (eatenFixMenu= 1; eatenFixMenu <= 3 ;eatenFixMenu++) {
    if(day < 3  && eatenFixMenu == 1){
      console.log(day + "'th day, "+ eatenFixMenu + " fix menu is  eaten");
      break innerLoop;
    }
    if(day == 4){
      eatenFixMenu -= 1;
      console.log(day + "'th day, "+ eatenFixMenu + " fix menu is  eaten");
      break;
    }
    console.log(day + "'th day, "+ eatenFixMenu + " fix menu is  eaten");

  }
}
```

```
1'th day, 1 fix menu is  eaten
2'th day, 1 fix menu is  eaten
3'th day, 1 fix menu is  eaten
3'th day, 2 fix menu is  eaten
3'th day, 3 fix menu is  eaten
4'th day, 0 fix menu is  eaten
5'th day, 1 fix menu is  eaten
5'th day, 2 fix menu is  eaten
5'th day, 3 fix menu is  eaten
```

3 - Lua

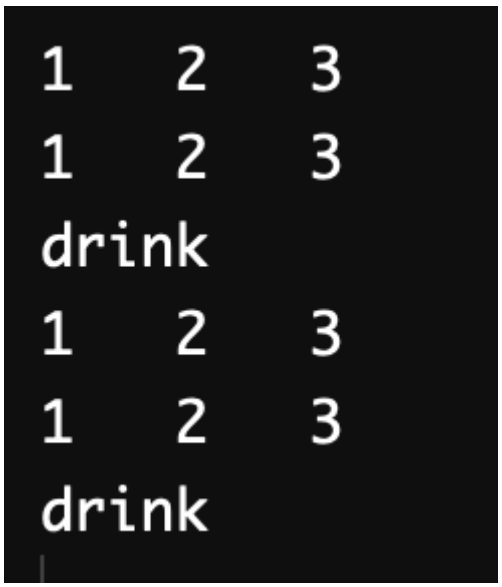
First two part of the code is for understand the hierarchy between conditional and unconditional exit from the unlabelled loops. The first code is about scarry capacity that how many jumpscare I can handle in a movie. After the capacity is satisfied, the function is finished. The condition is held by repeat until loop which is a an unique syntax for Lua language. After reaching capacity, the code is finished. In the second example, it is a depresion capacity for an arthouse movie. That is my preference so I have more tolerance than arthouse movie. There is also a conditional loop which is again repeat until loop. However, inside the loop there is if statement the cut loop because there exist a external problem which is an instant feeling that makes me depressive. So loop is supposed to cut when it does not come to capacity. Therefore it proofs the hierarchy conditional loop is below the unconditional loop and it is more controlable.

```lua
--conditional exit
--a work about being scarry
scarry = 0
repeat
   print("I am not scared enough.")
   scarry = scarry + 50
until( scarry > 200 )
print("I am really terrifying.")

--unconditional exit
--a work about being depressed
depressive = 0;
repeat
  print("I feel depressive.")
  depressive = depressive + 1
  if(depressive > 3)
  then
    print("Time to get up!")
    break
  end
until(depressive > scarry)
```

```
I am not scared enough.
I am not scared enough.
I am not scared enough.
I am not scared enough.
I am not scared enough.
I am really terrifying.
I feel depressive.
I feel depressive.
I feel depressive.
I feel depressive.
Time to get up!
```

After dealing with unlabeled loops, this example deals with the partly labeled example which is used goto method to make the same outcome.. It is for understand also the difference between labeled and unlabeled exit. Also, the capability of what could be doing in the labeled exit. The code is a basic printer that expected 1 2 3 1 2 3 drink and doing with the using both for loops and goto function's mechanics. The outerLoop and innerLoop is for having a control to continue or break the loop. That is not pretty functional to make simple printer which is more easy to write without labeled loop. Therefore, labeled loop is a distinct property which makes some things easier.

```
--label exit
bored = false
for i = 1,100,1
do
  if(bored)
  then
    print("drink")
  end
  for number = 1,3,1
  do
    if(i > 2)
    then
        goto throwThemBack
    end
    if(number > 2)
    then
      goto causeImBored
    end

    if(number < 2)
    then
        print(number , number + 1, number + 2)
    end
    if(number > 1)
    then
        print(number -1, number , number + 1)
    end
  end
  ::causeImBored::
  bored = true
end
::throwThemBack::
```



4 - PHP

First two part of the code is for understand the hierarchy between conditional and unconditional exit from the unlabelled loops. The first code is about hunger calculator of mine after breakfast according to time. After the time arrives to eat, the function is finished. In the second example, it is again hunger calculator for another day. However, I drink tea and my appetite is opened so I want to eat so iteration is over without the condition is satisfied. There is also a conditional loop which is again for loop. However, inside the loop there is if statement the cut loop because there exist a external problem which is tea. So loop is supposed to cut when it does not come to capacity. Therefore it proofs the hierarchy conditional loop is below the unconditional loop and it is more controlable.

```php
<?php
//conditional exit

for($time = 1; $time < 5; $time ++){
    echo("It is ".$time.", Me not hungry"."\n");
}
echo("It is ".$time.", and Me hungry!". "\n");

//unconditional exit
$tea = 3;
for($time= 1; $time < 5; $time ++){
    if($time == $tea){
        break;
    }
    echo("It is ".$time.", Me not hungry"."\n");
}
echo("It is ".$time.", and I can eat smt with tea!");
```

```
It is 1, Me not hungry
It is 2, Me not hungry
It is 3, Me not hungry
It is 4, Me not hungry
It is 5, and Me hungry!
It is 1, Me not hungry
It is 2, and Me hungry while drinking tea!
```

After dealing with unlabeled loops, this example deals with the partly labeled example which is used goto method to make the same outcome.. It is for understand also the difference between labeled and unlabeled exit. Also, the capability of what could be doing in the labeled exit. The code is a basic time counter.. The outerLoop and innerLoop is for having a control to continue or break the loop which makes the add time travel functionality to the code. That is pretty functional to make an ordinary time counter to time travel added time counter which is hard to write without a labeled loop. To illustrate more control of the code, the inner and outer loop is both used.

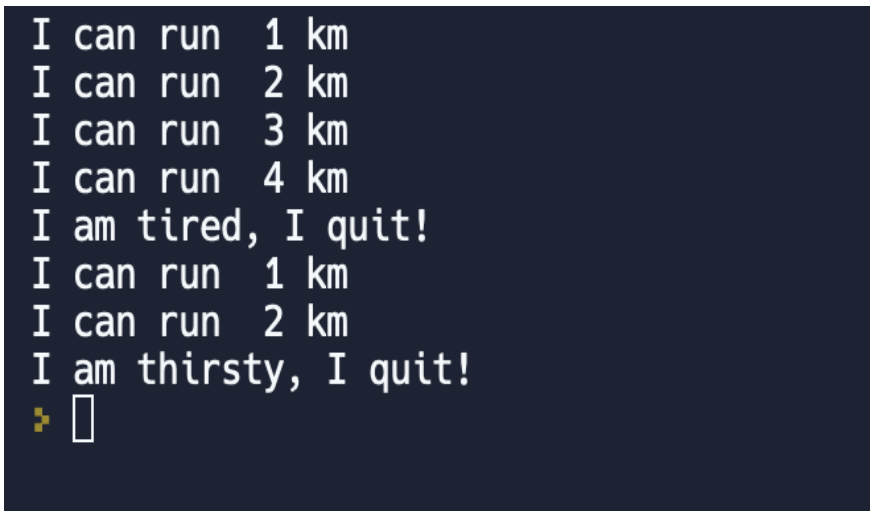
```php
//labeled exit
$timeTravel = 20;
$ninthDoctor = 3;
for($hour = 0; $hour < 12 ; $hour ++){
    $hour += 1;

    for($minute = 10; $minute < 60; $minute = $minute + 10){
        if($ninthDoctor < $hour){
            goto end;
        }
        if($timeTravel < $minute){
            goto future;
        }
        echo("Time is ".$hour.", ".$minute."\n");
    }
    future:
}
end:
echo("Time is over.");

?>
```

```
Time is 1, 10
Time is 1, 20
Time is 3, 10
Time is 3, 20
Time is over.
```

5 - Python

First two part of the code is for understand the hierarchy between conditional and unconditional exit from the unlabelled loops. The first code is about running capacity calculator. In fifth kilometer, the function is finished. In the second example, it is again running capacity calculator. However, I am thirsty in the moment of running so I quit. Thus, iteration is over without the condition is satisfied. There is also a conditional loop which is again for loop. However, inside the loop there is if statement the cut loop because there exist a external problem which is being thirsty. So loop is supposed to cut when it does not come to capacity. Therefore it proofs the hierarchy conditional loop is below the unconditional loop and it is more controlable.

```
#conditional exit
capacity = 4;
km = 1;
for km in range(capacity):

  km = km + 1;
  print("I can run ", km, "km");

print("I am tired, I quit!");

#unconditional exit
km = 1;
thirsty = 2;
for km in range(capacity):
  km = km + 1;
  print("I can run ", km, "km");
  if thirsty == km:
    break
print("I am thirsty, I quit!");
```



I could not totally create the labeled Python loop.

“PEP 3136 proposes labeled break/continue. Guido rejected it because "code so complicated to require this feature is very rare". The PEP does mention some workarounds, though (such as the exception technique), while Guido feels refactoring to use return will be simpler in most cases.”

6 - Ruby

First two parts of the code are to understand the hierarchy between conditional and unconditional exit from the unlabelled loops. The first code is the reading book calculator after four parts of reading the function are finished. In the second example, it is again reading book calculator. However, I slept in the moment of reading so It is interrupted. Thus, iteration is over without the condition is satisfied. There is also a conditional loop which is again for loop. However, inside the loop there is if statement the cut loop because there exist a external problem which is sleeping. So loop is supposed to cut when it does not come to capacity. Therefore it proofs the hierarchy conditional loop is below the unconditional loop and it is more controllable.

```ruby
#conditional exit
puts "..First Day.."
bragging = "I've read book"
for $page in 1..4
  puts bragging
end
puts  "It's enough for today."

#unconditional exit
sleeping = 2
puts "..Second Day.."
for $page in 1..4
  puts bragging
  break if sleeping == $page
end
puts  "Dreams."
```

```
bundle exec ruby main.rb
..First Day..
I've read book
I've read book
I've read book
I've read book
It's enough for today.
..Second Day..
I've read book
I've read book
Dreams.
```

It is tried to show labelled loop in Ruby with using throw catch method however, mistake occurs.

7 - Rust

First two parts of the code are to understand the hierarchy between conditional and unconditional exit from the unlabelled loops. The first code is the sleep track system according to hours between 1 a.m and 9 a.m. After 9 a.m, the function are finished. In the second example, it is again sleep track system. However, neighbour start to cry and shout and the sleep is interrupted. Thus, iteration is over without the condition is satisfied. There is also a conditional loop which is again while loop. However, inside the loop there is if statement the cut loop because there exist a external problem which is neighbour. So loop is supposed to cut when it does not come to the time. Therefore it proofs the hierarchy conditional loop is below the unconditional loop and it is more controllable.

```
fn main() {

  //conditional exit
  let mut sleep = 1;
  let wake_up = 9;
  while sleep < wake_up{
      sleep += 1;
      println!("Dreams..");
  }
  println!("Run!");

  //unconditional exit
  sleep = 1;
  let neighbour_on = 3;
  while sleep < wake_up{
      if neighbour_on == sleep{ break; }
      sleep += 1;
      println!("Dreams..");
  }
  println!("Waking up in the middle of the night");
}
```

```
Dreams..
Dreams..
Dreams..
Dreams..
Dreams..
Dreams..
Dreams..
Dreams..
Run!
Dreams..
Dreams..
Waking up in the middle of the night
```

My compiler does not work the labelled loop so The photograph is copied from stackoverflow to show labeled loop and exits.

```
fn main() {
    'outer: for x in 0..5 {
        'inner: for y in 0..5 {
            println!("{},{}", x, y);
            if y == 3 {
                break 'outer;
            }
        }
    }
}
```

8 - Evaluation Of Languages

8.1- Readability

The Dart language is easy to read and understand. The syntax is again simple and quite similar to other languages. For me it is pretty high readable. Javascript is again easy to read however Dart is more readable for me. However Lua is hard understand at first, then when I look at the language It's syntax is very simple and straightforward. No different symbols involved. PHP compared to other three is least readable language. However, my opinion it is a mediocre about readability. Python is definitely the most readable I am not sure if I knew it before. However in that assignment (User-Located Loop Control Mechanisms), it is not so readable. Compared to Lua and Dart it is slightly better about readability. In my opinion, there is a indentation instead of braces or keywords making the code easier to read. Ruby is quite readable for me too but slightly better than PHP worse than others. Rust is pretty bad about readability. However, because it is primitive I think the PHP is least readable language and Lua is the most readible language. Because the go to mechanism is quite readable and the language is also simple and easy to read.

8.2- Writability

I am not sure easy to write but Dart is definitely fun to write for me. Ruby is the worst for writability. Javascript is highly writable language but compared to Dart it least writable for me. Both have label and the comparison between label and go to. Label more easier to write for me so the for writability. The options are only Javascript, Dart and Rust. Rust is a language I could not adapt so The Dart is most writable language for me.

9 - Learning Strategy

In the process of doing homework, the major process is to make sure what the homework actually wants from me. The fear of writing the same thing for all the languages makes me not make sure about what I do. After reading ten or hundred times of the assignment, I made a pattern and I am now thinking that I solved the philosophy behind the homework. However, I am still not sure I totally understand the homework. The usage of languages is much more easier for this homework compared to first homework. I used for loop for the first homework so I just look the first homework to remember. To conclude, understanding part takes much more effort than doing the homework. I used online compiler again. The repetition part makes me exhaust a little while doing the homework.

https://stackoverflow.com/questions/1352120/how-to-break-outer-cycle-in-ruby

https://stackoverflow.com/questions/189645/how-can-i-break-out-of-multiple-loops

https://stackoverflow.com/questions/22905752/named-breaks-in-for-loops-in-rust

https://www.geeksforgeeks.org/ruby-loops-for-while-do-while-until/

https://doc.rust-lang.org/book/ch03-04-comments.html

https://www.geeksforgeeks.org/dart-break-statement/

https://stackoverflow.com/questions/9751207/how-can-i-use-goto-in-javascript

https://www.geeksforgeeks.org/how-to-use-goto-in-javascript/

https://www.educba.com/lua-goto/