
# CS 315

# Programming Languages

# Section 3


# Ömer Oktay Gültekin

# 21901413


# 2022-2023 / Fall Semester

## Section 1 – Explaning Code Examples:

All code examples will be exactly the same except they are written in different languages for a fair comparison. Only Python will have less code than others since it does not have a pre-defined way for continuing the outer loop. The codes will consist of two parts. The first part will investigate the unlabeled unconditional loop control mechanisms usage in the languages by printing a values from 1 to 6 while not printing some of the values because of unconditional loop control mechanisms. The second part will investigate the labeled unconditional loop control mechanisms usage in the languages by using nested 2 loops. The first loop will have the numbers from 1 to 3; whereas the second loop will have the numbers from 1 to 6. Normal execution of the loops will be intervened by labeled one inner loop continuity control statement, one outer loop continuity control statement, one inner loop break control statement and one outer loop break control statement (except Python). All the labels will be defined can be anything user wants but semantic meaning was concerned during coding. The evaluation of the general syntax of the languages was discussed in the previous homework; therefore, this report will only focus on the syntax of the user-controlled loop control mechanisms. Also, this report will not include a discussion of the equivalent ways of labeled exits. Using try catch statements or function return values to simulate labeled exits can be done in most if not all programming languages. This report's main concern is to investigate the keywords that are specified in languages' specifications for the usage of unconditional exits.

### Dart Example:

```dart
void main() {
  // Unconditional Exits in Dart
  // Unlabeled Exits in Dart
  print("----------Unlabeled Exits----------");
  for(var i = 1; i <= 6; i++) {
    if (i == 3) {
      print("Unlabeled continue is used to pass the current iteration when the value is $i");
      continue;
    } else if (i == 5) {
      print("Unlabeled break is used to break the loop when the value is: $i");
      print("The remaining value 6 will not be printed.");
```

```
    break;
    }
    print("Loop value: $i");
  }
  print("-------------------------------------------------");

  // Labeled Exits in Dart
  print("-------------------Labeled Exits--------------------");
  Outer: for(var i = 1; i <= 3; i++) {
    print("Outer value: $i");
    Inner: for(var i2 = 1; i2 <= 6; i2++) {
          if(i == 1) {
            print(" Labeled continue is used, skipped outer loop iteration value: $i");
            continue Outer;
          } else if(i == 2 && i2 == 3) {
            print(" Labeled break is used to terminate current inner loop iteration.");
            break Inner;
          } else if(i2 == 3) {
            print(" Labeled continue is used, skipped inner loop iteration value: $i2");
            continue Inner;
          } else if(i2 == 5) {
            print(" Labeled break is used to terminate current outer loop iteration");
            break Outer;
          }
    print(" Inner value: $i2");
    }
  }
  print("Outer Loop is successfully broken.");
  print("-------------------------------------------------");
}
```

Dart fully supports user-located loop control mechanisms. Unconditional exit keywords of the language are break and continue. Break is used to terminate all remaining loop execution whereas continue is used to terminate only current loop execution. By default, both only affect the loop that they are in. Labeled exits are available in the language by labeling each loop by user defined label and using those labels in the following syntax:
break [user-defined-label] or continue [user-defined-label]. The output of the code is:

```
----------------Unlabeled Exits----------------
Loop value: 1
Loop value: 2
Unlabeled continue is used to pass the current iteration when the value is 3
Loop value: 4
Unlabeled break is used to break the loop when the value is: 5
The remaining value 6 will not be printed.
--------------------------------------------------
------------------Labeled Exits-------------------
Outer value: 1
 Labeled continue is used, skipped outer loop iteration value: 1
Outer value: 2
 Inner value: 1
 Inner value: 2
 Labeled break is used to terminate current inner loop iteration.
Outer value: 3
 Inner value: 1
 Inner value: 2
 Labeled continue is used, skipped inner loop iteration value: 3
 Inner value: 4
 Labeled break is used to terminate current outer loop iteration
Outer Loop is successfully broken
--------------------------------------------------
```

**Javascript Example:**

```
<!DOCTYPE html>
<html>
<body>
  <script>
    // Unconditional Exits in JS
    // Unlabeled Exits in JS
    console.log("----------------Unlabeled Exits----------------");
    for (let i = 1; i <= 6; i++) {
      if (i == 3) {
        console.log(`Unlabeled continue is used to pass the current iteration when the value is ${i}`);
        continue;
      } else if (i == 5) {
        console.log(`Unlabeled break is used to break the loop when the value is: ${i}`);
        console.log("The remaining value 6 will not be printed.");
        break;
      }
      console.log(`Loop value: ${i}`);
    }
```

```
    console.log("-------------------------------------------------\n");

    // Labeled Exits in JS
    console.log("------------------Labeled Exits------------------");
    Outer: for (let i = 1; i <= 3; i++) {
      console.log(`Outer value: ${i}`);
      Inner: for (let i2 = 1; i2 <= 6; i2++) {
        if (i == 1) {
          console.log(` Labeled continue is used, skipped outer loop iteration value: ${i}`);
          continue Outer;
        } else if (i == 2 && i2 == 3) {
          console.log(` Labeled break is used to terminate current inner loop iteration.`);
          break Inner;
        } else if (i2 == 3) {
          console.log(` Labeled continue is used, skipped inner loop iteration value: ${i2}`);
          continue Inner;
        } else if (i2 == 5) {
          console.log(" Labeled break is used to terminate current outer loop iteration");
          break Outer;
        }
        console.log(` Inner value: ${i2}`);
      }
    }
    console.log("Outer Loop is successfully broken");
    console.log("-------------------------------------------------");
  </script>
</body>
</html>
```

Javascript fully supports user-located loop control mechanisms. Unconditional exit keywords of the language are break and continue. Break is used to terminate all remaining loop execution whereas continue is used to terminate only current loop execution. By default, both only affect the loop that they are in. Labeled exits are available in the language by labeling each loop by user defined label and using those labels in the following syntax: break [user-defined-label] or continue [user-defined-label]. Therefore, the syntax is the same as Dart. The output of the code is:

```
----------------Unlabeled Exits----------------
Loop value: 1
Loop value: 2
Unlabeled continue is used to pass the current iteration when the value is 3
Loop value: 4
Unlabeled break is used to break the loop when the value is: 5
The remaining value 6 will not be printed.
--------------------------------------------------
-------------------Labeled Exits-------------------
Outer value: 1
 Labeled continue is used, skipped outer loop iteration value: 1
Outer value: 2
 Inner value: 1
 Inner value: 2
 Labeled break is used to terminate current inner loop iteration.
Outer value: 3
 Inner value: 1
 Inner value: 2
 Labeled continue is used, skipped inner loop iteration value: 3
 Inner value: 4
 Labeled break is used to terminate current outer loop iteration
Outer Loop is successfully broken
--------------------------------------------------
```

**Lua Example:**

```lua
-- Unconditional Exits in Lua
-- Unlabeled Exits in Lua
print("----------------Unlabeled Exits----------------")
for i = 1, 6
do
   if i == 4 then
      print("Unlabeled break is used to break the loop when the value is:", i)
      print("The remaining values 5 and 6 will not be printed.")
      break
   end
   print("Loop value:", i)
end
print("--------------------------------------------------")
```

```lua
-- Labeled Exits in Lua
print("------------------Labeled Exits------------------")
for i = 1, 3
do
   print("Outer value:", i)
   for i2 = 1, 6
   do
      if i == 1 then
         print(" Labeled continue is used, skipped outer loop iteration value:", i)
         -- This works for lua >= 5.2 (goto introduced here)
         goto continueOuter
      elseif i == 2 and i2 == 3 then
         print(" Labeled break is used to terminate current inner loop iteration")
         goto breakInner
      elseif i2 == 3 then
         print(" Labeled continue is used, skipped inner loop iteration value:", i2)
         goto continueInner
      elseif i2 == 5 then
         print(" Labeled break is used to terminate current outer loop iteration")
         goto breakOuter
      end
      print(" Inner value:", i2)
      ::continueInner::
   end
   ::breakInner::
   ::continueOuter::
end
::breakOuter::
print("Outer Loop is successfully broken")
print("-------------------------------------------------")
```

Only Lua compilers with the version >= 5.2 fully supports user-located loop control mechanisms. Unconditional exit keyword of the language is break. Break is used to terminate all remaining loops. By default, it only affects the loop that it is in. There is no unlabeled continue statement in the language. The reason for that is the goto statement is already capable of what continue statement does in other languages [1]. Labeled exits are available in the language by labeling any place in the program and using the goto statement for both exit statements by using those labels in the following syntax: goto [user-defined-label]. It makes the control-mechanisms

very risky since one can easily “goto” the earlier places of the program that may cause the infinite loop of the program. The output of the code is:

```
----------------Unlabeled Exits----------------
Loop value:1
Loop value: 2
Loop value: 3
Unlabeled break is used to break the loop when the value is:    4
The remaining values 5 and 6 will not be printed.
--------------------------------------------------
-------------------Labeled Exits-------------------
Outer value:    1
 Labeled continue is used, skipped outer loop iteration value:1
Outer value:    2
 Inner value:   1
 Inner value:   2
 Labeled break is used to terminate current inner loop iterationOuter value:    3
 Inner value:   1
 Inner value:   2
 Labeled continue is used, skipped inner loop iteration value:  3
Inner value:    4
 Labeled break is used to terminate current outer loop iteration
Outer Loop is successfully broken
--------------------------------------------------
```

## PHP Example:

```php
<?php
  // Unconditional Exits in PHP
  // Unlabeled Exits in PHP
  echo("----------------Unlabeled Exits----------------\n");
  for($i = 1; $i <= 6; $i++) {
     if ($i == 3) {
        echo("Unlabeled continue is used to pass the current iteration when the value is $i\n");
        continue;
     } else if ($i == 5) {
        echo("Unlabeled break is used to break the loop when the value is: $i\n");
        echo("The remaining value 6 will not be printed.\n");
        break;
     }
     echo("Loop value: $i\n");
```

```
    }
    echo("-------------------------------------------------\n");

    // Labeled Exits in PHP
    echo("-------------------Labeled Exits-------------------\n");
    for($i = 1; $i <= 3; $i++) {
        echo("Outer value: $i\n");
        for($i2 = 1; $i2 <= 6; $i2++) {
            if($i == 1) {
                echo(" Labeled continue is used, skipped outer loop iteration value: $i\n");
                continue 2; // continue from 2 previous loop (imagine like a tree)
            } else if($i == 2 && $i2 == 3) {
                echo(" Labeled break is used to terminate current inner loop iteration.\n");
                break 1; // break current loop (default behaviour of using break alone)
            } else if($i2 == 3) {
                echo(" Labeled continue is used, skipped inner loop iteration value: $i2\n");
                continue 1; // default behaviour
            } else if($i2 == 5) {
                echo(" Labeled break is used to terminate current outer loop iteration\n");
                break 2;
            }
        echo(" Inner value: $i2\n");
        }
    }
    echo("Outer Loop is successfully broken\n");
    echo("-------------------------------------------------\n");
?>
```

PHP fully supports user-located loop control mechanisms. Unconditional exit keywords of the language are break and continue. Break is used to terminate all remaining loop execution whereas continue is used to terminate only current loop execution. By default, both only affect the loop that they are in. Labeled exits are available in the language by indicating the affecting loop number from inside to outside starting from 1 where 1 is the loop that keywords in. The syntax of the labeled exits: break [loop-number] or continue [loop-number]. The output of the code is:

```
----------Unlabeled Exits----------
Loop value: 1
Loop value: 2
Unlabeled continue is used to pass the current iteration when the value is 3
Loop value: 4
Unlabeled break is used to break the loop when the value is: 5
The remaining value 6 will not be printed.
--------------------------------------------------
----------Labeled Exits----------
Outer value: 1
 Labeled continue is used, skipped outer loop iteration value: 1
Outer value: 2
 Inner value: 1
 Inner value: 2
 Labeled break is used to terminate current inner loop iteration.
Outer value: 3
 Inner value: 1
 Inner value: 2
 Labeled continue is used, skipped inner loop iteration value: 3
 Inner value: 4
 Labeled break is used to terminate current outer loop iteration
Outer Loop is successfully broken
--------------------------------------------------
```

**Python Example:**

```python
# Unconditional Exits in Python
# Unlabeled Exits in Python
print("----------Unlabeled Exits----------")
for i in range(1, 7): # 1 inclusive 7 exclusive
  if i == 3:
    print(f"Unlabeled continue is used to pass the current iteration when the value is {i}")
    continue
  elif i == 5:
    print(f"Unlabeled break is used to break the loop when the value is: {i}")
    print("The remaining value 6 will not be printed.")
    break
    print(f"Loop value: {i}")
print("--------------------------------------------------")

# Labeled Exits in Python
# No labeled exits available in python but unique simulation way exists
# using for ... else ... block in which else block only runs if the corresponding
# for statement terminates normally without break. Otherwise, the remaining code
# will be run that breaks the outer loop.
```

```
print("------------------Labeled Exits------------------")
for i in range(1, 4):
  print(f"Outer value: {i}")
  for i2 in range(1, 7):
    if i2 == 5:
      print(f" Unlabeled break is used to terminate current inner loop iteration when the inner value
is {i2}")
      break
    print(f" Inner value: {i2}")
  else:
      continue
  print("Inner Break triggers termination of the current outer loop iteration")
  break

print("Outer Loop is successfully broken.")
print("-------------------------------------------------")
```

Python partially supports user-located loop control mechanisms. Unconditional exit keywords of the language are break and continue. Break is used to terminate all remaining loop execution whereas continue is used to terminate only current loop execution. By default, both only affect the loop that they are in. Labeled exits are not available in the language but a unique simulation way exists that uses for … else … syntax. In this syntax, else block only runs if the corresponding for statement terminates normally without break. Otherwise, the remaining code will be run and break the outer loop. However, there is no unique way for labeled continue statement or breaking multiple loops. Therefore, introduction of the labeled unconditionals exits was proposed with the title of PEP 3136 but adding labeled exists to the language specification was rejected by developers of the Python language [2]. The output of the code is:

```
---------------Unlabeled Exits---------------
Unlabeled continue is used to pass the current iteration when the value is 3
Unlabeled break is used to break the loop when the value is: 5
The remaining value 6 will not be printed.
-------------------------------------------------
------------------Labeled Exits------------------
Outer value: 1
 Inner value: 1
 Inner value: 2
 Inner value: 3
 Inner value: 4
 Unlabeled break is used to terminate current inner loop iteration when the inner value
is 5
Inner Break triggers termination of the current outer loop iteration
Outer Loop is successfully broken.
-------------------------------------------------
```

**Ruby Example:**

```
# Unconditional Exits in Ruby
# Unlabeled Exits in Ruby
puts("----------------Unlabeled Exits----------------")
for i in 1..6
   if i == 3

      puts("Unlabeled continue (next) is used to pass the current iteration when the value is %d"
%[i])
      next
   elsif i == 5
      puts("Unlabeled break is used to break the loop when the value is: %d" %[i])
      puts("The remaining value 6 will not be printed.")
      break
   end
   puts("Loop value: %d" %[i])
end
puts("--------------------------------------------------")

# Labeled Exits in Ruby
# Ruby does not have Labeled Exists but have a unique way of simulating them using catch
statement.
# Label is defined with ":" at the catch statement that will be a returning point when the
corresponding label is thrown.
   puts("------------------Labeled Exits------------------")
   catch :outerBreak do
   for i in 1..3
    catch :innerBreak do
    puts("Outer value: #{i}")
    for i2 in 1..6
     catch :innerContinue do
     if i == 1
      puts(" Labeled continue is used, skipped outer loop iteration value: #{i}")
      throw :innerBreak # inner break = outer continue
     elsif i == 2 and i2 == 3
      puts(" Labeled break is used to terminate current inner loop iteration.")
      throw :innerBreak
     elsif i2 == 3
```

```
      puts(" Labeled continue is used, skipped inner loop iteration value: #{i2}")
      throw :innerContinue
    elsif i2 == 5
      puts(" Labeled break is used to terminate current outer loop iteration")
      throw :outerBreak
    end
    puts(" Inner value: #{i2}")
    end
  end
  end
end
end
puts("Outer Loop is successfully broken.")
puts("--------------------------------------------------")
```

Ruby partially supports user-located loop control mechanisms. Unconditional exit keywords of the language are break and next. Break is used to terminate all remaining loop execution whereas next is used to terminate only current loop execution. By default, both only affect the loop that they are in. Labeled exits are not available in the language but a unique simulation way exists that uses catch statement. In this syntax, label is defined with ":" at the catch statement that will be a returning point when the corresponding label is thrown. This way can also handle exiting multiple loops. The output of the code is:

```
----------------Unlabeled Exits----------------
Loop value: 1
Loop value: 2
Unlabeled continue (next) is used to pass the current iteration when the value is 3
Loop value: 4
Unlabeled break is used to break the loop when the value is: 5
The remaining value 6 will not be printed.
--------------------------------------------------
------------------Labeled Exits------------------
Outer value: 1
 Labeled continue is used, skipped outer loop iteration value: 1
Outer value: 2
 Inner value: 1
 Inner value: 2
 Labeled break is used to terminate current inner loop iteration.
Outer value: 3
 Inner value: 1
 Inner value: 2
 Labeled continue is used, skipped inner loop iteration value: 3
 Inner value: 4
 Labeled break is used to terminate current outer loop iteration
Outer Loop is successfully broken.
--------------------------------------------------
```

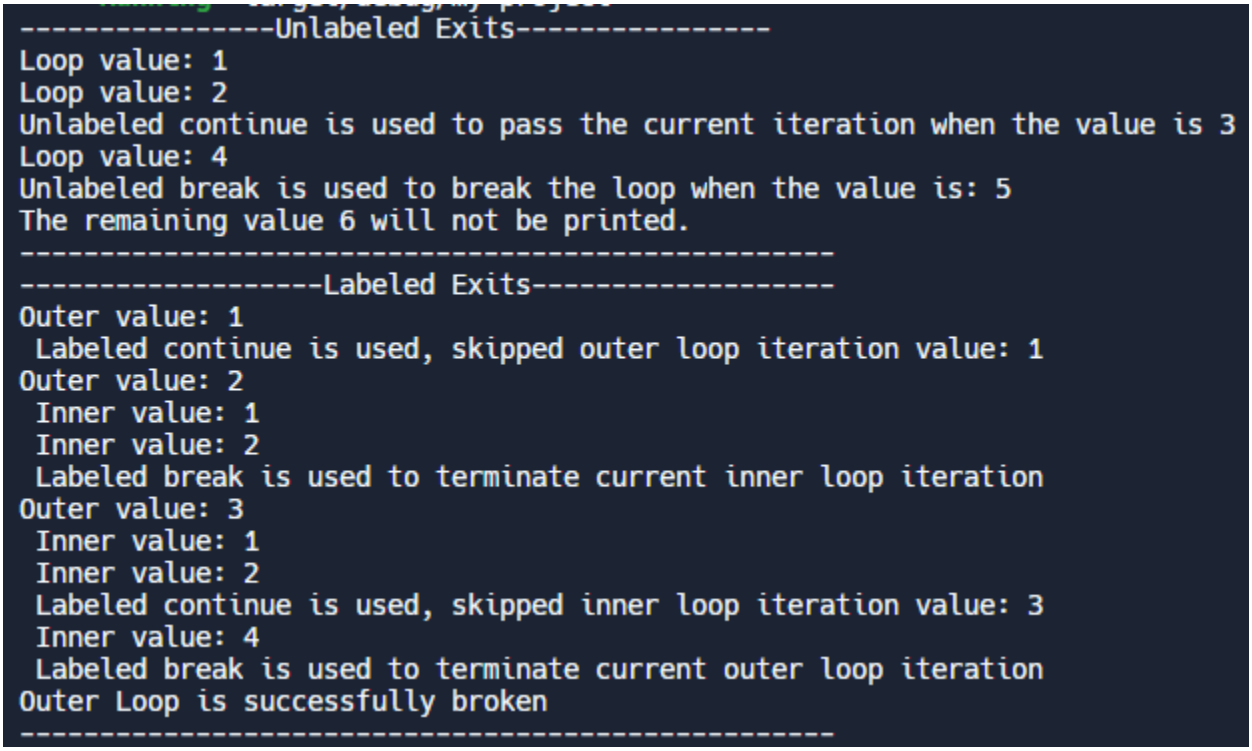
### Rust Example:

```rust
fn main() {
    // Unconditional Exits in Rust
    // Unlabeled Exits in Rust
    println!("----------------Unlabeled Exits----------------");
    for i in 1..7 { // 1..7 => 1 inclusive 7 exclusive
        if i == 3 {
            println!("Unlabeled continue is used to pass the current iteration when the value is {}", i);
            continue;
        } else if i == 5 {
            println!("Unlabeled break is used to break the loop when the value is: {}", i);
            println!("The remaining value 6 will not be printed.");
            break;
        }
        println!("Loop value: {}", i);
    }
    println!("------------------------------------------------");

    // Labeled Exits in Rust
    println!("-------------------Labeled Exits-------------------");
    'outer: for i in 1..4 {
        println!("Outer value: {}", i);
        'inner: for i2 in 1..7 {
            if i == 1 {
                println!(" Labeled continue is used, skipped outer loop iteration value: {}", i);
                continue 'outer;
            } else if  i == 2 && i2 == 3 {
                println!(" Labeled break is used to terminate current inner loop iteration");
                break 'inner;
            } else if i2 == 3 {
                println!(" Labeled continue is used, skipped inner loop iteration value: {}", i2);
                continue 'inner;
            } else if i2 == 5 {
                println!(" Labeled break is used to terminate current outer loop iteration");
                break 'outer;
            }
        println!(" Inner value: {}", i2);
        }
    }
    println!("Outer Loop is successfully broken");
```

```
    println!("-------------------------------------------------");
}
```

Rust fully supports user-located loop control mechanisms. Unconditional exit keywords of the language are break and continue. Break is used to terminate all remaining loop execution whereas continue is used to terminate only current loop execution. By default, both only affect the loop that they are in. Labeled exits are available in the language by labeling each loop by user defined label and using those labels in the following syntax: break '[user-defined-label] or continue '[user-defined-label]. Therefore, the syntax is the similar to Dart and JS. The output of the code is:




## Section 2 – Evaluating Languages in terms of Readability and Writability:

| | Dart | JS | Lua | PHP | Python | Ruby | Rust |
|---|---|---|---|---|---|---|---|
| Are keywords inherent? | **Yes** | **Yes** | **Yes** | **Yes** | **Yes** | **No (Next)** | **Yes** |
| Are there built-in labeled exits? | **Yes** | **Yes** | **Yes** | **Yes** | **No** | **No** | **Yes** |
| Can labeled exits only targets the starting point of the loops? | **Yes** | **Yes** | **No** | **Yes** | **No** | **No** | **Yes** |
| Are there built-in support for exiting more than two loops? | **Yes** | **Yes** | **Yes** | **Yes** | **No** | **No** | **No** |

In terms of unlabeled unconditional exists, all languages except Ruby uses natural way of expressing keywords which are break and keywords. The usage of next instead of continue in

Ruby makes it less writable than others since the developer needs to memorize something new but readability may not affected since next is inherent keyword too. In terms of labeled unconditional exists. Python immediately loses the competition since there is no built-in way to do the operation makes the code complex and both less writable and readable. In others, the ones that are uses the more flexible way of labels, namely Lua and Ruby, may less writable and readable than others since it is developers' responsibility to put the labels in correct places increases the effort of the developer to understand the correct places that labels should come, and the readers may also be suffered from keeping track of label places and needs of guessing the behaviour of the program. In the remainings, PHP differs from the others by using numerical method of labeling the loops may be less writable and readable than the others since both writer and reader must know the counting is starting from 1 and from inside to outside where inside indicates where the unconditional exit keyword exists. The reader or writer cannot even understand the numbers actually indicates loops; it may be anything. There is no clear sign of their label characteristics. In the remaining ones with labeling loops explicitly, Rust loses the competition by introducing "'" character in front of the label that makes the statement less writable since the developers again needs to remember that "'" character in front of the labels. However, the readability may not be affected since it is understandable by looking at the code that '[label-name] indicates labels. The remaining two languages, Dart and Javascript, are the same in terms of the syntax of the user-located control mechanisms and their's easy to remember and understandable syntax, in my opinion, makes them the best in terms of the writability and readability criteria.

## Section 3 – Learning Strategy:

In any language, I did not consult anyone, and I always use the sources on the internet.

### Learning Strategy for Dart:

My main resource was my previous knowledge about the language. For the parts that I didn't remember, I used the following online resources:
https://www.geeksforgeeks.org/labels-in-dart/
https://stackoverflow.com/questions/21028751/how-do-i-print-to-the-console-with-dart

### Learning Strategy for Javascript:

I was familiar with the general syntax of Javascript. I was just needed to look at the usage of break and continue in the language from the following link:
https://www.w3schools.com/jsref/jsref_break.asp

### Learning Strategy for Lua:

The documentation of the Lua was not suitable for beginners. Therefore, I used the following online resources to do the Lua part:

https://www.tutorialspoint.com/lua/lua_for_loop.htm
https://programming-idioms.org/idiom/43/break-outer-loop/1676/lua
https://www.educba.com/lua-goto/
https://www.educba.com/lua-continue/
https://www.educba.com/lua-break/

## Learning Strategy for PHP:

I mostly used my previous knowledge from the previous homework and the following link was enough for me to learn break and continue usage in this language:
https://www.w3schools.com/php/php_looping_break.asp

## Learning Strategy for Python:

My previous experience with the Python was enough to do the Python part except the for … else … block usage which I did not know before. I used the following links to do the Python part.
https://peps.python.org/pep-3136/
https://stackoverflow.com/questions/189645/how-can-i-break-out-of-multiple-loops
https://medium.com/techtofreedom/5-ways-to-break-out-of-nested-loops-in-python-4c505d34ace7

## Learning Strategy for Ruby:

I mostly used online forums to search for the problems, but I rarely used the official doc of the Ruby since it is not that bad. I specifically search the problems I have encountered using following links: The documentation of the Ruby was not helpful and finding the unique way of dealing with labeled unconditional exits in Ruby takes lots of the time, but I managed to the Ruby part using the following links:
https://www.geeksforgeeks.org/catch-and-throw-exception-in-ruby/
https://www.geeksforgeeks.org/ruby-break-and-next-statement/
https://www.tutorialspoint.com/ruby/ruby_if_else.htm
https://www.ruby-forum.com/t/how-to-break-out-deeply-nested-loops-newbie/56389
https://stackoverflow.com/questions/1352120/how-to-break-outer-cycle-in-ruby
https://www.rubyguides.com/2012/01/ruby-string-formatting/

## Learning Strategy for Rust:

My main reference to learning Rust was its documentation since it is the most comprehensive docs I have ever seen. I mostly used the doc of the Rust since it is amazing and every knowledge I needed was there. I did not need much to look at other online resources. I specifically used the following links:
https://doc.rust-lang.org/std/macro.println.html

https://stackoverflow.com/questions/27893223/how-do-i-iterate-over-a-range-with-a-custom-step
https://doc.rust-lang.org/rust-by-example/flow_control/if_else.html
https://doc.rust-lang.org/std/keyword.break.html
https://doc.rust-lang.org/std/keyword.continue.html

## Online Compilers I have use (The workspaces are public so you can see the code via link):

I have used replit code environment to test all my codes except Lua since goto statements only available with the Lua compilers greater than 5.1 which is not currently available in replit.
Python: https://replit.com/@MrKty/21901413hw2python#main.py
Dart: https://replit.com/@MrKty/21901413hw2dart#main.dart
Javascript: https://replit.com/@MrKty/21901413hw2js#index.html
Lua: http://tpcg.io/_DVYIHB (tutorialspoint online compiler which is normally available at https://www.tutorialspoint.com/execute_lua_online.php )
PHP: https://replit.com/@MrKty/21901413hw2php#main.php
Ruby: https://replit.com/@MrKty/21901413hw2ruby#main.rb
Rust: https://replit.com/@MrKty/21901413hw2rust#src/main.rs

## References:

[1] *Re: Upcoming changes in lua 5.2 [was re: Location of a package]*. [Online]. Available: http://lua-users.org/lists/lua-l/2008-02/msg01183.html. [Accessed: 03-Dec-2022].

[2] “Python enhancement proposals,” *PEP 3136 – Labeled break and continue*. [Online]. Available: https://peps.python.org/pep-3136/. [Accessed: 03-Dec-2022].