Homework 2 Report

Pınar Yücel
CS 315-3
21802188

Sample Code Strategy Clarification

Firstly, I searched for the different types of loops which can be used as logically controlled loops in the specified languages. Secondly, I confirmed their location of tests. After that, I looked for different ways of controlling the loops and tested if they are working as specified. I tried to imitate functionalities that are not present in a specific language by other mechanisms.

Dart

Sample Code

```
void main()
{
  // while loop [1]
  print('while loop test');

   // I am testing the while loop with a boolean expression that is false. If it prints, then it is using
posttest. Otherwise, its pretest [1].
  while (false)
  {
    print('while loop uses posttest');
  }

  // I am using the while loop to print numbers that are between 1 and 10 [1].
  int counter = 1;

  while(counter <= 10)
  {
    print(counter);
    counter = counter + 1;
  }

  // do while loop [1]
  print('\ndo while loop test');

   // I am testing the do while loop with a boolean expression that is false. If it prints, then it is using
posttest. Otherwise, its pretest [1].
  do
  {
    print('do while loop uses posttest');
  }
  while(false);

  // I am using the do while loop to print numbers that are between 1 and 10 [1].
  counter = 1;
  do
  {
    print(counter);
    counter = counter + 1;
  }
  while(counter <= 10);

  // for loop test [1]
  print('\nfor loop test');

  // I am testing the for loop with a boolean expression that is false. If it prints, then it is using posttest.
Otherwise, its pretest [1].
  for(;false;)
  {
    print('for loop uses posttest');
  }

  // I am using for loop to print numbers that are between 1 and 10 [1].
  counter = 1;

  for(;counter <= 10;)
  {
    print(counter);
    counter = counter + 1;
  }

  // break
  print('\nbreak statement test');

  // I am testing the break statement. If the while loop prints 1 to 4, then the break statement causes
to program to exit the loop [1].
  counter = 1;

  while(counter <= 10)
  {
    if(counter == 5)
    {
      break;
    }

    print("$counter");
    counter++;
  }

  print('');
```

```
 // I am testing how the break statement works when used inside a while loop inside a while loop. If it
prints 1 to 10, then it causes the program to only exit the inner loop. If it prints 1 to 4, then it causes
the program to exit both loops [1].
 counter = 1;

 while(counter <= 10)
 {
  while(counter == 5)
  {
   break;
  }

  print("$counter");
  counter++;
 }

 print('');

 // I am testing how the break statement works when used inside a for loop that is inside a while loop.
If it prints 1 to 10, then it causes the program to only exit the inner loop. If it prints 1 to 4, then it
causes the program to exit both loops [1].
 counter = 1;

 while(counter <= 10)
 {
  for(;counter == 5;)
  {
   break;
  }

  print("$counter");
  counter++;
 }

 // I am testing the break statement with a label to exit to the outermost loop. If it prints 0 and 1, then
it is working [1].

 print("");
 int i = 0;
 int j = 0;
 int k = 0;

 breakLabel:
 while(i < 3)
 {
   print("i = ${i}");
   j = 0;
   while(j < 3)
   {
     k = 0;
     while(k < 3)
     {
       if(i == 1)
       {
         break breakLabel;
       }
       k++;
     }
     j++;
   }
   i++;
 }

 // continue
 print('\ncontinue statement test');

 // I am testing the continue statement. If the while loop prints 1 to 10 except for 5, then the continue
statement skips the subsequent statements in the current iteration and takes the control back to the
beginning of the loop [1].
 counter = 1;

 while(counter <= 10)
 {
  if(counter == 5)
  {
   counter++;
   continue;
  }

  print("$counter");
  counter++;
 }

  // I am testing the continue statement with a label to continue from the next iteration of the
outermost loop. If it prints 2 and 3, then it is working [1].
 print("");
 i = 0;
 j = 0;
 k = 0;

 continueLabel:
 while(i < 3)
 {
  // Even though it seems meaningless, I am using it like this for the testing purposes. Otherwise it will
be an infitine loop [1].
  i++;
  j = 0;
  while(j < 3)
  {
      j++;
      k = 0;
      while(k < 3)
      {
          k++;
          if(i == 1)
          {
              continue continueLabel;
          }
      }
   }
   print("i = ${i}");
 }
}
```

Result of Execution

```
while loop test
1
2
3
4
5
6
7
8
9
10

do while loop test
do while loop uses posttest
1
2
3
4
5
6
```

```
7
8
9
10

for loop test
1
2
3
4
5
6
7
8
9
10

break statement test
1
2
3
4

1
2
3
4
5
6
7
8
9
10

1
2
3
4
5
6
7
8
9
10

i = 0
```

```
i = 1

continue statement test
1
2
3
4
6
7
8
9
10

i = 2
i = 3
```

Javascript

Sample Code

```
<script>
  // while loop [2]
  document.write('while loop test<br>');

   // I am testing the while loop with a boolean expression that is false. If it prints, then it is using posttest. Otherwise its pretest [2].
  while (false)
  {
    document.write('while loop uses posttest');
  }

  // I am using the while loop to print numbers that are between 1 and 10 [2].
  counter = 1;

  while(counter <= 10)
  {
    document.write(counter + "<br>");
    counter = counter + 1;
  }

  // do while loop
  document.write('<br><br>do while loop test<br>');

   // I am testing the do while loop with a boolean expression that is false. If it prints, then it is using posttest. Otherwise its pretest [2].
```

```
 do
 {
   document.write('do while loop uses posttest<br>');
 }
 while(false);

 // I am using the do while loop to print numbers that are between 1 and 10 [2].
 counter = 1;

 do
 {
   document.write(counter + "<br>");
   counter = counter + 1;
 }
 while(counter <= 10);

 // for loop test [2]
 document.write('<br><br>for loop test<br>');

 // I am testing the for loop with a boolean expression that is false. If it prints, then it is using posttest.
Otherwise its pretest [2].
 for(;false;)
 {
   document.write('for loop uses posttest');
 }

 // I am using for loop to print numbers that are between 1 and 10 [2].
 counter = 1;

 for(;counter <= 10;)
 {
   document.write(counter + "<br>");
   counter = counter + 1;
 }

 // break [2]
 document.write('<br><br>break statement test<br>');

 // I am testing the break statement. If the while loop prints 1 to 4, then the break statement causes
to program to exit the loop [2].
 counter = 1;

 while(counter <= 10)
 {
   if(counter == 5)
   {
```

```
   break;
  }

  document.write(counter + "<br>");
  counter++;
 }

 counter = 1;
 document.write("<br>");

 // I am testing how the break statement works when used inside a while loop inside a while loop. If it
prints 1 to 10, then it causes the program to only exit the inner loop. If it prints 1 to 4, then it causes
the program to exit both loops [2].
 while(counter <= 10)
 {
  while(counter == 5)
  {
   break;
  }

  document.write(counter + "<br>");
  counter++;
 }

      document.write("<br>");

 // I am testing how the break statement works when used inside a for loop inside a while loop. If it
prints 1 to 10, then it causes the program to only exit the inner loop. If it prints 1 to 4, then it causes
the program to exit both loops [2].
 counter = 1;

 while(counter <= 10)
 {
  for(;counter == 5;)
  {
   break;
  }

  document.write(counter + "<br>");
  counter++;
 }

 // I am testing the break statement with a label to exit to the outermost loop. If it prints 0 and 1, then
it is working [2].
 document.write("<br>");
 i = 0;
 j = 0;
 k = 0;

 breakLabel:
 while(i < 3)
 {
    document.write("i = " + i + "<br>");
    j = 0;
    while(j < 3)
    {
       k = 0;
       while(k < 3)
       {
          if(i == 1)
          {
             break breakLabel;
          }
          k++;
       }
       j++;
    }
    i++;
 }

 // continue [2]
 document.write('<br>continue statement test<br>');
```

 // I am testing the continue statement. If the while loop prints 1 to 10 except for 5, then the continue statement skips the subsequent statements in the current iteration and takes the control back to the beginning of the loop [2].

```
 counter = 1;

 while(counter <= 10)
 {
   if(counter == 5)
   {
     counter++;
     continue;
   }

   document.write(counter + "<br>");
   counter++;
 }
```

  // I am testing the continue statement with a label to continue from the next iteration of the outermost loop. If it prints 2 and 3, then it is working [2].

```
  document.write("<br>");
  i = 0;
  j = 0;
  k = 0;

  continueLabel:
  while(i < 3)
  {
    // Even though it seems meaningless, I am using it like this for the testing purposes. Otherwise it will
be an infitine loop [2].
    i++;
    j = 0;
    while(j < 3)
    {
        j++;
        k = 0;
        while(k < 3)
        {
            k++;
            if(i == 1)
            {
                continue continueLabel;
            }
        }
     }
     document.write("i = " + i + "<br>");
  }
</script>
```

Result of Execution

```
while loop test
1
2
3
4
5
6
7
8
9
10


do while loop test
do while loop uses posttest
1
2
3
4
5
6
7
8
9
10


for loop test
1
2
3
4
5
6
7
8
9
10


break statement test
1
2
3
4

1
2
3
4
5
6
7
8
9
10

1
2
3
```

```
4
5
6
7
8
9
10

i = 0
i = 1

continue statement test
1
2
3
4
6
7
8
9
10

i = 2
i = 3
```

Lua

Sample Code

```
-- while loop [3]
print('while loop test');

-- I am testing the while loop with a boolean expression that is false. If it prints, then it is using posttest. Otherwise, its pretest [3].
while (false)
do
   print('while loop uses posttest');
end

-- I am using the while loop to print numbers that are between 1 and 10 [3].
counter = 1;

while (counter <= 10)
do
   print(counter);
   counter = counter + 1
end

-- repeat until loop [3]
print('\nrepeat until loop test');

-- I am testing the repeat until loop with a boolean expression that is true. If it prints, then it is using
posttest. Otherwise, its pretest [3].
repeat
   print('repeat until loop uses posttest');
until(true);

-- I am using the repeat until loop to print numbers that are between 1 and 10 [3].
counter = 1;

repeat
   print(counter);
   counter = counter + 1
until(counter > 10);

-- break statement test
print('\nbreak statement test');

--I am testing the break statement. If the while loop prints 1 to 4, then the break statement causes to
program to exit the loop [3].
counter = 1;

while(counter <= 10)
do
   if(counter == 5)
   then
     break;
   end

   print(counter);
   counter = counter + 1;
end

--I am trying to use a label to exit all of the loops. If it is not infinite loop, then it works [3][4].
print();

while(true)
do
   print("Inside first loop...");
   while(true)
   do
```

```
      print("Inside second loop...");
      while(true)
      do
         print("Inside third loop...");
         goto breakAll
      end
   end
end

::breakAll::

-- I am testing how the break statement works when used inside a loop inside loop. If it prints 1 to 10,
then it causes the program to only exit the inner loop. If it prints 1 to 4, then it causes the program to
exit both loops [3].
counter = 1;
print();

while(counter <= 10)
do
   while(counter == 5)
   do
     break;
   end

   print(counter);
   counter = counter + 1;
end


-- goto statement test [3][4]
print('\ngoto statement test');

-- I am trying to use goto statement like a regular continue statement. If it works 1 to 10 except 5, then
it works [3][4].
counter = 1;
::continue::

while(counter <= 10)
do
   if(counter == 5)
   then
      counter = counter + 1;
      goto continue
   end

   print(counter);
```

```
    counter = counter + 1;
end
```

Result of Execution

```
while loop test
1
2
3
4
5
6
7
8
9
10

repeat until loop test
repeat until loop uses posttest
1
2
3
4
5
6
7
8
9
10

break statement test
1
2
3
4

Inside first loop...
Inside second loop...
Inside third loop...

1
2
3
```

```
4
5
6
7
8
9
10

goto statement test
1
2
3
4
6
7
8
9
10
```

PHP

Sample Code

```
<!DOCTYPE html>
<html>
<body>

<?php
// while loop [5]
echo 'while loop test<br>';

//I am testing the while loop with a boolean expression that is false. If it prints, then it is using
posttest. Otherwise, it is pretest [5].
while (false)
{
   echo 'while loop uses posttest';
}

// I am using the while loop to print numbers that are between 1 and 10 [5].
$counter = 1;

while($counter <= 10)
{
   echo "$counter<br>";
   $counter = $counter + 1;
}

// do while loop [5]
echo '<br>do while loop test<br>';

// I am testing the do while loop with a boolean expression that is false. If it prints, then it is using
posttest. Otherwise, it is pretest [5].
do
{
   echo 'do while loop uses posttest';
}
while(false);

// I am using the do while loop to print numbers that are between 1 and 10 [5].
echo "<br>";
$counter = 1;

do
{
   echo "$counter<br>";
   $counter = $counter + 1;
}
while($counter <= 10);

// for loop test
echo '<br>for loop test<br>';

// I am testing the for loop with a boolean expression that is false. If it prints, then it is using posttest.
Otherwise, it is pretest [5].
for(;false;)
{
   echo 'for loop uses posttest';
}

// I am using for loop to print numbers that are between 1 and 10 [5].
$counter = 1;

for(;$counter <= 10;)
{
   echo "$counter<br>";
   $counter = $counter + 1;
}

// break statement [5]
echo '<br><br>break statement test<br>';
```

```
// I am testing the break statement. If the while loop prints 1 to 4, then the break statement causes to
program to exit the loop [5].
$counter = 1;

while($counter <= 10)
{
   if($counter == 5)
   {
      break;
   }

   echo "$counter<br>";
   $counter = $counter + 1;
}

$counter = 1;
echo "<br>";

// I am testing how the break statement works when used inside a loop inside loop. If it prints 1 to 10,
then it causes the program to only exit the inner loop. If it prints 1 to 4, then it causes the program to
exit both loops [5].
while($counter <= 10)
{
   while($counter == 5)
   {
         break;
   }

   echo "$counter<br>";
   $counter = $counter + 1;
}

// I am trying to use a label to exit all of the loops. If it is not infinite loop, then it works [5][6].
echo "<br>";

while(true)
{
         echo "Inside first loop...<br>";
   while(true)
   {
         echo "Inside second loop...<br>";
      while(true)
      {
         echo "Inside third loop...<br>";
        goto breakAll;
      }
```

```
    }
}

breakAll:

// continue statement [5]
echo '<br>continue statement test<br>';

// I am testing the continue statement. If the while loop prints 1 to 10 except for 5, then the continue statement skips the subsequent statements in the current iteration and takes the control back to the beginning of the loop [5].
$counter = 1;

while($counter <= 10)
{
    if($counter == 5)
    {
        $counter = $counter + 1;
        continue;
    }

    echo "$counter<br>";
    $counter = $counter + 1;
}

?>

</body>
</html>
```

Result of Execution

```
while loop test
1
2
3
4
5
6
7
8
9
10

do while loop test
do while loop uses posttest
1
2
3
4
5
6
7
8
9
10

for loop test
1
2
3
4
5
6
7
8
9
10


break statement test
1
2
3
4

1
2
3
4
5
6
7
8
9
10

Inside first loop...
Inside second loop...
Inside third loop...
```

```
continue statement test
1
2
3
4
6
7
8
9
10
```

Python

Sample Code

```
# while loop
print('while loop test');

# I am testing the while loop with a boolean expression that is false. If it prints, then it is using
posttest. Otherwise, it is pretest [7].
while False:
   print('while loop uses posttest');

# I am using the while loop to print numbers that are between 1 and 10 [7].
counter = 1;

while counter <= 10:
   print(counter);
   counter = counter + 1;

# do while loop imitation [7]
print("\ndo while loop imitation test");
counter = 1;

# I am trying to see how can I imitate the regular do while (posttest) loop in python with the break
statement [7].
while True:
   print(counter);
   if counter >= 10:
      break

   counter = counter + 1;

# break statement [7]
print('\nbreak statement test');
```

```
# I am testing the break statement. If the while loop prints 1 to 4, then the break statement causes to
program to exit the loop [7].
counter = 1;

while counter <= 10:
    if counter == 5:
        break;

    print(counter);
    counter = counter + 1;

counter = 1;
print('');

# I am testing how the break statement works when used inside a loop inside loop. If it prints 1 to 10,
then it causes the program to only exit the inner loop. If it prints 1 to 4, then it causes the program to
exit both loops [7].
while counter <= 10:
    while counter == 5:
        break;

    print(counter);
    counter = counter + 1;

# continue statement [7]
print('\ncontinue statement test');
counter = 1;

# I am testing the continue statement. If the while loop prints 1 to 10 except for 5, then the continue
statement skips the subsequent statements in the current iteration and takes the control back to the
beginning of the loop [7].
while counter <= 10:
    if(counter == 5):
      counter = counter + 1;
      continue;

    print(counter);
    counter = counter + 1;
```

Result of Execution

```
while loop test
1
2
```

```
3
4
5
6
7
8
9
10

do while loop imitation test
1
2
3
4
5
6
7
8
9
10

break statement test
1
2
3
4

1
2
3
4
5
6
7
8
9
10

continue statement test
1
2
3
4
6
7
8
```

```
9
10
```

Ruby

Sample Code

```
# while loop test [8]
puts "while loop test"

# I am testing the while loop with a boolean expression that is false. If it prints, then it is using
posttest. Otherwise, it is pretest [8].
while false
   puts "while loop uses posttest";
end

# I am using the while loop to print numbers between 1 and 10 [8].
counter = 1;

while counter <= 10
   puts counter;
   counter = counter + 1;
end

# until loop test [8]
puts "\nuntil loop test"

# I am testing the until loop with a boolean expression that is true. If it prints, then it is using posttest.
Otherwise, it is pretest [8].
until true
   puts "until loop uses posttest";
end

# I am using the until loop to print numbers between 1 and 10 [8].
counter = 1;

until counter == 11
   puts counter
   counter = counter + 1;
end

# do while loop imitation test [8]
puts "\ndo while loop imitation test";
```

```
# I am trying to confirm that this is a correct imitation of do while loop, meaning that it is using
posttest [8].
begin
  puts "this imitation is using posttest"
end while false

# I am tring to print numbers between 1 and 10 using the imitation [8].
counter = 1;

begin
  puts counter;
  counter = counter + 1;
end while counter <= 10

# I am tring another way to imitate do while loop [8].
counter = 1;
puts "";

loop do
  puts counter;
  counter = counter + 1;
 if counter > 10
  break
 end
end

# Posttest loop with until modifier [8].
counter = 1;
puts "";

begin
  puts counter;
  counter = counter + 1;
end until counter > 10

# break statement test
puts " \nbreak statement test"

# I am testing the break statement. If the while loop prints 1 to 4, then the break statement causes to
program to exit the loop [8].
counter = 1;

while counter <= 10
  if counter == 5
   break;
  end

  puts counter;
  counter = counter + 1;
end

counter = 1;
puts;

# I am testing how the break statement works when used inside a loop inside loop. If it prints 1 to 10, then it causes the program to only exit the inner loop. If it prints 1 to 4, then it causes the program to exit both loops [8].
while counter <= 10
  while counter == 5
   break;
  end

  puts counter;
  counter = counter + 1;
end

# next statement test [9]
puts " \nnext statement test"

counter = 1;

# I am testing the next statement. If it prints 1 to 10 except for 5, then it is working as expected [9].
while counter <= 10
  if counter == 5
    counter = counter + 1;
    next;
  end

  puts counter;
  counter = counter + 1;
end

# redo statement test [9]
puts " \nredo statement test"

# I am testing the redo statement. It should print twice if restarts the iteration of the most internal loop, without checking loop condition [9].
control = true;
while control
  puts "inside loop...";
  if control
    control = false;
    redo;
  end
end
```

Result of Execution

```
while loop test
1
2
3
4
5
6
7
8
9
10

until loop test
1
2
3
4
5
6
7
8
9
10

do while loop imitation test
this imitation is using posttest
1
2
3
4
5
6
7
8
9
10
```

```
2
3
4
5
6
7
8
9
10

1
2
3
4
5
6
7
8
9
10

break statement test
1
2
3
4

1
2
3
4
5
6
7
8
9
10

next statement test
1
2
3
4
6
7
8
```

```
9
10

redo statement test
inside loop...
inside loop...
```

Rust

Sample Code

```
fn main()
{
  // while loop [10]
  println!("while loop test");

   // I am testing the while loop with a boolean expression that is false. If it prints, then it is using
posttest. Otherwise it is pretest [10].
  while false
  {
    println!("while loop uses posttest");
  }
  // I am using the while loop to print numbers that are between 1 and 10 [10].
  let mut counter = 1;

  while counter <= 10
  {
    println!("{}", counter);
    counter = counter + 1;
  }

  // posttest loop imitation [10]
   println!("\nposttest loop imitation");

  // I am trying to imitate a posttest loop [10].
  counter = 1;

  loop
  {
    println!("{}", counter);
    counter = counter + 1;

    if counter > 10
    {
        break;
    }
  }

  // break statement test [10]
   println!("\nbreak statement test");

  // I am testing the break statement. If the while loop prints 1 to 4, then the break statement causes
to program to exit the loop [10].
  counter = 1;

  while counter <= 10
  {
    if counter == 5
    {
      break;
    }

    println!("{}", counter);
    counter = counter + 1;
  }

  println!();

  // I am testing how the break statement works when used inside a while loop inside a while loop. If it
prints 1 to 10, then it causes the program to only exit the inner loop. If it prints 1 to 4, then it causes
the program to exit both loops [10].
  counter = 1;

  while counter <= 10
  {
    while counter == 5
    {
      break;
    }

    println!("{}", counter);
    counter = counter + 1;
  }

  println!();

  // I am testing the break statement with a label to exit to the outermost loop. If it prints 0 and 1, then
it is working [10].

  let mut i = 0;
  let mut j = 0;
```

```
 let mut k = 0;

 'breakLabel:
 while i < 3
 {
   println!("i = {}", i);
   j = 0;
   while j < 3
   {
     k = 0;
     while k < 3
     {
       if i == 1
       {
         break 'breakLabel;
       }
       k = k + 1;
     }
     j = j + 1;
   }
   i = i + 1;
 }

 // continue [10]
 println!("\ncontinue statement test");

 // I am testing the continue statement. If the while loop prints 1 to 10 except for 5, then the continue statement skips the subsequent statements in the current iteration and takes the control back to the beginning of the loop [10].
 counter = 1;

 while counter <= 10
 {
  if counter == 5
  {
   counter = counter + 1;
   continue;
  }

  println!("{}", counter);
  counter = counter + 1;
 }

  // I am testing the continue statement with a label to continue from the next iteration of the outermost loop. If it prints 2 and 3, then it is working [10].
 println!();
```

```
  i = 0;
  j = 0;
  k = 0;

  'continueLabel:
  while i < 3
  {
    // Even though it seems meaningless, I am using it like this for the testing purposes. Otherwise it will
be an infitine loop [10].
    i = i + 1;
    j = 0;
    while j < 3
    {
        j = j + 1;
        k = 0;
        while k < 3
        {
            k = k + 1;
            if i == 1
            {
                continue 'continueLabel;
            }
        }
     }
     println!("i = {}", i);
  }
}
```

Result of Execution

```
while loop test
1
2
3
4
5
6
7
8
9
10

posttest loop imitation
1
2
```

```
3
4
5
6
7
8
9
10

break statement test
1
2
3
4

1
2
3
4
5
6
7
8
9
10

i = 0
i = 1

continue statement test
1
2
3
4
6
7
8
9
10

i = 2
i = 3
```

Language Evaluations

Dart

I think for loops being used like logically controled loops is a minus in terms of readability because if sounds meaningless when the name of the loop is considered. Furthermore, programmers are prone to making mistakes with typos with for loops that are being used as logically controlled loops because you still need to seperate the boolean part with semicolons. Hence, its a minus for writeability. Dart allows the use of break and continue with labels which I think decrease the readability, since you have to find where the labels are and writeability since people are prone to making errors because of the low levelness of the labels, of the language. However, I think the use of while and do while loops are pretty straightforward and intuitive which is a plus for the readability and writeability of the language. Dart supports the regular break and continue statements which increase the readability of the code because otherwise programmers need to use solutions such as using controller booleans that make the code less readable.

Javascript

Again, I think for loops being used like logically controled loops is a minus in terms of readability because if sounds meaningless when the name of the loop is considered. Furthermore, programmers are prone to making mistakes with typos with for loops that are being used as logically controlled loops because you still need to seperate the boolean part with semicolons. Hence, its a minus for writeability. Javascript allows the use of break and continue with labels which I think decrease the readability, since you have to find where the labels are and writeability since people are prone to making errors because of the low levelness of the labels, of the language. However, I think the use of while and do while loops are pretty straightforward and intuitive which is a plus for the readability and writeability of the language. Javascript supports the regular break and continue statements which increase the readability of the code because otherwise programmers need to use solutions such as using controller booleans that make the code less readable.

Lua

I think the use of while and repeat until loops are pretty straightforward and intuitive which is a plus for the readability and writeability of the language. Even though Lua supports the regular break statement, there is no continue statement in Lua which I think is a minus for writeability because when there is a break statement, I automatically think that there is also a continue statement, and may try to use it which results in an error. However, break statement increase the readability of the code because otherwise programmers need to use solutions such as using controller booleans that make the code less readable. Labels with goto statements can be used to imitate the continue statement but they are not really readable because you have to locate the label everytime to understand the code.

PHP

I think the use of while and do while loops are pretty straightforward and intuitive which is a plus for the readability and writeability of the language. PHP supports the regular break and continue statements which increase the readability of the code because otherwise programmers need to use solutions such as using controller booleans that make the code less readable. For loops being used like logically controled loops is a minus in terms of readability because if sounds meaningless when the name of the loop is considered. Furthermore, programmers are prone to making mistakes with typos with for loops that are being used as logically controlled loops because you still need to seperate the boolean part with semicolons.

Python

I think the use of while loop is pretty straightforward and intuitive which is a plus for the readability and writeability of the language. Python does not have a do while loop which is a minus in terms of writeability because it limits the programmers. To imitate a do while loop, break statement can be used, however, it makes the code harder to read. Python supports the regular break and continue statements which increase the readability of the code because otherwise programmers need to use solutions such as using controller booleans that make the code less readable.

Ruby

I think the use of while and until loops are pretty straightforward and intuitive which is a plus for the readability and writeability of the language. There are various ways to control the logically controlled loops in Ruby language (see code), which means that there are more to learn to understand the language which is a minus in terms of both readability and writeability.

Rust

The use of while loop is pretty straightforward and intuitive which is a plus for the readability and writeability of the language. However, the lack of do while loop is a minus. Imitating a posttest loop is unreadable because of its use of breaks and also it makes the language harder to write. Rust supports the regular break and continue statements which increase the readability of the code because otherwise programmers need to use solutions such as using controller booleans that make the code less readable.

Best Language

In my opinion, Lua is the best language for the logically controlled loops for me. It supports pretest and posttest loops that have meaningful names. You can not use for loops as logically controlled loops in Lua which is good as explained previously.

Learning Strategy

I used Google to learn how logically controlled loops are used in the specified languages. I tried to use the websites that are the either offical websites of the languages or known sites. I tried to avoid question ask sites such as stack overflow. I reported every experiment I performed as comments in the code. I did not continue experimenting without completely understanding an operation. I did not try anything random to see if it would work. I followed the information that I found on the internet. Since online compilers/interpreters were allowed, I just tested my code on the online compilers/interpreters whose link can be found below. According to those, outputs are as they should be.

Online Compilers/Interpreters
Dart : https://dartpad.dev/
Javascript : https://js.do/
Lua : https://www.jdoodle.com/execute-lua-online/
PHP : https://www.w3schools.com/php/phptryit.asp?filename=tryphp_compiler
Python : https://www.programiz.com/python-programming/online-compiler/
Ruby : https://www.onlinegdb.com/online_ruby_compiler
Rust : https://play.rust-lang.org/

References

[1] "Dart Programming - Loops," *Tutorialspoint*. [Online]. Available: https://www.tutorialspoint.com/dart_programming/dart_programming_loops.htm. [Accessed: 14-Dec-2020].
[2] *JavaScript Tutorial*. [Online]. Available: https://www.w3schools.com/js/DEFAULT.asp. [Accessed: 14-Dec-2020].
[3] "Lua - Loops," *Tutorialspoint*. [Online]. Available: https://www.tutorialspoint.com/lua/lua_loops.htm. [Accessed: 14-Dec-2020].
[4] "Goto Statement," *lua*. [Online]. Available: http://lua-users.org/wiki/GotoStatement. [Accessed: 14-Dec-2020].
[5] *PHP Tutorial*. [Online]. Available: https://www.w3schools.com/php/default.asp. [Accessed: 14-Dec-2020].
[6] "goto - Manual," *php*. [Online]. Available: https://www.php.net/manual/tr/control-structures.goto.php. [Accessed: 14-Dec-2020].
[7] *Python While Loops*. [Online]. Available: https://www.w3schools.com/python/python_while_loops.asp. [Accessed: 14-Dec-2020].
[8] *Loops in Ruby - performing repeated operations on a data set*. [Online]. Available: https://launchschool.com/books/ruby/read/loops_iterators. [Accessed: 14-Dec-2020].
[9] T. Cadier, "Ruby's redo, retry and next keywords," *AppSignal Blog*. [Online]. Available: https://blog.appsignal.com/2018/06/05/redo-retry-next.html. [Accessed: 14-Dec-2020].
[10] "Rust By Example," *Flow of Control - Rust By Example*. [Online]. Available: https://doc.rust-lang.org/rust-by-example/flow_control.html. [Accessed: 14-Dec-2020].