CS 315
Homework Assignment 2
Turan Mert Duran
21601418
Section 2

```dart
void main(){
  print("-------*-----Pretest-----WHİLE--*--------");
  var num = 10;
  while(num>=0){
    if( num == 8){
      num--;
      print("-------*-----Continue at 8-----*--------");

      continue;
    }
    if(num % 6 == 0){
      print("-------*-----Break at 6-----*--------");
      break;
    }

    print(num);
    num--;
  }

    print("-------*-----Pretest-----FOR--*--------");
  for( var i = 10 ; i >= 0; i-- ) {

      if( i == 8){
      print("-------*-----Continue at 8-----*--------");

      continue;
    }
      if(i % 6 == 0){
      print("-------*-----Break at 6-----*--------");
      break;
    }


    print(i) ;

   }
print("-------*-----Protest-----*--------");
```

```
  var n = 10;
  do{
   if( n == 8){
      n--;
      print("-------*-----Continue at 8-----*--------");

      continue;
    }
    if(n % 6 == 0){
      print("-------*-----Break at 6-----*--------");
      break;
    }

    print(n);
    n--;
  }while(n>=0);
}

//https://www.tutorialspoint.com/dart_programming/dart_programming_continue_statement.htm
//https://www.tutorialspoint.com/dart_programming/dart_programming_do_while_loop.htm
```

Output

```
-------*-----Pretest-----WHILE--*--------
10
9
-------*-----Continue at 8-----*--------
7
-------*-----Break at 6-----*--------
-------*-----Pretest-----FOR--*--------
10
9
-------*-----Continue at 8-----*--------
7
-------*-----Break at 6-----*--------
-------*-----Protest-----*--------
10
9
-------*-----Continue at 8-----*--------
7
-------*-----Break at 6-----*--------
```

My Dart code is firstly showing how the pretest logically controlled loop of Dart language works. Dart language is using while (condition test) {code block} structure and for(; cond_test;) {code block} for pretest logically controlled loops [1]. Loop starts by checking first condition test. For controlling loop, there are two different options, break statement and continue statement. By using continue statement in loop, as example of it can be seen from above (program continues at when num is 8) programmers are able to just jump to beginning of next iteration and skipping all remaining codes in this iteration. By using break statement in loop, as it can be seen from above (program breaks at when num is equal to 6) programmers are able to terminate the loop immediately. Dart language is using do{code block}while(condition test) structure for posttest logically controlled loops [2]. This is similar to while loop with an exception that loop starts by executing code block before checking condition. Loop controlling mechanisms are same and available in posttest logically loops too.


```
<html>

<body>

  <script>
  document.write('-------*-----Pretest---While--*--------');
  let num = 10;
  while(num>=0){
    if( num == 8){
      num--;
      document.write("<br>");
      document.write('-------*-----Continue at 8-----*--------');
      continue;
    }
    if(num % 6 == 0){
        document.write("<br>");
        document.write('-------*-----Break at 6-----*--------');
        document.write("<br>");
      break;
    }
    document.write("<br>");
    document.write(num);
    num--;
```

```
  }
  document.write("<br>");
  document.write("<br>");
  document.write('-------*-----Pretest---For--*--------');
  document.write("<br>");

  for (var i = 10; i > 0; i--) {

    if( i == 8){
      document.write('-------*-----Continue at 8-----*--------');
      document.write("<br>");
      continue;
    }
    if(i % 6 == 0){
        document.write('-------*-----Break at 6-----*--------');
        document.write("<br>");
      break;
    }
    document.write(i);
    document.write("<br>");
  }
  document.write("<br>");
  document.write("<br>");
  document.write('-------*-----Protest-----*--------');
  let n = 10;
  do{
   if( n == 8){
      n--;
      document.write("<br>");
      document.write('-------*-----Continue at 8-----*--------');
      continue;
    }
    if(n % 6 == 0){
        document.write("<br>");
        document.write('-------*-----Break at 6-----*--------');
        document.write("<br>");
        break;
    }
    document.write("<br>");
    document.write(n);
    n--;
  }while(n>=0);

  </script>

</body>

</html>
```

Output

```
-------*-----Pretest---While--*--------
10
9
-------*-----Continue at 8-----*--------
7
-------*-----Break at 6-----*--------


-------*-----Pretest---For--*--------
10
9
-------*-----Continue at 8-----*--------
7
-------*-----Break at 6-----*--------


-------*-----Protest-----*--------
10
9
-------*-----Continue at 8-----*--------
7
-------*-----Break at 6-----*--------
```

My Javascript code is firstly showing how the pretest logically controlled loop of Javascript language works. Javascript language is using while (condition test) {code block} structure and for (; cond_test;) {code block} for pretest logically controlled loops as how it works in Dart language [3]. Pretest logically controlled loop starts by checking first condition test. For controlling loop, there are two different options, break statement and continue statement [4]. By using continue statement in loop, as example of it can be seen from above (program continues at when num is 8) programmers are able to just jump to beginning of next iteration and skipping all remaining codes in this iteration. By using break statement in loop, as it can be seen from above (program breaks at when num equals to 6) programmers are able to terminate the loop immediately. Javascript language is using do{code block}while(condition test) structure for posttest logically controlled loops [3]. This is similar to while loop with an

exception that loop starts by executing code block before checking condition as how it is in Dart Language. Loop controlling mechanisms are same and available in posttest logically loops too.


```
in.lua
n = 10
    print("-------*-----Pretest-----*--------")

while( n > 0 )
do
    if( n == 5)
    then
        print("-------*-----Break at
        5-----*--------")
       break
    end
    print(n)
    n = n -1
end

    print("-------*-----Posttest-----*--------")
num = 10;
repeat
    if(num == 5)
    then
        print("-------*-----Break at
        5-----*--------")
       break
    end
     print(num)
     num = num - 1
 until( num < 0 )
```

```
-------*-----Pretest-----*--------
10
9
8
7
6
-------*-----Break at 5-----*--------
-------*-----Posttest-----*--------
10
9
8
7
6
-------*-----Break at 5-----*--------
```

My Lua code is firstly showing how the pretest logically controlled loop of Lua language works. Javascript language is using while (condition test) do [code_block] end structure for pretest logically controlled loops as how it works in Dart language [5]. Pretest logically controlled loop starts by checking first condition test. For controlling loop, there are just one option, break statement. Unfortunately, there is nothing for continue and skipping the iteration and jumping to the beginning of the other iteration. By using break statement in loop, as it can be seen from above (program breaks loop at when num equals to 5) programmers are able to terminate the loop immediately [6]. Lua language is using repeat [code_block] until (condition test) structure for posttest logically controlled loops [7]. This is similar to while loop with an exception that loop starts by executing code block before checking condition. The difference that distinguishes Lua language from JS and Dart is syntax of posttest logically controlled loop. Language is including while-do for pretest but not including do-while for posttest. Loop controlling mechanism is same and available in posttest logically loops too.

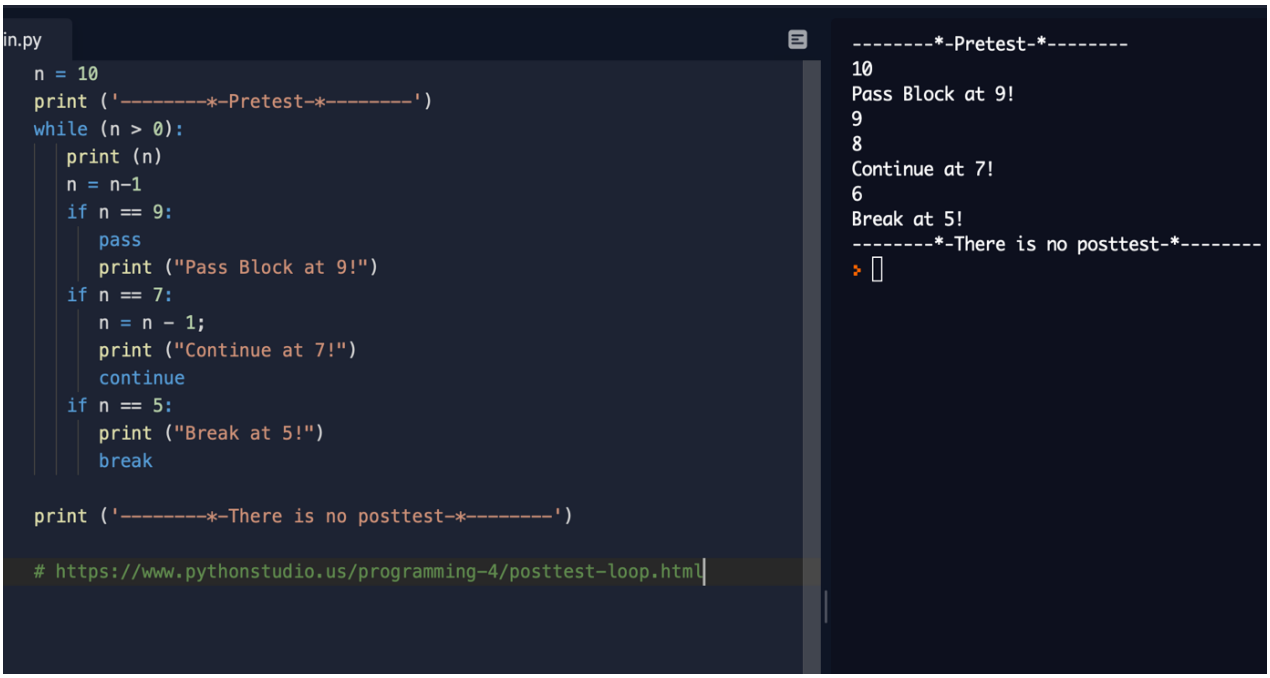

```php
<?php
$n = 10;
      echo("-----*-Pretest---While--*-------");
while($n > 0){
        echo("\r\n");
    if( $n == 7 ){
      $n--;
      echo ("-------*---continue at 7 *--------");
      continue;
    }
    if( $n == 5 ){
      echo ("-------*---Break at 5 *--------");
      break;
    }
    echo $n ;
    $n--;
}
echo("\r\n");
echo("-----*-Pretest---For---*-------");
for ($i = 10; $i > 0; $i--) {
```

```
  echo("\r\n");
  if( $i == 7 ){
    echo ("-------*---continue at 7 *--------");
    continue;
  }
  if( $i == 5 ){
    echo ("-------*---Break at 5 *--------");
    break;
  }
  echo $i ;
}
      echo("\r\n");
      echo("-----*-Posttest-*-------");

$num = 10;
do{
    echo("\r\n");
    if( $num == 7 ){
      $num--;
      echo ("-------*---continue at 7 *--------");
      continue;
    }
    if( $num == 5 ){
      echo ("-------*---Break at 5 *--------");
      break;
    }
    echo $num;
    $num--;
}
while($num > 0);
?>
```

Output

```
-----*-Pretest---While--*-------
10
9
8
-------*---continue at 7 *--------
6
-------*---Break at 5 *---------
-----*-Pretest---For---*-------
10
9
8
-------*---continue at 7 *--------
6
-------*---Break at 5 *---------
-----*-Posttest-*-------
10
9
8
-------*---continue at 7 *--------
6
-------*---Break at 5 *---------
```

My works. PHP language is using while (condition test) {code_block} structure and for(; (condition test);) {code_block} for pretest logically controlled loops as how it works in Dart language [8]. Pretest logically controlled loop starts by checking first condition test. For controlling loop, there are 2 options, break statement and continue statement. By using continue statement in loop, as example of it can be seen from above (program continues at when n is 7) programmers are able to just jump to beginning of next iteration and skipping all remaining codes in this iteration. By using break statement in loop, as it can be seen from above (program breaks at when n equals to 5) programmers are able to terminate the loop immediately [8]. PHP language is using do{code block}while(condition test) structure for posttest logically controlled loops [8]. This is similar to while loop with an exception that loop starts by executing code block before checking condition as how it is in Javascript Language. Loop controlling mechanisms are same and available in posttest logically loops too. PHP code is firstly showing how the pretest logically controlled loop of PHP language



My Python code is firstly showing how the pretest logically controlled loop of Python language works. Python language is using while (condition test) [code_block] structure for pretest logically controlled loops [9]. Pretest logically controlled loop starts by checking first condition test. For controlling loop, there are 3 options [10], pass statement break statement and continue statement. By using pass statement nothing is executed. As I understood, it is a null statement just for used as TODO comment. By using continue statement in loop, as example of it can be seen from above (program continues at when n is 7) programmers are able to just jump to beginning of next iteration and skipping all remaining codes in this iteration. By using break statement in loop, as it can be seen from above (program breaks at when n equals to 5) programmers are able to terminate the loop immediately [10]. Unfortunately, Python language does not include any posttest loop [11].

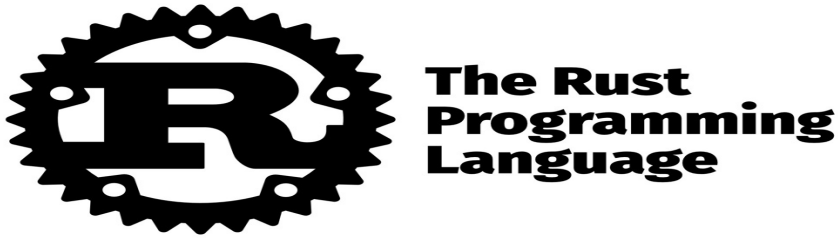

```ruby
$n = 10
  puts("--------*-Pretest-*--------")

while $n > 0  do
  if $n == 9 then
    puts("--------*-Next at 9-*--------")
    $n = $n-1
    next
  end
  if $n == 5 then
    puts("--------*-Break at 5-*--------")
    break
  end

  puts($n)
  $n = $n - 1
end
puts("--------*-Pretest---Until-do*--------")
$i = 10
until $i < 0  do
  if $i == 9 then
    puts("--------*-Next at 9-*--------")
    $i = $i-1
    next
```

```
  end

  if $i == 5 then
    puts("--------*-Break at 5-*--------")
    break
  end
puts($i)
$i = $i - 1
end

  puts("")
  puts("--------*-Posttest---Begin - while*--------")
$num = 10
begin
  if $num == 9 then
    puts("--------*-Next at 9-*--------")
    $num = $num-1
    next
  end
  if $num == 5 then
    puts("--------*-Break at 5-*--------")
    break
  end
   puts($num)
   $num -= 1
end while $num > 0
```

Output

```
--------*-Pretest-*--------
10
--------*-Next at 9-*--------
8
7
6
--------*-Break at 5-*--------
--------*-Pretest---Until-do*--------
10
--------*-Next at 9-*--------
8
7
6
--------*-Break at 5-*--------

--------*-Posttest---Begin - while*-------
10
--------*-Next at 9-*--------
8
7
6
--------*-Break at 5-*--------
```

My Ruby code is firstly showing how the pretest logically controlled loop of Ruby language works. Ruby language is using while condition test do code_block end OR until condition test do code_block end structure for pretest logically controlled loops [12]. Pretest logically controlled loop starts by checking first condition test. For controlling loop, there are 2 options, break statement and next statement. By using next statement in loop, as example of it can be seen from above (program next at when n is 9) programmers are able to just jump to beginning of next iteration and skipping all remaining codes in this iteration. By using break statement in loop, as it can be seen from above (program breaks at when n equals to 5) programmers are able to terminate the loop immediately. There are actually two more way of controlling loop retry and redo statements I couldn't show it on codes. Because retry and redo statements of Ruby causes infinite loops. Ruby language is using begin code_block end while condition test structure for posttest logically controlled loops [12]. This is similar to while loop with an exception that loop starts by executing code block before checking condition. Loop controlling mechanisms are same and available in posttest logically loops too.




```rust
fn main() {
    println!("------*-Pretest-*-------");
    let mut num = 10;
    while num > 0{
        if num == 5 {
            println!("------*-Break at 5-*-------");
            break;
        }
        if num == 8 {
            num = num - 1;
            println!("------*-Continue at 8-*-------");
            continue;
        }

        println!("{}",num);
        num-=1;
    }
    println!("");
    println!("------*-Posttest-*-------");
    let mut n = 10;
    loop {
        if n == 5 {
            println!("------*-Break at 5-*-------");
            break;
        }
        if n == 8 {
            n = n - 1;
            println!("------*-Continue at 8-*-------");
            continue;
        }

        println!("{}",n);
        n-=1;
    }
}
```

```
> ./main
------*-Pretest-*-------
10
9
------*-Continue at 8-*-------
7
6
------*-Break at 5-*-------

------*-Posttest-*-------
10
9
------*-Continue at 8-*-------
7
6
------*-Break at 5-*-------
>
```

My Rust code is firstly showing how the pretest logically controlled loop of Rust language works. Rust language is using while condition_test {code_block} structure for pretest logically controlled loops [13]. Pretest logically controlled loop starts by checking first condition test. For controlling loop, there are 2 options, break statement and next statement. By using next statement in loop, as example of it can be seen from above (program next at when n is 8) programmers are able to just jump to beginning of next iteration and skipping all remaining codes in this iteration. By using break statement in loop, as it can be seen from above (program breaks at when n equals to 5) programmers are able to terminate the loop immediately. Rust language does not include any posttest logically controlled loop[13] but by using break and infinite loop (that Rust offers) posttest logically controlled loop can be created by programmers. Example of it can be seen from above. Condition test of loop can be done by if condition of the break statement.

Evaluation of these languages in terms of readability and writability of logically controlled loops

In terms of readability and writability Dart, Lua and PHP programming languages were easy to write and read. Implementation of logically controlled were easy and understandable not too complex and similar with each other. Two of them except Lua have continue and break statements for controlling loops. By just looking example codes of these languages, I understand their syntax and started to write code. I think Dart and PHP would be best programming language among others. Because in other languages I forced to write logically controlled that took my time. However, in Dart and PHP syntax of it is so easy to write and understandable. The worst of among these languages in terms of readability and writability was the Rust and Python programming languages. There was no posttest logically controlled loop in these languages. Programmers forced to write their own posttest loops by using breaks and pretest loops. Remaining programming languages that I have never mentioned are not too easy to write and read as PHP Lua and Dart but they were not much difficult as Rust language.

My Learning Strategy

I started firstly by looking what logically controlled loops are. Then, I searched examples of logically controlled loops in different programming languages on Google. Actually, as I see there is not much different between different programming languages when writing logically controlled loops. Almost half of programming languages that I have skimmed their syntax of logically controlled loops were using same syntax and includes while-do or do-while structure. Moreover, almost all of them using break with same structure and in each of them meaning and functions of break is same. After searching languages on Google, I found a website called Tutorials Point (link is available at below the title of informative websites). This website was including syntaxes and information about logically controlled loops almost in all languages. I used Repl.it(link is available at below the title of Compilers/Interpreters) for compiling languages. It is including almost all of the languages' compilers or interpreters. Thanks to repl.it, I haven't forced to download compilers of all languages. On the other hand, I used VSCode that offers syntax support for all languages to write all languages.

Compilers / Interpreters:

- https://repl.it for (Dart, Lua, Rubby, PHP, Rust, Python) *(Acc. Date: 09.12.2020)*
- Google Chrome (for JS&HTML)

Informative websites:

- stackoverflow (https://stackoverflow.com/questions/9279768/how-do-i-loop-over-a-hash-of-hashes-in-ruby) *(Acc. Date: 09.12.2020)*
- W3schools (https://www.w3schools.com/python/python_dictionaries_loop.asp) *(Acc. Date: 09.12.2020)*
- Tutorial Republic (https://www.tutorialrepublic.com) *(Acc. Date: 09.12.2020)*
- Python Studio (https://www.pythonstudio.us)*(Acc. Date: 09.12.2020)*

- Tutorials Point

  (https://www.tutorialspoint.com) *(Acc. Date: 09.12.2020)*

References:

[1] "Dart Programming - While Loop (n.d.). Retrieved December 09,2020, from www.tutorialspoint.com/dart_programming/dart_programming_while_loop.htm.

[2] Dart Programming - do while Loop. (n.d.). Retrieved December 09, 2020, from https://www.tutorialspoint.com/dart_programming/dart_programming_do_while_loop.htm

[3] JavaScript - While Loops. (n.d.). Retrieved December 09, 2020, from https://www.tutorialspoint.com/javascript/javascript_while_loop.htm

[4] JavaScript - Loop Control. (n.d.). Retrieved December 09, 2020, from https://www.tutorialspoint.com/javascript/javascript_loop_control.htm

[5] Lua - while Loop. (n.d.). Retrieved December 09, 2020, from https://www.tutorialspoint.com/lua/lua_while_loop.htm

[6] Lua - break Statement. (n.d.). Retrieved December 09, 2020, from https://www.tutorialspoint.com/lua/lua_break_statement.htm

[7] Lua - repeat...until Loop. (n.d.). Retrieved December 09, 2020, from https://www.tutorialspoint.com/lua/lua_repeat_until_loop.htm

[8] PHP - Loop Types. (n.d.). Retrieved December 09, 2020, from https://www.tutorialspoint.com/php/php_loop_types.htm

[9](n.d.). Retrieved December 09, 2020, from https://www.w3schools.com/python/python_while_loops.asp

[10] 4. More Control Flow Tools¶. (n.d.). Retrieved December 09, 2020, from https://docs.python.org/3/tutorial/controlflow.html

[11] Correia, M. (2020, December 05). Post Test Loop - Python Programming. Retrieved December 09, 2020, from https://www.pythonstudio.us/programming-4/posttest-loop.html

[12] Ruby - Loops. (n.d.). Retrieved December 10, 2020, from https://www.tutorialspoint.com/ruby/ruby_loops.htm

[13] Rust - Loop. (n.d.). Retrieved December 10, 2020, from https://www.tutorialspoint.com/rust/rust_loop.htm