# CS 315
# Homework 2

**Counter-Controlled Loops in Dart, Javascript, PHP, Python, and Rust**

Zeynep Cankara
ID: 21703381
Section 1

## 1. Design Issues

### 1. What are types of loop control variables?

- **Dart:**
- Dart supports counter-controlled loops which **iterates over Iterables**, such type of counter-controlled loops can iterate over **Lists, Sets, keys of Maps** in Dart which have **loop control variables with built-in data types Numbers, Strings, Booleans**.
- **Program:**

```
// iterables
var listCourses = ["CS315", 2, true, 0.0];
print("For-Loop itrating over an Iterable (List)");
for (var i in listCourses){
  if(i is int){
    print("type loop (for) control variable: int");
  }
  if(i is double){
    print("type loop (for) control variable: double");
  }
  if(i is bool){
    print("type loop (for) control variable: Bool");
  }
   if(i is String){
    print("type loop (for) control variable: String");
  }
}
print("For-Loop itrating over an Iterable (Sets)");
var names = <String>{};
names.add("Zeynep");
for (var name in names){
  print("type loop (for) control variable: String");
}
```

- **Output**

```
For-Loop itrating over an Iterable (List)
type loop (for) control variable: String
type loop (for) control variable: int
type loop (for) control variable: double
type loop (for) control variable: Bool
type loop (for) control variable: int
type loop (for) control variable: double
For-Loop itrating over an Iterable (Sets)
type loop (for) control variable: String
```

- Loop control variable type can be either **double, int** for counter-controlled loop structures **(while, for, do-while)**.
- **Program:**

```
// While-Loops
// double, int
print("Counter Controlled While Loop");
int j = 0;
while(j < 1){
  if(j is int){
    print("type loop (while) control variable: int");
  }
  j += 1;
}
double i = 0.0;
while(i < 1){
  if(i is double){
    print("type loop (while) control variable: double");
  }
  i += 1;
}
// Do-While Loops
print("Counter Controlled Do-While Loop");
// double, int
int k = 2;
do{
  print("type loop (do-while) control variable: int");
  k -= 1;
}while(k > 1);
double z = 0.0;
do{
  print("type loop (do-while) control variable: double");
  z -= 1;
}while(z == 0);
// For-Loops
print("Counter Controlled For-Loop");
// double
for(double i = 0.0; i < 1; i++){
  if(i is double){
    print("type loop (for) control variable: double");
  }
}
// int
for(int i = 0; i < 1; i++){
  if(i is int){
    print("type loop (for) control variable: int");
  }
}
```

- **Output**

```
Counter Controlled While Loop
type loop (while) control variable: int
type loop (while) control variable: double
Counter Controlled Do-While Loop
type loop (do-while) control variable: int
type loop (do-while) control variable: double
Counter Controlled For-Loop
type loop (for) control variable: double
type loop (for) control variable: int
```

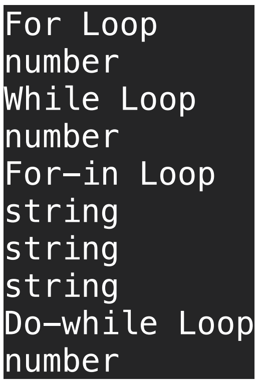
- **Javascript:**

- The example program demonstrates types of loops together with their data type of the loop control variables. `For, While, Do-While Loops` can take loop control variables of primitive data type `Number`. Javascript has a special For loops called as `For-in` loop which iterates over sequential data and has a loop control variable iterator. The example demonstrates that the For-in loop control variable data type being `string`.

- **Program**

```
// Question 1
console.log("For Loop")
for (let i = 0; i < 1; i++) {
  console.log(typeof i);
}
var counter = 1
console.log("While Loop")
while (counter >= 1){
  console.log(typeof counter);
  counter -= 1;
}
// looping over an iterable
var classes = ["CS315", 23, undefined];
console.log("For-in Loop")
for (i in classes) {
  console.log(typeof i);
}
console.log("Do-while Loop")
var i = 0;
do {
  i += 1;
  console.log(typeof i);
} while (i < 1);
```

- **Output**



```
For Loop
number
While Loop
number
For-in Loop
string
string
string
Do-while Loop
number
```

- **PHP:**
- The loop control variables can be data types `double or integer.` In the example programs, in the first loop the loop control variable has data type `double` where in the second loop the loop control variable has data type `integer`.
- **Program**

```
for ($q1 = 0.0; $q1 < 1.0; $q1++) {
    echo gettype($q1), "<br>";
}
for ($q1 = 0; $q1 < 1; $q1++) {
    echo gettype($q1), "<br>";
}
```

- **Output**

```
double
integer
```

- **Python:**
- The `for loops` in Python are iterates over items of any sequence (string or list). **Everything in Python is an object**. Type of a loop control variable can be anything an iterator pointing at; including data types of **Numbers, Strings, Lists, Tuples, Dictionaries, Functions**. For `while loops,` **Number types** such as **int, float** can be used as loops control variable which are both objects which can be understood by investigating how data types. associated with classes in the outputs of the following programs.

- **Program:**

```
    list_q1 = ["CS315", 2.3, 0, {'name': 'Zeynep'}, (2,3), 'a', [2,3], print]
    print("For-Loop")
    for element in list_q1:
        print("Type of the loop control variable: ", type(element))
```

- **Output:**

```
For-Loop
Type of the loop control variable:  <class 'str'>
Type of the loop control variable:  <class 'float'>
Type of the loop control variable:  <class 'int'>
Type of the loop control variable:  <class 'dict'>
Type of the loop control variable:  <class 'tuple'>
Type of the loop control variable:  <class 'str'>
Type of the loop control variable:  <class 'list'>
Type of the loop control variable:  <class
'builtin_function_or_method'>
```

- **Program:**

```
    counter = 1
    while counter < 2:
        print("Type of the loop control variable: ", type(counter))
        counter += 1
    counter = 1.0
    while counter < 2.0:
        print("Type of the loop control variable: ", type(counter))
        counter += 1.0
    return 0
```

- **Output**

```
While-Loop
Type of the loop control variable:  <class 'int'>
Type of the loop control variable:  <class 'float'>
```

- **Rust:**

```
for var in expression {
    code}
```

- The loop structure in Rust is different than loops in languages such as C. **The items in the expression is converted into an iterator**. The iterator gives series elements. The value of elements are bound to var during each iteration of the loop. **Thus, data type of the loop control variable becomes whatever the iterator is pointing at.**

## 2. What are the scopes of loop control variables?

- **Dart:**
- Scope of a  loop control variable limited to the **loop or block** the variable declared. In the example outside of the loop the variable i2 is not accessible.
- **Program**

```
  for(int i2=0;i2 < 2;i2++){
   print(i2);
 }
 // print(i2); Results in an compilation error
```

- **Output**

```
0
1
```

- **Javascript:**
- The loop control variable’s scope depends on the initialization expression. If the variable declared with `var` keyword then it will have wither `global or function scope`. If the variable declared with `let` keyword, the variable will have `blocked scope` which will only allow access within the loop. The example demonstrates how the loop variable `i2` initialized with `let` keyword has blocked scope and not accessible outside the loop and variable `i3` initialized with `var` keyword has a global scope. Thus, accessible outside the loop.
- **Program**

```
for (let i2 = 0; i2 < 1; i2++) {
  i2 = i2+1
  console.log(i2);
}
try{console.log(i2)}
catch(err){
  console.log("Can't access variable outside loop")
}
for (var i3 = 0; i3 < 1; i3++) {
  i3 = i3+1
  console.log(i3);
}
try{
  console.log(i3)
  console.log("Can access variable outside loop")
}
catch(err){
  console.log("Can't access variable outside loop")
}
```

- **Output**

```
1
Let init (Block Scope): Can't access variable outside loop
1
2
Var init (Global Scope): Can access variable outside loop
```

- **PHP:**
- PHP loop control variables has **global scope**. In the example program variable `a` accessible upon exiting the loop even though it is declared within the loop initialization part of the for loop.
- **Program**

```
echo "<br> Q2 ";
for ($a = 0; $a <= 5; $a++) {
    echo "<br> a = $a";
}
echo "<br> Accessing the loop control variable outside of the loop <br>";
echo "a = $a <br>";
```

- **Output**

```
a = 0
a = 1
a = 2
a = 3
a = 4
a = 5
Accessing the loop control variable outside of the loop
a = 6
```

- **Python:**
- Python has for loop control variables **local to the loop**. The **issue with control variables scope is local to the loop arises when the loop variable is still referenceable  upon exiting the loop**. Thus, I found out that upon exiting the loop, final value of the loop control variable is copied into the surrounding scope. Loop control variable will not change unless it is not referenced by a nested function defined in the loop body. The example program shows how new scope for the loop control variable. created when referenced within the loop and how the final value copied to the

outside scope of the loop to make the loop control variable referenceable outside the loop.

- **Program**

```
    for i in range(3):
        # both i and i2 shoud be same
        i = i + 2
        print("value of loop control variable: ", i)
    try:
        print("ACCESS to the loop control variable from outside of the loop")
        print("value of loop control variable:  ", i)
    except:
         print(" NO ACCESS to the loop control variable from outside of the loop")
```

- **Output**

```
value of loop control variable:  2
value of loop control variable:  3
value of loop control variable:  4
ACCESS to the loop control variable from outside of the loop
value of loop control variable:   4
```

- **Rust:**
- Scope of a  loop control variable limited to the **loop or block** the variable declared. In the example outside of the loop the variable x is not accessible.
- **Program**

```
      for x in 0..3 {
        println!("{}", x); // x: i32
    }
    //println!("{}", x); // not accessible outside of the loop
```

- **Output**

```
0
1
2
```

3. **Is it legal for the loop control variable or loop parameters to be changed in the loop, and if so, does the change affect loop control?**

- **Dart:**
- **It is legal for loop control variable and loop control parameters to changed within the loop**. In the example the loop control variables value advanced by one in each iteration. to decrease the number of iterations. Thus, it can change the loop control.
- **Program**

```
int start = 0;
int end = 5;
for(int i=start;i<end;i++){
  i =  i + 1;
  print(i);
}
```

- **Output**

```
1
3
5
```

- **Javascript:**
- Loop condition *evaluated at each iteration of the loop*. In the example program we can see that loop control variable `i4` *can be changed within the loop*, affecting the loop control. I manipulated the loop control variable to skip a step at each iteration by advancing it's value by one.
- **Program**

```
for (let i4 = 0; i4 < 5; i4++) {
  i4 = i4+1
  console.log("Current i4 (loop control variable) value: ",i4);
}
```

- **Output**

```
Current i4 (loop control variable) value: 1
Current i4 (loop control variable) value: 3
Current i4 (loop control variable) value: 5
```

- **PHP:**
- It is **legal** but in PHP `for` Loops, the. **initial expression executed once at the beginning** of the loop. When the loop control variable reassigned to another value within the loop, the loop control variable no longer updated by the loop-end expression. Example program goes into an infinite loop without the break statement since the loop-end expression will not be re-evaluated once the loop control variable assigned to a new value within the loop
- **Program**

```
for ($a = 0; $a <= 10; $a++) {
    $a =  8; // will not update a since now $a refers new variable causing infinite loop
    echo "a = $a";
    break;
}
```

- **Output**

```
a = 8
```

- **Python:**
- Allows the iterable object to be changed within the loop which in turn affects the control flow of the loop. *Example Program 1* demonstrates how we can add items to a list which the loop iterates over. It causes an infinite loop if I did not exit the loop after size of the list exceeds 5 showing that loop parameters can be manipulated. *Example Program 2* demonstrates that in Python loop control variable can't changed within the loop since by examining the program output it can be seen that loop control variable binds in the end of first iteration and can't be manipulated except the first iteration.

- **Program 1**

```
    list_exp = [1,2,3]
    for _ in list_exp:
        list_exp.append(2)
        if len(list_exp) > 5:
            print("list modified...")
            break
```

- **Output 1**

```
list modified…
```

- **Program 2**

```
    for i in range(3):
        # both i and i2 shouLd be same
        i = i + 2
        print("value of loop control variable: ", i)
```

- **Output 2**

```
value of loop control variable:  2
value of loop control variable:  3
value of loop control variable:  4
```

- **Rust:**
- **As a default loop control variable declared as the immutable and not legal to change loop control variable without declaring it as mutable**. In the example program **the loop control variable declared as mutable allowing it to change**. It did not affect the number of iterations the loop go into but do affect the value of the loop control variable.
- **Program**

```
    for mut x in 0..3 {
        x = 3;
        println!("{}", x); // x: i32
    }
```

- **Output**

```
3
3
3
```

## 4. Are the loop parameters evaluated only once, or once for every iteration?

- **Dart:**
- **Loop parameters evaluated once for every iteration**. The example program shows the end iteration condition changed within the loop and reflect back to the loop logic.
- **Program**

```
for(int i=start;i<end;i++){
  start =  3;
  end = 3;
  print(i);
}
```

- **Output**

```
0
1
2
```

- **Javascript:**
- **Loop parameters evaluated once for every iteration**. It can be seen from the example program that during the third iteration, condition which controls the end of the iteration updated and reflected to the program increasing the number of iterations of the loop.
- **Program**

```
var loopEnd = 3
console.log("End of the loop set to iteration: ", loopEnd);
for (var i5 = 0; i5 < loopEnd; i5++) {
  if(i5 == 2){
    loopEnd = 5
    console.log("End of the loop changed to iteration: ", loopEnd);
  }
}
console.log("Number of iterations: ",i5);
```

- **Output**

```
End of the loop set to iteration: 3
End of the loop changed to iteration: 5
Number of iterations: 5
```

- **PHP:**
- **Evaluated once for every iteration**. In PHP For loops, the conditional expression executed at the beginning of each loop. In the example program, it is shown that the loop parameters are being changed inside the loop. To be specific, `start` variable set to value 2 at the end of the first iteration and it keeps updated
- **Program**

```
for ($start = 0, $end = 5;$start <= $end;$start++, $end--) {
    echo "start = $start and end = $end";
    echo '<br>';
    $start = 2; // will be 3 at the end of the iteration
}
```

- **Output**

```
start = 0 and end = 5
start = 3 and end = 4
start = 3 and end = 3
```

- **Python:**
- The example program demonstrates that the **loop parameters evaluated at the the first iteration of the loop and remain unchanged** since when tried to manipulated within the loop it stays bounded to the tightest scope which is the conditional part of the loop. Example program shows that the loop parameters which are variables `start = 4` and `end = 7`, bound during the initial iteration and remain unchanged even though I tried to assign a new value to the `end` variable within the loop.
- **Program**

```
    start = 4
    end = 7
    for i in range(start, end):
        end = 9
        print(i)
```

- **Output**

```
4
5
6
```

- **Rust:**
- **Evaluated only once at the beginning of the loop**. The example program shows that even though we change the loop parameters, the change does not reflect to the new iteration.
- **Program**

```
      println!("Q4");
      let mut start = 0;
```

```
    let mut end = 5;
    for mut x in start..end {
        end = 3;
        start = 12;
        println!("{}", x); // x: i32
    }
```

- **Output**

```
0
1
2
3
4
```

## 2. Discussion

In my opinion Rust supports the most flexible counter-controlled loops and be my preferred programming language for the counter-controlled loops.  My reasoning behind is as follows:

- The loop control variable's data type is flexible, allowing any data type which can be accessed via an iterator.
- The scope limited to the loop/block which increases readability of the programming language. Making it easy to trace the variables since the variable known to be declared in the loop/block.
- Depending on the deceleration style of the loop control variable (mutable or unmutable) the loop control variable become accessible within the loop which increases the user control over the programming language. I think the default option does not allowing legal access to the loop control variable, increases the reliability of the language.
- Loop parameter changes do not affect the program logic which I think useful to increase reliability of the language preventing unwanted changes in the loop control.
- To conclusion,  Rust's counter controlled loops design choices reflects Rust's purpose of being performance and reliability oriented programming language. On top of that, Rust's loop structures incorporating reliability together with flexibility which makes the programming language user friendly as well.

## 3. Learning Strategy

First, I refreshed my memory about meaning of the counter-controlled loops and loop control variables from the lecture slides. Additionally, I found a resource in internet going through properties of counter-controlled loops in Java together with explanations [1]. Running the codes on Dijkstra takes significant amount of time since establishing a stable VPN connection is hard in my case. Hence, I found the online compilers of the programming languages and test the sample programs using the online compilers before running on Dijkstra.

*Dart*

*Online Compiler (DartPad): https://dartpad.dev/*

I checked the Dart documentation Language Tour to refresh my memory about built-in types in Dart. Also, learn more about the counter-controlled loops structures in Dart [2]. I found out that Dart supports looping over Iterables which allows Dart to iterate over any structure implementing Iterator getter/setter methods. Thus, I checked out the Iterable Class documentation [3]. These resources helped me to understand inner workings of counter controlled loops in Dart and write my own test programs.

*Python*

I checked Python3 Official. documentation and found out that in Python every data type is an Object underneath and loops in Python `for loops` able to iterate over any sequence of elements which allows flexibility for loop control variable being any Object (Numbers, Strings, Dict, Function, Tuple, …) [4]. Additionally, I checked the StackOverflow to see some examples to increase my understanding about loop control variables in Python.

I prepared simple experiments with the scoping rules that I learnt from class. I came across a good article about how Scoping works in Python to test whether my observations are compatible with the documentation [5]. For Question 3 and 4, I refreshed my memory about meaning of loop parameters and did experiments with the language to draw conclusions. Solved ambiguities by consulting the StackOverflow community.

*Javascript:*

*Online Compiler: https://playcode.io/*

First, I checked the Mozilla Web Developers Documentation for Javascript loop structures and data types [6]. I found out that Mozilla Web Developers Documentation has a credibility of an official documentation and extensively explaining the Javascript language. I checked the Javascript documentation put by w3schols.com to learn more about the loops in Javascript [7]. Additionally, I found a good Javascript resource going into detail what is the scope of loop control variables, when the condition and post-expression statements of the loop evaluated [8].

*PHP:*

*Online Compiler: http://www.writephponline.com/*

I checked out the official documentation of PHP to learn about control structures in PHP [9]. As a Additionally, I find out some resources explaining the inner workings of counter controlled loops in PHP [10]. I already know basic PHP syntax from Homework1 which allowed me to speed up my learning curve in PHP language. I wrote sample test programs to verify the things that I was reading in the official documentation.

*Rust:*

*Online Compiler: https://play.rust-lang.org/*

I followed the same strategy that I followed for other programming languages. First checking out the online documentation to see how loop control being done in Rust [11]. Also, experimenting my own how loop control variables are being treated in the Rust language.

## 4. References

[1] B. Kjell, “Declaring Loop Controlled Variables,” *Central Connecticut State University.* [Online]. Available:https://chortle.ccsu.edu/java5/notes/chap42/ch42_2.html [Accessed: Apr. 26, 2020].

[2] “A Tour of the Dart Language,” *Dart.* [Online]. Available: https://dart.dev/guides/language/language-tour#built-in-types [Accessed: Apr. 27, 2020].

[3] “Iterable<E> Class,” *Dart.* [Online]. Available: https://api.dart.dev/stable/2.8.0/dart-core/Iterable-class.html [Accessed: Apr. 27, 2020].

[4] “More Control Flow Tools,” *Python.* [Online]. Available: https://docs.python.org/3/tutorial/controlflow.html [Accessed: Apr. 28, 2020].

[5] G. Ewing, “[Python-ideas] For-loop variable scope: simultaneous possession and ingestion of cake,”. [Online]. Available:https://mail.python.org/pipermail/python-ideas/2008-October/002109.html [Accessed: Apr. 28, 2020].

[6] “Loops and iteration,” *MDN web docs*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration [Accessed: Apr. 28, 2020].

[7] “JavaScript for Statement,” *w3schools.com*. [Online]. https://www.w3schools.com/jsref/jsref_for.asp [Accessed: Apr. 28, 2020].

[8] “JavaScript for Loop,” *Javascript Tutorial*. [Online]. https://www.javascripttutorial.net/javascript-for-loop/ [Accessed: Apr. 28, 2020].

[9] “Control Structures (for),” *PHP Manual*. [Online]. https://www.php.net/manual/en/control-structures.for.php [Accessed: Apr. 28, 2020].

[10] S. Kavinda, “PHP Loops”. [Online]. https://tutorials.supunkavinda.blog/php/loops [Accessed: Apr. 28, 2020].

[11] “Loops,” *Rust*. [Online].https://doc.rust-lang.org/1.7.0/book/loops.html [Accessed: Apr. 28, 2020].