CS315 - HW2

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

Olcay Akman
21702671
Section 1

Introduction

In this homework report I will be discussing counter-controlled loop examples, implementations and characteristics in the following programming languages listed:

- Dart
- JavaScript
- PHP
- Python
- Rust

The general design issues related with counter-controlled loops are as follows:

1. What are the types of loop control variables?
2. What are the scopes of loop control variables?
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?
4. Are the loop parameters evaluated only once, or once for every iteration? [1]

1. Design Issues of Counter-Controlled Loops

1.1. Design Issues in Dart

In Dart, counter-controlled loops are called **for** loops. Dart language uses type inference [2], which is why the type of the loop variable can be of any type which is covered by **var**. The loop variable's scope is limited to the for loop in Dart.

```
var myList = [1,2,3,4,5];
//int as loop control variable
for (var i in myList) {
  print(i);
}
```

Ex1.1.1 This piece of code prints the value of i.

```
var myList = [1,2,3,4,5];
//int as loop control variable
for (var i in myList) {
  print(i);
}
print(i);
```

Ex1.1.2 This piece of code cannot execute the final print(i) method as explained above.

The loop variable can be changed inside the loop body, and this does affect loop control, depending on the condition of the loop. The loop is terminated once the loop condition statement becomes false [3]. Although the loop variable can be of any primitive type (covered by **var**), condition part of the for loop can only evaluate to a **bool** value.

```
var x = 3;
bool i;
for (i = true; i; x--) {
  print(x);
  if (x == 0) {
    i = false;
  }
}
```

Ex1.1.3 This piece of code shows the case explained above.

The loop parameter can be evaluated only once or once per every iteration depending on the form of implementation. If the loop control statement is a boolean variable then it is possible to not evaluate the loop parameter on each iteration but rather change its value only once among other iterations to change the control statement's result.

```
var x = 3;
bool i;
for (i = true; i; x--) {
  print(x);
  if (x == 0) {
    i = false;
  }
}
```

Ex1.1.4 Loop parameter **i** only evaluated once.

```
var myList = [1,2,3,4,5];
//int as loop control variable
for (var i in myList) {
  print(i);
}
```

Ex1.1.5 Loop parameter **i** evaluated on each iteration.

1.2. Design Issues in JavaScript

In JavaScript **for** loops, it is not possible to declare the loop variable in the initialization statement, at least not as a 'global' variable. In general, in Javascript the **let** keyword allows the coder to create a block scope local variable [4].

```
var x = 0;
for (var j = 0; j < 10; j++) {
    x++;
}
```

Ex1.2.1 Not legal to declare **var j** here.

```
var x = 0;
for (let j = 0; j < 10; j++) {
    x++;
}
```

Ex1.2.2 Legal with the **let** keyword.

The loop control variables in JavaScript can be of any variable type, like in Dart. However, different from Dart, they can be declared with the keyword **var** as well as **let**. The difference between the two keywords is that **let** lets the scope of the loop variable be limited to within the loop whereas **var** sets it to be global.

```
var x = 0;
for (let j = 0; j < 10; j++) {
    x++;
}
document.getElementById("loop-out").innerHTML = x;
document.getElementById("var-scope").innerHTML = j;
```

Ex1.2.3 In this piece of code element with id "var-scope" is not assigned anything.

```
var x = 0;
for (var j = 0; j < 10; j++) {
    x++;
}
document.getElementById("loop-out").innerHTML = x;
document.getElementById("var-scope").innerHTML = j;
```

Ex1.2.4 In this piece of code element with id "var-scope" is assigned 10 which is the latest value **j** was assigned.

The loop control variable can be changed within the loop body in JavaScript **for** loops. This is a legal action and this does, in turn, affect the loop control. Below are examples:

```
myArr = [];
var itr = 0;
var i;
for (i = 1; i < 50; i = i + 10) {
    myArr[itr] = i;
    itr = itr + 1;
}
document.getElementById("out1").innerHTML = myArr;
```

Ex1.2.5 In this piece of code **myArr** is **1,11,21,31,41**.

```
myArr = [];
var itr = 0;
var i;
for (i = 1; i < 50; i = i + 10) {
    myArr[itr] = i;
    itr = itr + 1;
    i = i + 1;
}
document.getElementById("out1").innerHTML = myArr;
```

Ex1.2.6 In this piece of code **myArr** is **1,12,23,34,45**.

In JavaScript, the loop parameters of the **for** loop structure can be evaluated within the body of the loop and it affects the loop control. In this manner JavaScript is similar to Dart. Below is an example of how this functions in JavaScript. As can be seen, **b** is set to **true** before execution, and it is the loop control statement. Within the loop body it is changed to **false**, which then results in the termination of the loop.

```
var k;
var b = true;
document.getElementById("loop-param-before").innerHTML = b;
for (k = 0; b; k++) {
    if (k == 7) {
        b = false;
    }
}
document.getElementById("loop-param-after").innerHTML = b;
```

Ex1.2.7 Example code in JavaScript.

1.3. Design Issues in PHP

Loop control variables in PHP can also be of any type. The scope of the loop control variable of a for loop in PHP is not limited to the loop. The loop control variables are recognized outside the loop as well.

```
for ( $i = 0; $i < 5; $i++) {
    $i++;
    echo $i;
}
echo "\n";
echo $i;
```

Ex1.3.1 In this piece of code, the final **echo $i;** gives **6**.

The loop control variable can be changed in the loop body and this does affect the loop control. In **Ex1.3.1** we can see that the loop control variable **$i** is changed and the output of this loop is **1 3 5**. However, in the next example, **Ex1.3.2**, we see that when the loop variable is not changed in the loop body for the same example, the loop output becomes **0 1 2 3 4**.

```
$i = 0;
for (; $i < 5; $i++) {
    echo $i;
}
```

Ex1.3.2 In this piece of code the output is **0 1 2 3 4**.

The loop parameters in for loops of PHP are evaluated once for every iteration due to the syntax. Its syntax is similar to those of "C-type" for loops, thus the final expression of the loop signature automatically evaluates the loop variable on each iteration.

1.4. Design Issues in Python

In Python, for loops are syntactically different from Dart, JavaScript or PHP. Although these languages have a similar structure to Python as well as the generic C-type for loop structure, they differ from Python in terms of the design issues discussed in this homework.

The type of the loop variable is not limited in Python **for** loops. For instance, it can be an **int**, a **char**, or of any other type. Since Python uses type inferencing [5], the structure of the **for** loop differs on whether we want to use **int** or **char**.

```
for i in range(5):
    print(i+1)
```

Ex1.4.1 In this piece of code, **for** loop variable is an **int**.

```
for i in "in-loop":
    print(i)
```

Ex1.4.2 In this piece of code, **for** loop variable is a **char**.

```
myArr = [True, False, True, True]
for i in myArr:
    print(i)
```

Ex1.4.3 In this piece of code, **for** loop variable is an array element, which are all **bool**s.

In Python, the scope of the loop control variable is not limited to within the loop. The loop control variable can be created within the loop declaration and its value will be recognized outside the loop.

```
i = "not in loop"
for i in "in-loop":
    print(i)
print(i)
```

Ex1.4.4 In this piece of code, the final **print(i)** statement will print the latest value **i** was assigned within the loop, which is **p**. It will not print **"not in loop"**.

The loop control variable can be changed within the loop body, however, this does not affect the loop control, unlike the languages examined so far.

```
i = "not in loop"
for i in "in-loop":
    i = 'p'
    print(i)
```

Ex1.4.5 In this piece of code, **print(i)** prints **p** as many times as the number of characters in the string **"in-loop"**.

```
for i in range(5):
    i = 4
    print(i+1)
```

Ex1.4.6 In this piece of code, **print(i)** prints **5** in a total of 5 times.

Loop parameters in Python are evaluated on every iteration, and this is always the case. The examples **Ex1.4.5** and **Ex1.4.6** illustrate this issue in Python.

1.5. Design Issues in Rust

As in all the above explained languages in this homework, the counter-controlled, also known as iterator, loops are called **for** loops in Rust language [6]. The syntax of **for** loops in Rust mirrors that of Python, thus it does not look like the common "C-style" **for** loop which is existent in the other languages mentioned in this homework [7].

However, unlike Python, the scope of the loop variable is limited to within the loop. That is, the loop variable, if not declared outside, will not be recognized outside the loop.

```
for i in 0..5 {
    println!("rust code - {}", i);
}
println!("i outside the loop is: {}", i);
```

Ex1.5.1 This piece of code does not compile.

```rust
let i = 0;
for i in 0..5 {
    println!("rust code - {}", i);
}
println!("i outside the loop is: {}", i);
```

Ex1.5.2 In this piece of code, **i** prints as **0** outside the loop.

The type of the loop control variable can be almost anything, as long as the expression part of the loop is iterable [7]. Arrays in Rust are not iterable in Rust by default [8], but by using a workaround as seen in **Ex1.5.3** it is possible to use as loop expression.

```rust
let myArr2 = ["s", "t", "r!"];
for elem in myArr2.iter() {
    print!("{} ", elem);
}
println!();
```

Ex1.5.3 In this piece of code, loop variable is a string.

Also, **Ex1.5.1** shows that the loop variable can be an **int**. Without the array workaround, the loop variable type in Rust can be of a limited list given in reference [8].

The loop variable cannot be changed within the loop body of the **for** loop, and thus such an interference does not affect the loop control.

```rust
let mut myArr: [i32; 5] = [0; 5];
for i in 1..5 {
    myArr[i] = 33;
    i++;
}
```

Ex1.5.4 This piece of code does not compile.

The loop parameters, due to its syntax, are evaluated on every iteration just like in Python.

2. Best Counter-Controlled Loop

I have examined the programming languages required by this homework. As stated in the homework instructions, I have done so to deduce the four given questions related to the design of counter-controlled loops. In the case of all the given languages, this construct was called a for loop. Each language has similar and different properties in terms of these design issues (given in Introduction as the four questions mentioned before). After having done my study, I deduced some results.

In my opinion the types of loop control variables should not be limited as in Rust. In terms of the scope issue of the loop control variables, I mostly like the approach of JavaScript because the scope of the loop variable is limited to the loop unlike Python, and via the let keyword it can first be declared inside the loop signature.

In terms of changing the loop control variable within the loop body, I mostly liked the approach of JavaScript again. I liked it because it is important that the interference with the loop variable changes the loop control. In certain cases of different coding problems, such an action may become necessary.

Due to all these observations, my favorite for loop structure is of JavaScript.

3. Learning Strategy

While doing this homework, I have completed certain steps to complete my work. Initially, before diving into doing some actual work by experimenting with code-writing by myself, I went through a ‘preparation phase’. During this phase I did some research on the languages to be examined for the homework. The first and probably most important step of learning a new programming language is by consulting its official documents from its website. The documents of a programming language gives basic outlines of the type and usage of it in the industry. By looking at those, I obtained a brief understanding of each language. Another important component of this preparation phase contained setting up and downloading the relevant packages for each language so that I could compile my code when I came to the experimentation phase.

After this preparation phase, I got started with the actual work of this homework by firstly examining and understanding the design issues for the counter-controlled loops. Then the next step was to clearly indicate and address each of these issues in the given programming languages. This consisted of two major parts: First, I examined the documentation and other official resources for the counter-controlled loops for these

languages. These gave me an idea about the specific cases for each different language. Then, I needed to code some examples for myself to see and understand the full effects of these design related issues in perspective.

Looking at the official documentation of a language is mostly sufficient enough to grasp the general basics of a language. However, usually these introduction documents for programming languages end up being “too broad”, and because of this I unfortunately was not able to clearly address all the design issues required by this homework. Some of these design issues had to be experimented with, via programming some pieces of code in each language.

Programming in each language required different techniques of compiling and running its code. This itself was a learning process in which I consulted the web on how to run each language on my operating system. When my programming environment was complete, and I could successfully compile and run code in them, I dived into further investigating the design issues for counter-controlled loops. By combining the knowledge I acquired from reading the documentations of each language and the mini introduction tutorials provided by the official websites I wrote some sample codes to highlight the issues which I could not address by the documentations themselves. By experimenting with what expressions are valid in each language and what are not valid, along with observing the sample outputs I output to consoles to check the variables. Such a method, combined with the other methods I used as explained before, gave me an insight to how each design issue was addressed in each programming language.

4. References

[1] Robert W. Sebesta, *Concepts of Programming Languages*, Global Edition, 11/E, Pearson.

[2] "The Dart Type System." *Dart*, dart.dev/guides/language/sound-dart.

[3] Jay Tillu. "The for Loop in Dart." *Medium*, Jay Tillu, 16 Sept. 2019, medium.com/jay-tillu/the-for-loop-in-dart-fdf6f87c8cc7.

[4] "Let." *MDN Web Docs*, developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let.

[5] "Type Inference and Type Annotations¶." *Type Inference and Type Annotations - Mypy 0.770 Documentation*, mypy.readthedocs.io/en/stable/type_inference_and_annotations.html.

[6] "The Rust Reference." *Loop Expressions - The Rust Reference*, doc.rust-lang.org/reference/expressions/loop-expr.html.

[7] *For Loops*, doc.rust-lang.org/1.2.0/book/for-loops.html.

[8] "1.0.0[−]Primitive Type Array." *Rust*, doc.rust-lang.org/std/primitive.array.html.