Logically Controlled Loops in Different Languages

1. Perl

- **What is the location of the tests? Pretest, posttest, or both?**
  Perl has both pretest and posttest loops. for, until and while loops are pretest. do-while and do-until loops are posttest loops [3].

- **What kind of user-located loop control mechanisms are provided?**
  last, next, and redo statements are provided, and also labels are provided with these statements [3].

  - **Should the conditional mechanism be an integral part of the exit (conditional or unconditional exit)?**
    redo label requires a condition, it repeats the loop with its own condition without checking the condition of the loop. This will be explained in further detail in the inspection of the code part. next and last can depend on conditions or they can be directly used and loops can be exited without conditions [1].

  - **Should only one loop body be exited or can enclosing loops also be existed (labeled or unlabeled exit)?**
    Enclosing loops can also be exited with user-defined labels in Perl.

Inspection of Code

```
for( $i = 0; $i < 4; $i++)
{
    print("LLOOOOPP $i\n");
}
Output:
LLOOOOPP 0
LLOOOOPP 1
LLOOOOPP 2
LLOOOOPP 3
```

The first type of for loops in Perl is the C-type for loop, it has initialization, condition, and increment/decrement parts. This is a pretest loop, the condition is checked each time, and the body is executed later. Moreover, as in C group languages, each part of for loop is optional, for example, if all parts of the loop were empty (i.e. for(;;)), it would be an infinite loop, an example can be found in the source code in Perl. Moreover, this is the only type of logically controlled for loop in Perl [3].

```
$noOfAssignments = 0;
while( $noOfAssignments < 4)
{
    print("There are $noOfAssignments assignment(s) left\n");
} continue {
    $noOfAssignments = $noOfAssignments + 1;
}
print("Well, I am dropping out the university...\n")
Output:
There are 0 assignment(s) left
There are 1 assignment(s) left
There are 2 assignment(s) left
There are 3 assignment(s) left
Well, I am dropping out the university…
```

Of course, there are while loops in Perl and they are pretests. The condition is checked before getting into the loop's body. Moreover, the continue block can be used as in the above example for incrementing operation. The continue block can be removed and incrementing can be done in the while body. Moreover, it is possible to create an infinite loop by not incrementing the noOfAssignments variable or a true boolean value can be put in the while condition checking part.

```
# The notorious "do-while" that I've never used
$i = 1;
do {
    print("$i brain cells left...\n");
    $i = $i - 1;
} while( $i >= 0);
Output:
1 brain cells left...
0 brain cells left...
```

There is also do-while loop in Perl language. This is a posttest loop, that is it would run once even if the initial condition was not met.

```
$isEven = 0;
$num = 1;
until ($isEven)
{
    print("Num is : $num\n");
    if( $num % 2 == 0)
    {
        $isEven = 1;
        print('Num is even now!\n');
    }
    $num = $num + 1;
}
Output:
Num is : 1
Num is : 2
Num is even now!
$i = 1;
do {
    print("$i brain cells left...\n");
    $i = $i - 1;
} until( $i < 0);
Output:
1 brain cells left...
0 brain cells left...
```

Different from other languages, there are until and do-until loops in Perl language. The logic of until loops are the opposite of while loops, that is, while loop is executed while the condition is true, and until loop is executed until the condition becomes true. As while loop, until loop is pretest and as do-while loop, do-until loop is posttest [3].

```
for( $i = 0; $i < 5; $i++){
    for( $j = 0; $j < 2; $j++){
        if( $i == 1){
            print("No, I warned this lad i about being 1! I will break it!\n");
            last;
        }
        print("i is $i, j is $j\n");
    }
}
Output:
i is 0, j is 0
i is 0, j is 1
No, I warned this lad i about being 1! I will break it!
i is 2, j is 0
i is 2, j is 1
i is 3, j is 0
i is 3, j is 1
i is 4, j is 0
i is 4, j is 1
```

One of the loop control mechanisms in Perl is the last statement. This statement is the equivalent of the break statement in other languages. When this statement is executed, the currently executing loop is exited. It can be provided with a condition as in the above example (when i equals one the inner loop is exited), or it can directly be located in the loop body without an if statement; however, it directly exits the loop in this case and does not provide a logic [5].

```
$i = 0;
while( $i < 6){
    $i++;
    if( $i % 2 == 0)
    {
        print("I hate even numbers!\n");
        next;
    }
    print("$i\n");
}
Output:
1
I hate even numbers!
3
I hate even numbers!
5
I hate even numbers!
```

The second loop control mechanism in Perl is the next statement. It is equivalent to the continue statement in the other languages. In the above example when an even number is encountered the remaining part of the loop body is not executed and loop continues for the next value. As in the last, it is possible to have a condition or not have condition for next statement [5].

```
countOut:
for($i = 1; $i < 100; $i++){
   for( $j = 1; $j < 10; $j++){
       print("Looping... i = $i, j = $j \n");
       if( $i == 2)
       {
           print("C'mon I do not have time for looping 1000 times!");
           last countOut; # This directly breaks the outer loop!
       }
   }
}
Output:
Looping... i = 1, j = 1
Looping... i = 1, j = 2
Looping... i = 1, j = 3
Looping... i = 1, j = 4
Looping... i = 1, j = 5
Looping... i = 1, j = 6
Looping... i = 1, j = 7
Looping... i = 1, j = 8
Looping... i = 1, j = 9
Looping... i = 2, j = 1
C'mon I do not have time for looping 1000 times!
```

There are also user defined labels in Perl. It is possible to label the loops and call the last or next statements with this label. In the above example, when the if statement is entered, last directly breaks the outer loop with the help of the label [5].

```
count:
for($i = 1; $i < 6; $i++){
  countIn: for( $j = 1; $j < 10; $j++){

       if( $i % 2 == 0 || $j % 2 == 0 )
       {
           print("You cannot be even numbers!\n");
           next count;
       }
       print("Looping... i = $i, j = $j \n");
   }
}
Output:
Looping... i = 1, j = 1
You cannot be even numbers!
You cannot be even numbers!
Looping... i = 3, j = 1
You cannot be even numbers!
You cannot be even numbers!
Looping... i = 5, j = 1
You cannot be even numbers!
```

The same is also possible for next statement, in the above example when the if statement executed, the outer loop is continued with its next value.

```
$i = 0;
repeatThis:
for($j = 0; $j <= 2; $j++){
    print("i is $i, j is $j\n");
    $i++;
    redo repeatThis if ($i < 3);
}
Output:
i is 0, j is 0
i is 1, j is 0
i is 2, j is 0
i is 3, j is 1
i is 4, j is 2
```

There is also one special loop control mechanism in Perl which is redo. It works with a condition and can be used with labels. In the above examples, the loop variable does not increment until the value of i satisfies the condition. Loop continues normally after the condition is satisfied. In this way, the loop is executed 5 times instead of 3 times (without redo 3 iterations) [5].

2. Python

- **What is the location of the tests? Pretest, posttest, or both?**

  There are only pretest loops in Python language. The only loop provides pretest is while loop.

- **What kind of user-located loop control mechanisms are provided?**

  User-located loop control mechanisms are break and continue statements in Python. Moreover, there are no labels in Python.

  - **Should the conditional mechanism be an integral part of the exit (conditional or unconditional exit)?**

    Again, as in Perl, the breaks and continues can be either located with conditionals or without conditionals. By their nature breaks and continues are providing unconditional exits but it makes more sense to locate them inside the conditionals [1].

  - **Should only one loop body be exited or can enclosing loops also be existed (labeled or unlabeled exit)?**

    There are no labels in Python; therefore, enclosing loops cannot be exited with provided loop control mechanisms. However, it maybe simulated with else statement can be located after the loops but again normally it is not provided by the language creators [8].

Inspection of Code

```
for i in range(0, 4, 1):
    print("LLOOOOPP " + str(i))
```

There is no C-type for loop unlike the other languages in this homework. Range function provides a similar structure for the for loops. It starts with zero, continues until 3 and increments by 1. However, for loops in Python are not logically-controlled loops.

```
name = "JAX"
for spell in name:
    print(spell)
else:
    print("Woof!") # This is executed after for is completed
Output:
J
A
X
Woof!
```

For loops or while loops can be combined with an else statement that is executed once after the loop is completed. I gave this example to demonstrate this but the for loop in the above example is NOT logically controlled [1], [8].

```
noOfAssignment = 0
while( noOfAssignment < 4):
    print("There are " + str(noOfAssignment) + " assignment(s) left")
    noOfAssignment = noOfAssignment + 1
Output:
There are 0 assignment(s) left
There are 1 assignment(s) left
There are 2 assignment(s) left
There are 3 assignment(s) left
```

While loops in Python are logically controlled, fortunately. Moreover, while loop is pretest. In the above example loop is executed while the condition is satisfied.

```
i = 1
while True:
    print(str(i) + " brain cells left...")
    i = i - 1
    if( i < 0):
        break # Yeah, there is break statement in Python
Output:
1 brain cells left...
0 brain cells left…
```

There is no do-while loop in Python, as a result there is no posttest loop. However, somehow the behavior of do-while loop can be simulated by using a structure as in the above example.

```
for i in range(5):
    for j in range(2):
        if i == 1:
            print("No, I warned this lad i about being 1! I will break it!")
            break
    print("i is " + str(i) +", j is " + str(j))
Output:
i is 0, j is 1
No, I warned this lad i about being 1! I will break it!
```

```
i is 1, j is 0
i is 2, j is 1
i is 3, j is 1
i is 4, j is 1
i = 0
while i < 6:
    i = i + 1
    if i % 2 == 0:
        print("I hate even numbers!")
        continue
    print(i)
```

Python's break and continue statements are very similar to Perl's last and next. When a break is countered the current loop is exited and when a continue is countered loop is continued with the next iteration. Moreover, user-located labels are no provided in Python.

3. JavaScript

- **What is the location of the tests? Pretest, posttest, or both?**
  There are both posttest and pretest loops in JS. for and while loops are pretest loops, do-while loop is posttest loop [9].

- **What kind of user-located loop control mechanisms are provided?**
  break and continue statements are provided as loop control mechanisms, there are also labels that can be used with break and continue.

  - **Should the conditional mechanism be an integral part of the exit (conditional or unconditional exit)?**
    As in other languages, by their nature break and continue provides unconditional exit; however, these can be combined with conditional statements.
  - **Should only one loop body be exited or can enclosing loops also be existed (labeled or unlabeled exit)?**
    With labels it is possible to exit also outer loops.

Inspection of Code

```
for( let i = 0; i < 4; i++)
{
    console.log(`LLOOOOPP ${i}`)
}
```

There is C-type logically controlled for loops in JS. This kind of for loop is pretest, and its output is same with other languages, thus is not provided.

```
let num = 0;
while( num < 4)
{
    console.log(`There are ${num} assignment(s) left`)
    num++;
}
```

There are while loops in JS which are pretest logically-controlled loops. The output is the same with other languages.

```
i = 1;
do{
    console.log(`${i} brain cells left...`)
    i--;
}while( i >= 0)
```

Like Perl, JS has do-while loop which is posttest, the condition is checked after the loop body is executed. The output is the same with Perl.

```
for( let i = 0; i < 5; i++){
    for( let j = 0; j < 2; j++){
        if( i === 1){
            console.log("No, I warned this lad i about being 1! I will break it!");
            break;
        }
        console.log(`i is ${i}, j is ${j}`);
    }
}
i = 0;
while( i < 6)
{
    i++;
    if( i % 2 === 0){
        console.log("I hate even numbers!");
        continue;
    }
    console.log(i)
}
```

There are break and continue in JS that provide unconditional exit, but they can be combined with conditional expressions to make sense. Break exits the current loop, and continue continues with the next value of current loop by passing the remaining part of the body. Output is the same with other languages.

```
countOut:
for( let i = 1; i < 100; i++){
    for( let j = 1; j < 10; j++){
        console.log(`Looping... i = ${i}, j = ${j}`)
        if( i === 2){
           console.log("C'mon I do not have time for looping 1000 times!");
          break countOut;
        }
    }
}
forLabel1:
for(let i = 1; i < 6; i++){
    forLabel2:
    for(let j = 1; j < 5; j++)
    {
        if( i % 2 === 0 && j % 2 === 0)
        {
            console.log("You cannot be even numbers!");
            continue forLabel1;
        }
        else if( j % 2 === 0){
            console.log("Go away even j!");
            continue forLabel2;
        }
        console.log(`Looping... i = ${i}, j = ${j}`)
    }
}
Output:
Looping... i = 1, j = 1
Go away even j!
Looping... i = 1, j = 3
Go away even j!
Looping... i = 2, j = 1
You cannot be even numbers!
Looping... i = 3, j = 1
Go away even j!
Looping... i = 3, j = 3
Go away even j!
Looping... i = 4, j = 1
You cannot be even numbers!
Looping... i = 5, j = 1
Go away even j!
Looping... i = 5, j = 3
Go away even j!
```

There are user-located labels in JS as in Perl. These labels can be used with loop control statements. This makes exiting enclosing loops possible.

4. PHP

- **What is the location of the tests? Pretest, posttest, or both?**
  There are both posttest and pretest loops in PHP. For and while loops are pretest whereas do-while is posttest [11].
- **What kind of user-located loop control mechanisms are provided?**
  There are break, continue and goto statements in PHP as user-located loop control mechanisms.
  - **Should the conditional mechanism be an integral part of the exit (conditional or unconditional exit)?**
    As in other languages, loop control mechanisms provide unconditional exit but conditional mechanism can be used with loop control mechanisms to make more sense.
  - **Should only one loop body be exited or can enclosing loops also be existed (labeled or unlabeled exit)?**
    Labels cannot be used with continue and break in PHP; however, it is possible to break enclosing loops by giving a number with break or continue

statements, for example *break 2*; can be used to break the loop 2, in other words outer loop.

Inspection of Code

```
for( $i = 0; $i < 4; $i++){
    echo "LLOOOOPP $i<br>";
}
```

There is C-type for loop in PHP, too. It is logically-controlled and pretest. Output is the same with other languages.

```
$num = 0;
while( $num < 4){
    echo "There are $num assignment(s) left<br>";
    $num++;
}
```

There is also while loop in PHP which is pretest. Output is the same with other languages.

```
$i = 1;
do{
    echo "$i brain cells left...<br>";
    $i--;
} while( $i > -1);
```

There is also do-while loop in PHP which is posttest.

```
for( $i = 0; $i < 5; $i++){
    for( $j = 0; $j < 2; $j++){
        if( $i == 1){
            echo "No, I warned this lad i about being 1! I will break it!<br>";
            break;
        }
        echo "i is $i, j is $j<br>";
    }
}
$i = 0;
while( $i < 6){
    $i++;
    if( $i % 2 == 0)
    {
        echo "I hate even numbers!<br>";
        continue;
    }
    echo "$i<br>";
}
```

Two of loop control mechanisms are continue and break, they are working in the same way with the other languages and they give the same output.

```
for($i = 1; $i < 100; $i++){
   for( $j = 1; $j < 10; $j++){
       echo "Looping... i = $i, j = $j<br>";
       if( $i == 2)
       {
           echo "C'mon I do not have time for looping 1000 times!<br>";
           break 2; # This breaks the second level, the outer loop, isn't it cool?
       }
   }
}
```

Labels cannot be used with break or continue but break 2; exits the outer loop in the above example. The loop numbers start from 1 and increments, it is possible to break or continue loops by providing their numbers.

```
for($i = 1; $i < 100; $i++){
    for( $j = 1; $j < 10; $j++){
        echo "Looping... i = $i, j = $j<br>";
        if( $i == 2)
        {
            echo "C'mon I do not have time for looping 1000 times!<br>";
            goto rightHereWaiting;
        }
    }
 }
 rightHereWaiting:
 echo "You really broke the loop!";
Output:
Looping... i = 1, j = 1
Looping... i = 1, j = 2
Looping... i = 1, j = 3
Looping... i = 1, j = 4
Looping... i = 1, j = 5
Looping... i = 1, j = 6
Looping... i = 1, j = 7
Looping... i = 1, j = 8
Looping... i = 1, j = 9
Looping... i = 2, j = 1
C'mon I do not have time for looping 1000 times!
You really broke the loop!
```

PHP is the only language in this homework that has goto statement. It provides unconditional branching and it can be used with labels as in the above example, in this example it works like a break statement. Moreover, goto can be located between the loops to break varying numbers of loops.

5. Dart

- **What is the location of the tests? Pretest, posttest, or both?**
  There are both posttest in pretest loops in Dart language. Pretest loops are for and while loops, the one and only posttest loop is do-while loop [12].

- **What kind of user-located loop control mechanisms are provided?**
  As user-located loop control mechanims break and continue statements and labels are provided.

- **Should the conditional mechanism be an integral part of the exit (conditional or unconditional exit)?**
  break and continue provides unconditional exit.
- **Should only one loop body be exited or can enclosing loops also be existed (labeled or unlabeled exit)?**
  It is possible to exit several loop bodies by using labeled breaks and continues.

Inspection of Code

```
for( var i = 0; i < 4; i++){
    print("LLOOOOPP $i");
  }
```

There is good old C-type for loop in Dart language and it is pretest, it prints LLOOOOPP and index for 4 times. The output is same with other languages.

```
  var num = 0;
  while( num < 4){
    print("There are $num assignment(s) left");
    num++;
  }
  print("Well, I am dropping out the university...");
```

There are while loops in Dart which are pretest loops. The above example checks if num is less than 4 and prints the remaining no of assignments and increments num. The output is same with other languages, and note that this example contains the unfunniest joke!

```
  var i = 1;
  do{
    print("$i brain cells left...");
    i--;
  }while( i > -1);
```

There is do-while loop in Dart which is posttest. The above example prints my remaining number of brain cells and decrements them, then it checks the condition. The magic of do-while! Output is the same with other languages.

```
  for( var i = 0; i < 5; i++){
    for( var j = 0; j < 2; j++){
      if( i == 1){
        print("No, I warned this lad i about being 1! I will break it!");
        break;
      }
      print("i is $i, j is $j");
    }
  }
```

Well, there is break in this language too. It breaks the inner loop if i is 2. The output is the same with other languages.

```
  i = 0;
  while( i < 6){
    i++;
    if( i % 2 == 0){
      print("I hate even numbers");
      continue;
    }
    print("$i");
  }
```

There is also continue in Dart. The above program continues the while loop without printing the number when i is an even number.

```
  countOut:
  for( var i = 1; i < 100; i++){
    for( var j = 1; j < 10; j++){
        print("Looping... i = $i, j = $j");
        if( i == 2)
        {
            print("C'mon I do not have time for looping 1000 times!");
            break countOut; // This directly breaks the outer loop!
        }
    }
  }
```

The above example reveals the magic of labeled break, the outer loop is exited when i becomes 2. Labels help us to break several layers of loops.

```
  count:
  for( var i = 1; i < 6; i++){
    countIn: for( var j = 1; j < 10; j++){

        if( i % 2 == 0 || j % 2 == 0 )
        {
            print("You cannot be even numbers!");
            continue count;
        }
        print("Looping... i = $i, j = $j");
    }
  }
```

Finally, continue can be also labeled. The above example continues with the next value of the outer for loop when both i and j are even numbers.

6. Evaluation of Readability and Writability

All languages are similar to each other in terms of the syntax of while loops except Python. However, I believe that writability increases when variables does not start with a special symbol since this symbols are requiring key combinations, it makes things hard while writing the C-type for loop. Moreover, logically-controlled for loops are also very similar to each other in all languages except Python. Python does not provide logically controlled for loops or do-while loops. Even if it has the simplest syntax and provides best readability (except range function, its purpose maybe vague if I saw it for the first time), it has the poorest writability with lacking these loops and not having labeled break and continue. Moreover, I believe goto and numbered breaks and continues in PHP decrease readability even if they increase the writability. They increase the writability since they provide a flexibility and makes branching possible. However, it might be hard to follow where goto statement is taking to the program and if we had too many nested for loops it might be difficult to see which loop has which number. JavaScript has a high readability in my opinion, names are self-defining, it follows the convention in other C-group languages mostly. It also has a high writability since it provides loop-control mechanisms to the user, it has labels that increase the writability by providing flexibility. It has all kinds of logically-controlled loops, that is, for, while, do-while. Moreover, Dart is highly similar to the JS in terms of logically controlled loops, they have the same set of loops, they have same loop-control mechanisms and labels (Dart's print statement is better :D). Finally, Perl has a high readability except the variable naming issue, I mentioned. It has all the loops provided by JS, Dart and PHP, and Perl adds two more loops: do-until and until. I believe these two loops are increasing the writability because it provides a room for flexibility. They also increase readability in my opinion because the intention of programmer can be understood more clearly. Perl also has labels, and loop-control mechanisms as PHP, Dart and JS. It also has one more control mechanism which is redo. It increases writability but it might reduce the readability since redo in a loop might be confusing, at least it confused me at first. As a result, I can say Perl is best in terms of writability. However, I cannot say Python is best for readability since it is missing lots of functionality. Therefore, I choose my current favorite language JS as the most readable language for logically controlled loops.

7. My Learning Strategy

As in the first homework, I have started by skimming through my lecture notes about loops. After doing that I have read chapter 8 of our textbook which is Statement-Level Control Structures [1]. I have already printed Altay Hoca's example codes with their outputs; therefore, I also took a look at them [2]. In this process, I have realized the difference between logically-controlled loops and counter-controlled loops. I did not clearly know which one is which before, I was thinking loops such as for(... in …) were logically-controlled. After this learning process, I have written the codes in the exact same order with the first homework: Perl, Python, JavaScript, PHP, and Dart. Last time, I choose Perl first out of curiosity but I have realized that in general it has the most unusual namings and functionalities. For learning the syntax, I have read the examples and documentations on GeeksForGeeks, Perl Tutorial and Perl Maven [3], [4], [5]. This time I have used OnlineGDB compiler for both writing the codes and executing them [6]. As I expected Perl had the most functionalities again, redo keyword confused me at first. I have used my Perl code as a template for other languages. Then, I have moved on to Python, I have written Python on VSCode since I have Python compiler installed on my local machine [7]. I have read the examples on LearnPython website before adapting the code [8]. It was the easiest part for me because Python was lacking some functionalities and it had the easiest syntax in my opinion. After Python, I have written the code for JS, mdn web docs was my beloved company while writing this code [9]. I have used VSCode for writing the code since node.js is installed on my computer. JavaScript was not introducing new functionalities about logically-controlled loops. After JS, I have written PHP using w3Schools compiler [10], I have looked at the examples of TutorialsPoint [11]. The numbered breaks and goto were interesting but I am not sure if they are really useful. After PHP, I finally wrote Dart code, it was really really similar to JS in terms of syntax and functionalities. I have used DartPad for writing the code and looked at TutorialsPoint for help [12], [13]. As JS, it was not introducing new features, one more time I realized how similar the languages are when you use them in the same day. After completing writing process, I have tried my codes on Dijkstra machine, I had no errors at this step and started the report by adding code snippets. After adding code snippets and outputs, I have answered the questions for all languages, then I have added explanations for each code snippet.

8. References

[1] Robert W. Sebesta, *Concepts of Programming Languages*. Pearson, 2016.

[2] Altay Güvenir, https://dijkstra.ug.bcc.bilkent.edu.tr/~guvenir/. [Accessed on: April 23, 2022].

[3] "Perl | Loops (for, foreach, while, do…while, until, Nested loops)", GeeksForGeeks. Available: https://www.geeksforgeeks.org/perl-loops-for-foreach-while-do-while-until-nested-loops/. [Accessed on: April 23, 2022].

[4] "Perl for Loop", Perl Tutorial. Available: https://www.perltutorial.org/perl-for-loop/ [Accessed on: April 23, 2022].

[5] "Loop controls: next, last, continue, break", Perl Maven. Available: https://perlmaven.com/loop-controls-next-last. [Accessed on: April 23, 2022].

[6] "Online Perl Compiler", OnlineGDB. Available: https://www.onlinegdb.com/online_perl_compiler. [Accessed on: April 23, 2022].

[7] "Visual Studio Code". Available: https://code.visualstudio.com. [Accessed on: April 23, 2022].

[8] "Loops", Learn Python. Available: https://www.learnpython.org/en/Loops. [Accessed on: April 23, 2022].

[9] "Loops and iteration", mdn web docs. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration. [Accessed on: April 23, 2022].

[10] "PHP Tryit Editor v1.2", w3Schools. Available: https://www.w3schools.com/php/phptryit.asp?filename=tryphp_compiler. [Accessed on: April 23, 2022].

[11] “PHP - Loop Types”, TutorialsPoint. Available: https://www.tutorialspoint.com/php/php_loop_types.htm. [Accessed on: April 23, 2022].

[12] “Dart Programming - Loops”, TutorialsPoint. Available: https://www.tutorialspoint.com/dart_programming/dart_programming_loops.htm. [Accessed on: April 23, 2022].

[13] “DartPad”. Available: https://dartpad.dev/. [Accessed on: April 23, 2022].