CS315
Homework 2
Ata Seren
21901575

Note: Throughout the report and my codes, I answered 2nd and 3rd questions with a single code segment for each language since it answers both questions.

Python:

1. How are the boolean values represented?

```
print("Boolean expressions with <, >, == and
!=:")
print(0 < 1)
print(0 > 1)
print(0 == 1)
print(0 != 1)
print("\n")
print("Boolean expressions with bool() function
that allows user to evaluate any value in terms
of Boolean:")
print(bool("string"))
print(bool(123))
print(bool(["Ata", "Ahmet", "Alp"]))
print("\n")
print("There are not many values that evaluate to
False in a Boolean evaluation")
print("except empty values such as (), [], {},
"", 0, the value None and the value False
itself.")
print(bool(()))
print(bool([]))
print(bool({}))
print(bool(""))
print(bool(0))
print(bool(None))
print(bool(False))
```

Boolean values can be represented as expressions with <, >, == and != operators. Other than that, values can be used to represent a Boolean value in a conditional statement or in this case, with “bool” keyword to convert value into Boolean.

Except empty values such as (), [], {}, "", 0, the value None and the value False itself, all values are evaluated to literal true value.

2. What operators are short-circuited?
   and
3. How are the results of short-circuited operators computed?

```
def helper1():
    return "Further value has been checked."
def helper2(i):
    print("Further value in comparison has been
checked.")
    return i

#prints 0 without calling helper function (short
circuit)
print("-"*10)
print(0 and helper1())
print("-"*10)

#prints helper function value (no short circuit)
print(1 and helper1())
print("-"*10)

#prints 1 without calling helper function (short
circuit)
print(1 or helper1())
print("-"*10)

#prints helper function value (no short circuit)
print(0 or helper1())
print("-"*10)

#calls helper function and doesn't print 1 (short
circuit)
print(0 or helper1() or 1)
print("-"*10)

#directly prints 1 (short circuit)
print(0 or helper1() and 1)
print("-"*10)
print("-"*10)
print("-"*10)

#prints false without calling helper function
(short circuit)
```

```
print( 5 > 10 > helper2(1) )
print("-"*10)

#prints true with calling helper function (no
short circuit)
print( 5 < 10 > helper2(1) )
print("-"*10)

#prints false with calling helper function (no
short circuit)
print( 5 < 10 > helper2(15) )
```

"and", "or" and conditional operators such as "<, >, <=, >=" are short circuited. Since compiler allows, I directly printed the elements in comparison to test the short circuit results.

Also, I declared functions to print a value which indicates that short circuit doesn't happen.

4. What are the advantages about short-circuit evaluation?

   Short circuiting prevents calling and/or evaluating unnecessary elements in an expression and in this way, increases performance.
   In addition, it prevents errors such as a loop tries to access an out-of-bounds element:

```
list = ["a", "b", "c", "d", "e"]
index = 0
while ((index < 5) and (list[index] != "f")):
    print(list[index])
    index += 1
```

   If there wouldn't be a short circuit mechanism, list[index] in following loop would try to access a non-existent index and program would give an error.

5. What are the potential problems about short-circuit evaluation?

   Despite the fact that short circuit prevents performance issues and errors, it sometimes prevents required functions from being called (side effect):

```
a = 3
b = 2
def helper3():
    return "Program successful"
print((a > b) or helper3())
#"Program successful" hasn't been printed.
```

If helper3 function would be an important one for a function, loop or any code segment, the program depends on it would fail because short circuit has occurred and function didn't even be evaluated.

JavaScript

1. How are the boolean values represented?

```
console.log("Different types of boolean
expressions in JavaScript:");
console.log(Boolean(1 < 2));
console.log(Boolean(1 > 2));
console.log(Boolean(1 != 2));
console.log(Boolean(1 == 2));
console.log("Other than conditions and
comparisons, values can be evaluated as booleans
with Boolean().");
console.log(Boolean(13));
console.log(Boolean(13.12));
console.log(Boolean(-13));
console.log(Boolean("string"));
console.log(Boolean('char'));
console.log(Boolean(13 + 12 + 68));
console.log("Most values are evaluated to
\"true\" except empty values.");
console.log(Boolean(0));
console.log(Boolean(-0));
console.log(Boolean(""));
console.log(Boolean(''));
let x;
console.log(Boolean(x));
console.log(Boolean(null));
console.log(Boolean(false));
console.log(Boolean(1/"x"));
```

Boolean values can be represented as expressions with <, >, == and != operators. Other than that, values can be used to represent a Boolean value in a conditional statement or in this case, with Boolean() function to represent value as Boolean.

Except empty values such as (), [], {}, "", 0, -0 the value “null”, the value “false” itself and undefined values or expressions, all values are evaluated to literal true value.

2. What operators are short-circuited?
   and
3. How are the results of short-circuited operators computed?

```
function helper1() {
    return "No short circuit";
}
function helper2() {
            return "Short circuit in expression
with multiple \"||\" operators";
}
console.log("\"&&\" and \"||\" operators are
short circuited.");
console.log("To show the short circuit, helper
functions \"helper1()\" and \"helper2()\" that
returns a text when called has been declared.");
console.log("If a short circuit happens, the
function won't be called and we don't see its
value.");
console.log(true && helper1());
console.log(false && helper1());
console.log(true || helper1());
console.log(false || helper1());
console.log(false || false || helper2() || true);
//true haven't been evaluated
```

"&&" and "||" are short circuited. Since compiler allows, I directly printed the elements in comparison to test the short circuit results.

Also, I declared functions to print a value which indicates that short circuit doesn’t happen.

4. What are the advantages about short-circuit evaluation?

   Short circuiting prevents calling and/or evaluating unnecessary elements in an expression and in this way, increases performance.
   In addition, it prevents errors such as a loop tries to access an out-of-bounds element:

```
const list = ["a", "b", "c", "d", "e"];
let index = 0;
while ((index < 5) && (list[index] != "f")){
    console.log(list[index]);
    index += 1;
}
```

   If there wouldn't be a short circuit mechanism, list[index] in following loop would try to access a non-existent index and program would print an undefined value.

5. What are the potential problems about short-circuit evaluation?

   Despite the fact that short circuit prevents performance issues and errors, it sometimes prevent required functions from being called (side effect):

```
let a = 3;
let b = 2;
function helper3(){
    return "Program successful";
}
console.log((a > b) || helper3());
```

   If helper3 function would be an important one for a function, loop or any code segment, the program depends on it would fail because short circuit has occured and function didn't even be evaluated.

PHP

1. How are the boolean values represented?

```
$equalVal = (1 == 1);
$notEqualVal = (1 != 1);
$lessVal = (1 < 2);
$moreVal = (1 > 2);
var_dump($equalVal);
var_dump($notEqualVal);
var_dump($lessVal);
var_dump($moreVal);
echo "Boolean expressions with \"bool\" keyword
that allows user to evaluate any value in terms
of Boolean:";

var_dump((bool) "");        // bool(false)
var_dump((bool) "0");       // bool(false)
var_dump((bool) 0);         // bool(false)
var_dump((bool) 1);         // bool(true)
var_dump((bool) -2);        // bool(true)
var_dump((bool) "foo");     // bool(true)
var_dump((bool) 2.3e5);     // bool(true)
var_dump((bool) array(12)); // bool(true)
var_dump((bool) array());   // bool(false)
var_dump((bool) "false");   // bool(true)
```

Boolean values can be represented as expressions with <, >, == and != operators. Other than that, values can be used to represent a Boolean value in a conditional statement or in this case, with Boolean() function to represent value as Boolean.

Except empty values such as (), {}, "", 0, -0 , the value “false” itself and empty arrays, all values are evaluated to literal true value.

2. What operators are short-circuited?

and

3. How are the results of short-circuited operators computed?

```
$a = (false && helper());
var_dump($a);
$b = (true  || helper());
```

```
var_dump($b);
$c = (false and helper());
var_dump($c);
$d = (true  or  helper());
var_dump($d);
echo PHP_EOL;
```

"and", "or","&&" and "||" are short circuited. To show the short circuit, a helper function helper() that returns a text when called has been declared. If a short circuit happens, the function won't be called and we don't see its value.

Null coalescent operator "??" is used to check if a variable exists and it is short-circuited too. However it is in PHP 7 so it may cause some issues on Dijkstra machine.

```
var_dump(false ?? "no short circuit");// false
var_dump(true ?? "no short circuit");// true
var_dump('' ?? "no short circuit");// ""
var_dump(0 ?? "no short circuit");// 0
var_dump([] ?? "no short circuit");// array()
var_dump(null ?? "no short circuit");// no
short circuit in this case
```

4. What are the advantages about short-circuit evaluation?

   Short circuiting prevents calling and/or evaluating unnecessary elements in an expression and in this way, increases performance. In addition, it prevents errors such as a loop tries to access an out-of-bounds element:

```
$arr = array("a", "b", "c", "d", "e");
$index = 0;

while (($index < 5) and ($arr[$index] != "f")){
    echo "$arr[$index]" . PHP_EOL;
    $index += 1;
}
```

If there wouldn't be a short circuit mechanism, array accessing code segment in following loop would try to access a non-existent index and program would give an error.

5. What are the potential problems about short-circuit evaluation?

   Despite the fact that short circuit prevents performance issues and errors, it sometimes prevent required functions from being called (side effect):

```
$x = 3;
$y = 2;
function helper2(){
    echo "Program successful";
}
print_r(($x > $y) or helper2());
echo PHP_EOL;
```

   If helper2 function would be an important one for a function, loop or any code segment, the program depends on it would fail because short circuit has occurred and function didn't even be evaluated.

Dart

1. How are the boolean values represented?

```
var a = 5;
var b = 10;
print(a == b);
print(a != b);
print(a < b);
print(a > b);
```

   Unlike many other languages, Boolean data type in Dart recognizes only the literal true as true. Any other value is considered as false.

```
/*
if(str) {
    print('String is not empty');
}
else {
```

```
    print('Empty String');
}
*/
```

In some compilers, this codes segment works and "str" is considered as false in if statement.

2. What operators are short-circuited?

and

3. How are the results of short-circuited operators computed?

Only "&&" and "||" are short circuited in Dart language.

```
print(false && helperTrue());
print("--------------------");
print(true && helperTrue());
print("--------------------");
print(false || helperFalse());
print("--------------------");
print(true || helperFalse());
```

To show the short circuit, helperTrue() and helperFalse() functions have been declared. They return true and false values, respectively. If a short circuit happens, functions won't be called and we won't see a printed value.

4. What are the advantages about short-circuit evaluation?

Short circuiting prevents calling and/or evaluating unnecessary elements in an expression and in this way, increases performance. In addition, it prevents errors such as a loop tries to access a out-of-bounds element:

```
var list = ["a", "b", "c", "d", "e"];
var index = 0;

while ((index < 5) && (list[index] != "f")){
    print(list[index]);
    index += 1;
}
```

If there wouldn't be a short circuit mechanism, list[index] in following loop would try to access a non-existent index and program would give an error.

5. What are the potential problems about short-circuit evaluation?

   Despite the fact that short circuit prevents performance issues and errors, it sometimes prevents required functions from being called (side effect):

```
    var x = 3;
    var y = 2;

    print((x > y) || helperError());
    //"Program successful." haven't been
printed.
```

   If helperError function would be an important one for a function, loop or any code segment, the program depends on it would fail because short circuit has occurred and function didn't even be evaluated.

   Declared functions:

```
bool helperTrue() {
    print("No short circuit");
    return true;
}

bool helperFalse() {
    print("No short circuit");
    return false;
}

bool helperError(){
    print("Program successful.");
    return false;
}
```

Rust

1. How are the boolean values represented?

```
let bool_val1 = true;
let bool_val2 = false;
println!("{}", bool_val1);
println!("{}", bool_val2);
```

Other than literal values, Boolean values can be represented with conditionals in Rust:

```
let a = 20;
let b = 30;
let bool_val3 = a > b;
let bool_val4 = a < b;
let bool_val5 = a == b;
let bool_val6 = a != b;
println!("{}", bool_val3);
println!("{}", bool_val4);
println!("{}", bool_val5);
println!("{}", bool_val6);
```

2. What operators are short-circuited?

and

3. How are the results of short-circuited operators computed?

"&&" and "||" are short circuited. To show the short circuit, helper() function has been declared. If a short circuit happens, function won't be called and we won't see a printed value:

```
let a = false && helper();
println!("{}", a);
println!("--------------------");
let b = true && helper();
println!("{}", b);
println!("--------------------");
let c = false || helper();
println!("{}", c);
println!("--------------------");
let d = true || helper();
println!("{}", d);
```

4. What are the advantages about short-circuit evaluation?

   Short circuiting prevents calling and/or evaluating unnecessary elements in an expression and in this way, increases performance. In addition, it prevents errors such as a loop tries to access a out-of-bounds element:

```
let arr = ["a", "b", "c", "d", "e"];
let mut index = 0;

while index < 5 && arr[index] != "f"{
    println!("{}", arr[index]);
    index += 1;
}
```

   If there wouldn't be a short circuit mechanism, arr[index] in following loop would try to access a non-existent index and program would give an error.

5. What are the potential problems about short-circuit evaluation?

   Despite the fact that short circuit prevents performance issues and errors, it sometimes prevent required functions from being called (side effect):

```
let x = 3;
let y = 2;

let result = (x > y) || helper_success();
println!("{}", result);
//"Program successful." haven't been printed.
```

   If helper_success function would be an important one for a function, loop or any code segment, program depends on it would fail because short circuit has occurred and function didn't even be evaluated which prints a text.

   Declared functions:

```
fn helper() -> bool {
    println!("No short circuit");
    true
```

```
}

fn helper_success() -> bool {
    println!("Program successful.");
    true
}
```

Which language is the best for array operations?

In my opinion, Python is the best language for short-circuit evaluation among the evaluated languages. First of all, it supports value evaluation to Boolean values which means that different values can be represented as Boolean values. This is actually both an advantage and disadvantage. It's an advantage because it offers different options for Boolean representations to users that makes writing a code easier. It's a disadvantage because it may be difficult to remember all types of representations and using wrong values for Boolean evaluation can be harmful for the code execution. However, it's still an advantage since only empty values must be considered to get the desired Boolean value in the evaluation. Other than the mechanics, Python has a great readability and writability in all areas and it is a huge plus among other languages. On the other hand, JavaScript has similar properties to Python's. However, "<" and ">" operators are supported for short-circuiting in Python different from JavaScript which can be one of the few differences between these languages in terms of short-circuiting. To summarize, Python is the best language for short-circuiting among evaluated languages because it supports various features for short-circuiting different from other languages and it's easy to read and write.

My learning strategy for this homework:

From our project, my previous experiences about learning the code and first homework, I know that documentations of a languages include literally everything about that language. Therefore, I thought that I can find the information about the questions of second homework in these documentations. I go language by language and started to check the documentation of Python, JavaScript, PHP, Dart and Rust, respectively.

For the first question, documentations helped me a lot because information about the Boolean values is very precise and definitely correct in these documentations. In websites like StackOverflow, this information can be

found too, but when people use them in their own code, it is difficult to know that it is reliable since people can manipulate the Boolean expressions to fit them in their code.

Unfortunately, documentations were not useful for 2nd and 3rd questions because some of them don't directly explain the short-circuit evaluation of the operators (except PHP, it has a very comprehensive and useful documentation). Therefore, I used websites such as W3schools, GeeksForGeeks and StackOverflow where I can find both the explanation of mechanism of short-circuit and example codes specific to the language. By inspiring from these examples, I declared functions which I used in expressions with short-circuited operators and make me understand when short-circuit happens or not by returning and/or printing values. With these expressions, I experimented on all short-circuited operators by declaring and printing them in different ways according to the language.

For the 4th and 5th questions, different than the previous homework and questions, I used our textbook. Under 7.6 Short-Circuit Evaluation, there are 2 examples about advantage and disadvantage of short-circuiting. In the advantage example, short-circuiting prevented a while loop to access non-existent part of an array by checking a condition and not evaluating other. In the disadvantage example, a line of code is given. According to an explanation, it's from a loop too. It includes an increment as a part of expression and if short-circuit happens, this increment won't happen which may be important for the rest of the code. By using these examples, I recreated the scenarios and wrote codes that shows the advantage and disadvantage of short-circuiting.

To summarize my learning strategy, I used different types of sources for each question. For first question, I checked the official documentations of the languages for Boolean values. Since I'm familiar with the syntax of the languages, I easily understood the representation ways of Booleans values for each language and implemented them in my codes. For second and third questions, documentations were insufficient because many of them don't directly give the information about short-circuiting mechanisms in their language. Therefore, I used educational and Q&A coding platforms since they have an explanation and example codes of short-circuit operators. I created experimental codes by inspiring from these examples. For fourth and fifth questions, our textbook helped me to write experimental codes for 2 scenarios of advantage and disadvantage of short circuiting. I can say that this homework has improved my skill of learning from different types of sources at the same time.

REFERENCES

[1] https://www.w3schools.com/python/python_booleans.asp

[2] https://www.geeksforgeeks.org/short-circuiting-techniques-python/

[3] https://stackoverflow.com/questions/2580136/does-python-support-short-circuiting

[4] https://codeburst.io/javascript-what-is-short-circuit-evaluation-ff22b2f5608c

[5] https://www.geeksforgeeks.org/javascript-short-circuiting/

[6] https://www.w3schools.com/js/js_functions.asp

[7] https://www.php.net/manual/en/language.operators.logical.php

[8] https://www.php.net/manual/en/language.types.boolean.php

[9] https://www.phptutorial.net/php-tutorial/php-null-coalescing-operator/

[10] https://www.tutorialspoint.com/dart_programming/dart_programming_boolean.htm

[11] https://api.dart.dev/stable/2.14.4/dart-core/bool-class.html

[12] https://stackoverflow.com/questions/53644809/do-logical-operators-short-circuit-in-rust

[13] https://doc.rust-lang.org/reference/expressions/operator-expr.html#lazy-boolean-operators

[14] https://doc.rust-lang.org/std/primitive.bool.html