CS315 Homework 2

Rüzgar Ayan 21801984

1 Question 1. How are the boolean values represented?

1.1 Dart

After trying all possible data types including booleans, numbers, strings, arrays, objects and undefined or null values, we see that only the variables of type bool can evaluate to booleans when used in if statements. When we try to use a String or a number, it gives the error seen in the example below. So, boolean values are represented in bool variables and also as the results of logical and comparison operators.

```
bool boolTrue = true;
bool boolFalse = false;
double negNumber = -1.1;
int zeroNumber = 0;
String emptyString = "";
String nonemptyString = "abc";
var undefinedVar;
var nullVar = null;

if (boolTrue) print("boolTrue"); //Prints
if (boolFalse) print("boolFalse");

//Error: A value of type '***' can't be assigned to a variable of type
'bool'
/*
if (negNumber) print("negNumber");
if (zeroNumber) print("zeroNumber");
if (emptyString) print("emptyString");
if (nonemptyString) print("nonemptyString");
*/

//Uncaught Error: TypeError: null: type 'JSNull' is not a subtype of type
'bool'
//if (nullVar) print("nullVar");
//if (undefinedVar) print("undefinedVar");
```

1.2 Javascript

After trying all possible data types including booleans, numbers, strings, arrays, objects and undefined or null values, we see that all of these can evaluate to a boolean value and be used in if statements. In the example below, different values (zero/nonzero or empty/nonempty) for all these data types are tried. The variables that evaluate to false boolean values are the number 0, empty strings and undefined or null values. All the other values evaluate to true when used in an if statement and it doesn't matter whether or not an array or object is empty.

```
var boolTrue = true;
var boolFalse = false;
var negNumber = -1.1;
var zeroNumber = 0;
var posNumber = 1.1;
var emptyString = "";
var nonemptyString = "abc";
var emptyArray = [];
var nonemptyArray = ["abc", 123];
var emptyObject = {};
var nonemptyObject = {key: "value"};
var undefinedVar;
var nullVar = null;

if (boolTrue) console.log("boolTrue"); //Prints
if (boolFalse) console.log("boolFalse");
if (negNumber) console.log("negNumber"); //Prints
if (zeroNumber) console.log("zeroNumber");
if (posNumber) console.log("posNumber"); //Prints
if (emptyString) console.log("emptyString");
if (nonemptyString) console.log("nonemptyString"); //Prints
if (emptyArray) console.log("emptyArray"); //Prints
if (nonemptyArray) console.log("nonemptyArray"); //Prints
if (emptyObject) console.log("emptyObject"); //Prints
if (nonemptyObject) console.log("nonemptyObject"); //Prints
if (undefinedVar) console.log("undefinedVar");
if (nullVar) console.log("nullVar");
```

1.3 PHP

The answer to this is almost as same as the Javascript answer with one exception. In Javascript, empty arrays were evaluated as true but in PHP, they are evaluated as false when used in if statements. The example code below shows this.

```
    $boolTrue = True;
    $boolFalse = False;
    $negNumber = -1.1;
    $zeroNumber = 0;
    $posNumber = 1.1;
    $emptyString = "";
    $nonemptyString = "abc";
    $undefinedVar;
    $nullVar = NULL;
    $empty_array = [];
    $nonempty_array = ["abc", 1, 2];

    if ($boolTrue) echo "boolTrue\n"; //Prints
    if ($boolFalse) echo "boolTrue\n";
    if ($negNumber) echo "negNumber\n"; //Prints
    if ($zeroNumber) echo "zeroNumber\n";
    if ($posNumber) echo "posNumber\n"; //Prints
    if ($emptyString) echo "emptyString\n";
    if ($nonemptyString) echo "nonemptyString\n"; //Prints
    if ($undefinedVar) echo "undefinedVar\n";
    if ($nullVar) echo "nullVar\n";
    if ($empty_array) echo "empty_array\n";
    if ($nonempty_array) echo "nonempty_array\n"; //Prints
```

1.4 Python

The answer to this is almost as same as the PHP answer. We can put all kinds of data types into if statements. Nonzero numbers, nonempty strings and nonempty arrays evaluate to True. The only difference from PHP is that there are no undefined variables in Python.

```
boolTrue = True;
boolFalse = False;
negNumber = -1.1;
zeroNumber = 0;
posNumber = 1.1;
emptyString = "";
nonemptyString = "abc";
#undefinedVar; not possible in Python
nullVar = None;
empty_array = [];
nonempty_array = ["abc", 1, 2];

if (boolTrue) :
    print("boolTrue") #Prints
if (boolFalse) :
    print("boolFalse")
if (negNumber) :
    print("negNumber") #Prints
if (zeroNumber) :
    print("zeroNumber")
if (posNumber) :
    print("posNumber") #Prints
if (emptyString) :
    print("emptyString")
if (nonemptyString) :
    print("nonemptyString") #Prints
if (nullVar) :
    print("nullVar")
if (empty_array) :
    print("empty_array")
if (nonempty_array) :
    print("nonempty_array") #Prints
```

1.5 Rust

Similar to Dart, if we try to use any variable other than “bool” type inside if statements, we get an error. So, we understand that booleans are represented with only the bool variables (or as the result of logical and comparison operators).

```
    let boolTrue = true;
    let boolFalse = false;
    let string = "abc";
    let number = 1;

    if boolTrue { println!("boolTrue"); }
    if boolFalse { println!("boolFalse"); }

    //These below give error[E0308]: mismatched types
    //expected `bool`, found integer/&str
    if string { println!("string"); }
    if number { println!("number"); }
    // We understand that only boolean variables are used
```

2 Question 2. What operators are short-circuited?

2.1 Dart

The best-known short-circuit operators “&&” and “||” are tested together with their bitwise versions “&” and “|” and also the multiplication operator. For the bitwise and, the first operand is set to 0 (number with all bits set to 0). For the bitwise or, the first operand is set to “~0” (number with all bits set to 1). Although the first operands in these bitwise operators already determine the end result, they still look at the second operand. The bitwise xor operator “^” is not checked because it is not possible to determine the end result from one operand as in the other examples.

```
bool printBool(String input) {
    print("Printed at " + input + "!"); return true;
}
int printInt(String input) {
    print("Printed at " + input + "!"); return 1;
}

bool and = (false && printBool("and"));
bool or = (true || printBool("or"));
int bitwiseAnd = (0 & printInt("bitwiseAnd")); //Prints
int bitwiseOr = (~0 | printInt("bitwiseOr")); //Prints
int multiplication = (0 * 5 * printInt("multiplication")); //Prints
```

2.2 Javascript

The answer is the same as the Dart language. Supported operators are “&&” and “||”.

```
function print(input) {
    console.log("Printed at " + input + "!"); return 1;
}
const and = (false && print("and"));
const or = (true || print("or"));
const bitwiseAnd = (0 & print("bitwiseAnd")); //Prints
const bitwiseOr = (~0 | print("bitwiseOr")); //Prints
const multiplication = (0 * 5 * print("multiplication")); //Prints
```

2.3 PHP

The answer is almost the same as the other languages. There are also two operators “and” and “or” in PHP that are equivalent to “&&” and “||” except their precedence. These two operator are also short-circuit evaluated but I won’t use them in the later examples because they do the same thing with the ordinary ones.

```
    function printMsg($msg) {
        echo "$msg \n";
        return True;
    }

    $and_ = (False && printMsg("and"));
    $and2_ = (False and printMsg("and2"));
    $or_ = (True || printMsg("or"));
    $or2_ = (True or printMsg("or2"));
    $bitwiseAnd = (0 & printMsg("bitwiseAnd")); //Prints
    $bitwiseOr = (~0 | printMsg("bitwiseOr")); //Prints
    $multiplication = (0 * 5 * printMsg("multiplication")); //Prints
```

2.4 Python

The answer is the same as the other languages, just the operator symbols are different. Other languages use “&&” and “||” whereas Python uses “and” and “or” but still with short-circuit evaluation.

```
def printMsg(msg):
    print(msg)
    return True

and_ = (False and printMsg("and"))
or_ = (True or printMsg("or"))
bitwiseAnd = (0 & printMsg("bitwiseAnd")) #Prints
bitwiseOr = (~0 | printMsg("bitwiseOr")) #Prints
multiplication = (0 * 5 * printMsg("multiplication")) #Prints
```

2.5 Rust

The answer is the same as the other languages. Only “&&” and “||” operators support short-circuit evaluation.

```
    fn printBool(msg: &str) -> bool {
        println!("{}", msg);
        return true;
    }
    fn printInt(msg: &str) -> i32 {
        println!("{}", msg);
        return 1;
    }

    let and_ = (false && printBool("and"));
    let or_ = (true || printBool("or"));
    let bitwiseAnd = (0 & printInt("bitwiseAnd")); //Prints
    let bitwiseOr = (!0 | printInt("bitwiseOr")); //Prints
    let multiplication = (0 * 5 * printInt("multiplication")); //Prints
```

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

3.1 Dart

As shown in Question 2, only the operators “&&” and “||” are short-circuited. The way these are computed is that if the left operand of a “&&” operator is evaluated as false, then its right operand is not evaluated at all since the overall result of this operation will be false. Similarly for the “||” operator, if the first operand is true, then its right operand is not evaluated at all since the overall result of this operation will be true.

```
bool printBool(String input) {
    print("Printed at " + input + "!"); return true;
}

//Prints a and b
bool shortcircuit = printBool("a") && printBool("b") || printBool("c");
//Prints a and b
bool shortcircuit2 = false || printBool("a") && (printBool("b") ||
printBool("c"));
```

3.2 Javascript

The answer is the same as the Dart language.

```
function print(input) {
    console.log("Printed at " + input + "!"); return 1;
}

//Prints a and b
const shortcircuit = print("a") && print("b") || print("c");
//Prints a and b
const shortcircuit2 = 0 || print("a") && (print("b") || print("c"));
```

3.3 PHP

The answer is the same as the other languages.

```
    //Prints a and b
    $shortcircuit = printMsg("a") && printMsg("b") || printMsg("c");
    //Prints a and b
    $shortcircuit2 = False || printMsg("a") && (printMsg("b") ||
printMsg("c"));
```

3.4 Python

The answer is the same as the other languages.

```
#Prints a and b
shortcircuit = printMsg("a") and printMsg("b") or printMsg("c")
#Prints a and b
shortcircuit2 = False or printMsg("a") and (printMsg("b") or printMsg("c"))
```

3.5 Rust

The answer is the same as the other languages.

```
    //Prints a and b
    let shortcircuit = printBool("a") && printBool("b") || printBool("c");
    //Prints a and b
    let shortcircuit2 = false || printBool("a") && (printBool("b") ||
printBool("c"));
```

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

4.1 Dart

If there were no short-circuit evaluation, the program would have to do unnecessary computation in case the operands contain expensive computations. In the example below, short-circuiting allows the program to not evaluate the expensiveComputation() function. Also note that this could be achieved with nested if statements for both “&&” and “||” operations as shown in the example again. But these nested if loops reduce readability and writability, or in other words, short-circuit evaluation increases the readability and writability in these cases. It is shown that the unnecessary expensive computation is done only in the last if statement.

```
int expensiveComputation(String name) {
  int sum = 0;
  for (int i = 0; i < 10000; i++) {
    sum += i;
  }
  print("Did expensive computation at " + name);
  return sum;
}


var smallNumber = 1;
//With short-circuit
if (smallNumber > 10 && expensiveComputation("if1") > 10000) {
  print("Inside if 1");
}

//Simulating the if above without short-circuit
if (smallNumber > 10) {
  if (expensiveComputation("if2") > 10000) {
    print("Inside if 2");
  }
}

//Not doing short-circuit evaluation because of wrong order
//Prints "Did expensive computation at if3" only for this one
if (expensiveComputation("if3") > 10000 && smallNumber > 10) {
  print("Inside if 3");
}
```

4.2 Javascript

The answer is the same as the Dart language.

```
function expensiveComputation() {
    var sum = 0;
    for (var i = 0; i < 10000; i++) sum += i;
    return sum;
}

var smallNumber = 1;
//With short-circuit
if (smallNumber > 10 && expensiveComputation() > 10000) {
    console.log("Inside if 1");
}

//Without short-circuit
if (smallNumber > 10) {
    if (expensiveComputation() > 10000) {
        console.log("Inside if 2");
    }
}
```

4.3 PHP

The answer is the same as the other languages.

```php
    function expensiveComputation($name) {
        $sum = 0;
        for ($i = 0; $i <= 10000; $i++) {
            $sum += $i;
        }
        echo "Did expensive computation at $name \n";
        return $sum;
    }

    $smallNumber = 1;
    //With short-circuit
    if ($smallNumber > 10 && expensiveComputation("if1") > 10000) {
        echo "Inside if 1\n";
    }

    //Simulating the if above without short-circuit
    if ($smallNumber > 10) {
        if (expensiveComputation("if2") > 10000) {
          echo "Inside if 2\n";
        }
    }

    //Not doing short-circuit evaluation because of wrong order
    //Prints "Did expensive computation at if3" only for this one
    if (expensiveComputation("if3") > 10000 && $smallNumber > 10) {
        echo "Inside if 3\n";
    }
```

4.4 Python

The answer is the same as the other languages.

```python
def expensiveComputation(name):
    sum = 0
    for i in range(10000):
        sum = sum + i
    print("Did expensive computation at " + name)
    return sum

smallNumber = 1;
#With short-circuit
if smallNumber > 10 and expensiveComputation("if1") > 10000:
    print("Inside if 1")

#Simulating the if above without short-circuit
if smallNumber > 10:
    if expensiveComputation("if2") > 10000:
        print("Inside if 2");

#Not doing short-circuit evaluation because of wrong order
#Prints "Did expensive computation at if3" only for this one
if expensiveComputation("if3") > 10000 and smallNumber > 10:
    print("Inside if 3")
```

4.5 Rust

The answer is the same as the other languages.

```
    fn expensiveComputation(name: &str) -> i32 {
        let mut sum = 0;
        for i in 0..10000 {
            sum += i;
        }
        println!("Did expensive computation at {}", name);
        return sum;
    }

    let smallNumber = 1;
    //With short-circuit
    if smallNumber > 10 && expensiveComputation("if1") > 10000 {
        println!("Inside if 1");
    }

    //Simulating the if above without short-circuit
    if smallNumber > 10 {
        if expensiveComputation("if2") > 10000 {
        println!("Inside if 2");
        }
    }

    //Not doing short-circuit evaluation because of wrong order
    //Prints "Did expensive computation at if3" only for this one
    if expensiveComputation("if3") > 10000 && smallNumber > 10 {
        println!("Inside if 3");
    }
```

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

5.1 Dart

If the parts of the boolean evaluation that are skipped because of the short-circuit evaluation contains some functions/statements that have side-effects, then these side-effects won't occur. Programmers must consider this when adding functions with side-effects as operands in logical evaluations.

```
int globalInt = 1;
bool sideEffectComputation() {
  globalInt = 55;
  return true;
}

var var1 = smallNumber > 10 && sideEffectComputation();
print(globalInt); //Prints 1, not changed
var var2 = sideEffectComputation() && smallNumber > 10;
```

```
print(globalInt); //Prints 55, changed this time
```

5.2 Javascript

The answer is the same as the Dart language.

```
function sideEffectComputation() {
    globalVariable = "I have been changed";
    return 1;
}

var globalVariable = "Initial value";
const var1 = smallNumber > 10 && sideEffectComputation();
console.log(globalVariable); //Prints "Initial Value"
const var2 =  sideEffectComputation() && smallNumber > 10;
console.log(globalVariable); //Prints "I have been changed"
```

5.3 PHP

The answer is the same as the other languages.

```
    $globalVariable = "Initial value";
    function sideEffectComputation() {
        global $globalVariable;
        $globalVariable = "I have been changed";
        return True;
    }
    $var1 = $smallNumber > 10 && sideEffectComputation();
    echo "$globalVariable \n"; //Prints "Initial value"
    $var2 = sideEffectComputation() && $smallNumber > 10;
    echo "$globalVariable \n"; //Prints "I have been changed"
```

5.4 Python

The answer is the same as the other languages.

```
globalVariable = "Initial value";
def sideEffectComputation():
    global globalVariable
    globalVariable = "I have been changed"
    return True

var1 = smallNumber > 10 and sideEffectComputation()
print(globalVariable) #Prints "Initial value"
var2 = sideEffectComputation() and smallNumber > 10
print(globalVariable) #Prints "I have been changed"
```

5.5 Rust

The answer is the same as the other languages.

```
    static mut globalVariable: &str = "Initial value";

    fn sideEffectComputation() -> bool {
        unsafe {
            globalVariable = "I have been changed";
        }
        return true;
    }

    unsafe {
        let var1 = smallNumber > 10 && sideEffectComputation();
        println!("{}", globalVariable); //Prints "Initial value"
        let var2 = sideEffectComputation() && smallNumber > 10;
        println!("{}", globalVariable); //Prints "I have been changed"
    }
```

Discussion Question 1. Best Language for Short-circuit Evaluation

First of all, I would eliminate Dart and Rust from the alternatives because the representation of booleans in these languages only covers the boolean variables. We cannot use values such as empty strings/arrays or the number zero in these short-circuit evaluations which is usually the case. Or we cannot even use null values in short-circuiting in Dart. Three options remain: Javascript, PHP and Python. Since the short-circuit evaluation operators are all the same (“&&” and “||” equivalents) in these languages, I cannot comment from this perspective. One little detail I have noticed between these choices were that in PHP and Python, empty arrays evaluate to false as booleans whereas they evaluate to true in Javascript. This gives slightly more usage to short-circuiting in PHP and Python, so I eliminate Javascript as well.

Between the last two alternatives PHP and Python, there is not much of a difference in terms of short-circuit evaluation. So I make my choice as Python only on the fact that it already tries to be a highly readable language and this short-circuiting feature fits better in Python programs.

Discussion Question 2. Learning Strategy

I started with reading the section from the course textbook about short-circuit evaluation and seeing the possible answers to these questions from there. Other than the textbook, I also found the Wikipedia page for short-circuit evaluation very helpful as it was listing the short-circuit operators in many languages. Although it was listing only the and/or operators in the languages I am interested in, I still tried to short-circuit some other operations such as multiplication and bitwise and/or. That way, I was able to make sure that the information I have seen on the Wikipedia page was correct.

In comparison to the first homework, I noticed that there was almost no difference between the languages in terms of the way they do short-circuit evaluation. Other than the first question, the answers to the other 4 questions were almost the same for all languages. After noticing that, I have written an inclusive test program for these questions and converted the same program into the other languages to verify that they all give the same result. That way, I was able to make sure that short-circuiting was working as intended.

Also different from the first homework, there was very little information in the official programming language documentations. This was probably because short-circuit evaluation was a much more limited topic compared to the arrays and array operators. This caused me to learn by playing with the code and experimenting with different cases.

The hardest question for me was the first one. I couldn’t find online sources such as language documentation on what kinds of data types can evaluate to booleans. So, I had to try a lot of possible data types and values to determine how a boolean value is represented. I tried to choose these smartly so that I can understand something about the boolean representations from its result. For example, I have tried both empty and nonempty arrays and seen that they might evaluate to different or the same boolean values in different languages. In some cases such as Dart and Rust, I directly got errors giving me the results without much effort. But the other languages were harder because they are much more flexible and they were allowing me to put any data type into if statement conditions.