CS315 - Programming Languages

Homework 1

Deniz Çalkan
21703994
Section 01

1. Design Issues **2**
1.1 Boolean Operators Provided 2
1.2 Data Types for Operands of Boolean Operators 7
1.3 Operator Precedence Rules 14
1.4 Operator Associativity Rules 19
1.5 Operand Evaluation Order 24
1.6 Short Circuit Evaluation 27
2. Evaluation of Languages **34**
3. Learning Strategy **35**

1. Design Issues

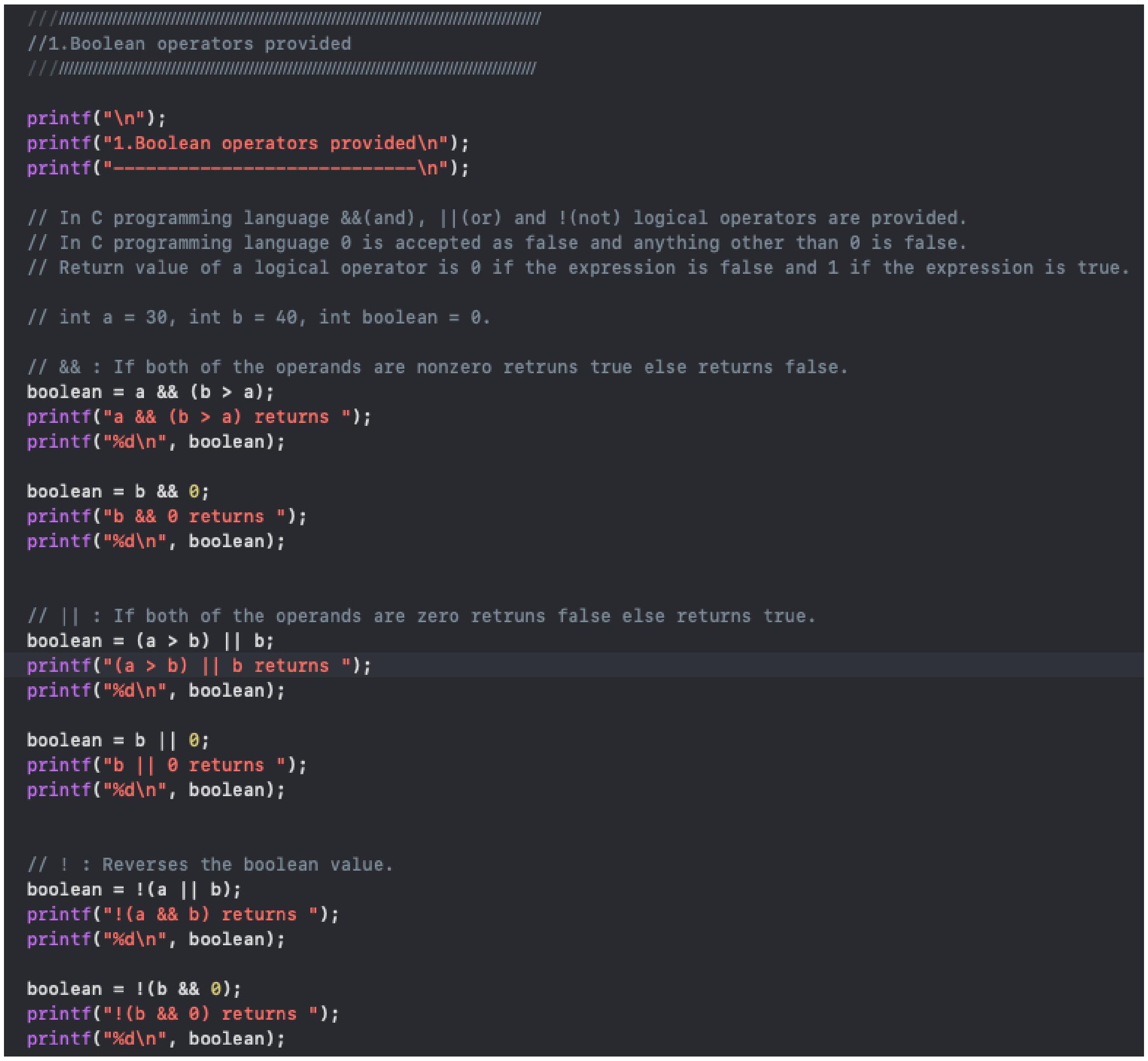
Note: All of the code examples use variables a = 30, b = 40, bool = false and 3 functions func1 (returns false) func2 and func3 (returns true). Also functions have print statements inside to make it easier to understand which function was executed.

1.1 Boolean Operators Provided

In this part examples are provided to show the existing logical operators in each language. The way these logical operators work and the resulting return values are provided as well. Further explanation is done with comments inside code segments.

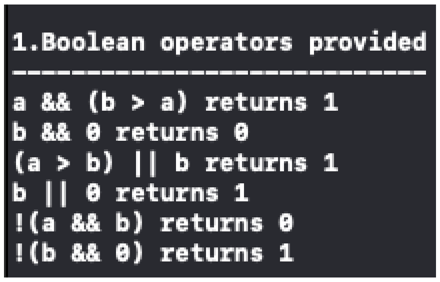
C

Code Segment



```c
///////////////////////////////////////////////////////////////////////////////////////////////////
//1.Boolean operators provided
///////////////////////////////////////////////////////////////////////////////////////////////////

printf("\n");
printf("1.Boolean operators provided\n");
printf("----------------------------\n");

// In C programming language &&(and), ||(or) and !(not) logical operators are provided.
// In C programming language 0 is accepted as false and anything other than 0 is false.
// Return value of a logical operator is 0 if the expression is false and 1 if the expression is true.

// int a = 30, int b = 40, int boolean = 0.

// && : If both of the operands are nonzero retruns true else returns false.
boolean = a && (b > a);
printf("a && (b > a) returns ");
printf("%d\n", boolean);

boolean = b && 0;
printf("b && 0 returns ");
printf("%d\n", boolean);


// || : If both of the operands are zero retruns false else returns true.
boolean = (a > b) || b;
printf("(a > b) || b returns ");
printf("%d\n", boolean);

boolean = b || 0;
printf("b || 0 returns ");
printf("%d\n", boolean);


// ! : Reverses the boolean value.
boolean = !(a || b);
printf("!(a && b) returns ");
printf("%d\n", boolean);

boolean = !(b && 0);
printf("!(b && 0) returns ");
printf("%d\n", boolean);
```

Output



```
1.Boolean operators provided
----------------------------
a && (b > a) returns 1
b && 0 returns 0
(a > b) || b returns 1
b || 0 returns 1
!(a && b) returns 0
!(b && 0) returns 1
```

GO

Code Segment

```
////////////////////////////////////////////////////////////////////////////////////////////////////////
//1.Boolean operators provided
////////////////////////////////////////////////////////////////////////////////////////////////////////

fmt.Println();
fmt.Println("1.Boolean operators provided");
fmt.Println("----------------------------");

// In GO programming language &&(and), ||(or) and !(not) logical operators are provided.
// In GO programming language false and true keywords are reserved for boolean type.
// If te result of a boolean operator is true return value is true which is type bool else return value is false.

// var a = 30, var b = 40, boolean = false.

// && : If both of the operands are true retruns true else returns false.
boolean = (a < b) && (b == b);
fmt.Print("(a < b) && (b == b) returns ");
fmt.Print(boolean,"\n");


// || : If both of the operands are false retruns false else returns true.
boolean = (b > a) || (a > b);
fmt.Print("(b > a) || (a > b) returns ");
fmt.Print(boolean,"\n");


// ! : Reverses the boolean value.
boolean = !((b > a) || (a > b));
fmt.Print("!((b > a) || (a > b)) returns ");
fmt.Print(boolean,"\n");
```

Output

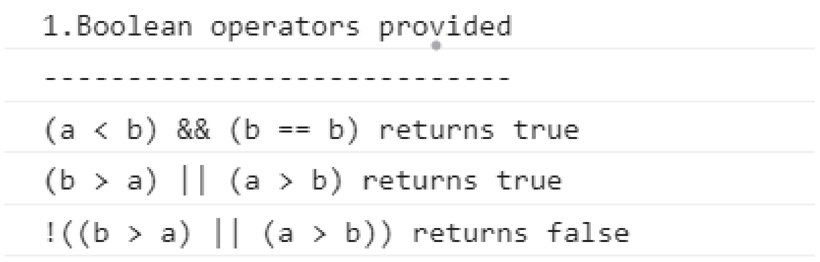
```
1.Boolean operators provided
----------------------------
(a < b) && (b == b) returns true
(b > a) || (a > b) returns true
!((b > a) || (a > b)) returns false
```

Javascript

Code Segment

```
console.log("1.Boolean operators provided")
console.log("----------------------------")


// In Javascript programming language &&(and), ||(or) and !(not) logical operators are provided.
// In Javascript programming language false and true keywords are reserved for boolean type.
// If the result of a boolean operator is true return value is true which is type bool else return value is false.

// var a = 30, var b = 40, boolean = false.

// && : If both of the operands are true returns true else returns false.
boolean = (a < b) && (b == b)
console.log("(a < b) && (b == b) returns " + boolean + "\n")
//console.log(boolean,"\n")


// || : If both of the operands are false returns false else returns true.
boolean = (b > a) || (a > b)
console.log("(b > a) || (a > b) returns "+ boolean + "\n")



// ! : Reverses the boolean value.
boolean = !((b > a) || (a > b))
console.log("!((b > a) || (a > b)) returns "+ boolean + "\n")
```

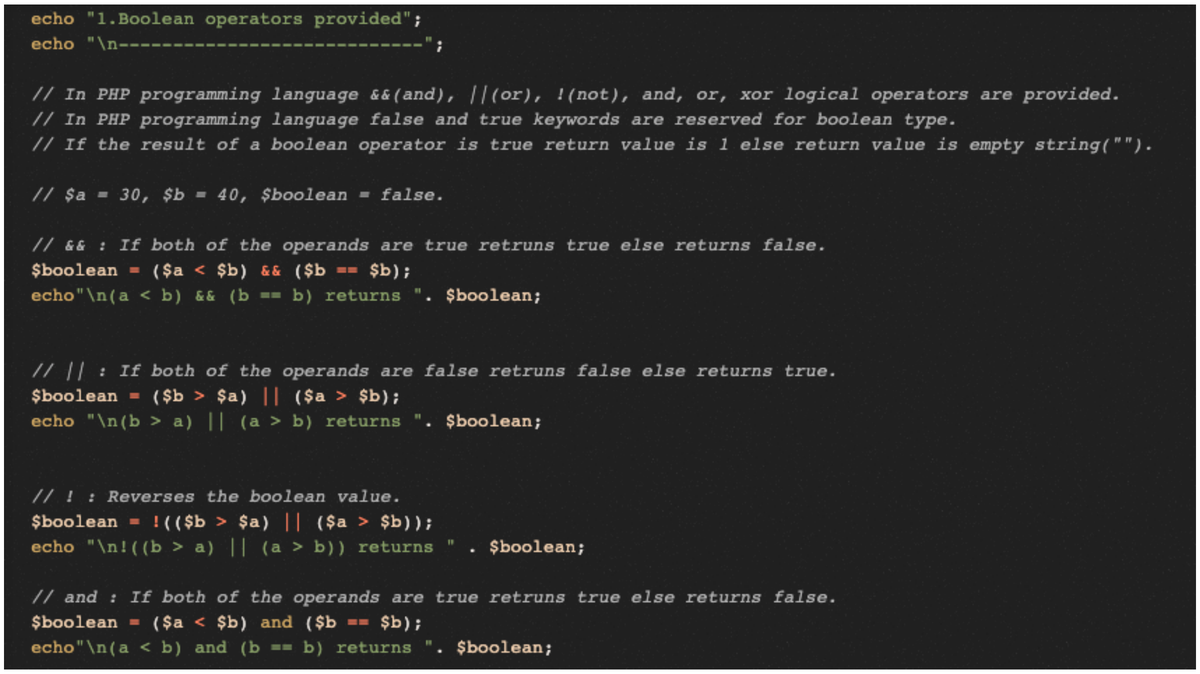
Output



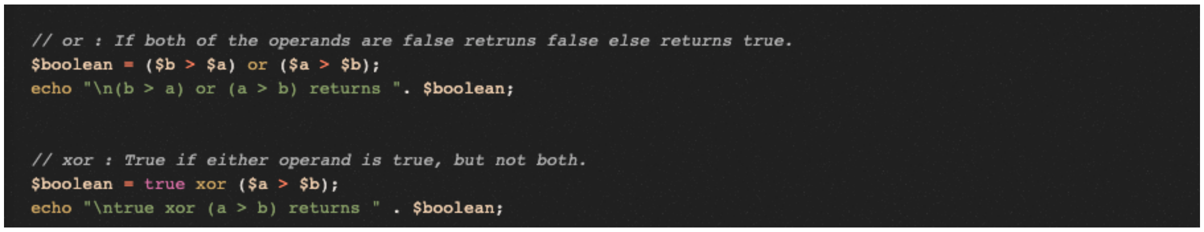
```
1.Boolean operators provided
----------------------------
(a < b) && (b == b) returns true
(b > a) || (a > b) returns true
!((b > a) || (a > b)) returns false
```

PHP

Code Segment

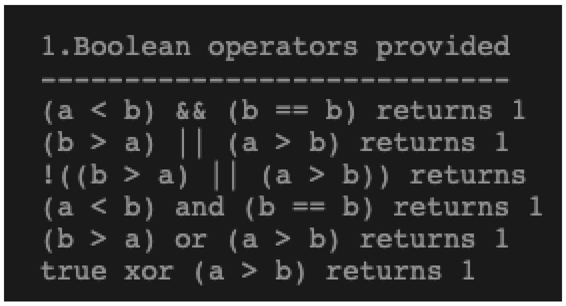


```
echo "1.Boolean operators provided";
echo "\n----------------------------";

// In PHP programming language &&(and), ||(or), !(not), and, or, xor logical operators are provided.
// In PHP programming language false and true keywords are reserved for boolean type.
// If the result of a boolean operator is true return value is 1 else return value is empty string("").

// $a = 30, $b = 40, $boolean = false.

// && : If both of the operands are true retruns true else returns false.
$boolean = ($a < $b) && ($b == $b);
echo"\n(a < b) && (b == b) returns ". $boolean;


// || : If both of the operands are false retruns false else returns true.
$boolean = ($b > $a) || ($a > $b);
echo "\n(b > a) || (a > b) returns ". $boolean;


// ! : Reverses the boolean value.
$boolean = !(($b > $a) || ($a > $b));
echo "\n!((b > a) || (a > b)) returns " . $boolean;

// and : If both of the operands are true retruns true else returns false.
$boolean = ($a < $b) and ($b == $b);
echo"\n(a < b) and (b == b) returns ". $boolean;
```



```
// or : If both of the operands are false retruns false else returns true.
$boolean = ($b > $a) or ($a > $b);
echo "\n(b > a) or (a > b) returns ". $boolean;


// xor : True if either operand is true, but not both.
$boolean = true xor ($a > $b);
echo "\ntrue xor (a > b) returns " . $boolean;
```

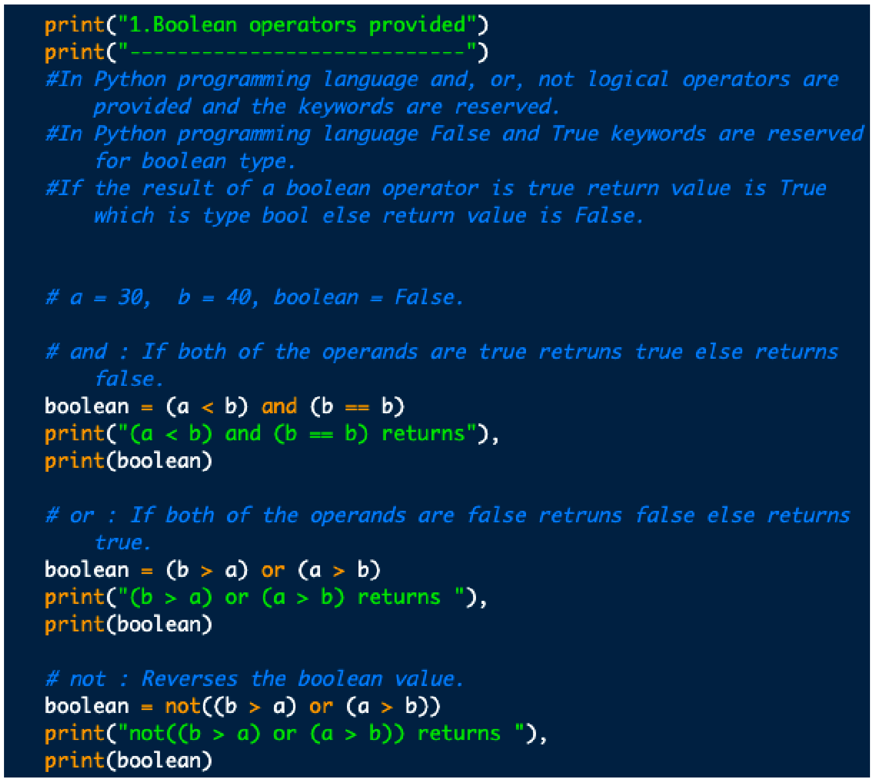
Output



```
1.Boolean operators provided
----------------------------
(a < b) && (b == b) returns 1
(b > a) || (a > b) returns 1
!((b > a) || (a > b)) returns
(a < b) and (b == b) returns 1
(b > a) or (a > b) returns 1
true xor (a > b) returns 1
```

Python

Code Segment



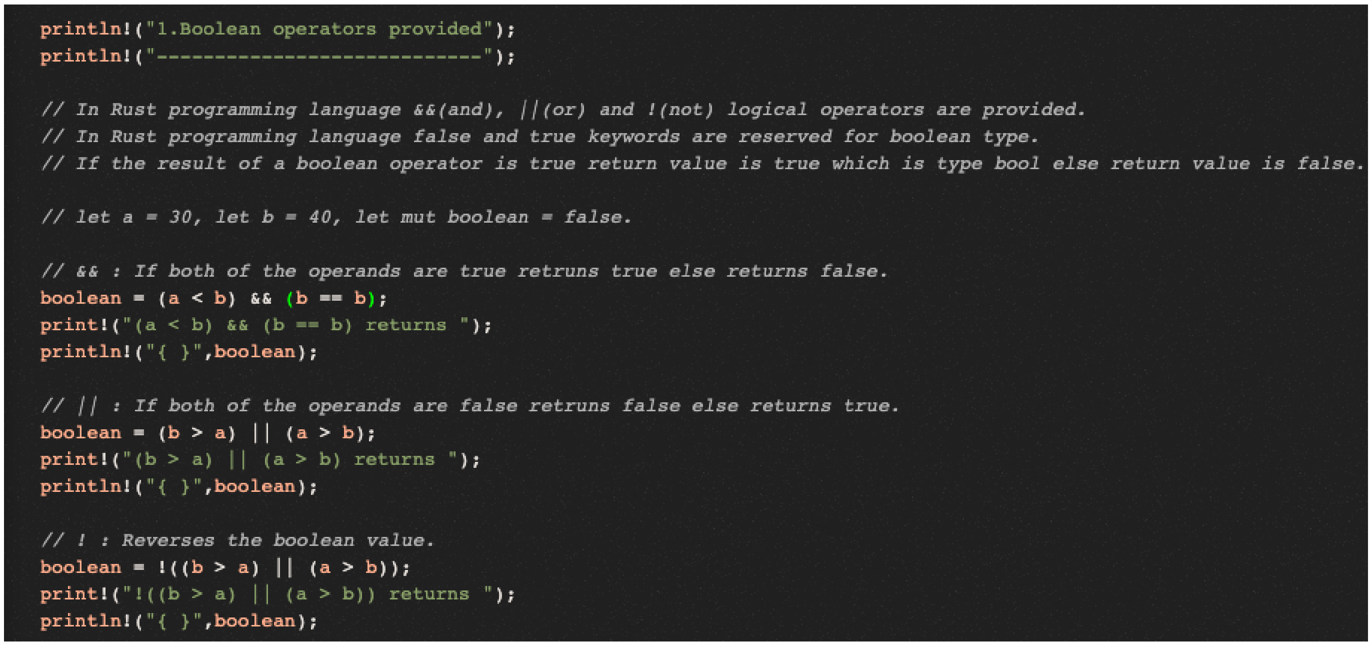
```
print("1.Boolean operators provided")
print("----------------------------")
#In Python programming language and, or, not logical operators are
    provided and the keywords are reserved.
#In Python programming language False and True keywords are reserved
    for boolean type.
#If the result of a boolean operator is true return value is True
    which is type bool else return value is False.


# a = 30,  b = 40, boolean = False.

# and : If both of the operands are true retruns true else returns
    false.
boolean = (a < b) and (b == b)
print("(a < b) and (b == b) returns"),
print(boolean)

# or : If both of the operands are false retruns false else returns
    true.
boolean = (b > a) or (a > b)
print("(b > a) or (a > b) returns "),
print(boolean)

# not : Reverses the boolean value.
boolean = not((b > a) or (a > b))
print("not((b > a) or (a > b)) returns "),
print(boolean)
```

Output

```
1.Boolean operators provided
----------------------------
(a < b) and (b == b) returns True
(b > a) or (a > b) returns  True
not((b > a) or (a > b)) returns  False
```

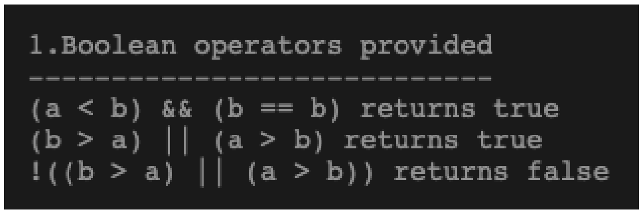
Rust

Code Segment



```
println!("1.Boolean operators provided");
println!("----------------------------");

// In Rust programming language &&(and), ||(or) and !(not) logical operators are provided.
// In Rust programming language false and true keywords are reserved for boolean type.
// If the result of a boolean operator is true return value is true which is type bool else return value is false.

// let a = 30, let b = 40, let mut boolean = false.

// && : If both of the operands are true retruns true else returns false.
boolean = (a < b) && (b == b);
print!("(a < b) && (b == b) returns ");
println!("{ }",boolean);

// || : If both of the operands are false retruns false else returns true.
boolean = (b > a) || (a > b);
print!("(b > a) || (a > b) returns ");
println!("{ }",boolean);

// ! : Reverses the boolean value.
boolean = !((b > a) || (a > b));
print!("!((b > a) || (a > b)) returns ");
println!("{ }",boolean);
```

Output



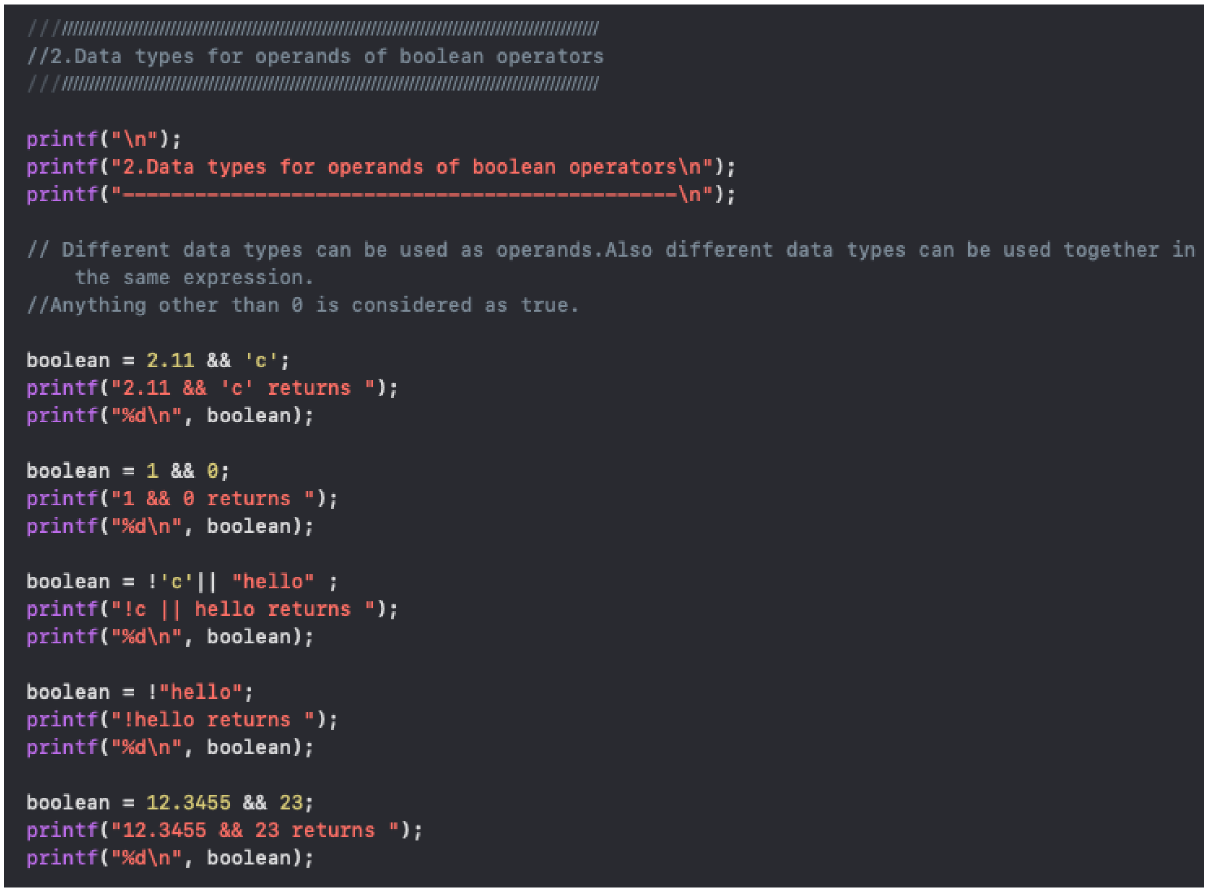
```
1.Boolean operators provided
----------------------------
(a < b) && (b == b) returns true
(b > a) || (a > b) returns true
!((b > a) || (a > b)) returns false
```

1.2 Data Types for Operands of Boolean Operators

In this part experiments with different data types are done and the results are observed for each language. Further explanation is done with comments inside code segments.

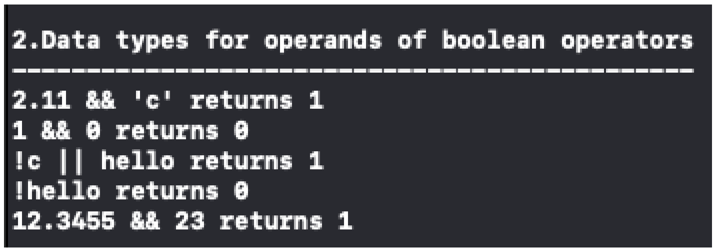
C

Code Segment



```
/////////////////////////////////////////////////////////////////////////////////////////////
//2.Data types for operands of boolean operators
/////////////////////////////////////////////////////////////////////////////////////////////

printf("\n");
printf("2.Data types for operands of boolean operators\n");
printf("----------------------------------------------\n");

// Different data types can be used as operands.Also different data types can be used together in
    the same expression.
//Anything other than 0 is considered as true.

boolean = 2.11 && 'c';
printf("2.11 && 'c' returns ");
printf("%d\n", boolean);

boolean = 1 && 0;
printf("1 && 0 returns ");
printf("%d\n", boolean);

boolean = !'c'|| "hello" ;
printf("!c || hello returns ");
printf("%d\n", boolean);

boolean = !"hello";
printf("!hello returns ");
printf("%d\n", boolean);

boolean = 12.3455 && 23;
printf("12.3455 && 23 returns ");
printf("%d\n", boolean);
```

Output



```
2.Data types for operands of boolean operators
----------------------------------------------
2.11 && 'c' returns 1
1 && 0 returns 0
!c || hello returns 1
!hello returns 0
12.3455 && 23 returns 1
```

GO

Code Segment

```
////////////////////////////////////////////////////////////////////////////////////////////////////////
//2.Data types for operands of boolean operators
////////////////////////////////////////////////////////////////////////////////////////////////////////

fmt.Println();
fmt.Println("2.Data types for operands of boolean operators");
fmt.Println("----------------------------------------------");

// Operands should be bool.

// Gives error because operands should be bool type
//boolean = 0 && 1;
//fmt.Print("0 && 1 returns ");
//fmt.Print(boolean,"\n");

boolean = true && false;
fmt.Print("true && false returns ");
fmt.Print(boolean,"\n");

boolean = (a > b) || (b > a);
fmt.Print("(a > b) || (b > a) returns ");
fmt.Print(boolean,"\n");
```

Output

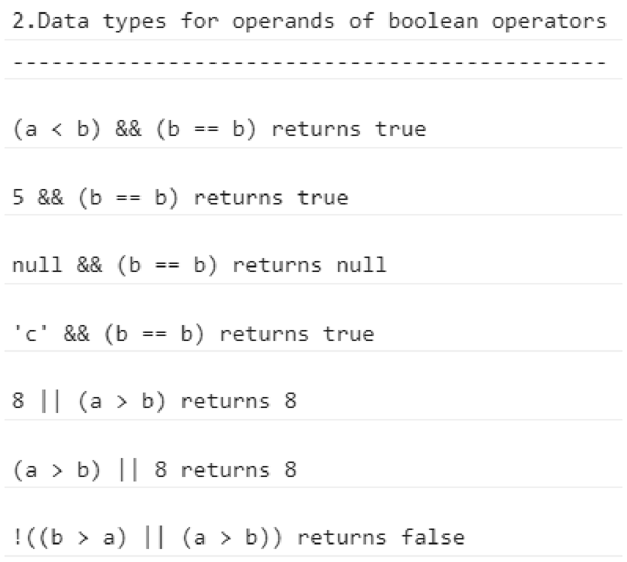
```
2.Data types for operands of boolean operators
----------------------------------------------
true && false returns false
(a > b) || (b > a) returns true
```

Javascript

Code Segment

```
console.log("2.Data types for operands of boolean operators")
console.log("----------------------------------------------")

//Different types can be used as operands but some of the results are not reasonable.
// For ! operator, if the operand is 0, null, "", NaN, undefined or false it is accepted as false and anything other accepted as true.


boolean = (a < b) && (b == b)
console.log("\n(a < b) && (b == b) returns "+ boolean + "\n")

boolean = 5 && (b == b)
console.log("\n5 && (b == b) returns "+ boolean + "\n")

boolean = null && (b == b)
console.log("\nnull && (b == b) returns "+ boolean + "\n")

boolean = 'c' && (b == b)
console.log("\n'c' && (b == b) returns "+ boolean + "\n")

boolean = 8 || (a > b)
console.log( "\n8 || (a > b) returns "+ boolean + "\n")

boolean = (a > b) || 8
console.log("\n(a > b) || 8 returns "+ boolean + "\n")

boolean = !((b > a) || (a > b))
console.log("\n!((b > a) || (a > b)) returns "+ boolean + "\n")
```

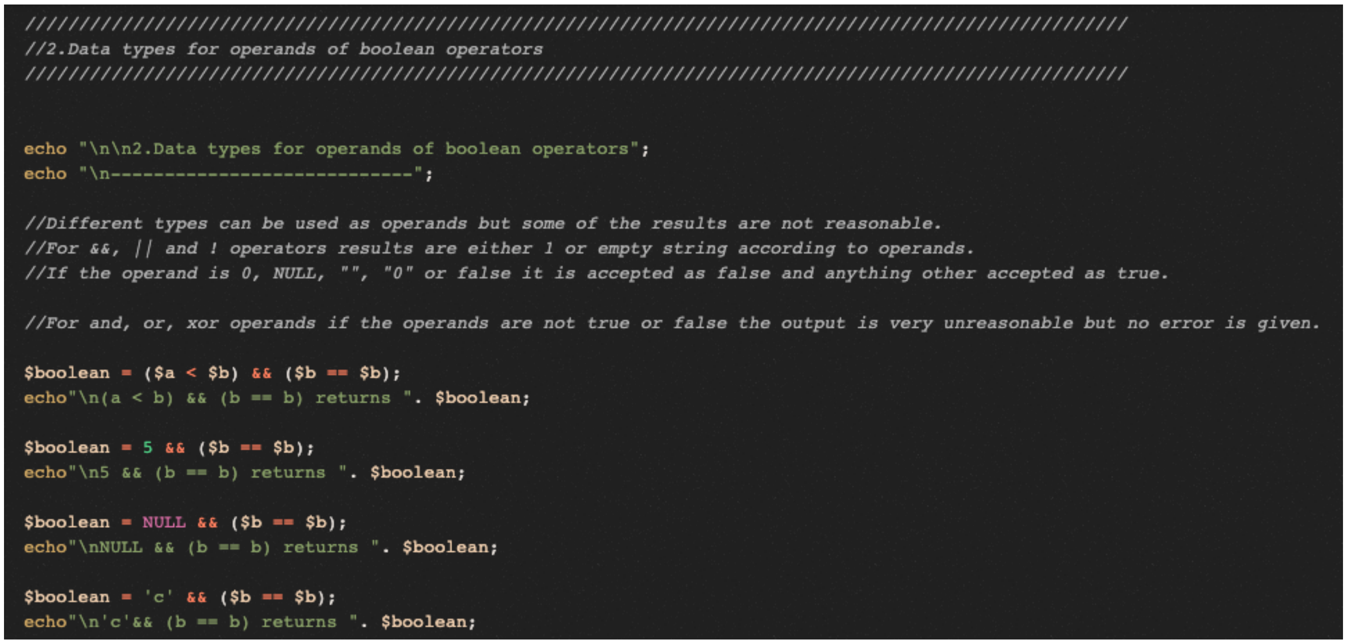
Output



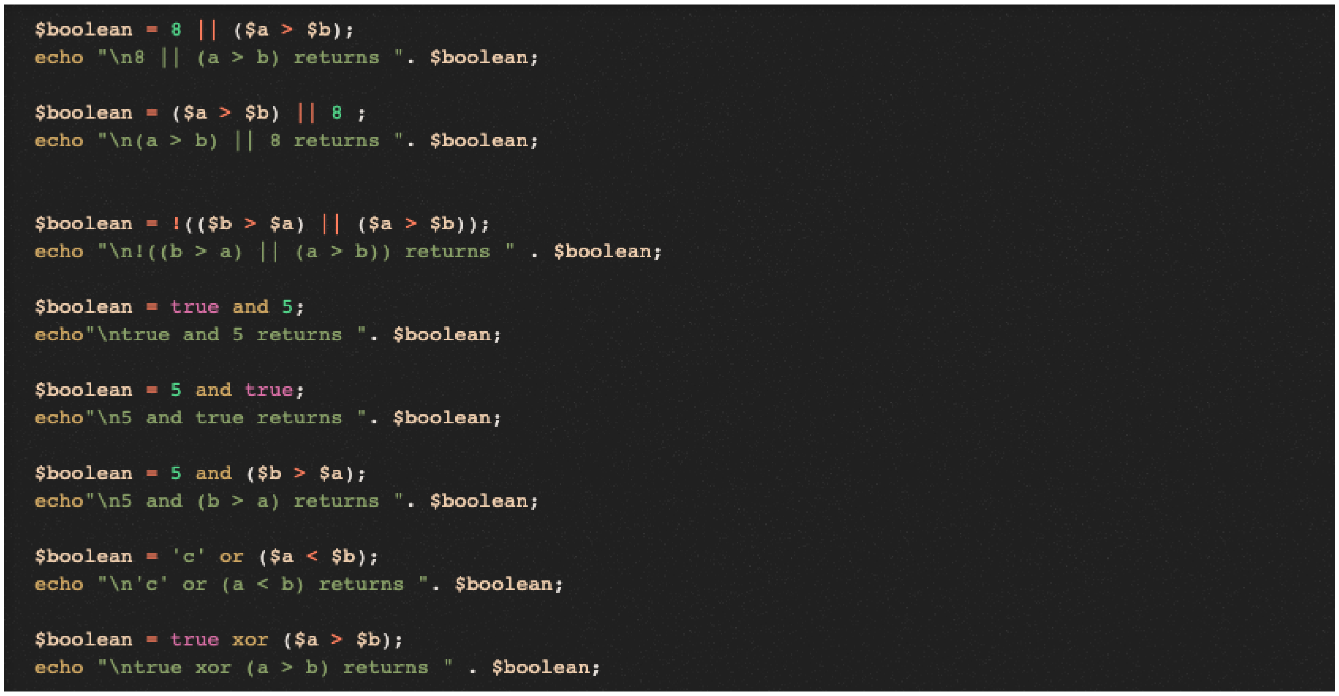
```
2.Data types for operands of boolean operators
----------------------------------------------

(a < b) && (b == b) returns true

5 && (b == b) returns true

null && (b == b) returns null

'c' && (b == b) returns true

8 || (a > b) returns 8

(a > b) || 8 returns 8

!((b > a) || (a > b)) returns false
```

PHP

Code Segment

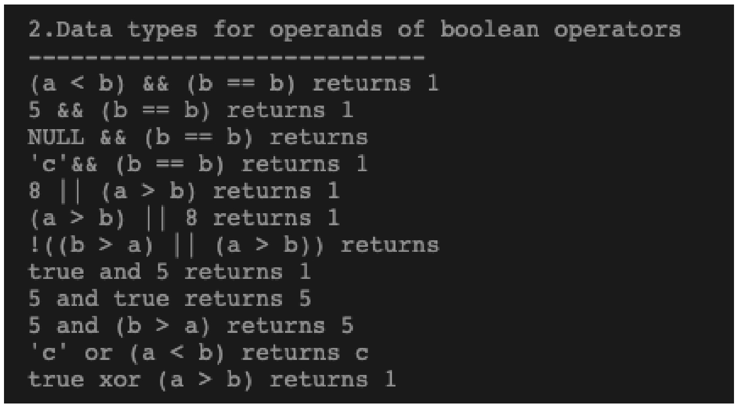


```php
//////////////////////////////////////////////////////////////////////////////////////////////////////
//2.Data types for operands of boolean operators
//////////////////////////////////////////////////////////////////////////////////////////////////////


echo "\n\n2.Data types for operands of boolean operators";
echo "\n----------------------------";

//Different types can be used as operands but some of the results are not reasonable.
//For &&, || and ! operators results are either 1 or empty string according to operands.
//If the operand is 0, NULL, "", "0" or false it is accepted as false and anything other accepted as true.

//For and, or, xor operands if the operands are not true or false the output is very unreasonable but no error is given.

$boolean = ($a < $b) && ($b == $b);
echo"\n(a < b) && (b == b) returns ". $boolean;

$boolean = 5 && ($b == $b);
echo"\n5 && (b == b) returns ". $boolean;

$boolean = NULL && ($b == $b);
echo"\nNULL && (b == b) returns ". $boolean;

$boolean = 'c' && ($b == $b);
echo"\n'c'&& (b == b) returns ". $boolean;
```



```php
$boolean = 8 || ($a > $b);
echo "\n8 || (a > b) returns ". $boolean;

$boolean = ($a > $b) || 8 ;
echo "\n(a > b) || 8 returns ". $boolean;


$boolean = !(($b > $a) || ($a > $b));
echo "\n!((b > a) || (a > b)) returns " . $boolean;

$boolean = true and 5;
echo"\ntrue and 5 returns ". $boolean;

$boolean = 5 and true;
echo"\n5 and true returns ". $boolean;

$boolean = 5 and ($b > $a);
echo"\n5 and (b > a) returns ". $boolean;

$boolean = 'c' or ($a < $b);
echo "\n'c' or (a < b) returns ". $boolean;

$boolean = true xor ($a > $b);
echo "\ntrue xor (a > b) returns " . $boolean;
```

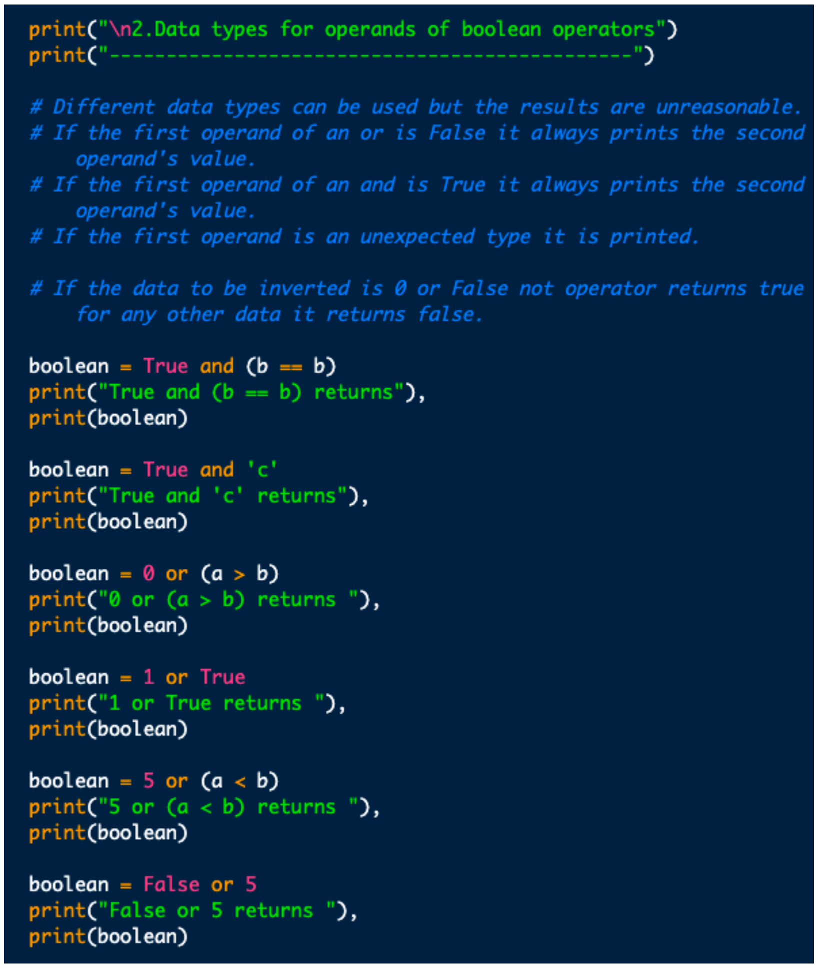
Output



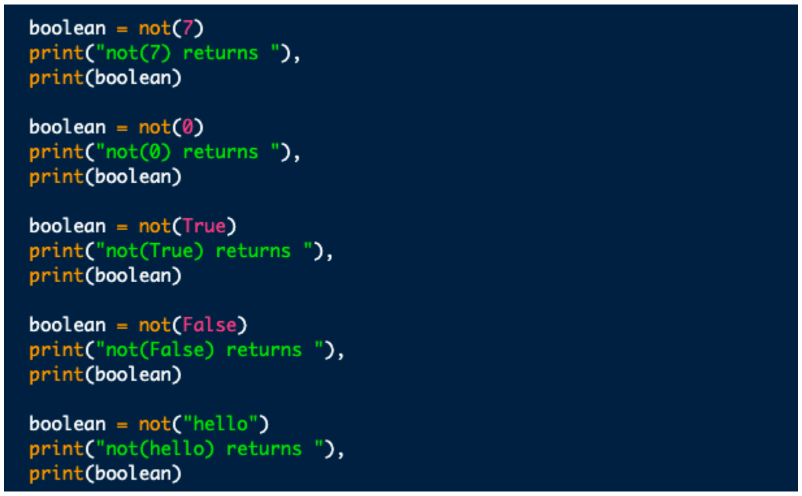
```
2.Data types for operands of boolean operators
----------------------------
(a < b) && (b == b) returns 1
5 && (b == b) returns 1
NULL && (b == b) returns
'c'&& (b == b) returns 1
8 || (a > b) returns 1
(a > b) || 8 returns 1
!((b > a) || (a > b)) returns
true and 5 returns 1
5 and true returns 5
5 and (b > a) returns 5
'c' or (a < b) returns c
true xor (a > b) returns 1
```

Python

Code Segment



```python
print("\n2.Data types for operands of boolean operators")
print("---------------------------------------------")

# Different data types can be used but the results are unreasonable.
# If the first operand of an or is False it always prints the second
    operand's value.
# If the first operand of an and is True it always prints the second
    operand's value.
# If the first operand is an unexpected type it is printed.

# If the data to be inverted is 0 or False not operator returns true
    for any other data it returns false.

boolean = True and (b == b)
print("True and (b == b) returns"),
print(boolean)

boolean = True and 'c'
print("True and 'c' returns"),
print(boolean)

boolean = 0 or (a > b)
print("0 or (a > b) returns "),
print(boolean)

boolean = 1 or True
print("1 or True returns "),
print(boolean)

boolean = 5 or (a < b)
print("5 or (a < b) returns "),
print(boolean)

boolean = False or 5
print("False or 5 returns "),
print(boolean)
```



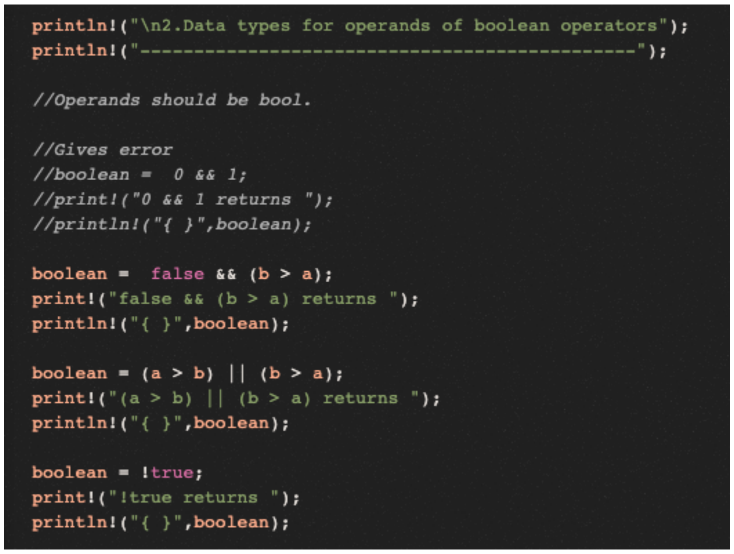
```
boolean = not(7)
print("not(7) returns "),
print(boolean)

boolean = not(0)
print("not(0) returns "),
print(boolean)

boolean = not(True)
print("not(True) returns "),
print(boolean)

boolean = not(False)
print("not(False) returns "),
print(boolean)

boolean = not("hello")
print("not(hello) returns "),
print(boolean)
```

Output

```
2.Data types for operands of boolean operators
----------------------------------------------
True and (b == b) returns True
True and 'c' returns c
0 or (a > b) returns  False
1 or True returns  1
5 or (a < b) returns  5
False or 5 returns  5
not(7) returns  False
not(0) returns  True
not(True) returns  False
not(False) returns  True
not(hello) returns  False
```

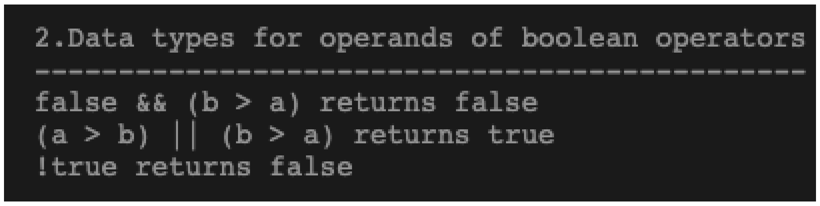
Rust

Code Segment



```
println!("\n2.Data types for operands of boolean operators");
println!("----------------------------------------------");

//Operands should be bool.

//Gives error
//boolean =  0 && 1;
//print!("0 && 1 returns ");
//println!("{ }",boolean);

boolean =  false && (b > a);
print!("false && (b > a) returns ");
println!("{ }",boolean);

boolean = (a > b) || (b > a);
print!("(a > b) || (b > a) returns ");
println!("{ }",boolean);

boolean = !true;
print!("!true returns ");
println!("{ }",boolean);
```

Output



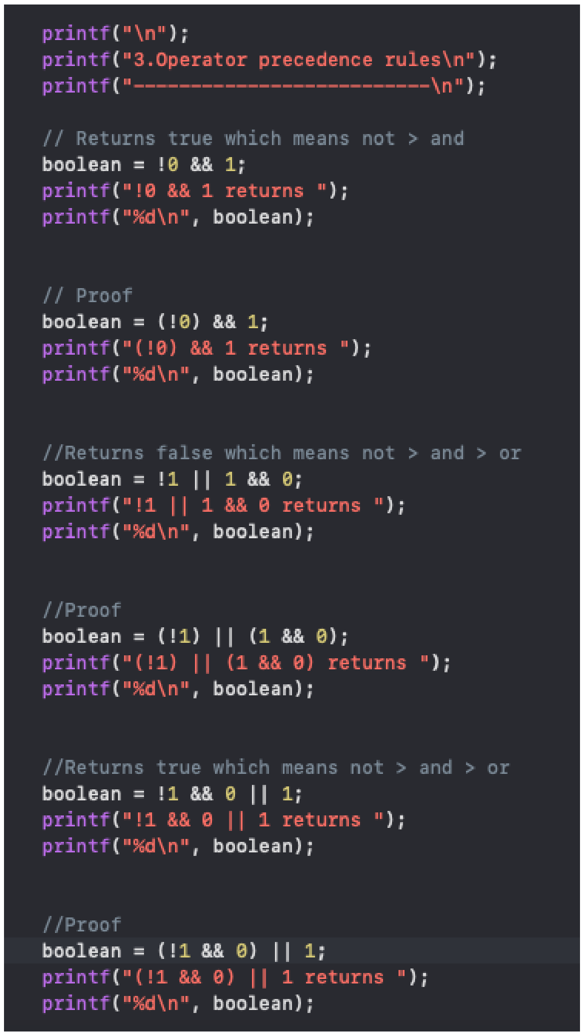
```
2.Data types for operands of boolean operators
----------------------------------------------
false && (b > a) returns false
(a > b) || (b > a) returns true
!true returns false
```

1.3 Operator Precedence Rules

In this part operators with different precedences compared to find the precedence levels of logical operators. Further explanation is done with comments inside code segments.

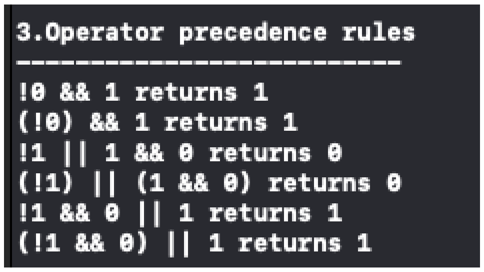
C

Code Segment



```c
printf("\n");
printf("3.Operator precedence rules\n");
printf("--------------------------\n");

// Returns true which means not > and
boolean = !0 && 1;
printf("!0 && 1 returns ");
printf("%d\n", boolean);


// Proof
boolean = (!0) && 1;
printf("(!0) && 1 returns ");
printf("%d\n", boolean);


//Returns false which means not > and > or
boolean = !1 || 1 && 0;
printf("!1 || 1 && 0 returns ");
printf("%d\n", boolean);


//Proof
boolean = (!1) || (1 && 0);
printf("(!1) || (1 && 0) returns ");
printf("%d\n", boolean);


//Returns true which means not > and > or
boolean = !1 && 0 || 1;
printf("!1 && 0 || 1 returns ");
printf("%d\n", boolean);


//Proof
boolean = (!1 && 0) || 1;
printf("(!1 && 0) || 1 returns ");
printf("%d\n", boolean);
```

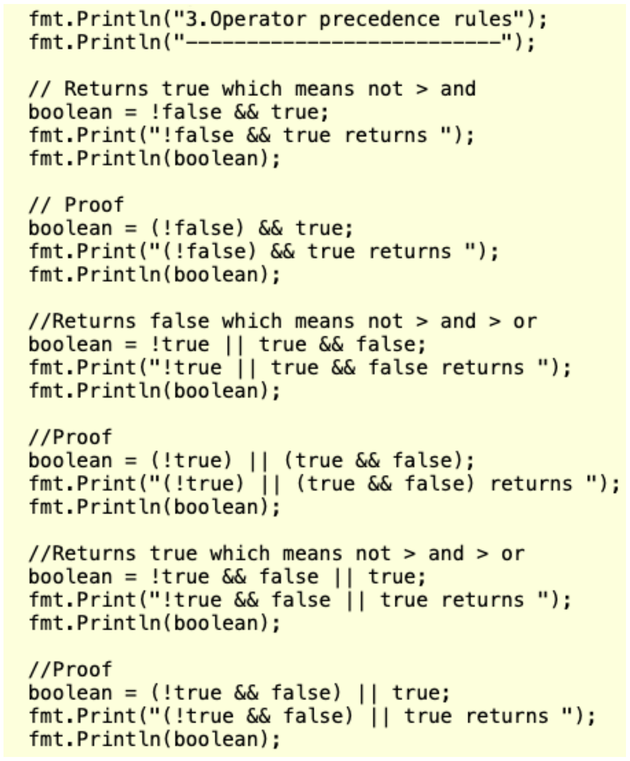
Output



```
3.Operator precedence rules
--------------------------
!0 && 1 returns 1
(!0) && 1 returns 1
!1 || 1 && 0 returns 0
(!1) || (1 && 0) returns 0
!1 && 0 || 1 returns 1
(!1 && 0) || 1 returns 1
```

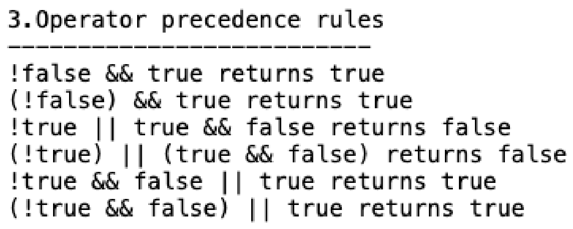
GO

Code Segment



```
fmt.Println("3.Operator precedence rules");
fmt.Println("--------------------------");

// Returns true which means not > and
boolean = !false && true;
fmt.Print("!false && true returns ");
fmt.Println(boolean);

// Proof
boolean = (!false) && true;
fmt.Print("(!false) && true returns ");
fmt.Println(boolean);

//Returns false which means not > and > or
boolean = !true || true && false;
fmt.Print("!true || true && false returns ");
fmt.Println(boolean);

//Proof
boolean = (!true) || (true && false);
fmt.Print("(!true) || (true && false) returns ");
fmt.Println(boolean);

//Returns true which means not > and > or
boolean = !true && false || true;
fmt.Print("!true && false || true returns ");
fmt.Println(boolean);

//Proof
boolean = (!true && false) || true;
fmt.Print("(!true && false) || true returns ");
fmt.Println(boolean);
```

Output



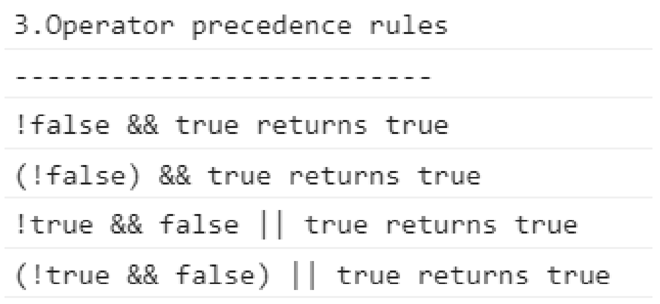
```
3.Operator precedence rules
--------------------------
!false && true returns true
(!false) && true returns true
!true || true && false returns false
(!true) || (true && false) returns false
!true && false || true returns true
(!true && false) || true returns true
```

Javascript

Code Segment

```
console.log("\n3.Operator precedence rules")
console.log("--------------------------")

// not > and
boolean =  !false && true
console.log("!false && true returns "+ boolean + "\n")

//Proof
boolean =  (!false) && true
console.log("(!false) && true returns "+ boolean + "\n")

// not > and > or
boolean =  !true && false || true
console.log("!true && false || true returns "+ boolean + "\n")

//Proof
boolean =  (!true && false) || true
console.log("(!true && false) || true returns "+ boolean + "\n")
```

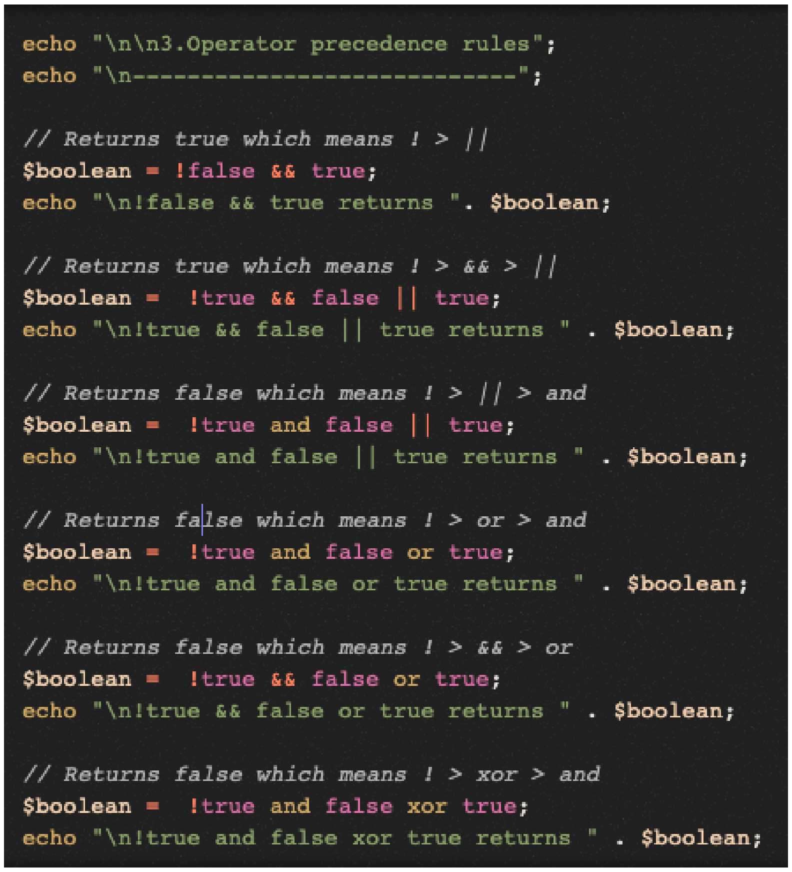
Output



```
3.Operator precedence rules
--------------------------
!false && true returns true
(!false) && true returns true
!true && false || true returns true
(!true && false) || true returns true
```

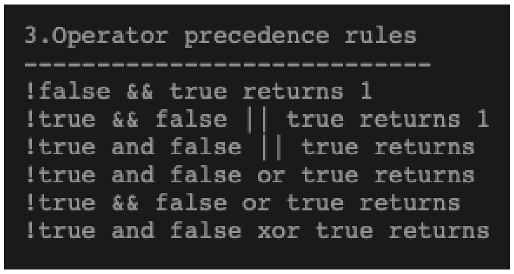
PHP

Code Segment



```
echo "\n\n3.Operator precedence rules";
echo "\n----------------------------";

// Returns true which means ! > ||
$boolean = !false && true;
echo "\n!false && true returns ". $boolean;

// Returns true which means ! > && > ||
$boolean =  !true && false || true;
echo "\n!true && false || true returns " . $boolean;

// Returns false which means ! > || > and
$boolean =  !true and false || true;
echo "\n!true and false || true returns " . $boolean;

// Returns false which means ! > or > and
$boolean =  !true and false or true;
echo "\n!true and false or true returns " . $boolean;

// Returns false which means ! > && > or
$boolean =  !true && false or true;
echo "\n!true && false or true returns " . $boolean;

// Returns false which means ! > xor > and
$boolean =  !true and false xor true;
echo "\n!true and false xor true returns " . $boolean;
```

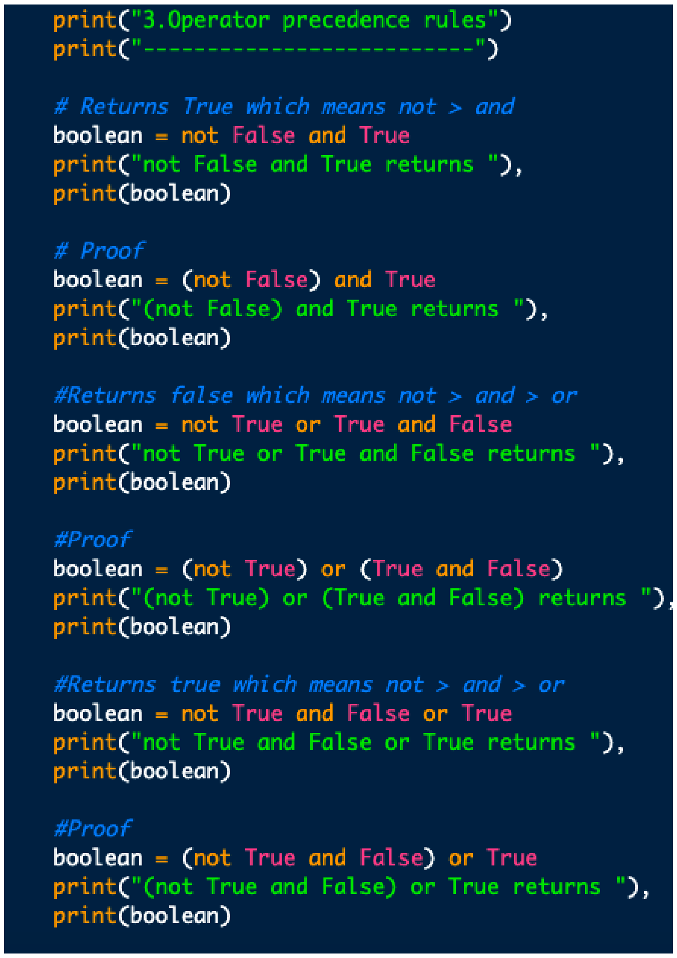
Output



```
3.Operator precedence rules
----------------------------
!false && true returns 1
!true && false || true returns 1
!true and false || true returns
!true and false or true returns
!true && false or true returns
!true and false xor true returns
```

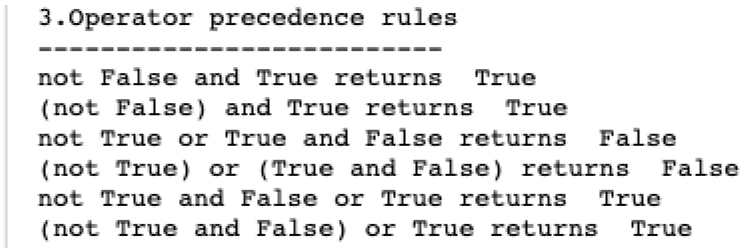
Python

Code Segment



```
print("3.Operator precedence rules")
print("------------------------")

# Returns True which means not > and
boolean = not False and True
print("not False and True returns "),
print(boolean)

# Proof
boolean = (not False) and True
print("(not False) and True returns "),
print(boolean)

#Returns false which means not > and > or
boolean = not True or True and False
print("not True or True and False returns "),
print(boolean)

#Proof
boolean = (not True) or (True and False)
print("(not True) or (True and False) returns "),
print(boolean)

#Returns true which means not > and > or
boolean = not True and False or True
print("not True and False or True returns "),
print(boolean)

#Proof
boolean = (not True and False) or True
print("(not True and False) or True returns "),
print(boolean)
```

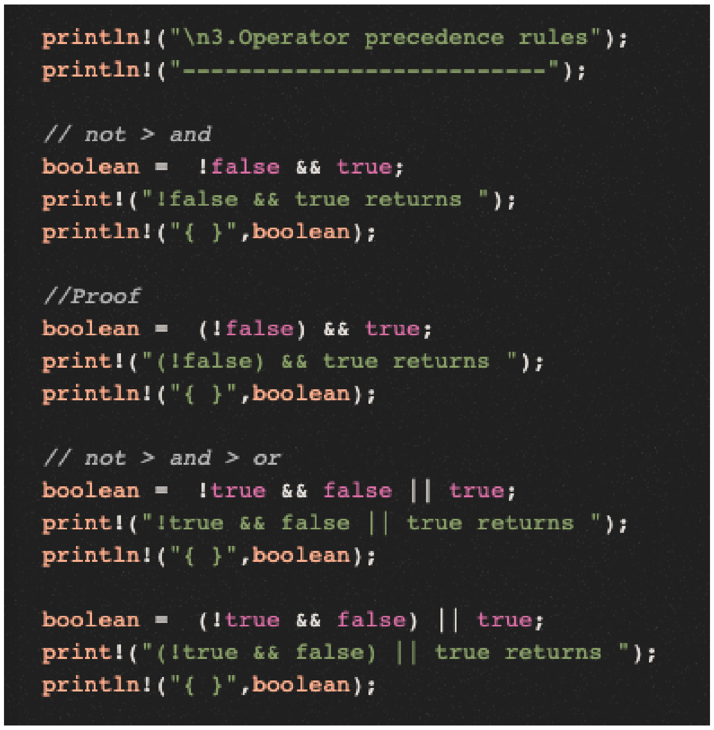
Output



```
3.Operator precedence rules
--------------------------
not False and True returns  True
(not False) and True returns  True
not True or True and False returns  False
(not True) or (True and False) returns  False
not True and False or True returns  True
(not True and False) or True returns  True
```

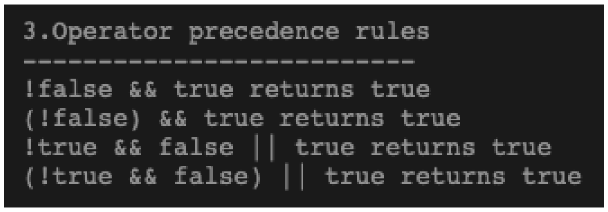
Rust

Code Segment



```
println!("\n3.Operator precedence rules");
println!("--------------------------");

// not > and
boolean =  !false && true;
print!("!false && true returns ");
println!("{ }",boolean);

//Proof
boolean =  (!false) && true;
print!("(!false) && true returns ");
println!("{ }",boolean);

// not > and > or
boolean =  !true && false || true;
print!("!true && false || true returns ");
println!("{ }",boolean);

boolean =  (!true && false) || true;
print!("(!true && false) || true returns ");
println!("{ }",boolean);
```

Output



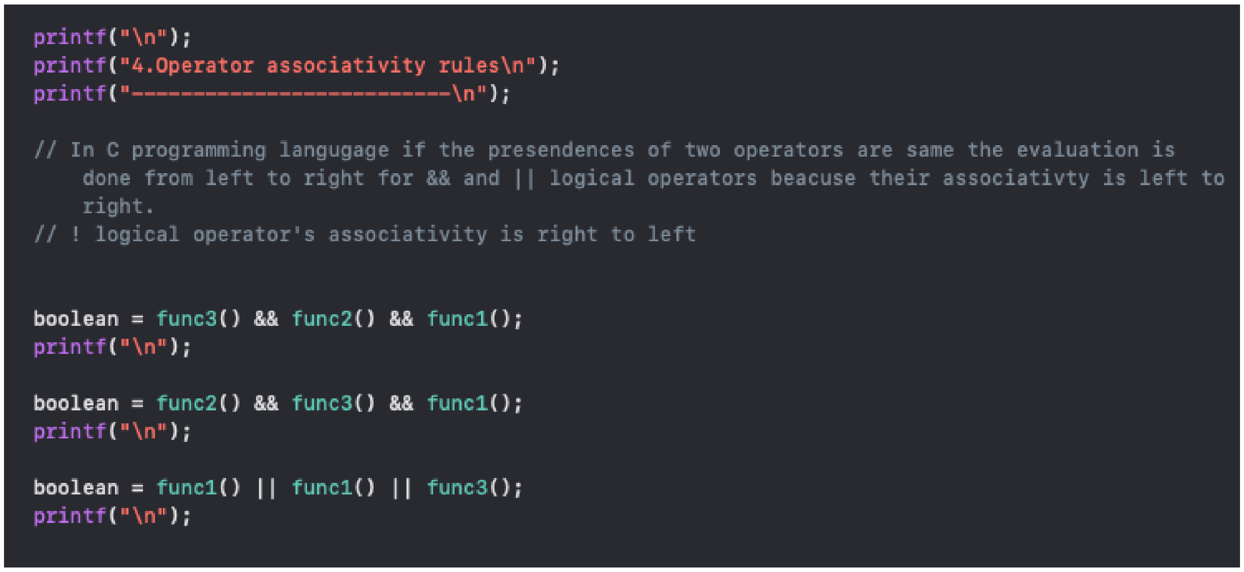
```
3.Operator precedence rules
--------------------------
!false && true returns true
(!false) && true returns true
!true && false || true returns true
(!true && false) || true returns true
```

1.4 Operator Associativity Rules

In this part, to be able to show the associativity of the logical operators, operators with the same precedence level are used in a single expression. Also, to be able to analyze the associativity a function call is used which prints function name and returns true or false depending on the function. Further explanation is done with comments inside code segments.

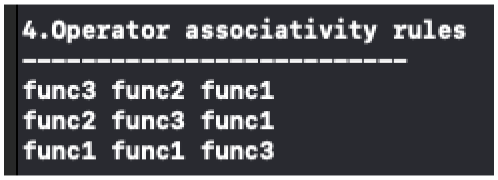
C

Code Segment



```c
printf("\n");
printf("4.Operator associativity rules\n");
printf("---------------------------\n");

// In C programming langugage if the presendences of two operators are same the evaluation is
    done from left to right for && and || logical operators beacuse their associativty is left to
    right.
// ! logical operator's associativity is right to left


boolean = func3() && func2() && func1();
printf("\n");

boolean = func2() && func3() && func1();
printf("\n");

boolean = func1() || func1() || func3();
printf("\n");
```

Output



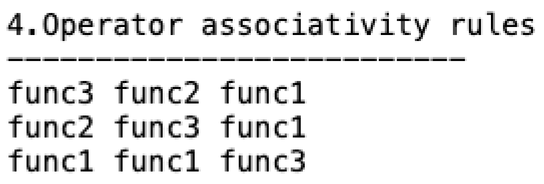
```
4.Operator associativity rules
--------------------------
func3 func2 func1
func2 func3 func1
func1 func1 func3
```

GO

Code Segment

```go
//////////////////////////////////////////////////////////////////////////////////////////////////////
//4.Operator associativity rules
//////////////////////////////////////////////////////////////////////////////////////////////////////

fmt.Println();
fmt.Println("4.Operator associativity rules");
fmt.Println("--------------------------");

// In GO programming langugage if the presendences of two operators are same the evaluation is
// done from left to right for && and || logical operators beacuse their associativty is left to right.
// ! logical operator's associativity is right to left.


boolean = func3() && func2() && func1();
fmt.Print("\n");

boolean = func2() && func3() && func1();
fmt.Print("\n");

boolean = func1() || func1() || func3();
fmt.Print("\n");
```

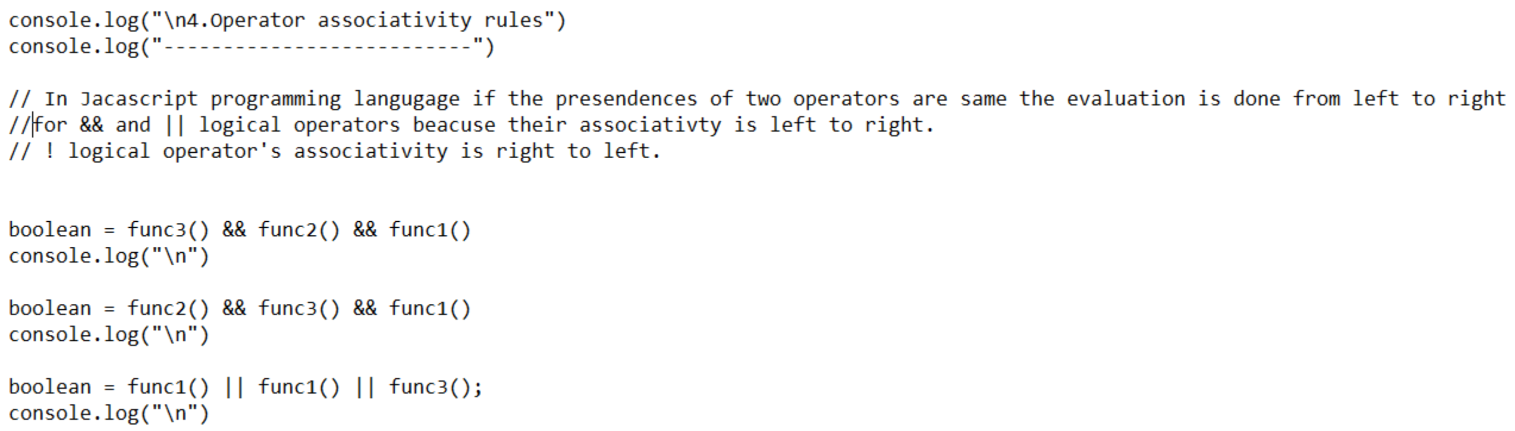
Output



```
4.Operator associativity rules
--------------------------
func3 func2 func1
func2 func3 func1
func1 func1 func3
```

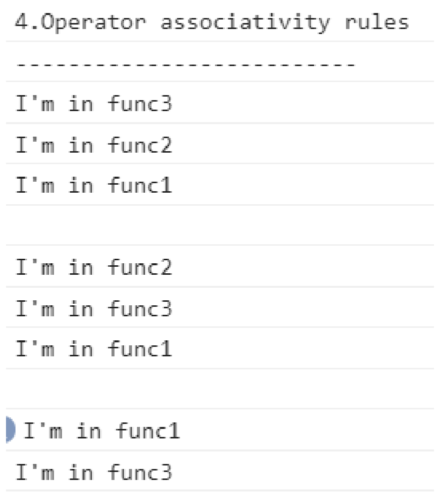
Javascript

Code Segment



```
console.log("\n4.Operator associativity rules")
console.log("-------------------------")

// In Jacascript programming langugage if the presendences of two operators are same the evaluation is done from left to right
//for && and || logical operators beacuse their associativty is left to right.
// ! logical operator's associativity is right to left.


boolean = func3() && func2() && func1()
console.log("\n")

boolean = func2() && func3() && func1()
console.log("\n")

boolean = func1() || func1() || func3();
console.log("\n")
```

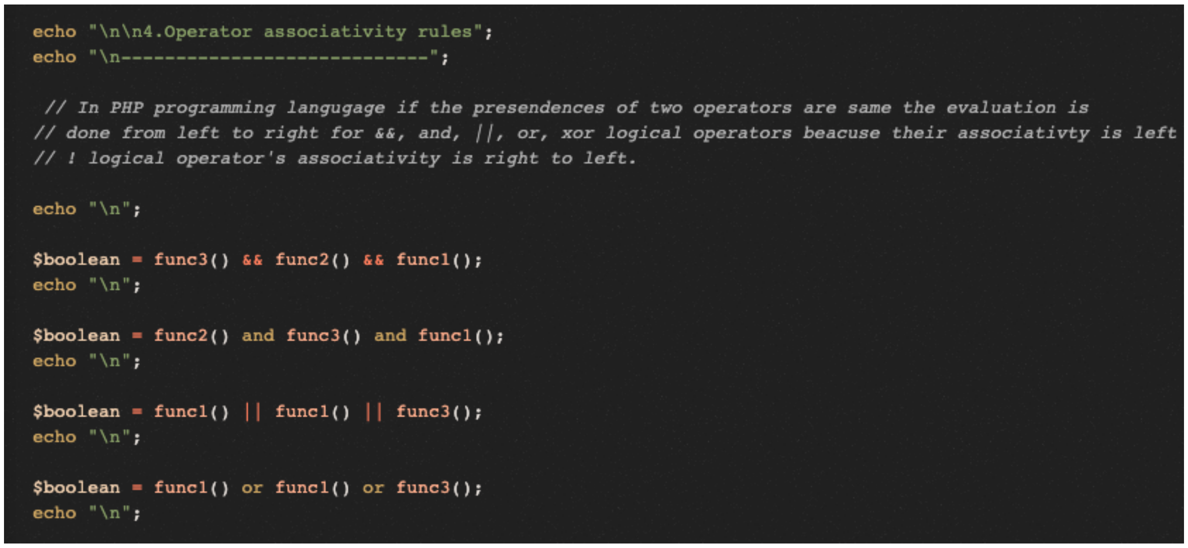
Output



```
4.Operator associativity rules
--------------------------
I'm in func3
I'm in func2
I'm in func1

I'm in func2
I'm in func3
I'm in func1

I'm in func1
I'm in func3
```

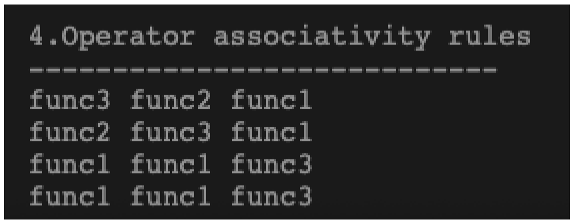
PHP

Code Segment



```
echo "\n\n4.Operator associativity rules";
echo "\n---------------------------";

 // In PHP programming langugage if the presendences of two operators are same the evaluation is
// done from left to right for &&, and, ||, or, xor logical operators beacuse their associativty is left
// ! logical operator's associativity is right to left.

echo "\n";

$boolean = func3() && func2() && func1();
echo "\n";

$boolean = func2() and func3() and func1();
echo "\n";

$boolean = func1() || func1() || func3();
echo "\n";

$boolean = func1() or func1() or func3();
echo "\n";
```

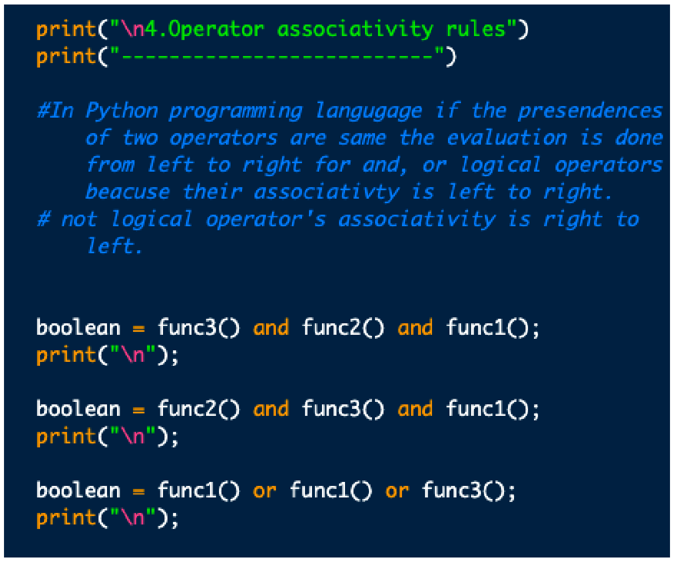
Output



```
4.Operator associativity rules
---------------------------
func3 func2 func1
func2 func3 func1
func1 func1 func3
func1 func1 func3
```

Python

Code Segment



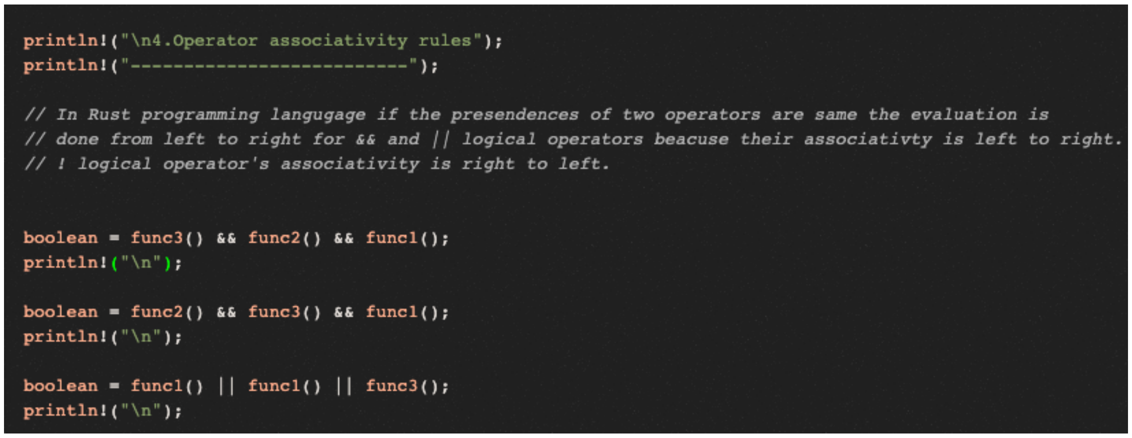
```
print("\n4.Operator associativity rules")
print("-------------------------")

#In Python programming langugage if the presendences
    of two operators are same the evaluation is done
    from left to right for and, or logical operators
    beacuse their associativty is left to right.
# not logical operator's associativity is right to
    left.


boolean = func3() and func2() and func1();
print("\n");

boolean = func2() and func3() and func1();
print("\n");

boolean = func1() or func1() or func3();
print("\n");
```

Output

```
4.Operator associativity rules
--------------------------
I'm in func3
I'm in func2
I'm in func1


I'm in func2
I'm in func3
I'm in func1


I'm in func1
I'm in func1
I'm in func3
```

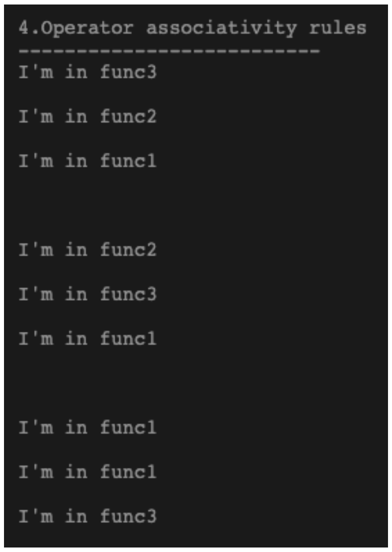
Rust

Code Segment



```
println!("\n4.Operator associativity rules");
println!("--------------------------");

// In Rust programming langugage if the presendences of two operators are same the evaluation is
// done from left to right for && and || logical operators beacuse their associativty is left to right.
// ! logical operator's associativity is right to left.


boolean = func3() && func2() && func1();
println!("\n");

boolean = func2() && func3() && func1();
println!("\n");

boolean = func1() || func1() || func3();
println!("\n");
```

Output



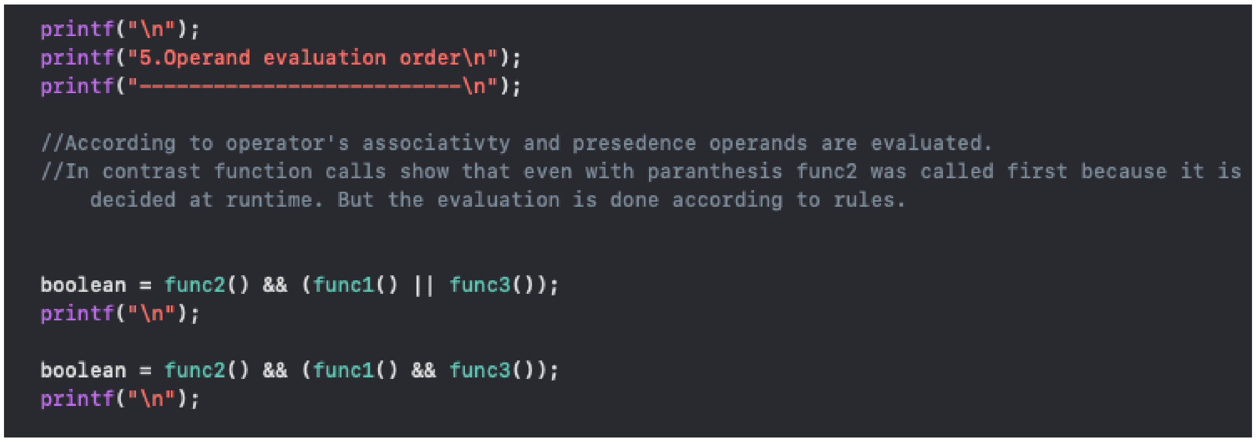
```
4.Operator associativity rules
--------------------------
I'm in func3

I'm in func2

I'm in func1



I'm in func2

I'm in func3

I'm in func1



I'm in func1

I'm in func1

I'm in func3
```

1.5 Operand Evaluation Order

In this part to be able to understand operand evaluation order, function calls are used. Further explanation is done with comments inside code segments.

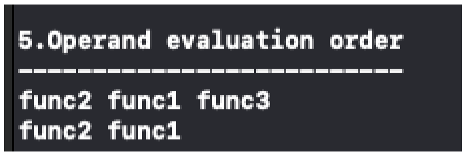
C

Code Segment



```c
printf("\n");
printf("5.Operand evaluation order\n");
printf("--------------------------\n");

//According to operator's associativty and presedence operands are evaluated.
//In contrast function calls show that even with paranthesis func2 was called first because it is
    decided at runtime. But the evaluation is done according to rules.


boolean = func2() && (func1() || func3());
printf("\n");

boolean = func2() && (func1() && func3());
printf("\n");
```

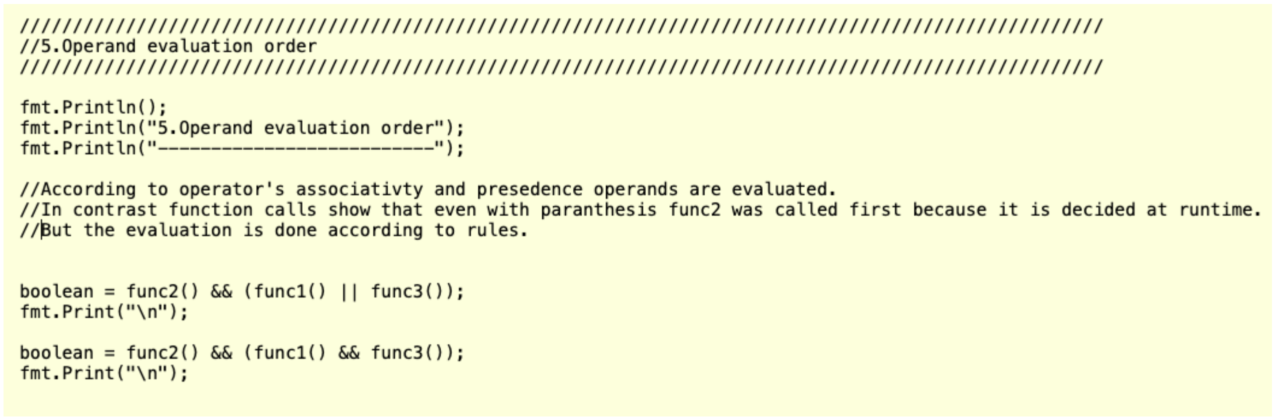
Output



```
5.Operand evaluation order
--------------------------
func2 func1 func3
func2 func1
```

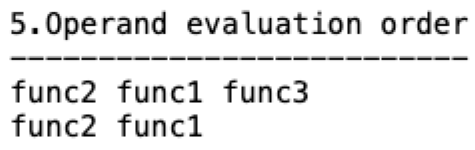
GO

Code Segment



```go
////////////////////////////////////////////////////////////////////////////////////////////////////////
//5.Operand evaluation order
////////////////////////////////////////////////////////////////////////////////////////////////////////

fmt.Println();
fmt.Println("5.Operand evaluation order");
fmt.Println("--------------------------");

//According to operator's associativty and presedence operands are evaluated.
//In contrast function calls show that even with paranthesis func2 was called first because it is decided at runtime.
//But the evaluation is done according to rules.


boolean = func2() && (func1() || func3());
fmt.Print("\n");

boolean = func2() && (func1() && func3());
fmt.Print("\n");
```

Output



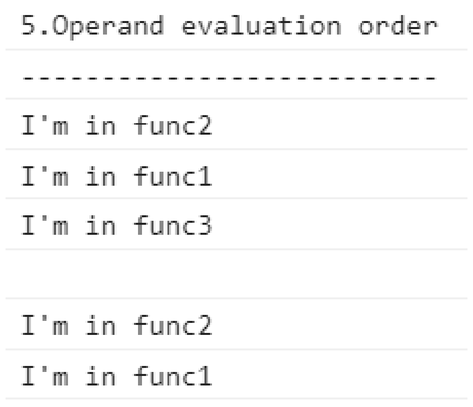
```
5.Operand evaluation order
--------------------------
func2 func1 func3
func2 func1
```

Javascript

Code Segment

```
console.log("\n5.Operand evaluation order")
console.log("--------------------------")

//According to the operator's associativity and precedence, operands are evaluated.
//In contrast function calls show that even with parentheses func2 was called first because it is decided at runtime.
//But the evaluation is done according to rules.


boolean = func2() && (func1() || func3());
console.log("\n")

boolean = func2() && (func1() && func3());
console.log("\n")
```

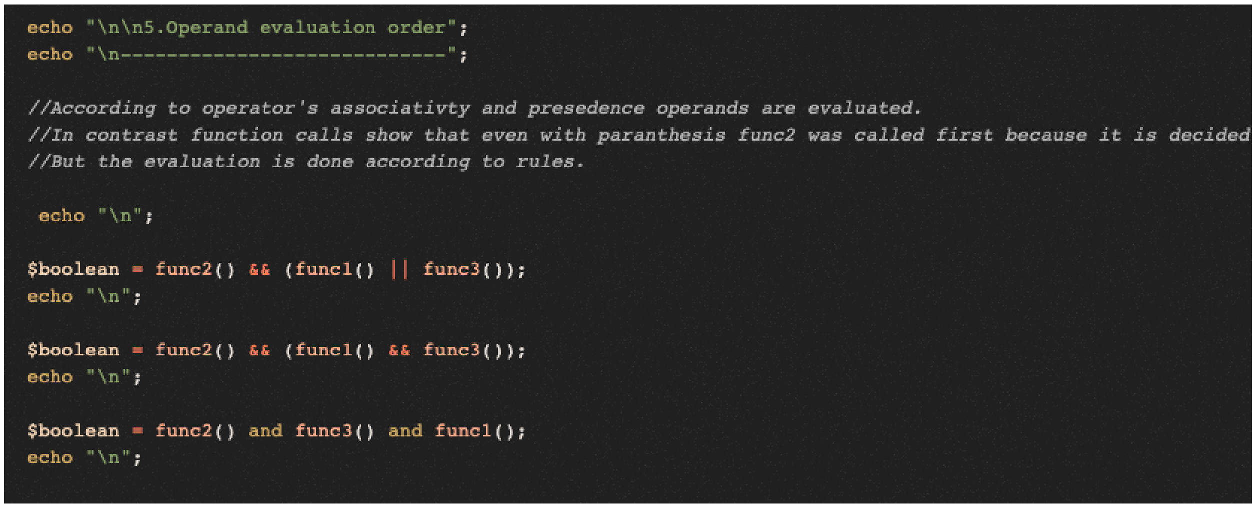
Output



```
5.Operand evaluation order
--------------------------
I'm in func2
I'm in func1
I'm in func3

I'm in func2
I'm in func1
```

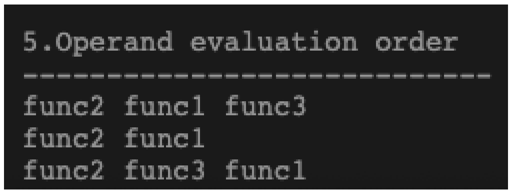
PHP

Code Segment



```
echo "\n\n5.Operand evaluation order";
echo "\n----------------------------";

//According to operator's associativty and presedence operands are evaluated.
//In contrast function calls show that even with paranthesis func2 was called first because it is decided
//But the evaluation is done according to rules.

 echo "\n";

$boolean = func2() && (func1() || func3());
echo "\n";

$boolean = func2() && (func1() && func3());
echo "\n";

$boolean = func2() and func3() and func1();
echo "\n";
```

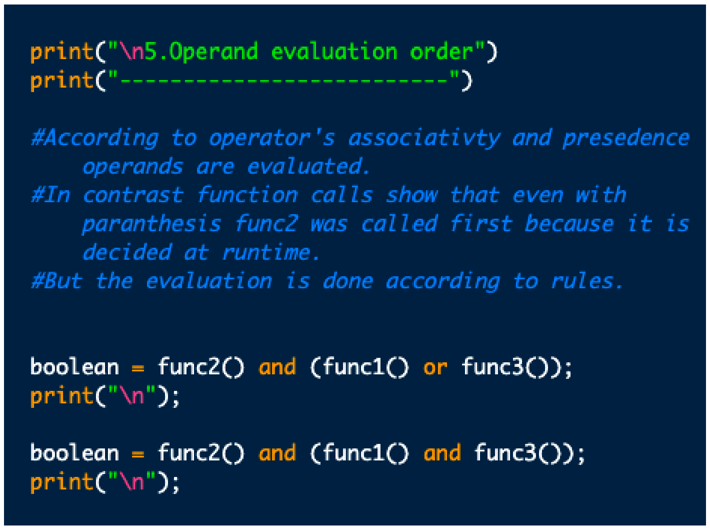
Output



```
5.Operand evaluation order
----------------------------
func2 func1 func3
func2 func1
func2 func3 func1
```

Python

Code Segment



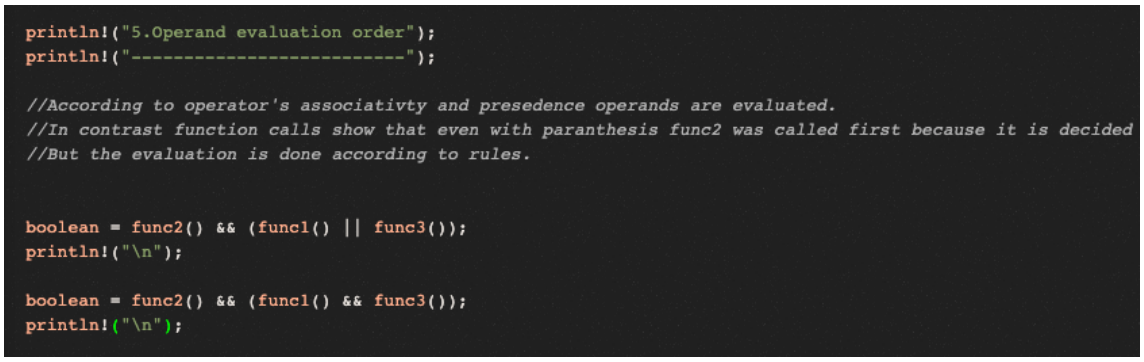
```
print("\n5.Operand evaluation order")
print("--------------------------")

#According to operator's associativty and presedence
    operands are evaluated.
#In contrast function calls show that even with
    paranthesis func2 was called first because it is
    decided at runtime.
#But the evaluation is done according to rules.


boolean = func2() and (func1() or func3());
print("\n");

boolean = func2() and (func1() and func3());
print("\n");
```

Output

```
5.Operand evaluation order
--------------------------
I'm in func2
I'm in func1
I'm in func3


I'm in func2
I'm in func1
```

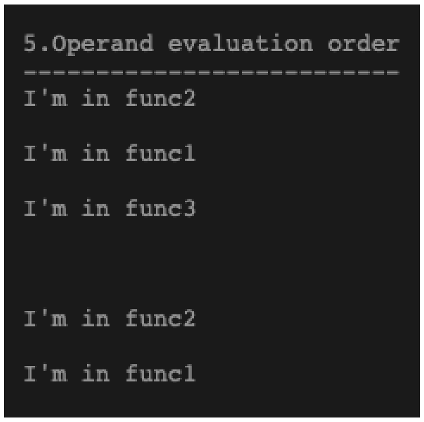
Rust

Code Segment



```
println!("5.Operand evaluation order");
println!("-------------------------");

//According to operator's associativty and presedence operands are evaluated.
//In contrast function calls show that even with paranthesis func2 was called first because it is decided
//But the evaluation is done according to rules.


boolean = func2() && (func1() || func3());
println!("\n");

boolean = func2() && (func1() && func3());
println!("\n");
```

Output



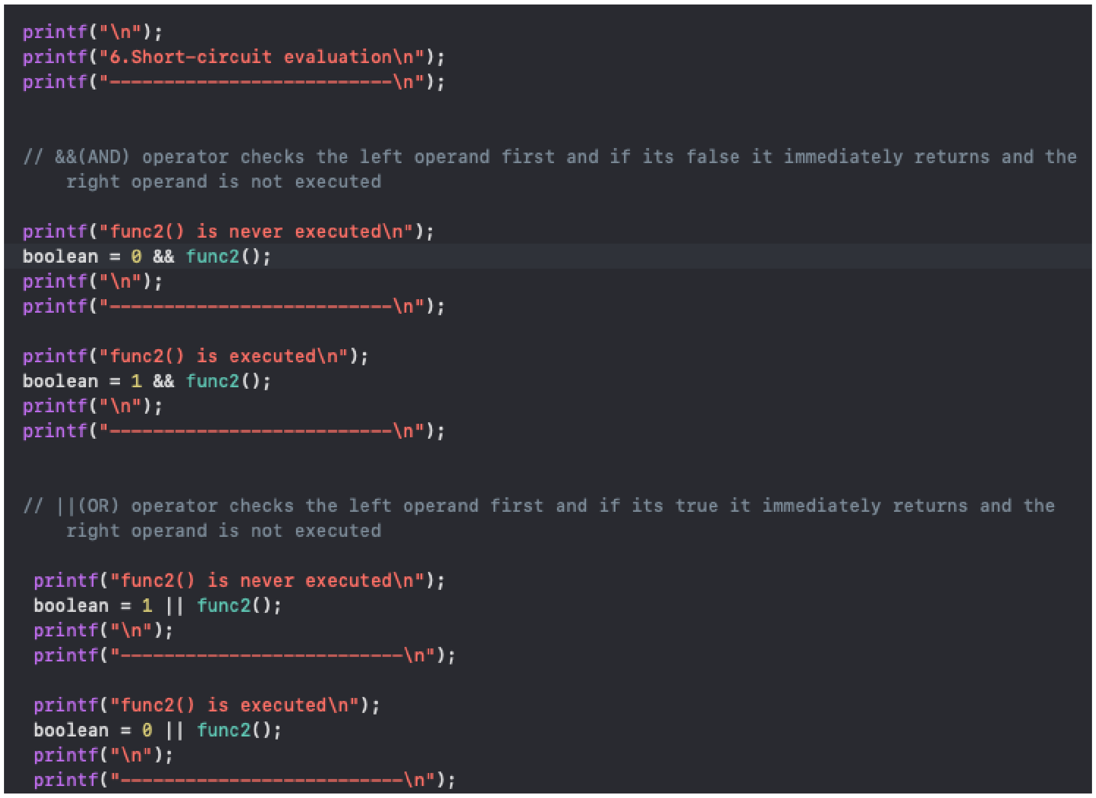
```
5.Operand evaluation order
-------------------------
I'm in func2

I'm in func1

I'm in func3


I'm in func2

I'm in func1
```

1.6 Short Circuit Evaluation

In this part to be able to understand short circuit evaluation function calls are used. Further explanation is done with comments inside code segments.

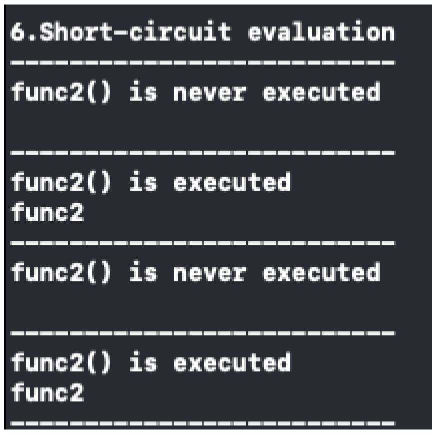
C

Code Segment



```
printf("\n");
printf("6.Short-circuit evaluation\n");
printf("--------------------------\n");


// &&(AND) operator checks the left operand first and if its false it immediately returns and the
    right operand is not executed

printf("func2() is never executed\n");
boolean = 0 && func2();
printf("\n");
printf("--------------------------\n");

printf("func2() is executed\n");
boolean = 1 && func2();
printf("\n");
printf("--------------------------\n");


// ||(OR) operator checks the left operand first and if its true it immediately returns and the
    right operand is not executed

 printf("func2() is never executed\n");
 boolean = 1 || func2();
 printf("\n");
 printf("--------------------------\n");

 printf("func2() is executed\n");
 boolean = 0 || func2();
 printf("\n");
 printf("--------------------------\n");
```

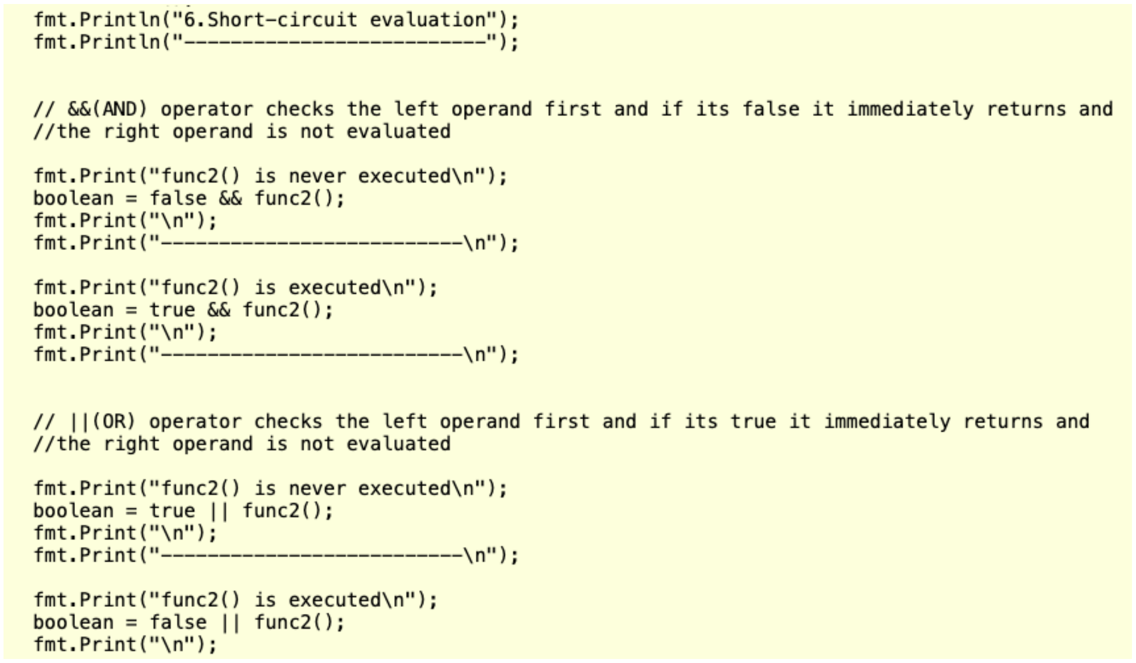
Output



```
6.Short-circuit evaluation
--------------------------
func2() is never executed

--------------------------
func2() is executed
func2
--------------------------
func2() is never executed

--------------------------
func2() is executed
func2
--------------------------
```

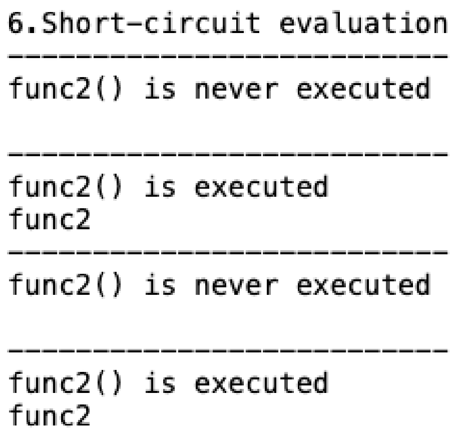
GO

Code Segment



```
fmt.Println("6.Short-circuit evaluation");
fmt.Println("-------------------------");


// &&(AND) operator checks the left operand first and if its false it immediately returns and
//the right operand is not evaluated

fmt.Print("func2() is never executed\n");
boolean = false && func2();
fmt.Print("\n");
fmt.Print("-------------------------\n");

fmt.Print("func2() is executed\n");
boolean = true && func2();
fmt.Print("\n");
fmt.Print("-------------------------\n");


// ||(OR) operator checks the left operand first and if its true it immediately returns and
//the right operand is not evaluated

fmt.Print("func2() is never executed\n");
boolean = true || func2();
fmt.Print("\n");
fmt.Print("-------------------------\n");

fmt.Print("func2() is executed\n");
boolean = false || func2();
fmt.Print("\n");
```

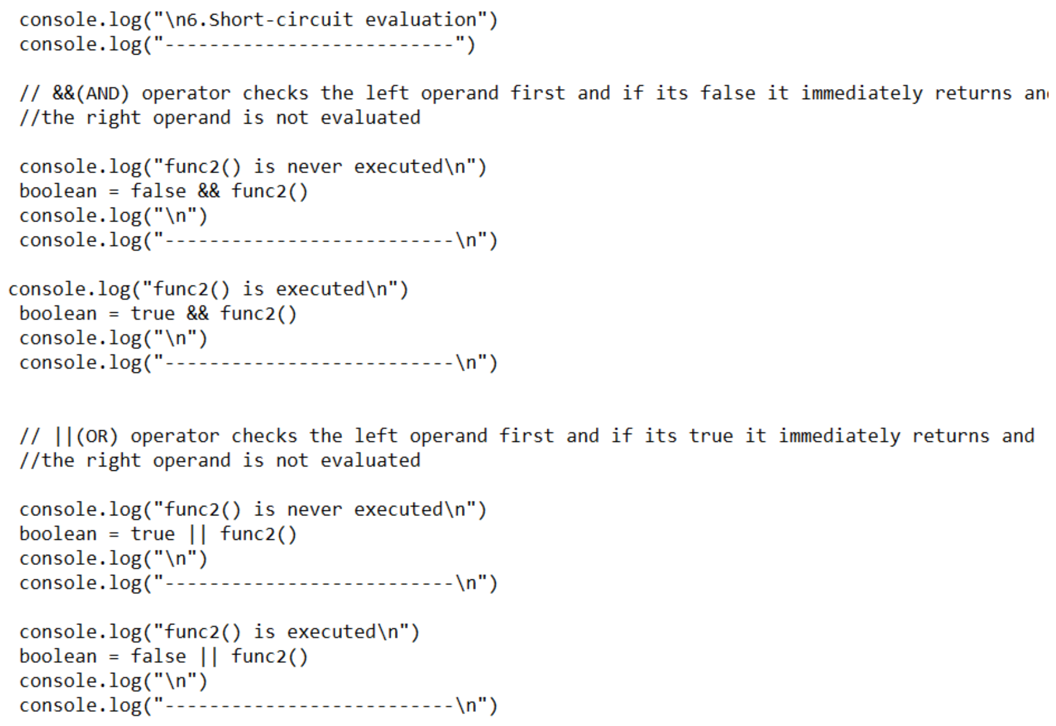
Output



```
6.Short-circuit evaluation
--------------------------
func2() is never executed

--------------------------
func2() is executed
func2
--------------------------
func2() is never executed

--------------------------
func2() is executed
func2
```

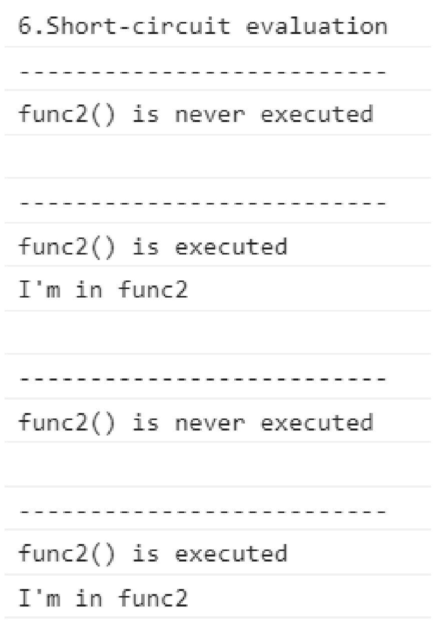
Javascript

Code Segment



```
 console.log("\n6.Short-circuit evaluation")
 console.log("-------------------------")

 // &&(AND) operator checks the left operand first and if its false it immediately returns an
 //the right operand is not evaluated

 console.log("func2() is never executed\n")
 boolean = false && func2()
 console.log("\n")
 console.log("-------------------------\n")

console.log("func2() is executed\n")
 boolean = true && func2()
 console.log("\n")
 console.log("-------------------------\n")


 // ||(OR) operator checks the left operand first and if its true it immediately returns and
 //the right operand is not evaluated

 console.log("func2() is never executed\n")
 boolean = true || func2()
 console.log("\n")
 console.log("-------------------------\n")

 console.log("func2() is executed\n")
 boolean = false || func2()
 console.log("\n")
 console.log("-------------------------\n")
```

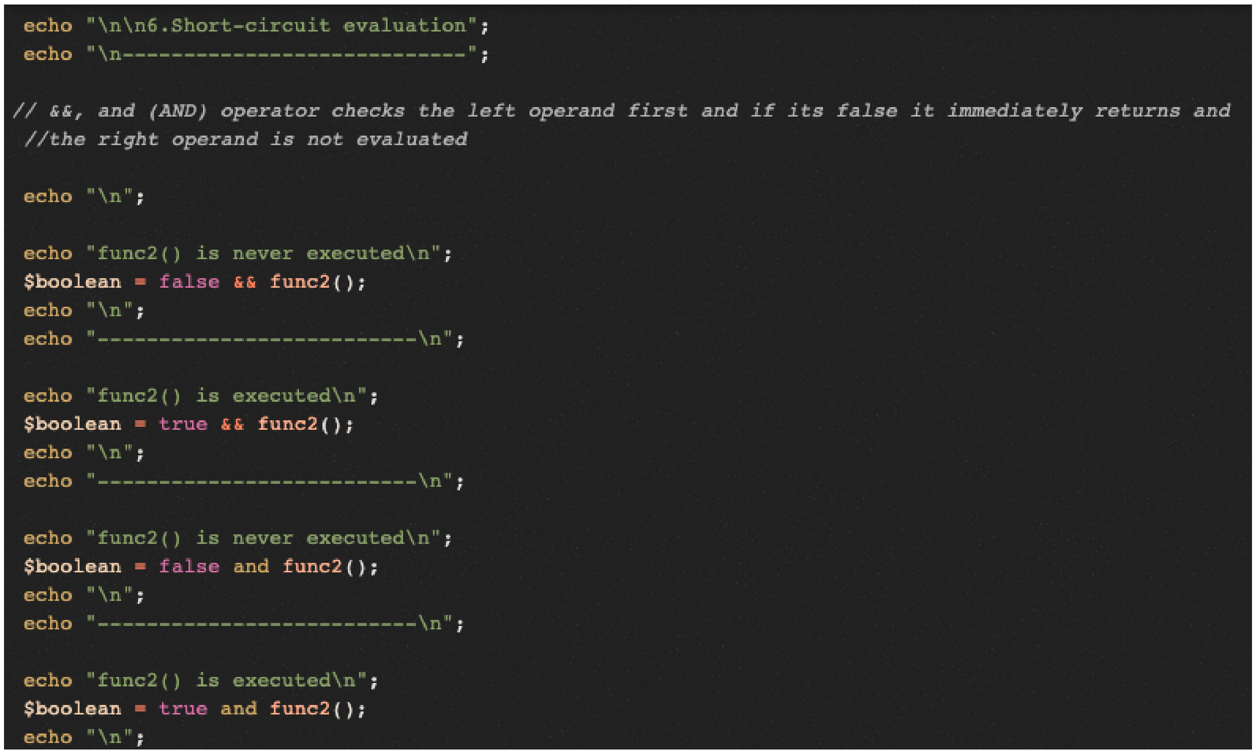
Output



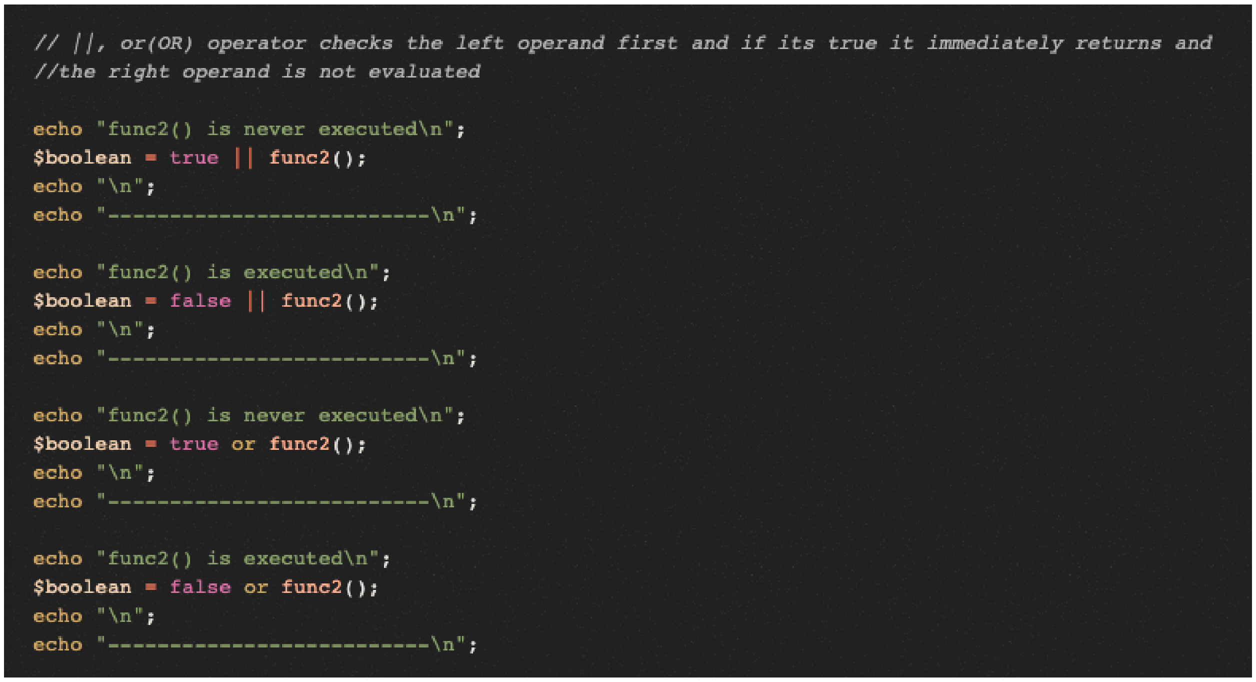
```
6.Short-circuit evaluation
--------------------------
func2() is never executed

--------------------------
func2() is executed
I'm in func2

--------------------------
func2() is never executed

--------------------------
func2() is executed
I'm in func2
```

PHP

Code Segment

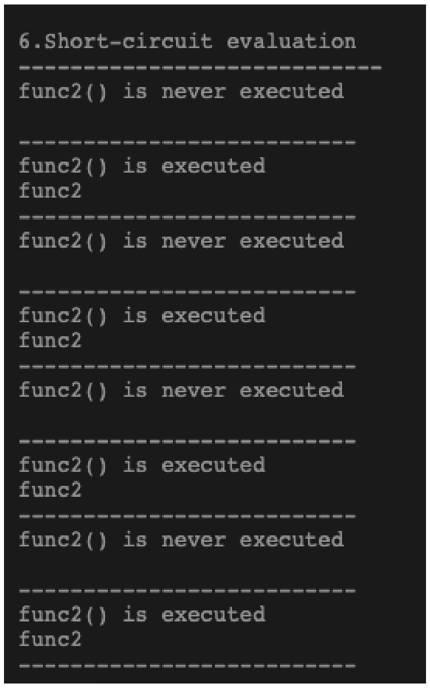


```
echo "\n\n6.Short-circuit evaluation";
echo "\n-----------------------------";

// &&, and (AND) operator checks the left operand first and if its false it immediately returns and
//the right operand is not evaluated

echo "\n";

echo "func2() is never executed\n";
$boolean = false && func2();
echo "\n";
echo "--------------------------\n";

echo "func2() is executed\n";
$boolean = true && func2();
echo "\n";
echo "--------------------------\n";

echo "func2() is never executed\n";
$boolean = false and func2();
echo "\n";
echo "--------------------------\n";

echo "func2() is executed\n";
$boolean = true and func2();
echo "\n";
```



```
// ||, or(OR) operator checks the left operand first and if its true it immediately returns and
//the right operand is not evaluated

echo "func2() is never executed\n";
$boolean = true || func2();
echo "\n";
echo "--------------------------\n";

echo "func2() is executed\n";
$boolean = false || func2();
echo "\n";
echo "--------------------------\n";

echo "func2() is never executed\n";
$boolean = true or func2();
echo "\n";
echo "--------------------------\n";

echo "func2() is executed\n";
$boolean = false or func2();
echo "\n";
echo "--------------------------\n";
```

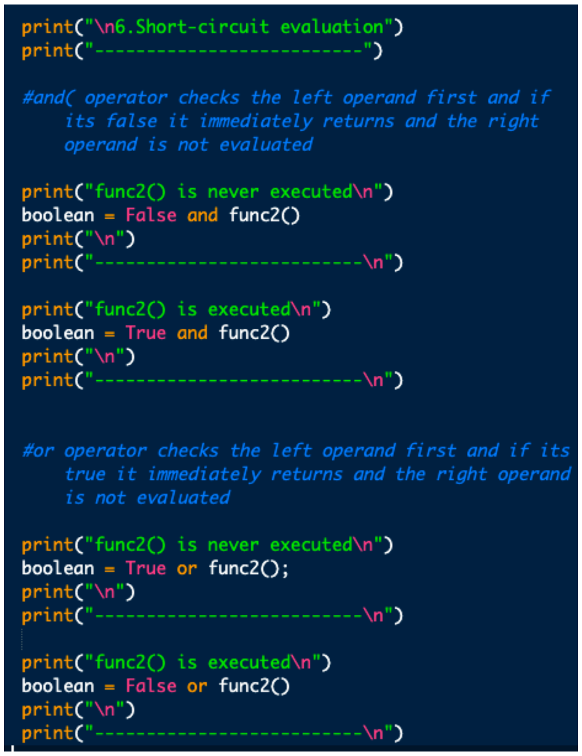
Output



```
6.Short-circuit evaluation
----------------------------
func2() is never executed

--------------------------
func2() is executed
func2
--------------------------
func2() is never executed

--------------------------
func2() is executed
func2
--------------------------
func2() is never executed

--------------------------
func2() is executed
func2
--------------------------
func2() is never executed

--------------------------
func2() is executed
func2
--------------------------
```

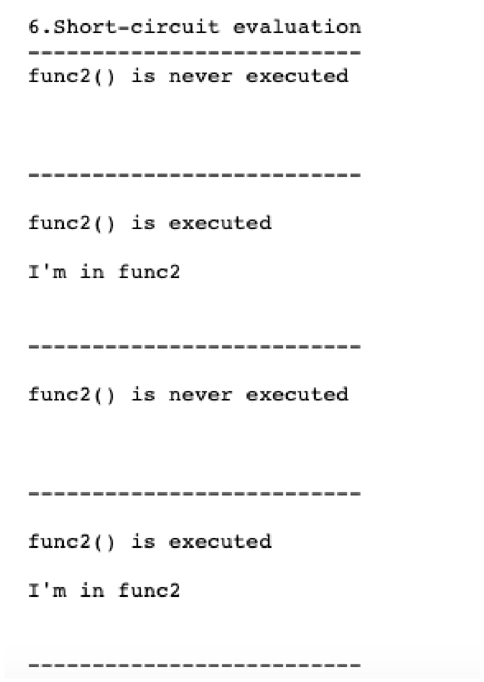
Python

Code Segment



```python
print("\n6.Short-circuit evaluation")
print("--------------------------")

#and( operator checks the left operand first and if
    its false it immediately returns and the right
    operand is not evaluated

print("func2() is never executed\n")
boolean = False and func2()
print("\n")
print("-------------------------\n")

print("func2() is executed\n")
boolean = True and func2()
print("\n")
print("-------------------------\n")


#or operator checks the left operand first and if its
    true it immediately returns and the right operand
    is not evaluated

print("func2() is never executed\n")
boolean = True or func2();
print("\n")
print("-------------------------\n")

print("func2() is executed\n")
boolean = False or func2()
print("\n")
print("-------------------------\n")
```

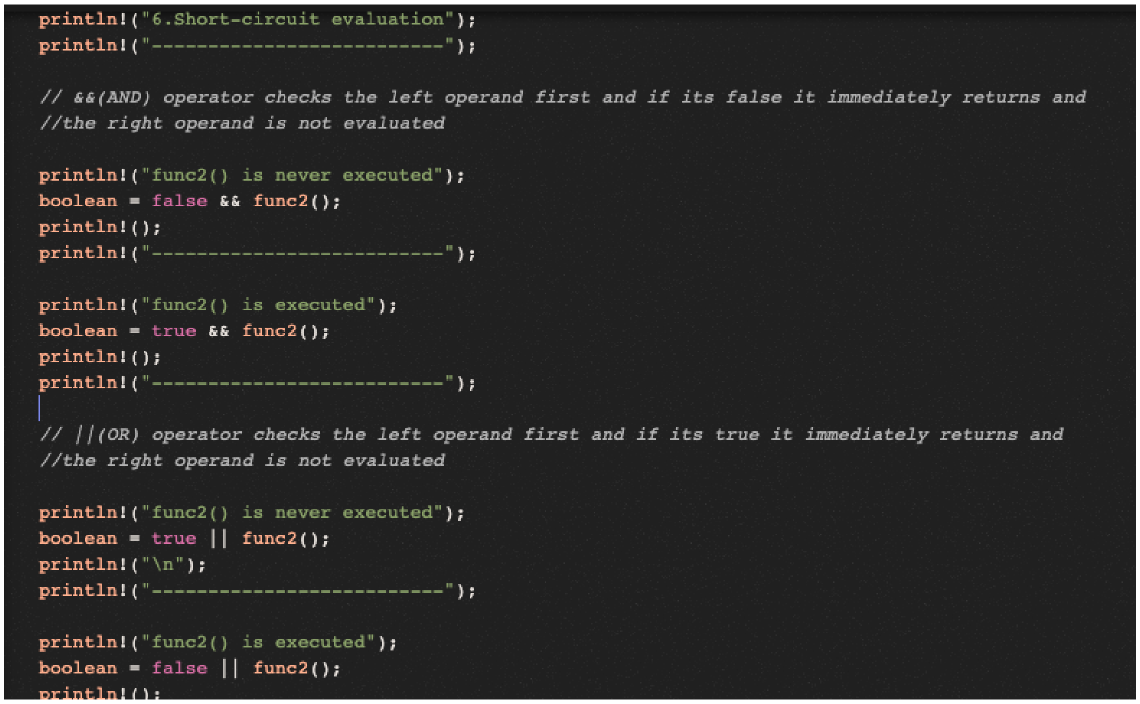
Output



```
6.Short-circuit evaluation
--------------------------
func2() is never executed


--------------------------

func2() is executed

I'm in func2


--------------------------

func2() is never executed



--------------------------

func2() is executed

I'm in func2


--------------------------
```

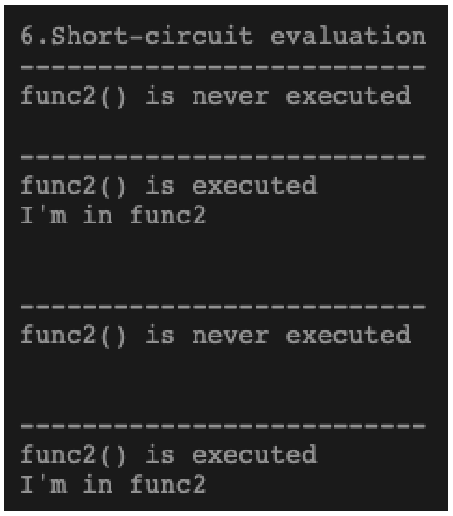
Rust

Code Segment



```
println!("6.Short-circuit evaluation");
println!("--------------------------");

// &&(AND) operator checks the left operand first and if its false it immediately returns and
//the right operand is not evaluated

println!("func2() is never executed");
boolean = false && func2();
println!();
println!("--------------------------");

println!("func2() is executed");
boolean = true && func2();
println!();
println!("--------------------------");

// ||(OR) operator checks the left operand first and if its true it immediately returns and
//the right operand is not evaluated

println!("func2() is never executed");
boolean = true || func2();
println!("\n");
println!("--------------------------");

println!("func2() is executed");
boolean = false || func2();
println!();
```

Output



```
6.Short-circuit evaluation
-------------------------
func2() is never executed

-------------------------
func2() is executed
I'm in func2


-------------------------
func2() is never executed


-------------------------
func2() is executed
I'm in func2
```

2. Evaluation of Languages

My least favourite language in terms of readability and writability was definitely PHP because this language has feature multiplicity. For example there are two ways to write some logical operators such as or operator (‘or’ - ||) and and operator (‘and’ - &&). Also different data types can be used as operands without any errors and logical operators who are supposed to do the same job behave differently when the operands are unexpected. Another thing that bothered me was the return value when the expression is false which is an empty string. This situation made me recheck my code multiple times till I figured that my code was right from the beginning and the reason that I’m not getting any output was actually meaning the expression returned false. Python and Javascript are also not very readable and writable because again different data types can be used as operands without any errors and sometimes the return values are very unreasonable. On the other hand, in C again different data types can be used as operands but there’s a rule which is anything other than 0 is false and this rule is very straightforward. So, the only outputs are 0 or 1. Although C is straightforward, I think having a boolean type is better in terms of readability and writability because reading a statement like ‘c’ && 90 wouldn’t make sense to someone who doesn’t know the language. This brings us to my favourite languages which are GO and Rust. In these languages operands of logical operators should be boolean and they give error if you try to use different data types and the return values are always true or false. In other parts of the experiment I did not observe much differences between languages because they tend to act in a similar manner.

3. Learning Strategy

First of all I started with a skeleton program to give me an idea of what I will do to complete this task. I wrote my skeleton program in C programming language because it's the language that I have most knowledge about and also the one that I’m most comfortable with. I wrote a C program that will address all of the design issues specified in the homework description and separated all parts with meaningful comments. I also added information and clarifications on top of each part which I obtained while doing research about these parts. I wrote the C code using the Xcode development environment in MacOS. After I finished the C program I started writing the same program in other languages: GO, Python, Rust and PHP respectively using online compilers (will be provided later) and I wrote the Javascript program in Linux using a text editor while making the necessary syntax changes. Then, I started my report. For each part I went over all of the programs again one by one and looked for differences in each one of them and modified the programs to be able to show the important aspects. I did lots of research to be able to understand the dynamics of each language. Also, did lots of experiments with different data types, variables, print outs and functions. Then, I added the code segments and outputs to the report.

URLs to online compilers:

GO: https://play.golang.org/
Python: https://www.tutorialspoint.com/execute_python_online.php
Rust: https://rextester.com/l/rust_online_compiler
PHP: https://rextester.com/l/php_online_compiler