Name(s)____ Period ______ Date ___________________

Worksheet - Compound Conditionals

Chained and Nested Conditionals

In order to express more complex decisions, we've used **chained conditionals** (`else` / `else-if`) and **nested conditionals** (`if` statements inside of `if` statements). These are powerful tools which **can be combined to express any complex boolean condition.**

Practice: Using what you know about writing `if`, `if-else`, and `if-else-if` statements, write ***pseudocode*** to accomplish the tasks given below.[1] Assume that you are writing an application which asks the user to input the day of the week (a string) and his age (a number) and that these are stored in variables called **`day`** and **`age`.**

- **EXAMPLE:** If the day is Friday, write, "Thank goodness it's Friday." Otherwise, write, "How long till Friday?"

```
if day == Friday
        write "Thank goodness it's Friday."
else
        write "How long till Friday?"
```

- If the day is any day but "Monday" the program writes, "At least it's not Monday."

- If the user is a teenager, the program writes, "You are a teenager." Otherwise, it writes, "You are not a teenager." (Remember that you can assume there is a variable called **age** that you can refer to.)

- If the day is a weekend day, the program writes, "It is the weekend." Otherwise, it writes, "It is not the weekend."

[1] Writing **pseudocode** means you should use the structural conventions of a programming language, but it is intended for human reading rather than machine reading. Even if pseudocode sometimes looks very close to a real programming language, there is no expectation that it be syntactically perfect - it's about expressing and communicating your ideas for a program.

Improving Conditionals with Boolean Operators

As we noted before, **nested and chained conditionals can be used to create any possible boolean condition**. Unfortunately, however, the resulting code will often be long, cumbersome, and redundant. To help simplify the expression of complex conditionals we will be introducing three new logical operators: **NOT**, **AND**, and **OR**.

NOT

When creating more complex boolean expressions you will encounter instances when you want to know if a statement **is false** rather than true. You have seen the **!=** ("is not equal") operator before, but previously we might also have accomplished this by using the `else` statement. Sometimes, expressing logical conditions is a challenge and you might feel like you need to write an `if` statement *just* to use the `else` clause. Here is simple example.

Example:

```
if(day == "Monday"){
}
else {
  write("At least it's not Monday.");
}
```

But writing an empty `if` statement just to use the `else` clause is bad practice (some would call it ugly code). There are many instances in programming where an operation doesn't have a convenient logical inverse (like **==** and **!=**) and we simply want to say **if** some condition is **NOT** true. The single **!** is the **NOT** operator. Here's an example:

Operator	Description	Truth Table	JavaScript Syntax
NOT statement1	Evaluates to the opposite truth value of the statement provided as input	*expr* | !(*expr*) T | F F | T	**!**

Note the **truth table** shown above. A truth table is a simple tool used to show how every possible value of the input will be treated by a logical operation. Here we use "*expr*" to stand in for any expression that might evaluate to **true** or **false**. This table shows that when **NOT** is applied to a boolean expression, the result is the opposite truth value.

Also note that the **JavaScript syntax** for NOT is a single exclamation point. As good practice, you should place the expression to which you want to apply the NOT within parentheses. Here's our example from before, simplified with the **NOT** operation.

Updated Example:

```
if(!(day == "Monday")){
  write("At least it's not Monday.");
}
```

Practice: Circle whether each expression evaluates to `true` or `false`. The variable `day` is initialized as shown.

```
var day = "Monday";
```

1. `!(day == "Monday")` (Evaluates to `true`) (Evaluates to `false`)
2. `!(2 > 3)` (Evaluates to `true`) (Evaluates to `false`)
3. `!(day == "Tuesday")` (Evaluates to `true`) (Evaluates to `false`)

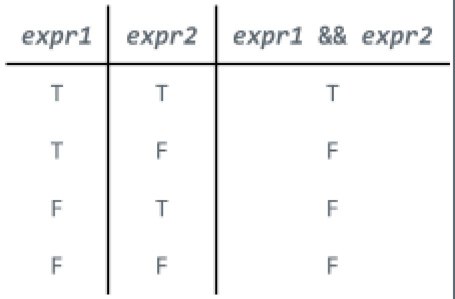
AND

When we've wanted to check whether **multiple conditions are true,** we have been making use of nested conditionals. Here's how we might approach determining whether a user is a teenager, based on the variable `age`.

Example:

```
if(age >= 13){
  if(age < 20){
    write("You are a teenager.");
   } else {
     write("You are not a teenager.");
   }
} else {
  write("You are not a teenager.");
}
```

Not only is this code long, but there is redundancy introduced by having to write "You are not a teenager." at two different points. We can improve the expression of this complex condition with the **AND** operator.

Operator	Description	Truth Table	JavaScript Syntax
statement1 **AND** statement2	Evaluates to true only when both boolean statements it connects are true		`&&`

expr1	*expr2*	*expr1 && expr2*
T	T	T
T	F	F
F	T	F
F	F	F

Notice that the **truth table** for **AND** includes columns for two statements, "a" and "b," and that every possible true / false combination of their values is shown in the columns.

When using the **AND** operator, we can significantly improve the code we wrote in the example above (see below). We've actually improved the code in two ways. First, notice that we've removed the redundant lines of code. Second, since all of the logic of the conditional can now be found in one line, it is easier to read and understand, making it a better expression of what you are trying say.

Updated Example:

```
if((age >= 13) && (age < 20)){
  write("You are a teenager.");
} else {
   write("You are not a teenager.");
}
```

It is recommended that you **place each boolean expression inside of its own parentheses**, as shown here. This ensures that the expressions are evaluated in the order you intend.

Practice: Circle whether each expression evaluates to `true` or `false`. The variable `age` is initialized as shown.

```
var age = 16;
```

1. `(age > 12) && (age < 18)` (Evaluates to `true`) (Evaluates to `false`)
2. `(age != 12) && (age > 18)` (Evaluates to `true`) (Evaluates to `false`)
3. `(age != 12) && (2 > 3)` (Evaluates to `true`) (Evaluates to `false`)

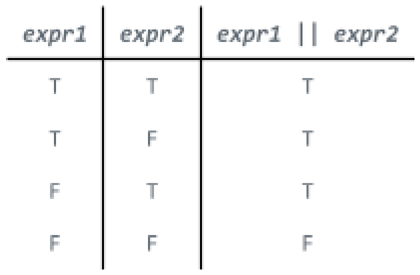
OR

When we've wanted to check whether **at least one** of many conditions is true we have been making use of chained conditionals. Here's how we might approach determining whether it is the weekend based on the variable "day".

Example:

```
if(day == "Saturday"){
  write("It is the weekend.");
} else if(day == "Sunday"){
  write("It is the weekend.”);
} else {
  write("It is not the weekend.");
}
```

Once again, this code is long and introduces redundancies. We can improve the expression of this complex condition with the **OR** operator.

Operator	Description	Truth Table	JavaScript Syntax
statement1 **OR** statement2	Evaluates to true as long as at least one of the boolean statements it connects is true		||

expr1	expr2	expr1 || expr2
T	T	T
T	F	T
F	T	T
F	F	F

The syntax for **OR** in JavaScript is **two vertical pipe characters** that you probably have not used very much outside of programming. When using the **OR** operator, we can significantly improve the code we wrote in the example above. Once again we've removed redundancy and improved the overall readability of our code.

Updated Example:

```
if((day == "Saturday") || (day == "Sunday")){
  write("It is the weekend.");
} else {
  write("It is not the weekend.");
}
```

Notice that once again we have **placed each boolean expression inside of its own parentheses**. This is not strictly necessary, but it makes it clear which expressions are being grouped together.

Practice: Circle whether each expression evaluates to `true` or `false`. Assume the variable day is initialized as shown.

```
var day = "Monday";
```

1. `(day == "Mon") || (day == "Monday")` (Evaluates to `true`) (Evaluates to `false`)
2. `(day == "Tues") || (day == "Tuesday")` (Evaluates to `true`) (Evaluates to `false`)
3. `(day == "Tues") || (5 < 10)` (Evaluates to `true`) (Evaluates to `false`)

Activity

Evaluating Compound Conditionals: Determine if the follow statements evaluate to `true` or `false.` Assume in every case that the two variables `age` and `day` have been initialized with the values shown.

```
var age = 16;
var day = "Monday";
```

4. `(age > 10) && (age < 20)` (Evaluates to `true`) (Evaluates to `false`)

5. `!(age > 10)` (Evaluates to `true`) (Evaluates to `false`)

6. `(day == "Tuesday") || (age < 12)` (Evaluates to `true`) (Evaluates to `false`)

7. `!((age == 16) || (day == "Monday"))` (Evaluates to `true`) (Evaluates to `false`)

8. `!((age == 16) && !(day == "Monday"))` (Evaluates to `true`) (Evaluates to `false`)

Challenge Problems

9. `((age == 16) && (day == "Monday")) && (day == "Tuesday")`

   (Evaluates to `true`) (Evaluates to `false`)

10. `((age == 16) && (day == "Monday")) || (day == "Tuesday")`

    (Evaluates to `true`) (Evaluates to `false`)

11. `((age > 10) && ((age + 5) > 20))`

    (Evaluates to `true`) (Evaluates to `false`)