the shortcut

Fullstack Web Development Tutorial Lesson 4

Today's lesson will cover

- Logical operators
- Loops
- Switch statement

JavaScript fundamentals

Logical Operators

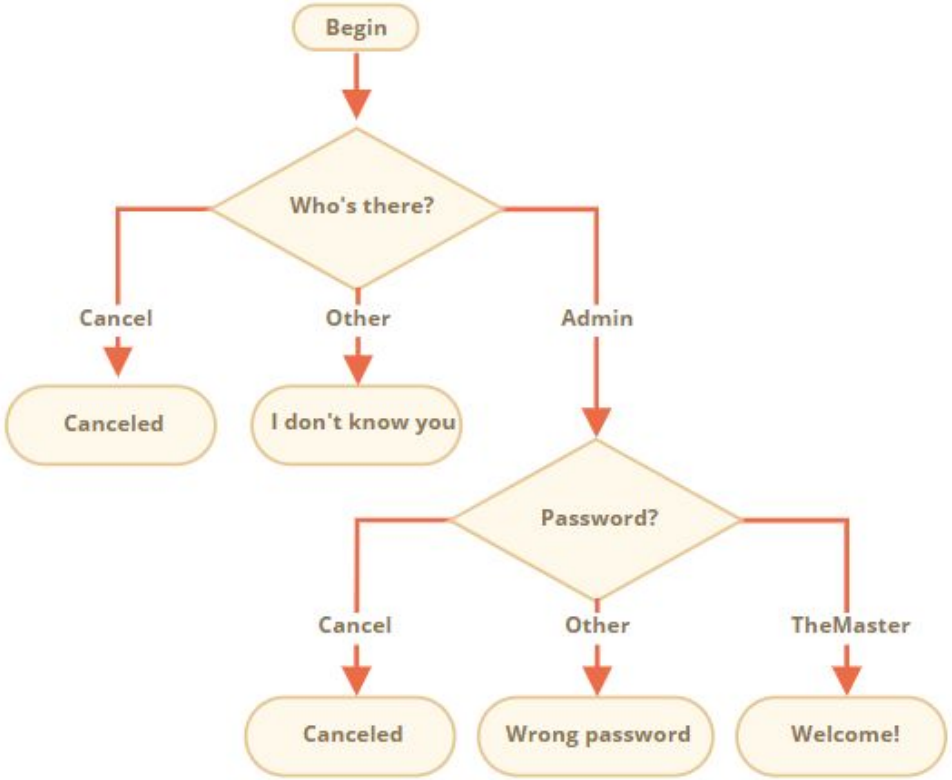
- There are three logical operators in JavaScript: `||` (OR), `&&` (AND), `!` (NOT)
- Although they are called “logical”, they can be applied to values of any type, not only boolean. Their result can also be of any type.
- In classical programming, the logical OR is meant to manipulate boolean values only. If any of its arguments are `true`, it returns `true`, otherwise it returns `false`.
    - In JavaScript, the operator is a little bit trickier and more powerful
- In classical programming, AND returns `true` if both operands are truthy, otherwise `false`
- The boolean NOT operator is represented with an exclamation sign `!`
    - Accepts a single argument and does the following:
        - Converts the operand to boolean type: `true/false`.
        - Returns the inverse value.
        - A double NOT `!!` is sometimes used for converting a value to boolean type

Exercise: Logical Operators

- Write the code which asks for a login username with `prompt`.
- If the visitor enters `"Admin"`, then `prompt` for a password, if the input is an empty line or `Esc` – show “Canceled”, if it’s another string – then show “I don’t know you”.

The password is checked as follows:

- If it equals “TheMaster”, then show “Welcome!”,
- Another string – show “Wrong password”,
- For an empty string or cancelled input, show “Canceled”
- Hint: passing an empty input to a prompt returns an empty string `''`. Pressing `ESC` during a prompt returns `null`.



Loop: while

- *Loops* are a way to repeat the same code multiple times
- While loop
  - While the `condition` is truthy, the `code` from the loop body is executed
  - If `i++` was missing from the example above, the loop would repeat (in theory) forever
- Do...while loop
  - condition check can be moved *below* the loop body using the `do..while` syntax
  - The loop will first execute the body, then check the condition, and, while it's truthy, execute it again and again
  - This form of syntax should only be used when you want the body of the loop to execute **at least once** regardless of the condition being truthy

Loop: for

- The `for` loop is more complex, but it's also the most commonly used loop
- Syntax:
  - ```
    for (begin; condition; step) {
      // ... loop body ...
    }
    ```
- For loop steps:
  - begin: Executes once upon entering the loop.
  - condition: Checked before every loop iteration. If false, the loop stops.
  - body: Runs again and again while the condition is truthy.
  - step: Executes after the body on each iteration.
- General loop algorithm
  - Run begin
  - → (if condition → run body and run step)
  - → (if condition → run body and run step)
  - → (if condition → run body and run step)
  - → ...
    ```

Loop: for *(Contd.)*

- Inline variable declaration: The “counter” variable `i` is declared right in the loop. This is called an “inline” variable declaration. Such variables are visible only inside the loop.
- You can skip parts of the loop and it will still work
  - note that the two `for` semicolons `;` must be present. Otherwise, there would be a syntax error.
- Breaking loop:
  - Normally, a loop exits when its condition becomes falsy.
  - But we can force the exit at any time using the special `break` directive.
- Continue directive:
  - The `continue` directive is a “lighter version” of `break`. It doesn’t stop the whole loop. Instead, it stops the current iteration and forces the loop to start a new one (if the condition allows).

Exercise: Loops

- An integer number greater than `1` is called a prime if it cannot be divided without a remainder by anything except `1` and itself.
- In other words, `n > 1` is a prime if it can't be evenly divided by anything except `1` and `n`.
- For example, `5` is a prime, because it cannot be divided without a remainder by `2`, `3` and `4`.
- **Write the code which outputs prime numbers in the interval from `2` to `n`.**
- For `n = 10` the result will be `2,3,5,7`.
- P.S. The code should work for any `n`, not be hard-tuned for any fixed value.

The Switch Statement

- A `switch` statement can replace multiple `if` checks.
- It gives a more descriptive way to compare a value with multiple variants.
- The `switch` has one or more `case` blocks and an optional default
  - ```
    switch(x) {
      case 'value1':  // if (x === 'value1')
         ...
         [break]

      case 'value2':  // if (x === 'value2')
         ...
         [break]

      default:
         ...
         [break]
    }
    ```
- Type matters
    ```

Exercise: Switch statement

- Rewrite the code below using a single `switch` statement:
  - ```
    let a = +prompt('a?', '');

    if (a == 0) {
      alert( 0 );
    }
    if (a == 1) {
      alert( 1 );
    }

    if (a == 2 || a == 3) {
      alert( '2,3' );
    }
    ```
    ```

the shortcut

Self Study Assignments

To Dos

- Continue freecodecamp Javascript. Ideally finish before we resume after summer.
- Continue with FCC HTML, CSS lessons. Ideally finish all the lessons by end of this month.
- Work on the HTML, CSS assignments to make the projects as complete as you desire and push latest version on Git repository
- Share your freecodecamp profile link with Elena so that we can track your progress by Friday 05 June, 2020

the shortcut

Feedback

What works best for you?

- What do you prefer when it comes to individual and peer to peer tasks?