JavaScript – W1S3
Conditional Statements and Loops

Cyrille JEGOUREL – Singapore University of Technology and Design

SINGAPORE UNIVERSITY OF
TECHNOLOGY AND DESIGN

Outline (Week1, Session 3)

- The if statement
- The else statement
- Dead code and code structure
- Nested ifs
- While statements
- Infinite looks and how to kill them
- The break statement

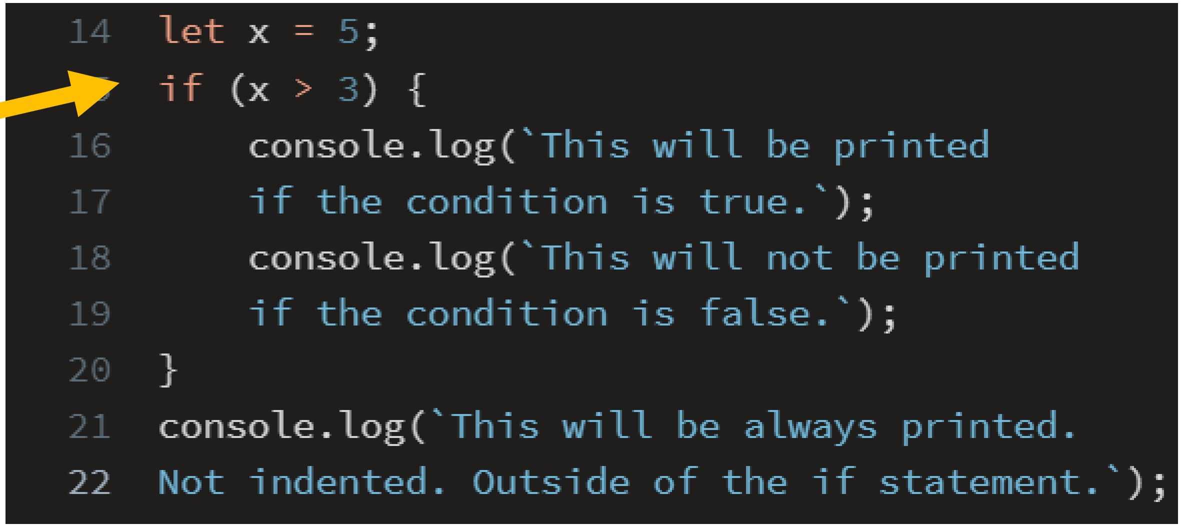
The if statement

The **if** statement is the simplest **conditional structure**.

The if statement

The **if** statement is the simplest **conditional structure**.

- **Structure:**
  - Use the keyword **if**,

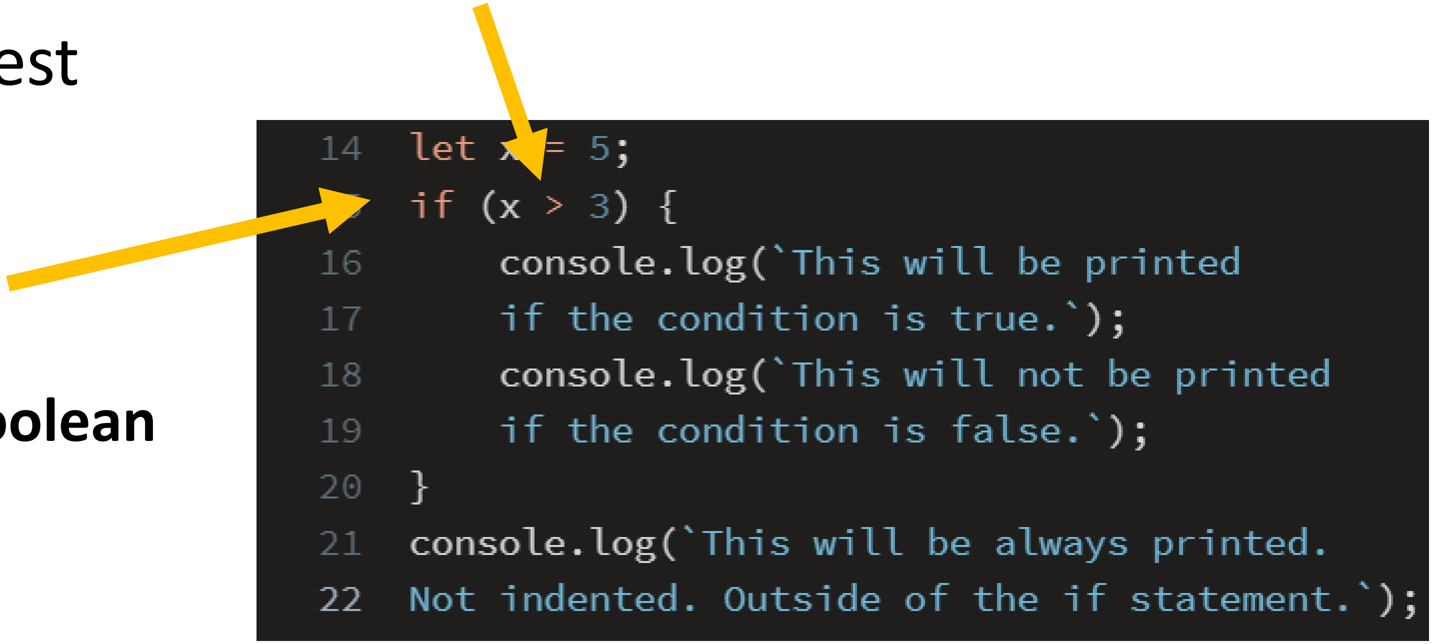


```
let x = 5;
if (x > 3) {
    console.log(`This will be printed
    if the condition is true.`);
    console.log(`This will not be printed
    if the condition is false.`);
}
console.log(`This will be always printed.
Not indented. Outside of the if statement.`);
```

The if statement

The **if** statement is the simplest **conditional structure**.

- **Structure:**
  - Use the keyword **if**,
  - Immediately after, pass a **Boolean** or write an **expression that returns a Boolean**,

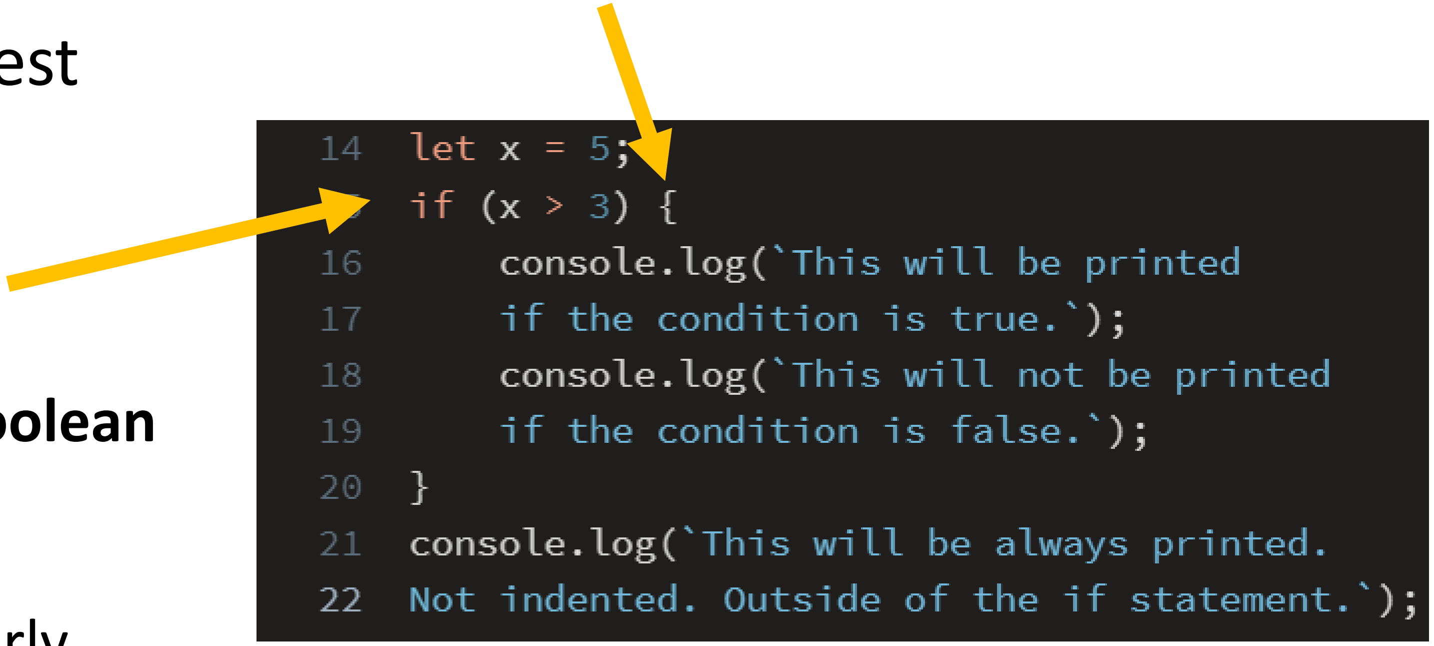


```
let x = 5;
if (x > 3) {
    console.log(`This will be printed
    if the condition is true.`);
    console.log(`This will not be printed
    if the condition is false.`);
}
console.log(`This will be always printed.
Not indented. Outside of the if statement.`);
```

The if statement

The if statement is the simplest **conditional structure**.

- **Structure:**
  - Use the keyword **if**,
  - Immediately after, pass a **Boolean** or write an **expression that returns a Boolean**,
  - Open an if block with the curly bracket **{** after the Boolean term,

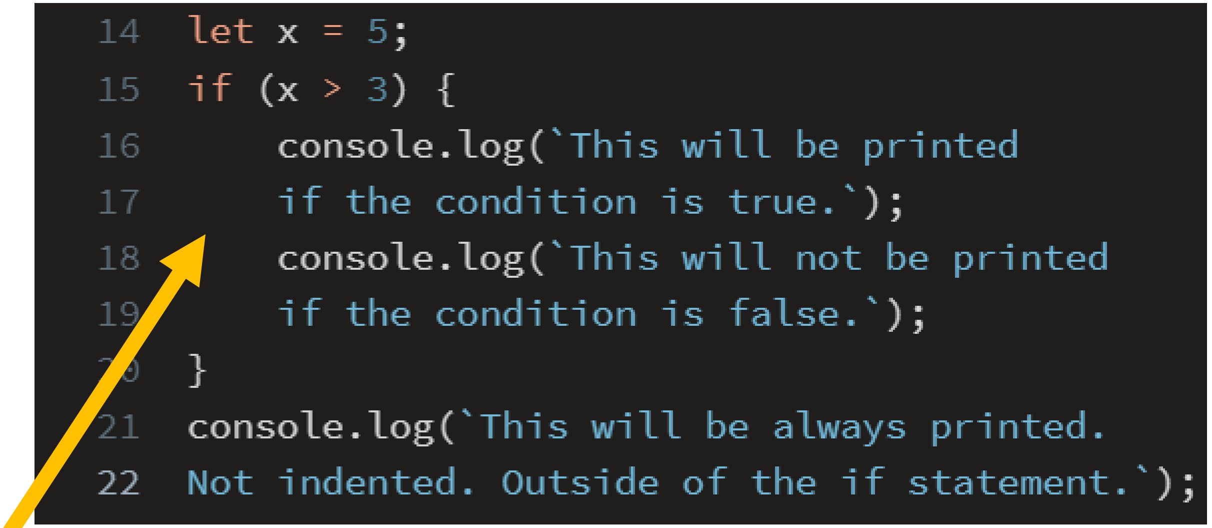


```
let x = 5;
if (x > 3) {
    console.log(`This will be printed
    if the condition is true.`);
    console.log(`This will not be printed
    if the condition is false.`);
}
console.log(`This will be always printed.
Not indented. Outside of the if statement.`);
```

The if statement

The **if** statement is the simplest **conditional structure**.

- **Structure:**
  - Use the keyword **if**,
  - Immediately after, pass a **Boolean** or write an **expression that returns a Boolean**,
  - Open an if block with the curly bracket **{** after the Boolean term,
  - Add a block of instructions **inside** the **if** statement, which will be executed if and only if the Boolean is **true** and close the block with a curly bracket **}** afterwards.

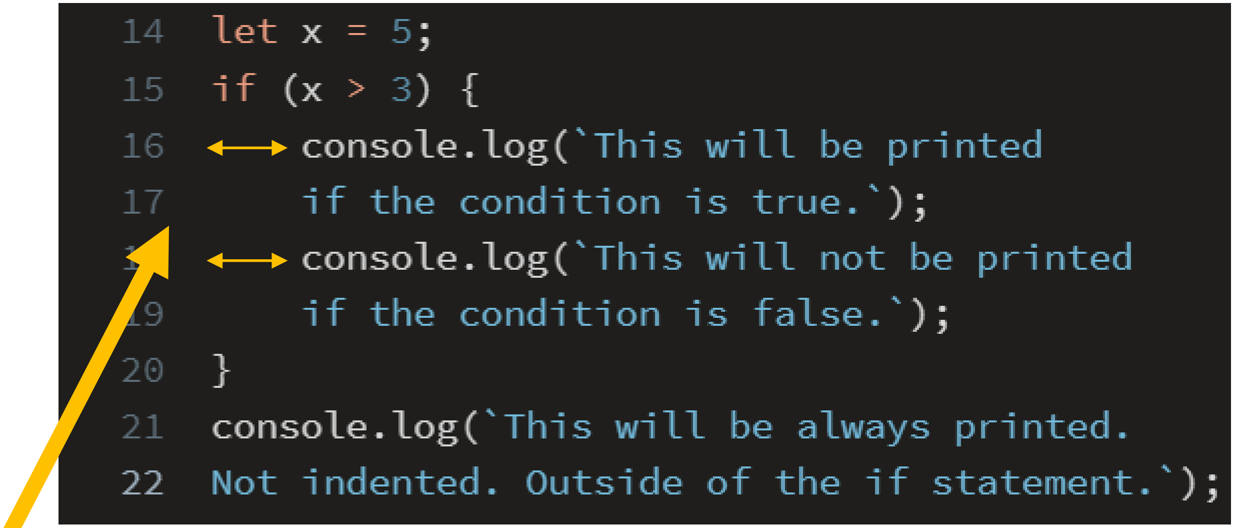


```
let x = 5;
if (x > 3) {
    console.log(`This will be printed
    if the condition is true.`);
    console.log(`This will not be printed
    if the condition is false.`);
}
console.log(`This will be always printed.
Not indented. Outside of the if statement.`);
```

The if statement

The if statement is the simplest **conditional structure**.

- **Structure:**
  - Use the keyword **if**,
  - Immediately after, pass a **Boolean** or write an **expression that returns a Boolean**,
  - Open an if block with the curly bracket **{** after the Boolean term,
  - Add a block of instructions **inside** the **if** statement, which will be executed if and only if the Boolean is **true**.



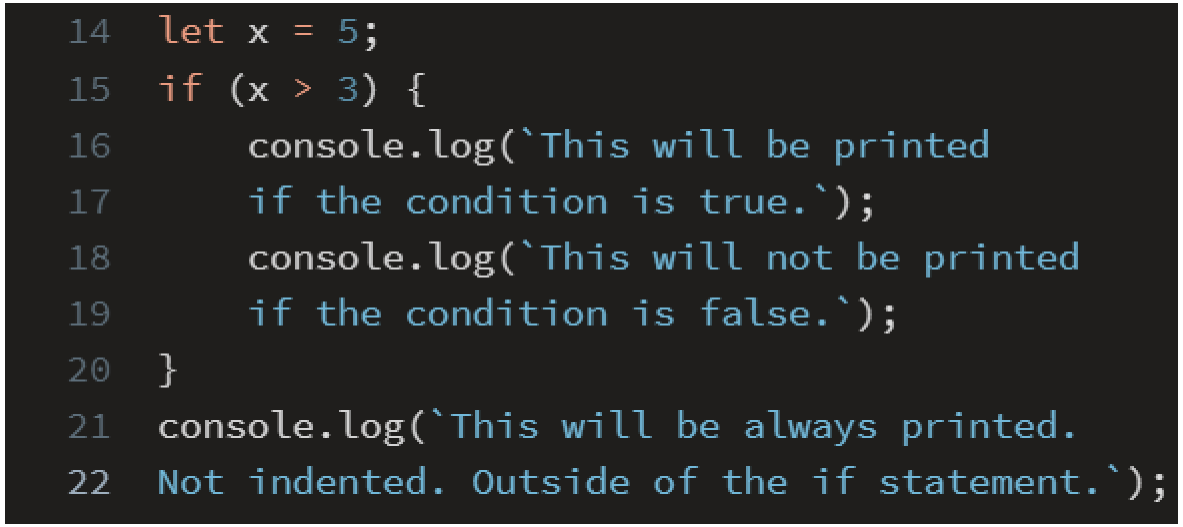
```
let x = 5;
if (x > 3) {
    console.log(`This will be printed
    if the condition is true.`);
    console.log(`This will not be printed
    if the condition is false.`);
}
console.log(`This will be always printed.
Not indented. Outside of the if statement.`);
```

Note: “inside” is a convention that means your instruction should be **indented** with 4 spaces more than the if statement.

The if statement

The **if** statement is the simplest **conditional structure**.

- **How it works:**
    - If the Boolean condition specified for the **if** statement is **true**, then execute the block of code inside the **if** statement.

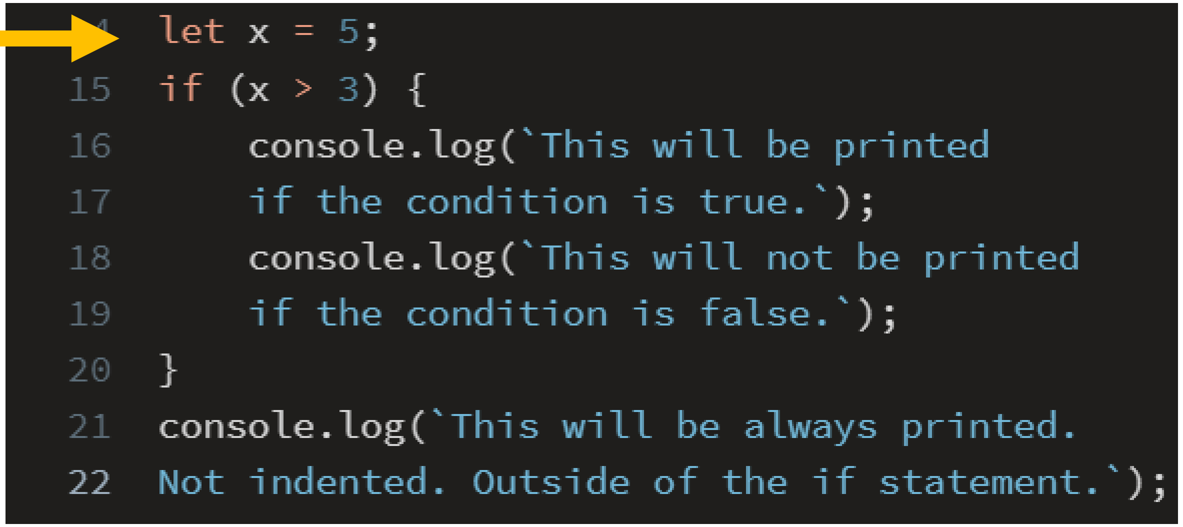


```
let x = 5;
if (x > 3) {
    console.log(`This will be printed
    if the condition is true.`);
    console.log(`This will not be printed
    if the condition is false.`);
}
console.log(`This will be always printed.
Not indented. Outside of the if statement.`);
```

The if statement

The if statement is the simplest **conditional structure**.

- **How it works:**
  - If the Boolean condition specified for the if statement is true, then execute the block of code inside the if statement.

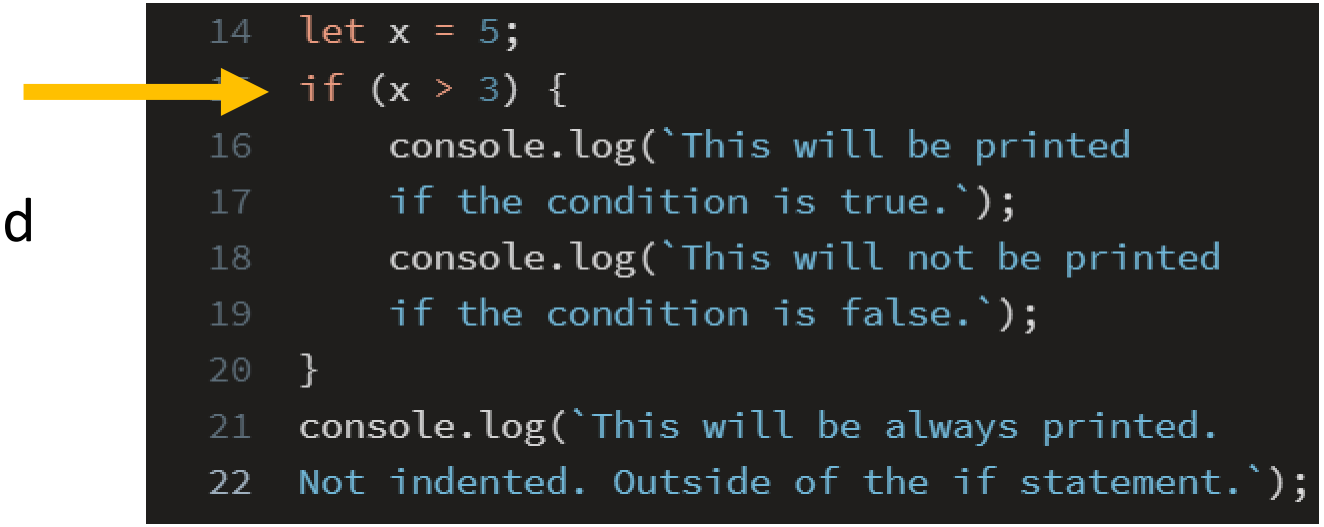


```
let x = 5;
if (x > 3) {
    console.log(`This will be printed
    if the condition is true.`);
    console.log(`This will not be printed
    if the condition is false.`);
}
console.log(`This will be always printed.
Not indented. Outside of the if statement.`);
```

The if statement

The **if** statement is the simplest **conditional structure**.

- **How it works:**
  - If the Boolean condition specified for the **if** statement is **true**, then execute the block of code inside the **if** statement.

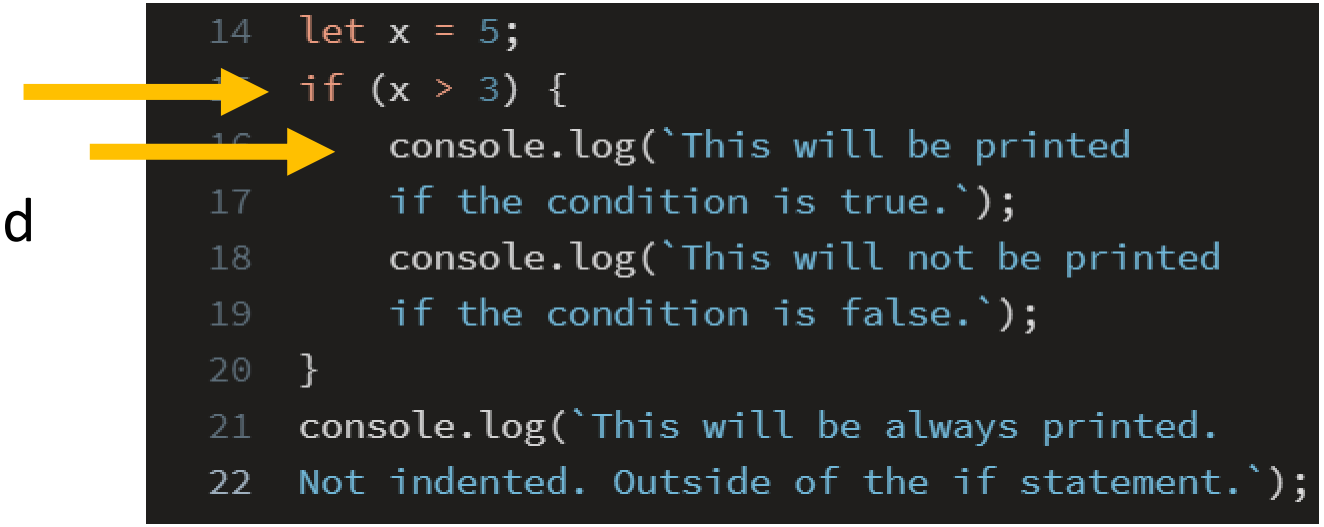


```
let x = 5;
if (x > 3) {
    console.log(`This will be printed
    if the condition is true.`);
    console.log(`This will not be printed
    if the condition is false.`);
}
console.log(`This will be always printed.
Not indented. Outside of the if statement.`);
```

The if statement

The **if** statement is the simplest **conditional structure**.

- **How it works:**
  - If the Boolean condition specified for the **if** statement is **true**, then execute the block of code inside the **if** statement.

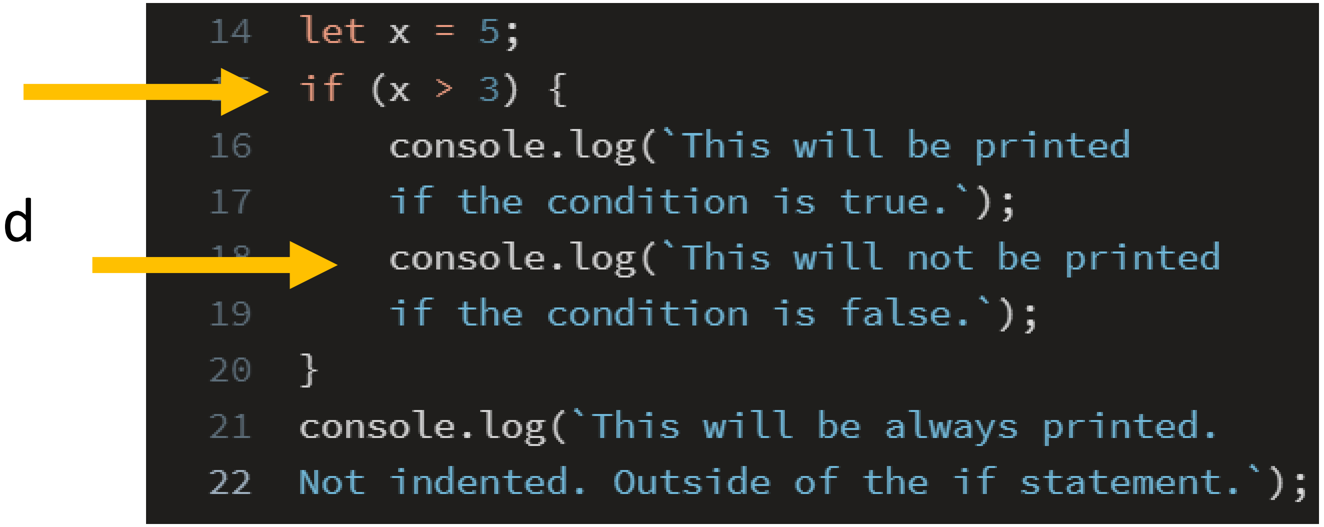


```
let x = 5;
if (x > 3) {
    console.log(`This will be printed
    if the condition is true.`);
    console.log(`This will not be printed
    if the condition is false.`);
}
console.log(`This will be always printed.
Not indented. Outside of the if statement.`);
```

The if statement

The **if** statement is the simplest **conditional structure**.

- **How it works:**
  - If the Boolean condition specified for the **if** statement is **true**, then execute the block of code inside the **if** statement.

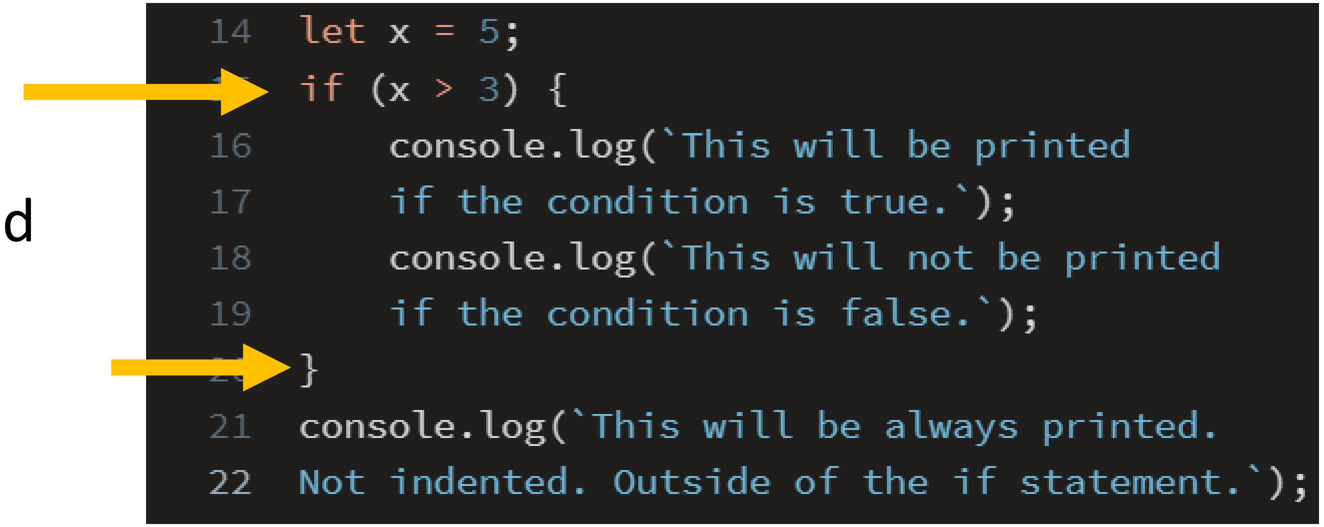


```
let x = 5;
if (x > 3) {
    console.log(`This will be printed
    if the condition is true.`);
    console.log(`This will not be printed
    if the condition is false.`);
}
console.log(`This will be always printed.
Not indented. Outside of the if statement.`);
```

The if statement

The **if** statement is the simplest **conditional structure**.

- **How it works:**
  - If the Boolean condition specified for the **if** statement is **true**, then execute the block of code inside the **if** statement.

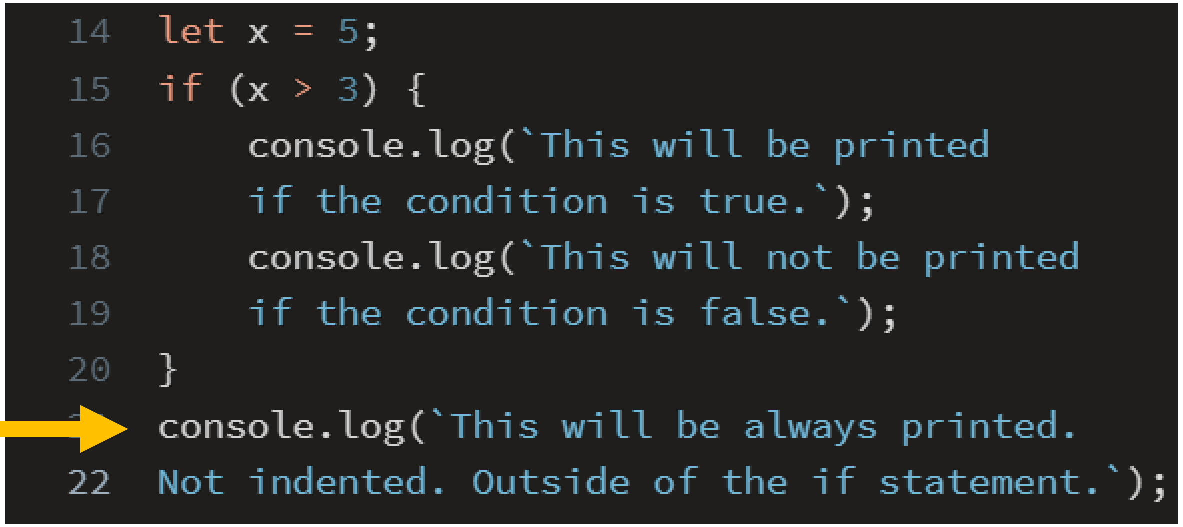


```
let x = 5;
if (x > 3) {
    console.log(`This will be printed
    if the condition is true.`);
    console.log(`This will not be printed
    if the condition is false.`);
}
console.log(`This will be always printed.
Not indented. Outside of the if statement.`);
```

The if statement

The if statement is the simplest **conditional structure**.

- **How it works:**
    - If the Boolean condition specified for the if statement is **true**, then execute the block of code inside the if statement.

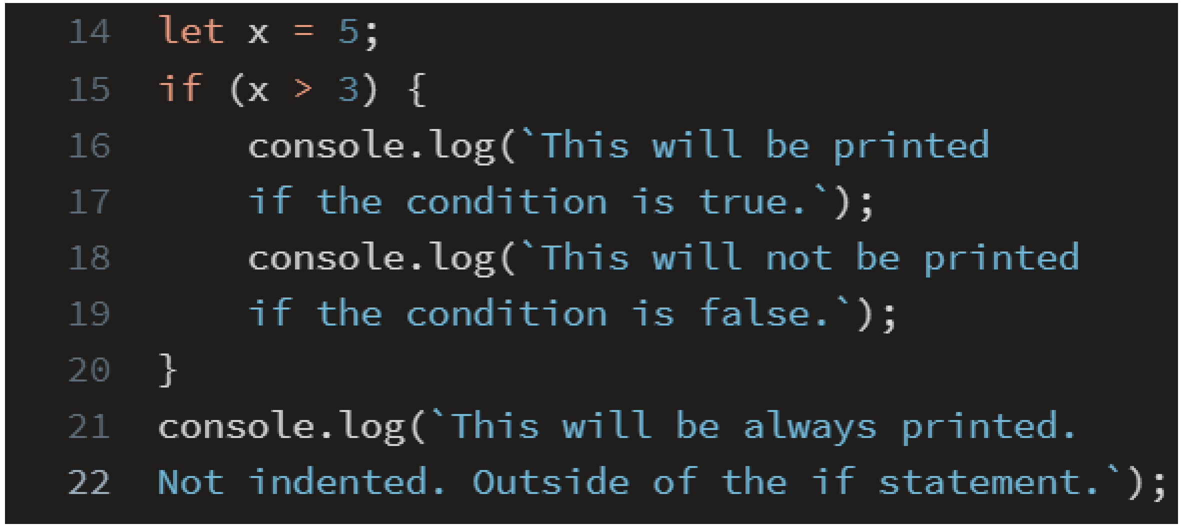


```
let x = 5;
if (x > 3) {
    console.log(`This will be printed
    if the condition is true.`);
    console.log(`This will not be printed
    if the condition is false.`);
}
console.log(`This will be always printed.
Not indented. Outside of the if statement.`);
```

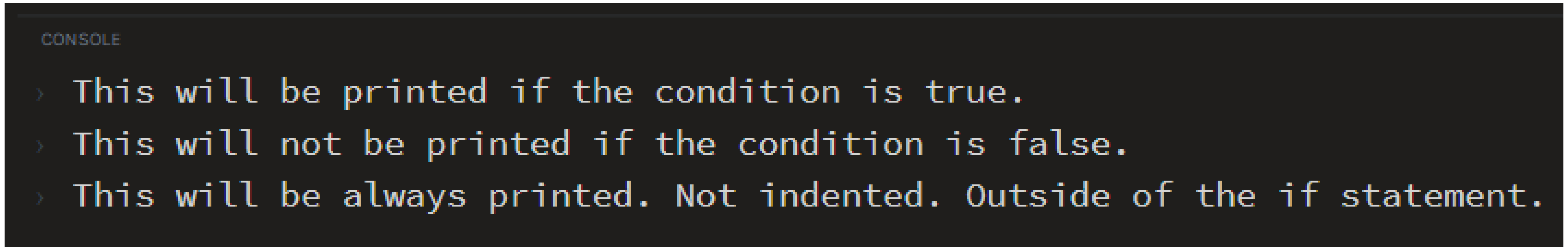
The if statement

The if statement is the simplest **conditional structure**.

- **How it works:**
  - If the Boolean condition specified for the if statement is **true**, then execute the block of code inside the if statement.



```
let x = 5;
if (x > 3) {
    console.log(`This will be printed
    if the condition is true.`);
    console.log(`This will not be printed
    if the condition is false.`);
}
console.log(`This will be always printed.
Not indented. Outside of the if statement.`);
```

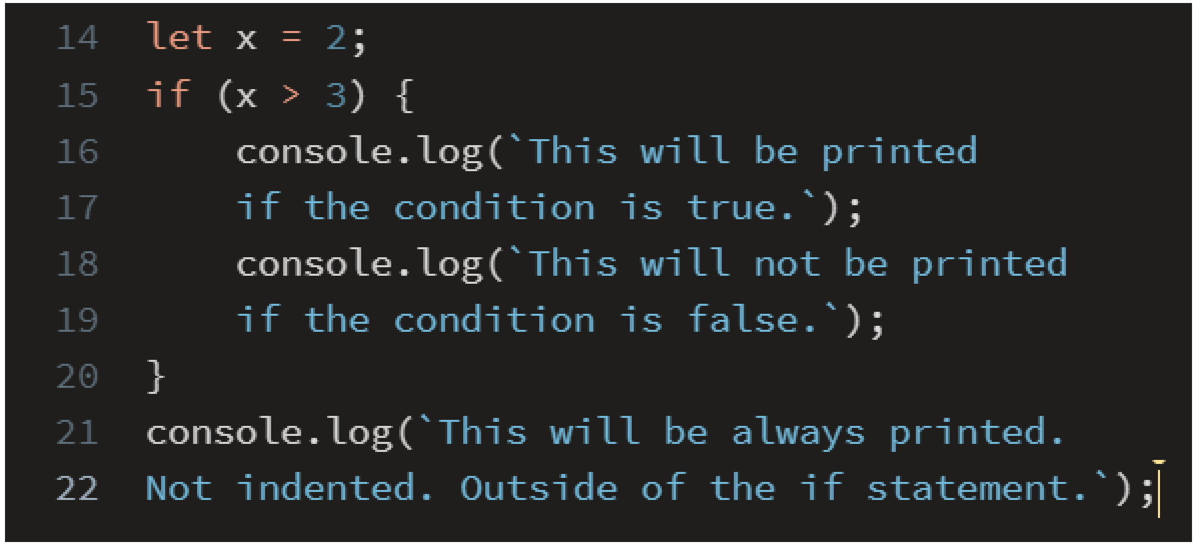


```
› This will be printed if the condition is true.
› This will not be printed if the condition is false.
› This will be always printed. Not indented. Outside of the if statement.
```

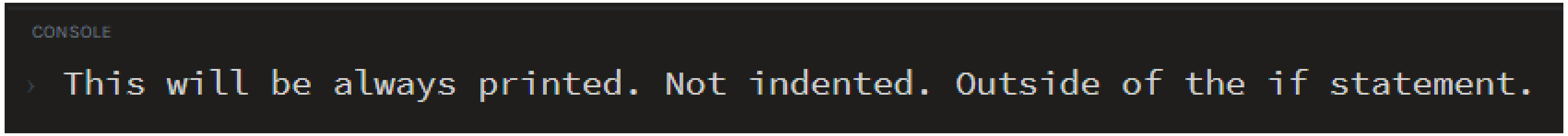
The if statement

The if statement is the simplest **conditional structure**.

- **How it works:**
  - If the Boolean condition specified for the if statement is **true**, then execute the block of code inside the if statement.
  - If the Boolean condition is **false**, ignore the block of code in the if statement.



```javascript
let x = 2;
if (x > 3) {
    console.log(`This will be printed
    if the condition is true.`);
    console.log(`This will not be printed
    if the condition is false.`);
}
console.log(`This will be always printed.
Not indented. Outside of the if statement.`);
```

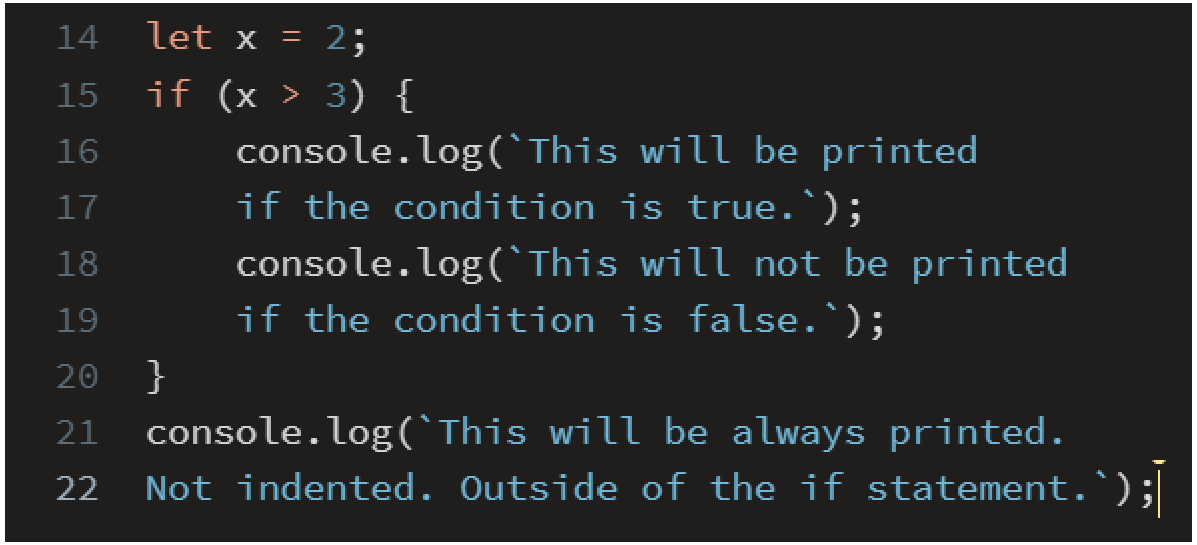


```
This will be always printed. Not indented. Outside of the if statement.
```

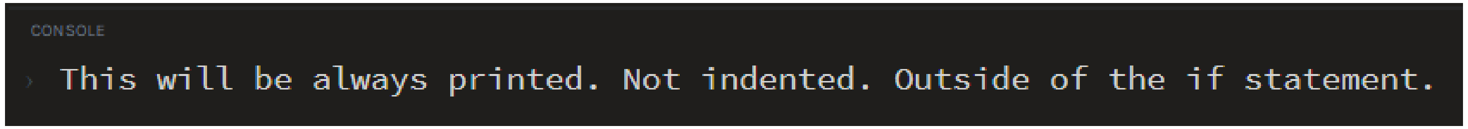
The if statement

The if statement is the simplest **conditional structure**.

- **How it works:**
  - If the Boolean condition specified for the if statement is **true**, then execute the block of code inside the if statement.
  - If the Boolean condition is **false**, ignore the block of code in the if statement.
  - Once we are done executing the code in if (or ignoring it), move on to the next (non-indented) line.



```
let x = 2;
if (x > 3) {
    console.log(`This will be printed
    if the condition is true.`);
    console.log(`This will not be printed
    if the condition is false.`);
}
console.log(`This will be always printed.
Not indented. Outside of the if statement.`);
```



```
This will be always printed. Not indented. Outside of the if statement.
```

The else statement

The **else** statement is used to define a block of code to execute, if and only if the previous **if** statement has failed.

The else statement

The **else** statement is used to define a block of code to execute, if and only if the previous **if** statement has failed.

Same structure as **if**, but…

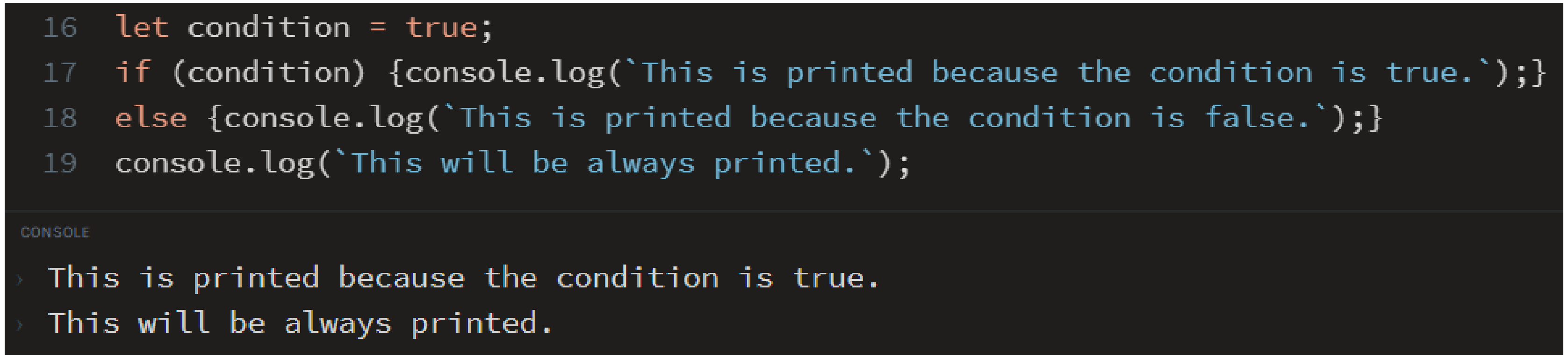
- **Comes last**, after the **if** statement.
- **No Boolean condition** to be checked.

The else statement

The else statement is used to define a block of code to execute, if and only if the previous if statement has failed.

Same structure as if, but…

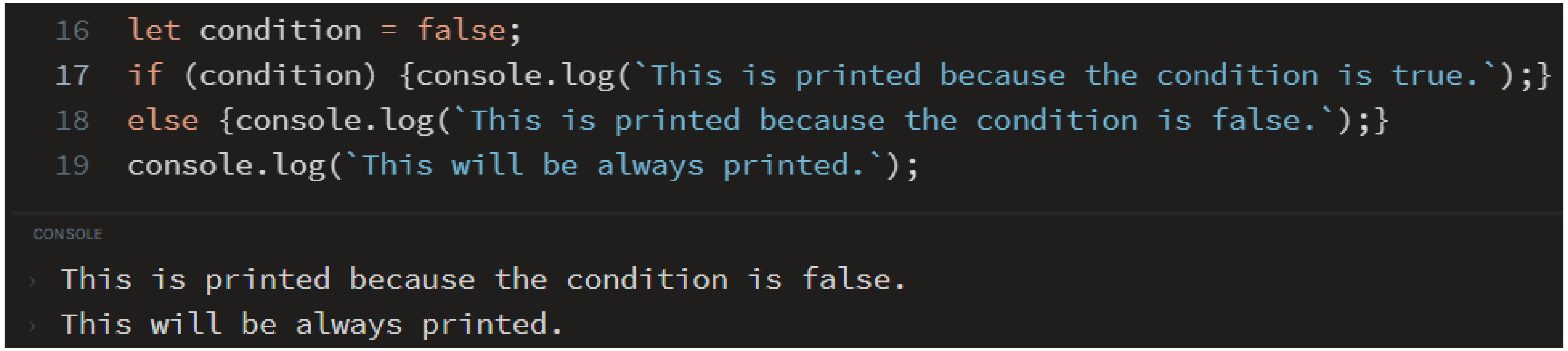
- **Comes last**, after all the if statement.
- **No Boolean condition** to be checked.



```
let condition = true;
if (condition) {console.log(`This is printed because the condition is true.`);}
else {console.log(`This is printed because the condition is false.`);}
console.log(`This will be always printed.`);
```

CONSOLE

```
This is printed because the condition is true.
This will be always printed.
```



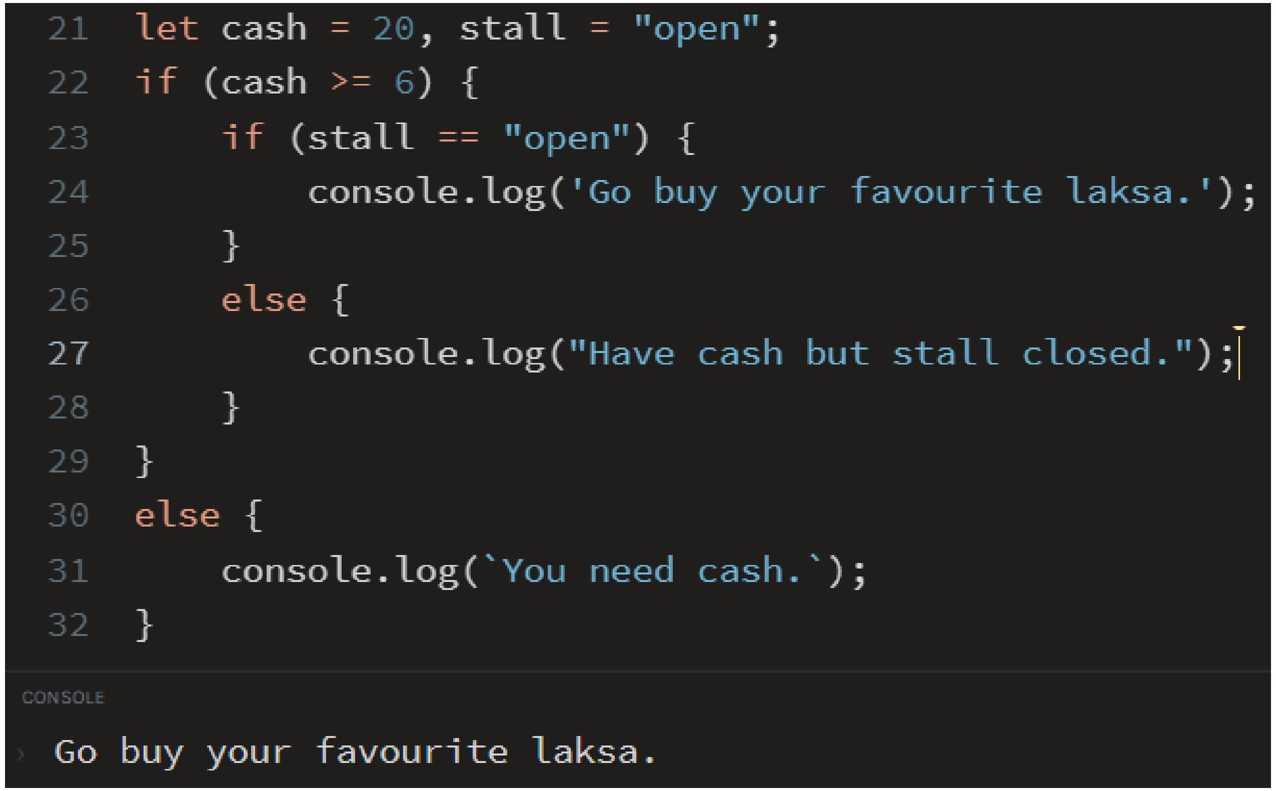
```
let condition = false;
if (condition) {console.log(`This is printed because the condition is true.`);}
else {console.log(`This is printed because the condition is false.`);}
console.log(`This will be always printed.`);
```

CONSOLE

```
This is printed because the condition is false.
This will be always printed.
```

Block nesting

- You can nest other if/else statements within an if block after the comparison or within the else block. In the following example,
- First, you evaluate the Boolean of the first if block (cash >= 6).
- If true (it is the case here), you find a “nested” if block.
- You test the condition of the nested if block (stall == “open”). If true again, you can finally display the “go buy” message. The nested else is then ignored.
- The program exits the nested blocks and eventually reaches the end of the outer if block.
- The last else is ignored.

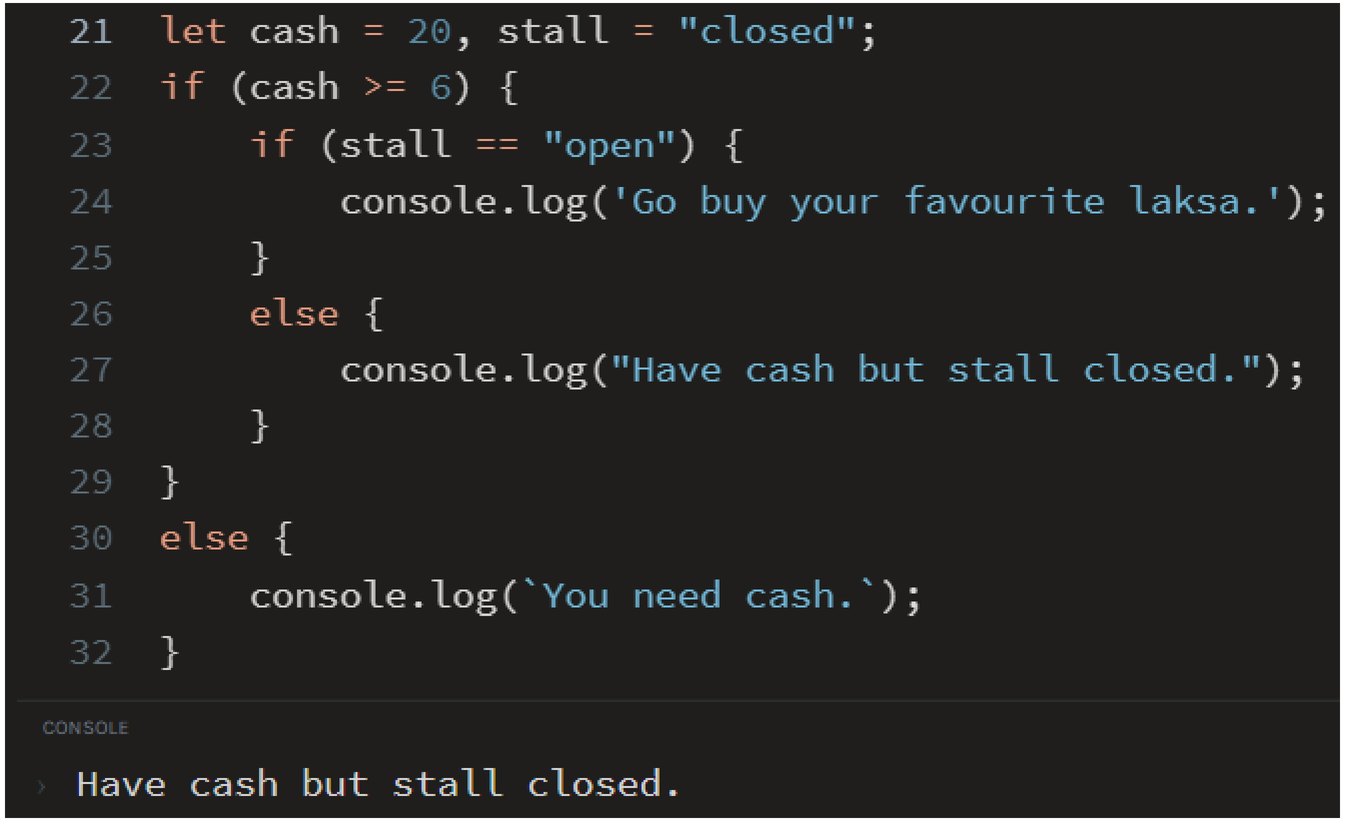


```javascript
let cash = 20, stall = "open";
if (cash >= 6) {
    if (stall == "open") {
        console.log('Go buy your favourite laksa.');
    }
    else {
        console.log("Have cash but stall closed.");
    }
}
else {
    console.log(`You need cash.`);
}
```

```
Go buy your favourite laksa.
```

Block nesting

- You can nest other if/else statements within an if block after the comparison or within the else block.
- With different values of cash (= 4) and stall (= “closed”), you can enter within the outer if block but not in the inner if block. In that case, the inner else block is executed and displays the “have cash” message.
- The outer else block is ignored.

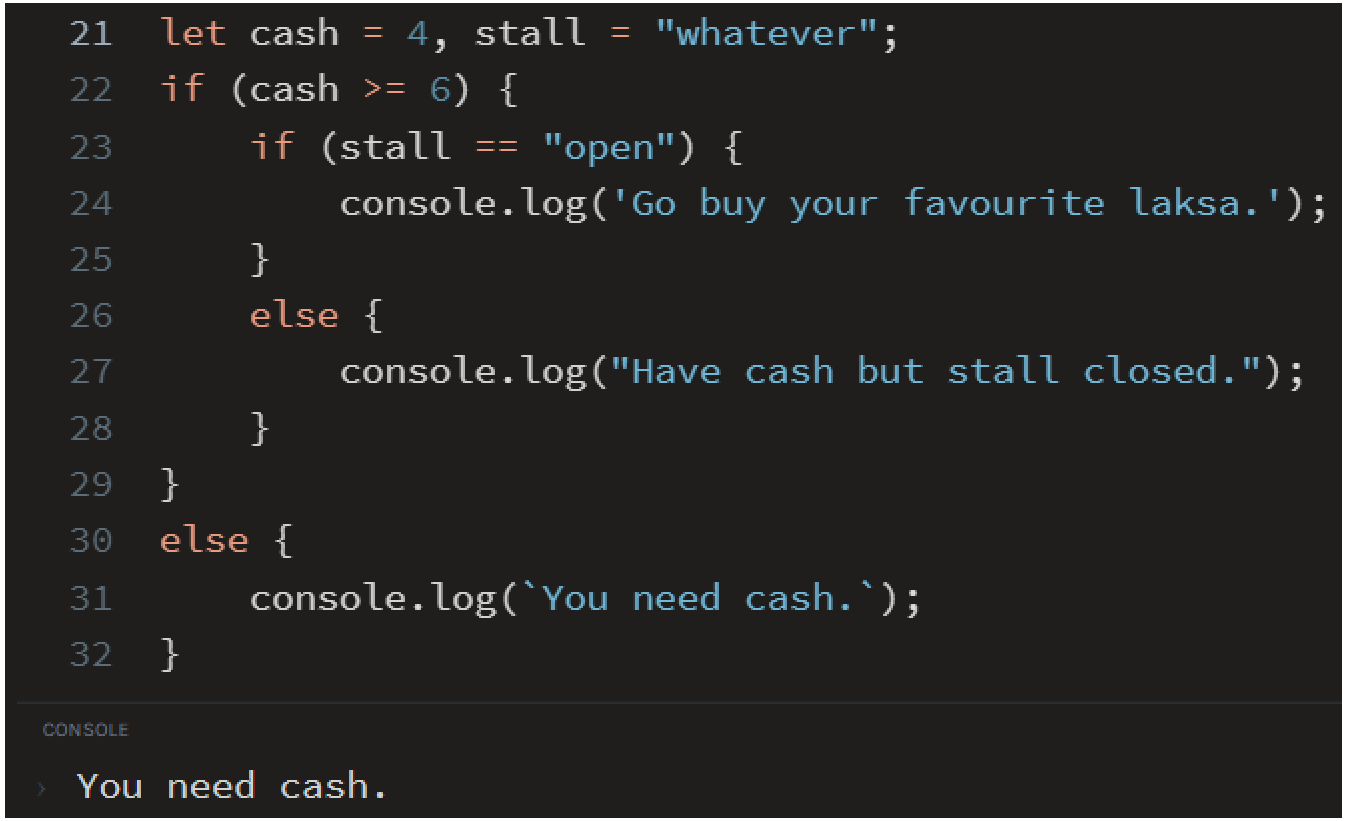


```
let cash = 20, stall = "closed";
if (cash >= 6) {
    if (stall == "open") {
        console.log('Go buy your favourite laksa.');
    }
    else {
        console.log("Have cash but stall closed.");
    }
}
else {
    console.log(`You need cash.`);
}
```

```
Have cash but stall closed.
```

Block nesting

- You can nest other if/else statements within an if block after the comparison or within the else block.
- Finally, if cash < 6, the outer if block can not be executed; the program executes the outer else block and displays the “You need cash” message.
- **Advice**: Be consistent with your indentations and the placing of the curly brackets. As you can see, it can become quickly messy…



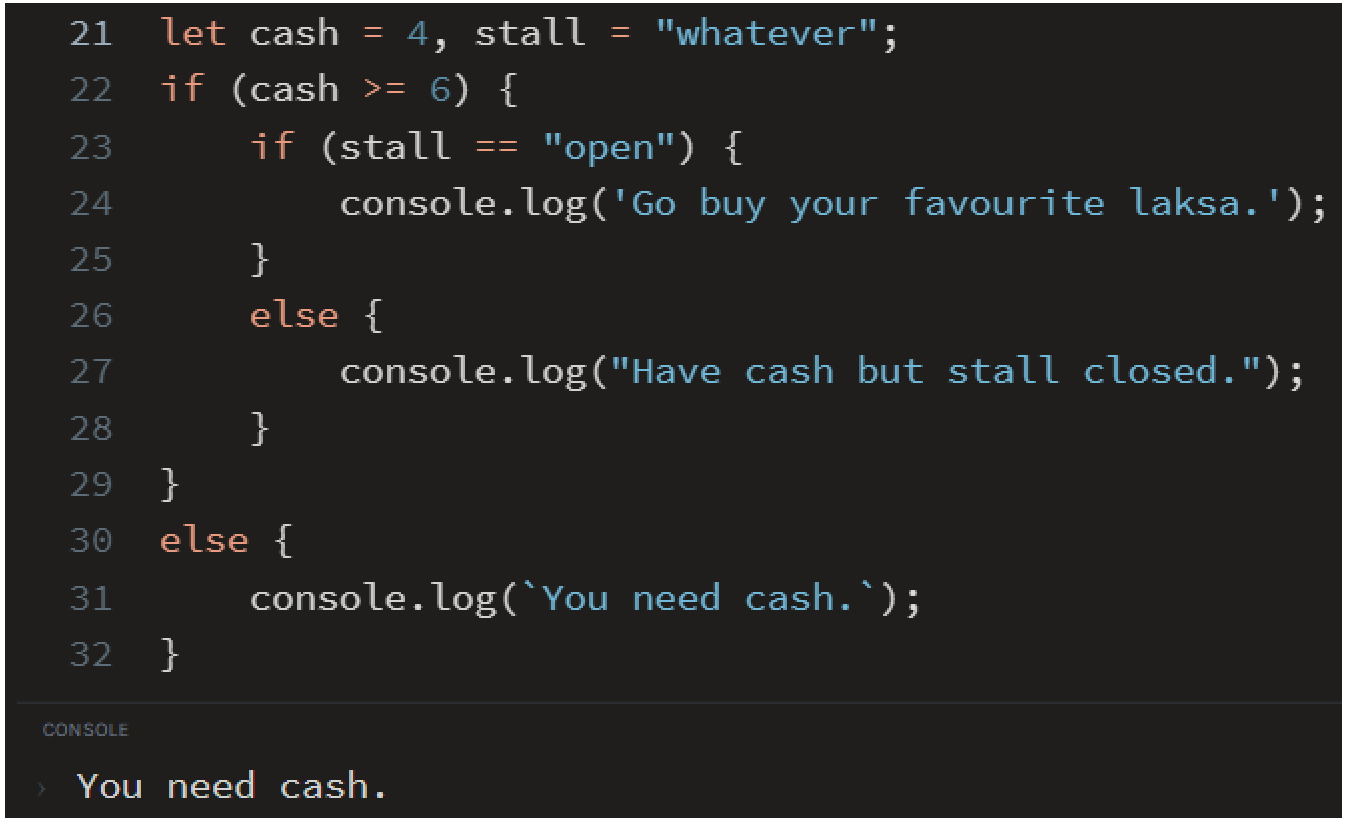
```
let cash = 4, stall = "whatever";
if (cash >= 6) {
    if (stall == "open") {
        console.log('Go buy your favourite laksa.');
    }
    else {
        console.log("Have cash but stall closed.");
    }
}
else {
    console.log(`You need cash.`);
}
```

CONSOLE

```
You need cash.
```

Block nesting

- You can nest other if/else statements within an if block after the comparison or within the else block.
- Finally, if cash < 6, the outer if block can not be executed; the program executes the outer else block and displays the "You need cash" message.



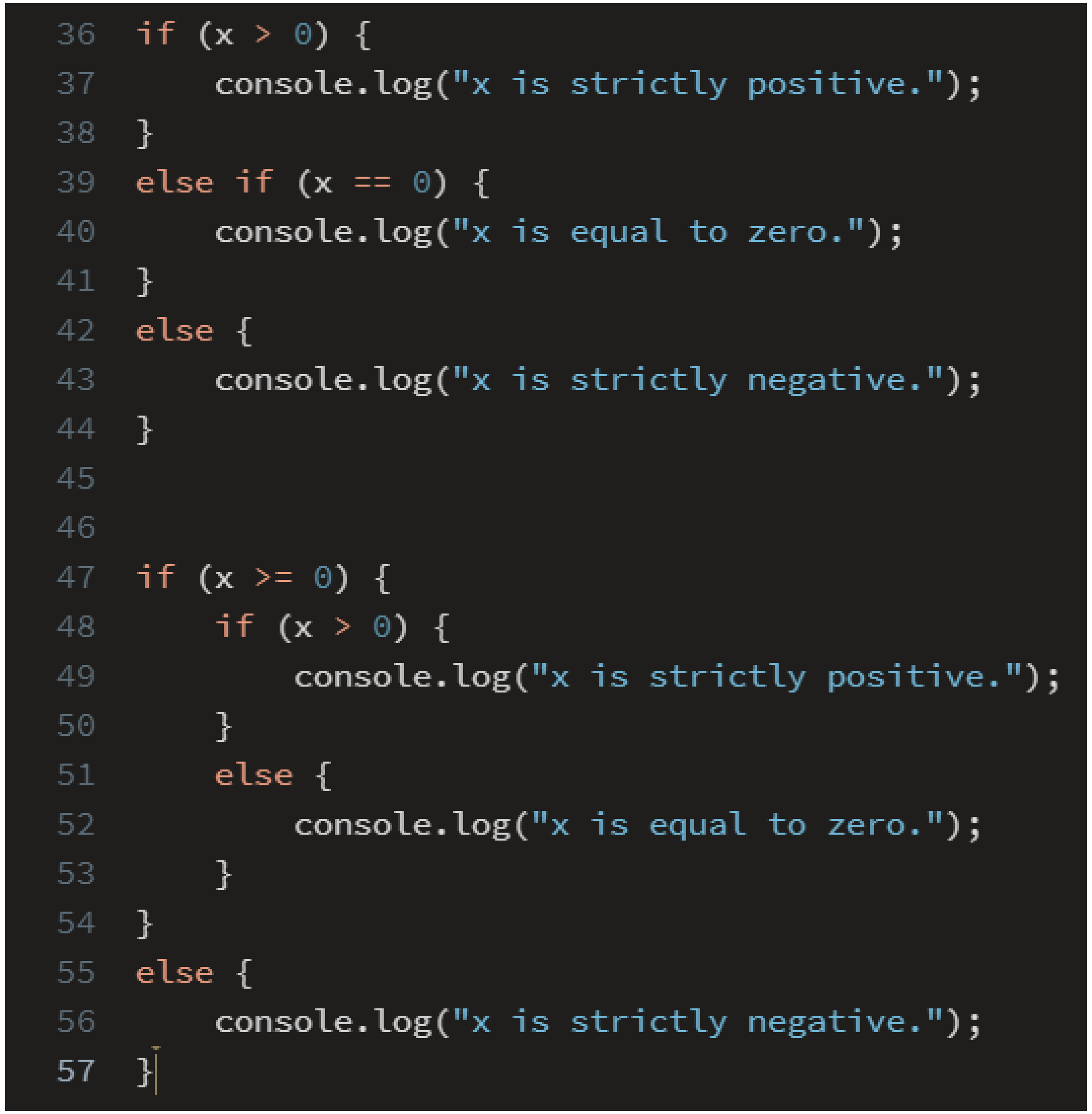
```
let cash = 4, stall = "whatever";
if (cash >= 6) {
    if (stall == "open") {
        console.log('Go buy your favourite laksa.');
    }
    else {
        console.log("Have cash but stall closed.");
    }
}
else {
    console.log(`You need cash.`);
}
```

```
You need cash.
```

- **Advice**: Be consistent with your indentations and the placing of the curly brackets. As you can see, it can become quickly messy…

The else-if statement

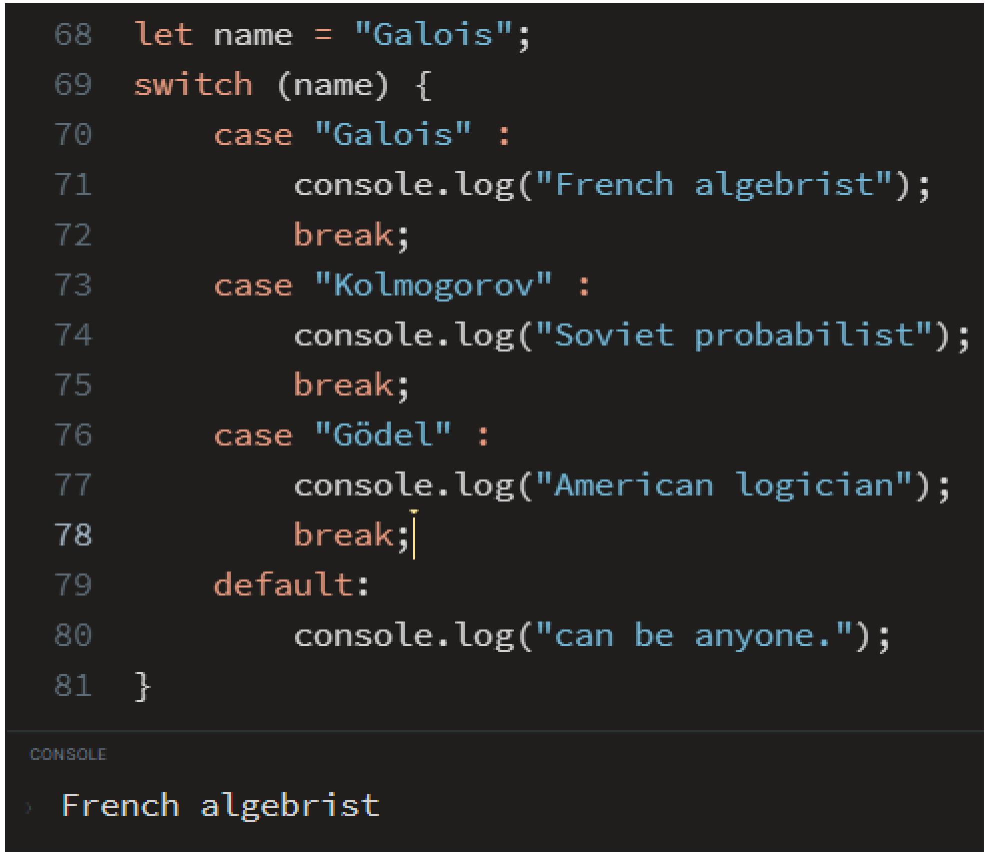
- The “else-if” is used to define another conditional test to be executed, if and only if the previous **if** statement has failed.
- It sometimes makes some scripts more readable. E.g., both programs from line 36 to 44 and from line 47 to 57 are equivalent.
- The first program is based on an if/else-if/else structure whereas the second program is based on block nesting.
- The first program is shorter and more readable due to less indentations and curly brackets.



```
36  if (x > 0) {
37      console.log("x is strictly positive.");
38  }
39  else if (x == 0) {
40      console.log("x is equal to zero.");
41  }
42  else {
43      console.log("x is strictly negative.");
44  }
45
46
47  if (x >= 0) {
48      if (x > 0) {
49          console.log("x is strictly positive.");
50      }
51      else {
52          console.log("x is equal to zero.");
53      }
54  }
55  else {
56      console.log("x is strictly negative.");
57  }
```

The switch statement

- The switch statement is used to execute a different block of code depending on whether a case is equal to the value of an expression.
- **Structure:**
  - Use the keyword **switch** followed by an **expression**.
  - The expression can be a value, a variable, a Boolean.
  - Open a block with the curly bracket **{** after the expression.
  - **For each case statement**, use the **case** keyword followed by a possible value. The expression is compared to this value. If it is true, the case block after the **colon** is executed until the **break** (to exit the switch statement and prevent the other case statements to be evaluated and executed).
  - The last case statement is introduced by the keyword **default** and no break is necessary.
  - Close the switch statement with the closing curly bracket **}**.

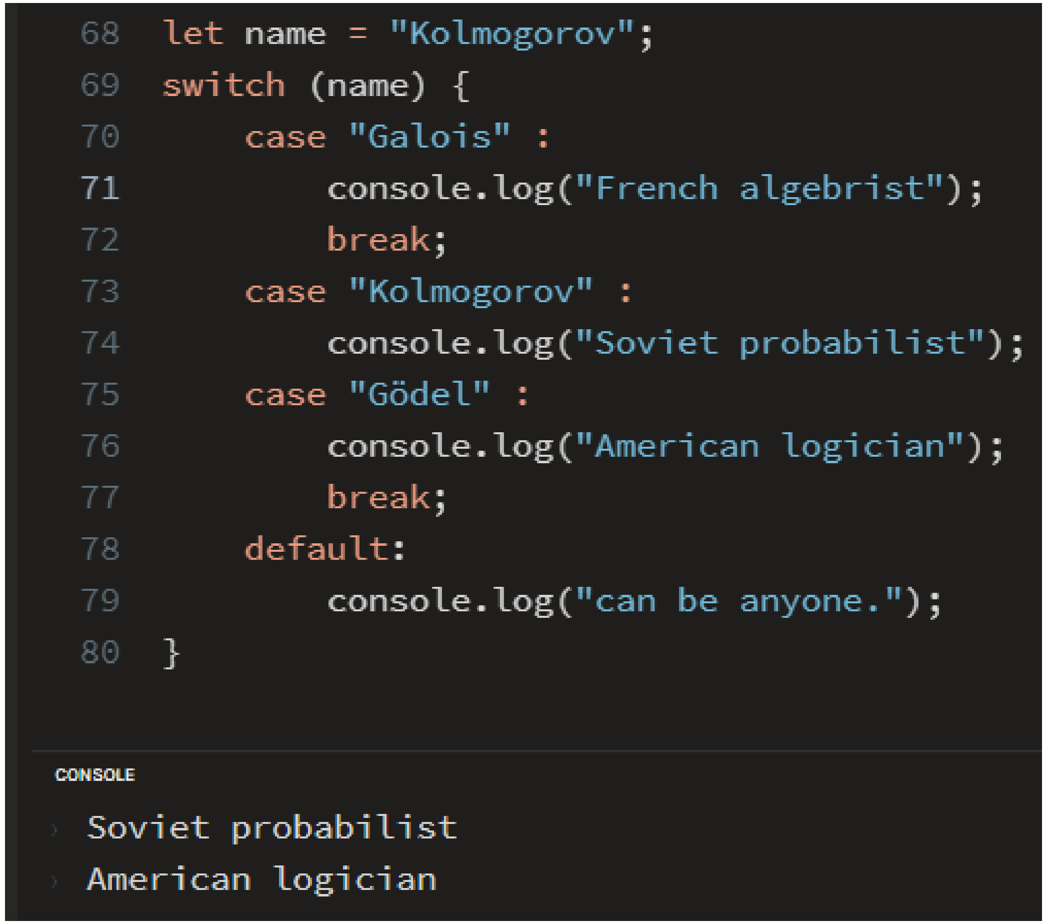


```
let name = "Galois";
switch (name) {
    case "Galois" :
        console.log("French algebrist");
        break;
    case "Kolmogorov" :
        console.log("Soviet probabilist");
        break;
    case "Gödel" :
        console.log("American logician");
        break;
    default:
        console.log("can be anyone.");
}
```

```
French algebrist
```

The switch statement

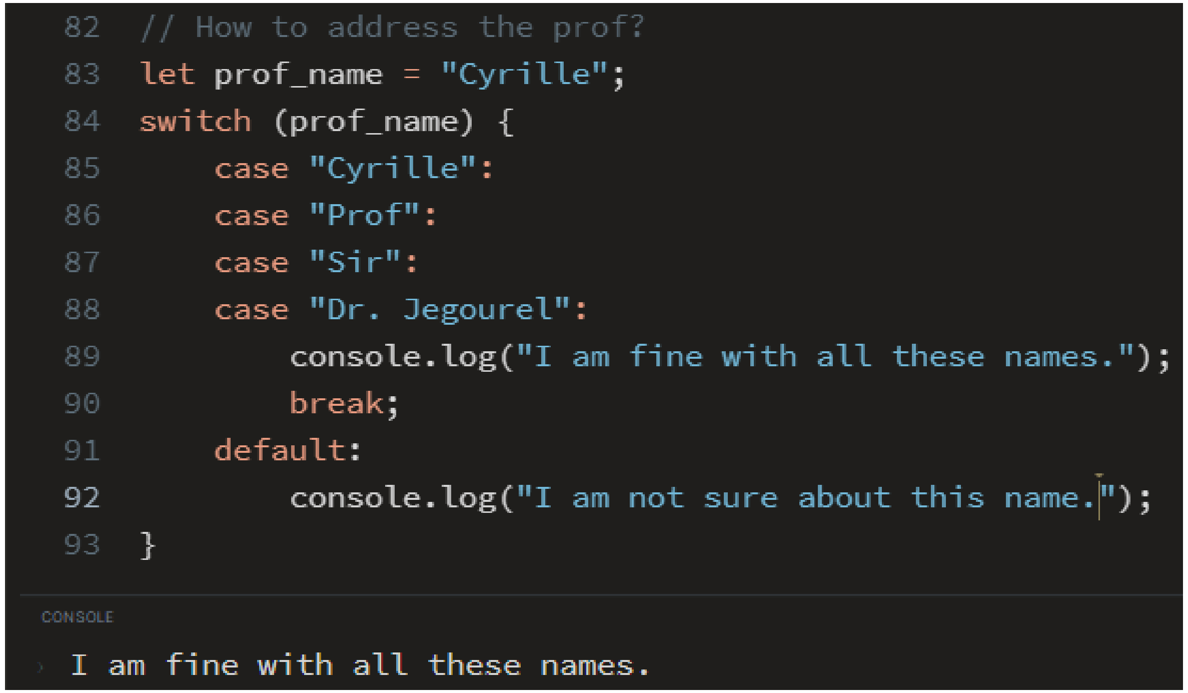
- The switch statement can replace a set of chained if/elseif/else statements.
- Don’t forget the break statement into each case. Otherwise, all the statements in the following cases will be also executed until the end of the switch statement, or a break is reached.



```
let name = "Kolmogorov";
switch (name) {
    case "Galois" :
        console.log("French algebrist");
        break;
    case "Kolmogorov" :
        console.log("Soviet probabilist");
    case "Gödel" :
        console.log("American logician");
        break;
    default:
        console.log("can be anyone.");
}
```

CONSOLE

```
Soviet probabilist
American logician
```



```
// How to address the prof?
let prof_name = "Cyrille";
switch (prof_name) {
    case "Cyrille":
    case "Prof":
    case "Sir":
    case "Dr. Jegourel":
        console.log("I am fine with all these names.");
        break;
    default:
        console.log("I am not sure about this name.");
}
```

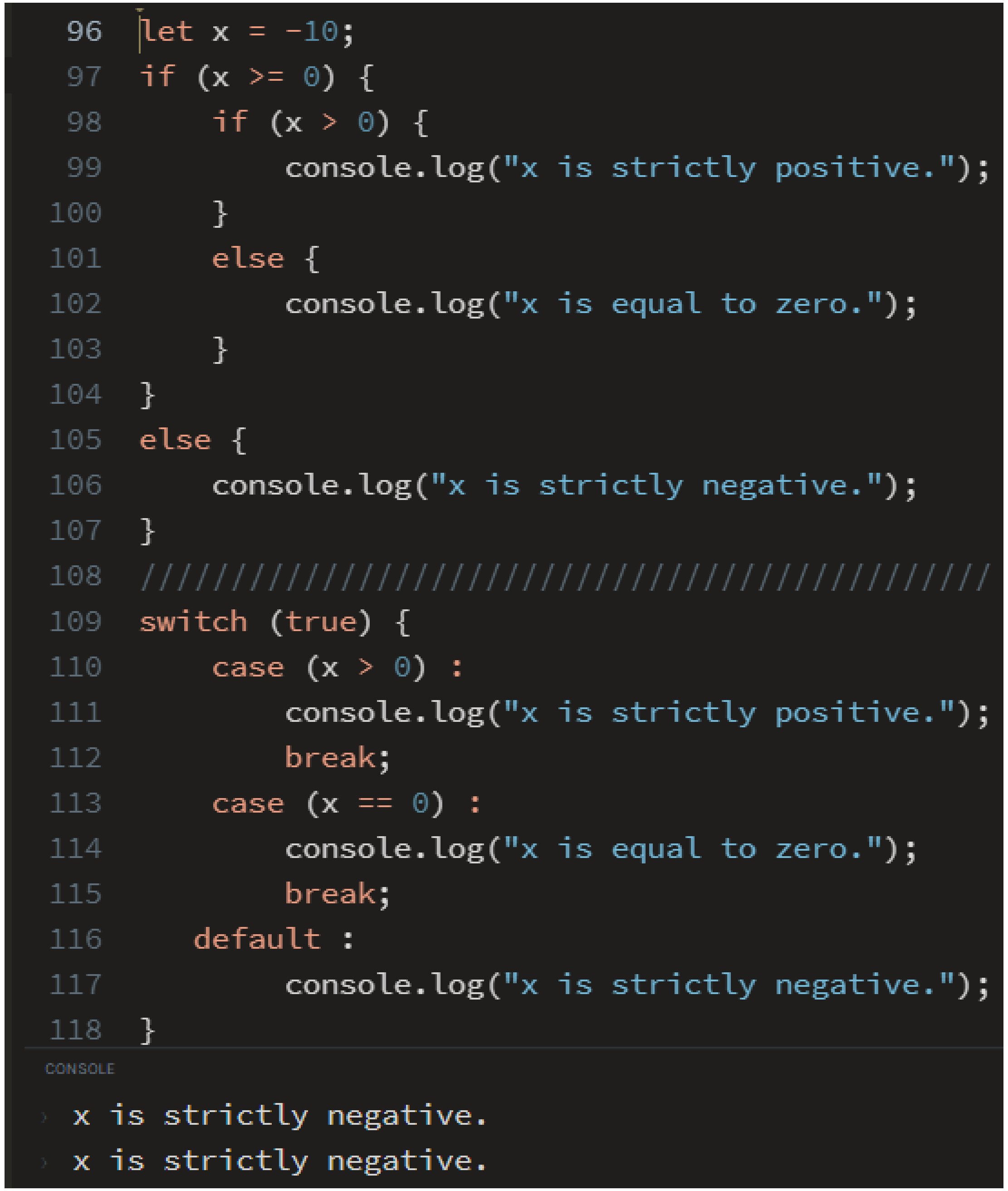
CONSOLE

```
I am fine with all these names.
```

However, this “falling through” technique can be useful when multiple cases are suitable.

The switch statement

- You can also use a Boolean value for the expression.
- Doing so allows to write conditions that return a Boolean value, allowing to test for more than a single value at a time.
- Look at the example: we use the Boolean value **true** as an expression.
- Each case is also a Boolean expression. If this condition is true, it matches with the value of the switch expression and that case will be executed.
- In the example, the default case is executed since (x>0) and (x==0) are **false**.

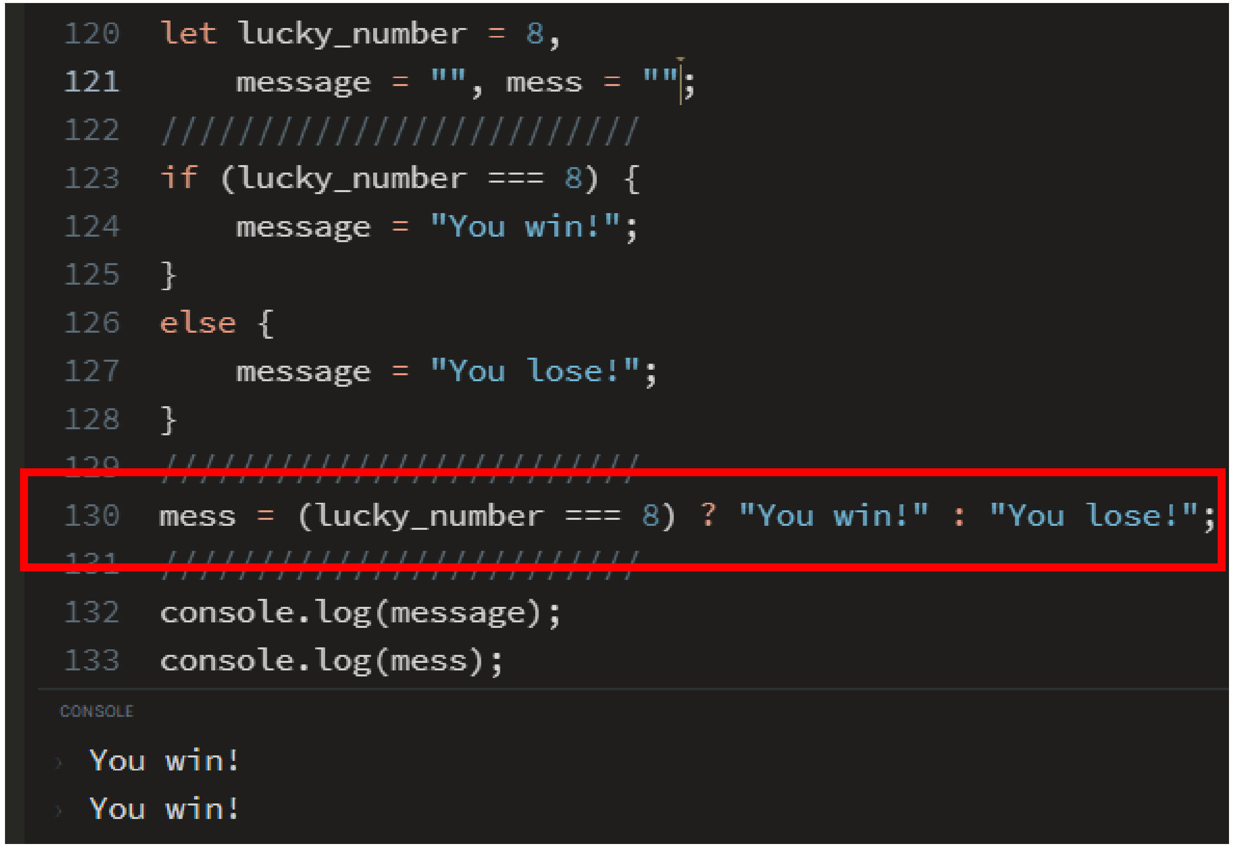


```
let x = -10;
if (x >= 0) {
    if (x > 0) {
        console.log("x is strictly positive.");
    }
    else {
        console.log("x is equal to zero.");
    }
}
else {
    console.log("x is strictly negative.");
}
//////////////////////////////////////////////
switch (true) {
    case (x > 0) :
        console.log("x is strictly positive.");
        break;
    case (x == 0) :
        console.log("x is equal to zero.");
        break;
   default :
        console.log("x is strictly negative.");
}
```

```
x is strictly negative.
x is strictly negative.
```

The conditional operator (*cond* ? *val1* : *val2*)

- The conditional (or ternary) operator can be used in place of simple if/else structure in your code to assign a value to a variable depending on the result of a conditional statement.
- In the example, we show that it can be done in one line instead of 6.
- **Structure**:
- After the assignment operator, write a Boolean condition followed by **?**.
- After the question mark, write two values on each side of a colon symbol **:**.
- JS evaluates the condition and if it returns true, assigns the value at the left side of the colon to the variable. Otherwise, it assigns the value at the right side of the colon to the variable.



```
let lucky_number = 8,
    message = "", mess = "";
/////////////////////////
if (lucky_number === 8) {
    message = "You win!";
}
else {
    message = "You lose!";
}
/////////////////////////
mess = (lucky_number === 8) ? "You win!" : "You lose!";
/////////////////////////
console.log(message);
console.log(mess);
```

CONSOLE

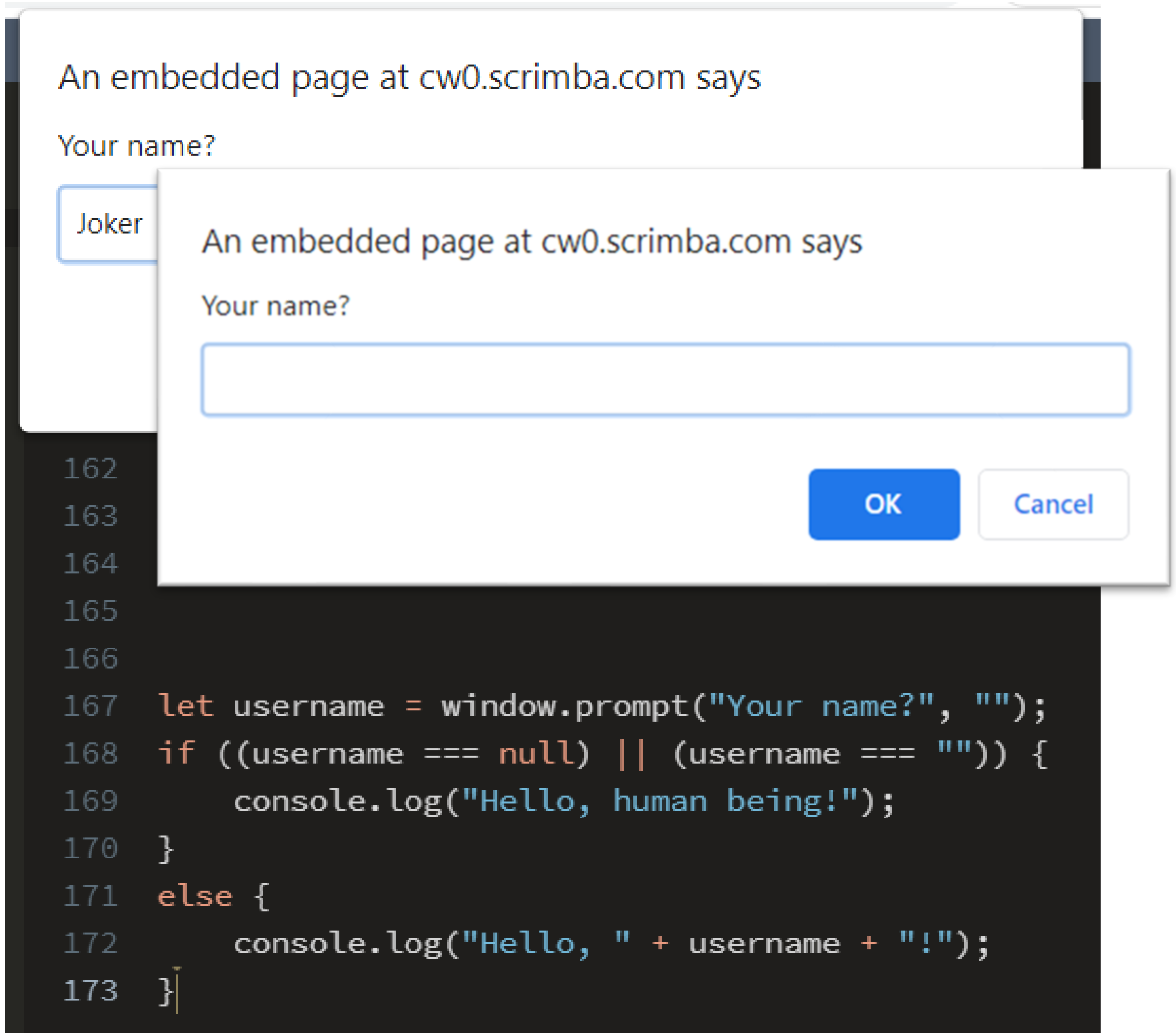
```
You win!
You win!
```

Exercise: Tax income

- Income tax rates on your annual income are charged as follows:
  - If your annual income is lower than 20000 dollars, no tax is charged.
  - Else, the tax rate is 5% on any income earned above 20000 dollars.
- Write a JavaScript function to carry out this calculation. Display the result as follows:
  - “The tax payable on an income of 30000 dollars is 500 dollars.”

User input

- In the following slides, we will use user inputs in the example.
- One way to get a user input is to use the **window.prompt()** method.
- This method takes two arguments: the text to display and the default value for the text box that retrieves user's reply.
- In the example, the prompt box asks for a name. Once the user clicks on OK, CANCEL or presses ENTER, the user value is stored in the username variable.
- The default value is null.
- If/else statements allow to perform basic testing on the user input.



An embedded page at cw0.scrimba.com says

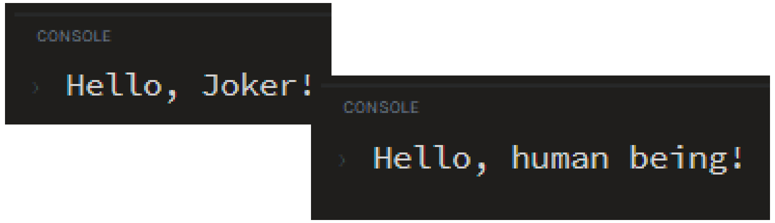
Your name?

Joker

An embedded page at cw0.scrimba.com says

Your name?

```javascript
let username = window.prompt("Your name?", "");
if ((username === null) || (username === "")) {
    console.log("Hello, human being!");
}
else {
    console.log("Hello, " + username + "!");
}
```



CONSOLE

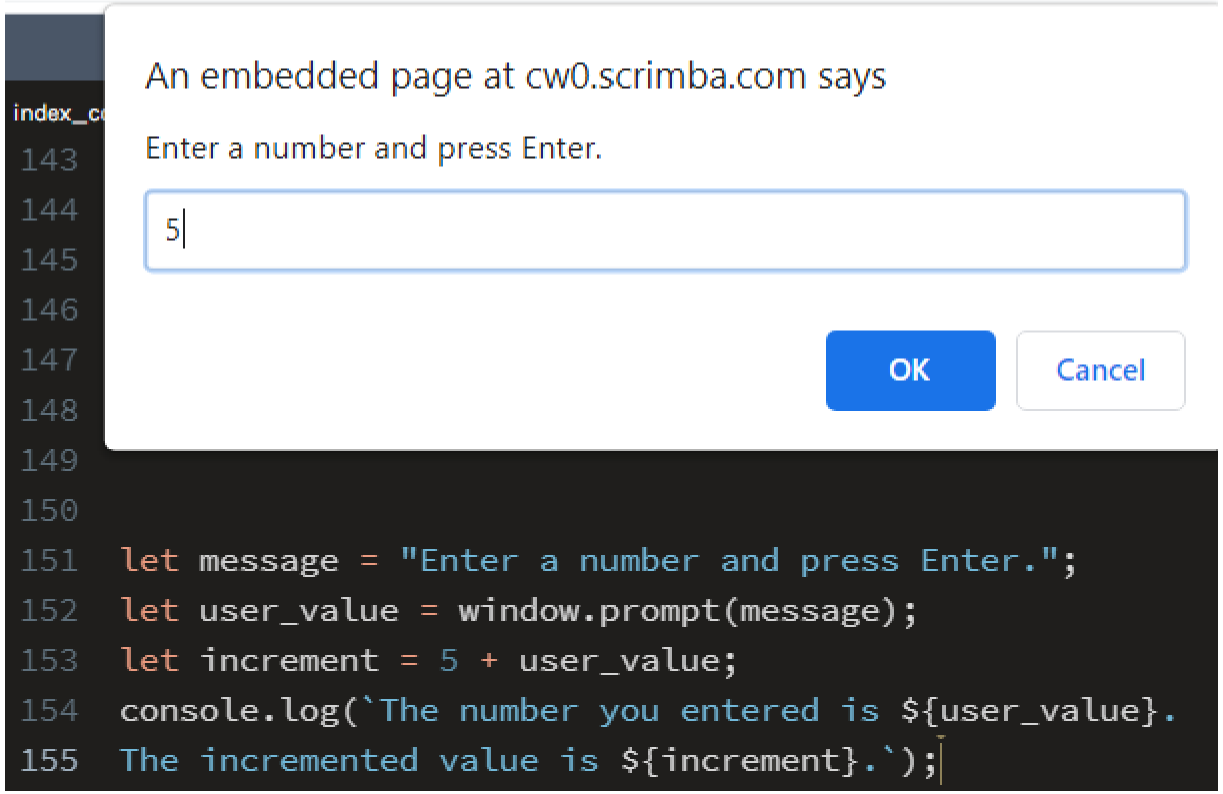
```
Hello, Joker!
```

CONSOLE

```
Hello, human being!
```

Getting values from a user with **prompt()**

- Be careful when getting values with **window.prompt()**!
- In this program,
  1. I asked the user to enter a number,
  2. Increment the value by 5,
  3. And later displayed both the original number and its incremented value.



An embedded page at cw0.scrimba.com says

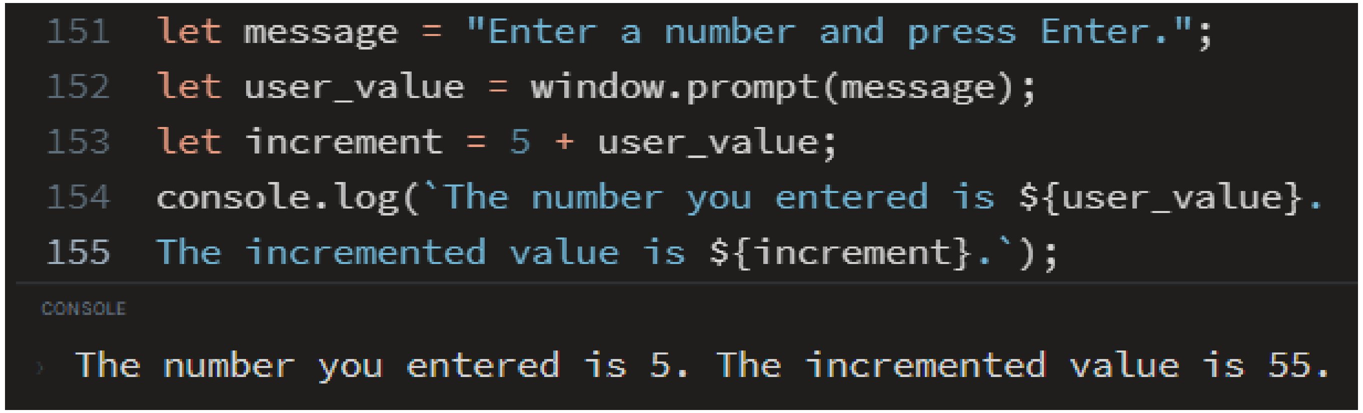
Enter a number and press Enter.

5

```
let message = "Enter a number and press Enter.";
let user_value = window.prompt(message);
let increment = 5 + user_value;
console.log(`The number you entered is ${user_value}.
The incremented value is ${increment}.`);
```

Getting values from a user with **prompt()**

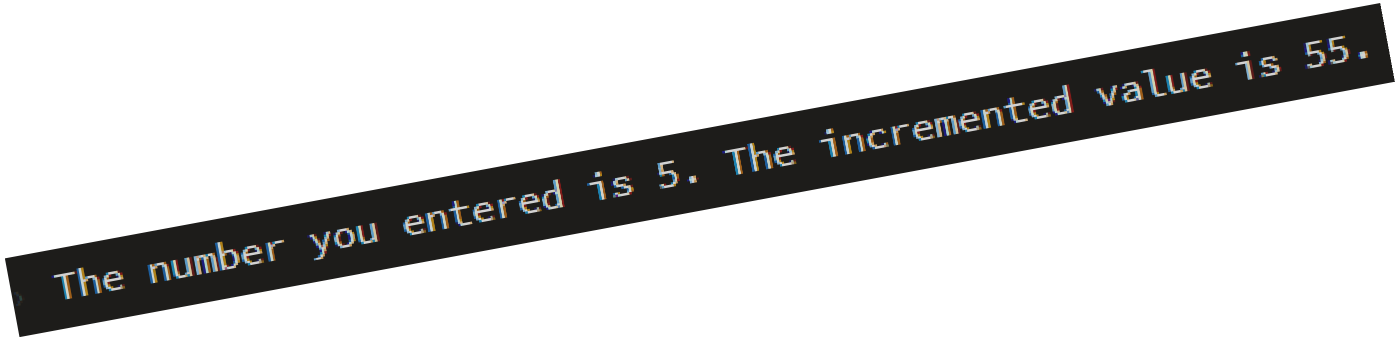
- Be careful when getting values with **window.prompt()**!
- In this program,
  1. I asked the user to enter a number,
  2. Increment the value by 5,
  3. And later displayed both the original number and its doubled value.



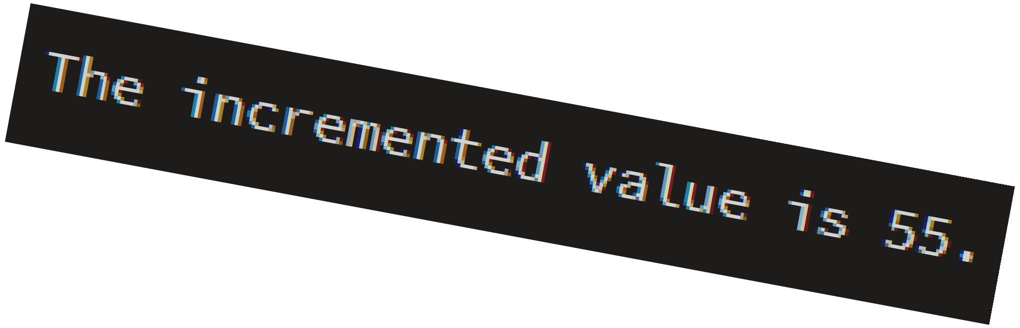
```
let message = "Enter a number and press Enter.";
let user_value = window.prompt(message);
let increment = 5 + user_value;
console.log(`The number you entered is ${user_value}.
The incremented value is ${increment}.`);
```

```
The number you entered is 5. The incremented value is 55.
```

- Something strange occurs…
  But why?

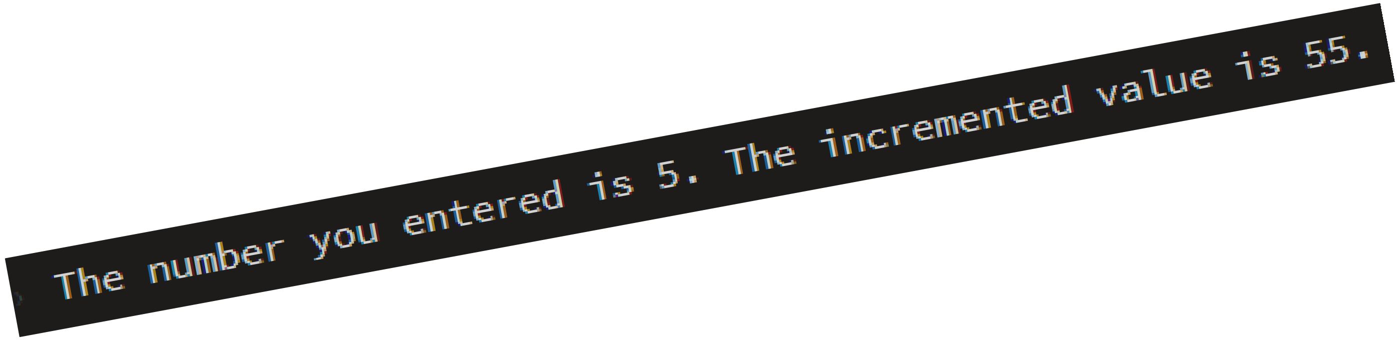


```
The number you entered is 5. The incremented value is 55.
```

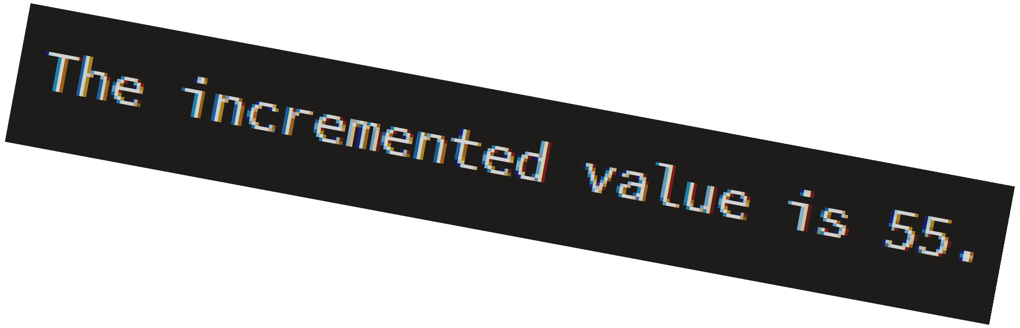


```
The incremented value is 55.
```

Has JavaScript gone mad?



```
The number you entered is 5. The incremented value is 55.
```



```
The incremented value is 55.
```

Has JavaScript gone mad?

Getting values from a user with **prompt()**

- **Important:** The value retrieved from users with **window.prompt()** will always be typed as **string variables**!

Getting values from a user with **prompt()**

- **Important:** The value retrieved from users with **window.prompt()** will always be typed as **string variables**!

- **Remember:** JS performs type coercion. Adding a string “5” with an integer number n simply **concatenates** the string with the integer n coerced into a string!
- Hence, **5 + “5”** is indeed **“55”**. But **2 * “5”** will return **10** because **“5”** is coerced into an integer here.

Getting values from a user with **prompt()**

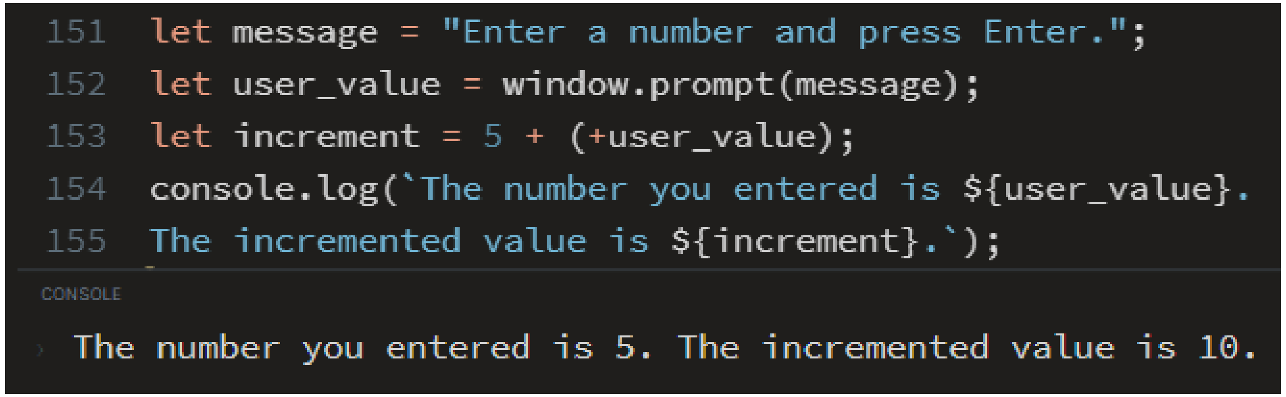
- **Important:** The value retrieved from users with **window.prompt()** will always be typed as **string variables**!

- **Remember:** JS performs type coercion. Adding a string “5” with an integer number n simply **concatenates** the string with the integer n coerced into a string!
- Hence, **5 + “5”** is indeed **“55”**. But **2 * “5”** will return **10** because **“5”** is coerced into an integer here.

- **We need to convert our string variable to a number variable before doing any math!**

Getting values from a user with **prompt()**

- Don’t forget to use the unary operator (+) to convert a string into a number.



```
let message = "Enter a number and press Enter.";
let user_value = window.prompt(message);
let increment = 5 + (+user_value);
console.log(`The number you entered is ${user_value}.
The incremented value is ${increment}.`);
```

```
The number you entered is 5. The incremented value is 10.
```

Practice activities for if/else if/else

Let us practice the if/else if/else concepts a bit, with an activity.

Activity 1 – Ask for user’s age

Activity 1 – Ask for user’s age

Write a function **ask_user_age()**, as described below.

- It **receives no parameters** and **returns no parameters**.
- It first **asks for the user to input its age**, and retrieves the info from the user.
- **If the age is negative** (0 included), the function should **print** a message that reads "Your age cannot be negative, it must be at least 1."
- **If the age given by the user is larger than 122** (oldest person on record, Jeanne Calment), then you should display "I really doubt you are ___ years old..." with the blank filled accordingly.
- **Otherwise**, the function should print "Oh, you are **___** years old? That's cool!", with the **blank filled** accordingly.

The while statement

The **while** statement is another type of **conditional structure.**

The while statement

The **while** statement is another type of **conditional structure.**

The **if** statement is the simplest **conditional structure**.

- **How it works:**
  - If the Boolean condition specified for the **if** statement is **true**, then execute the block of code inside the **if** statement.
  - If the Boolean condition is not **true**, ignore the block of code in the **if** statement.
  - Once we are done executing the code in **if** (or ignoring it), move on to the next (non-indented) line.

The while statement

The **while** statement is another type of **conditional structure.**

- **How it works:**
  - If the Boolean condition specified for the **while** statement is **true**, then execute the block of code inside the **while** statement.
  - If the Boolean condition **false**, ignore the block of code in the **while** statement.

The **if** statement is the simplest **conditional structure**.

- **How it works:**
  - If the Boolean condition specified for the **if** statement is **true**, then execute the block of code inside the **if** statement.
  - If the Boolean condition is **false**, ignore the block of code in the **if** statement.
  - Once we are done executing the code in **if** (or ignoring it), move on to the next (non-indented) line.

The **while** statement

The **while** statement is another type of **conditional structure.**

- **How it works:**
  - If the Boolean condition specified for the **while** statement is **true**, then execute the block of code inside the **while** statement.
  - If the Boolean condition is **false**, ignore the block of code in the **while** statement.
  - Once we are done executing the code in **while**, **move back to the while statement, and repeat until the condition is no longer true.**

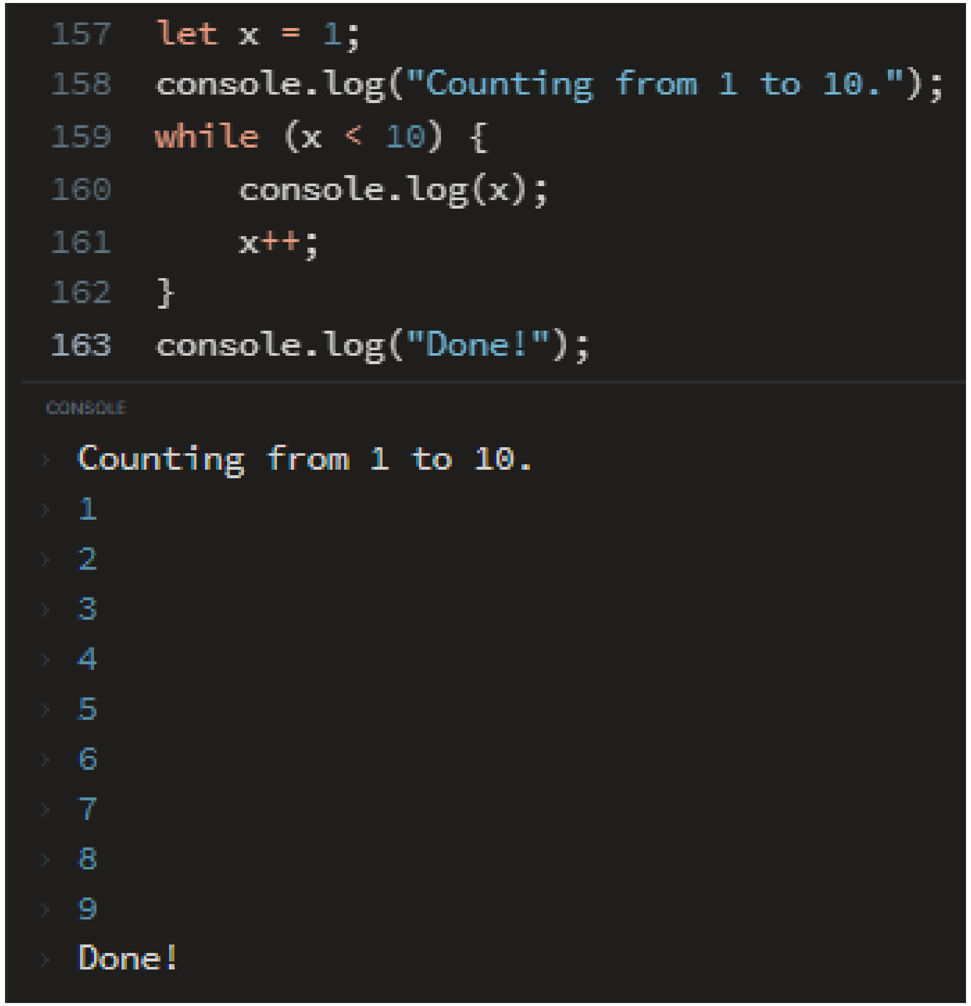
The **if** statement is the simplest **conditional structure**.

- **How it works:**
  - If the Boolean condition specified for the **if** statement is **true**, then execute the block of code inside the **if** statement.
  - If the Boolean condition is **false**, ignore the block of code in the **if** statement.
  - Once we are done executing the code in **if** (or ignoring it), **move on to the next (non-indented) line.**

The while statement

The **while** statement is another type of **conditional structure.**

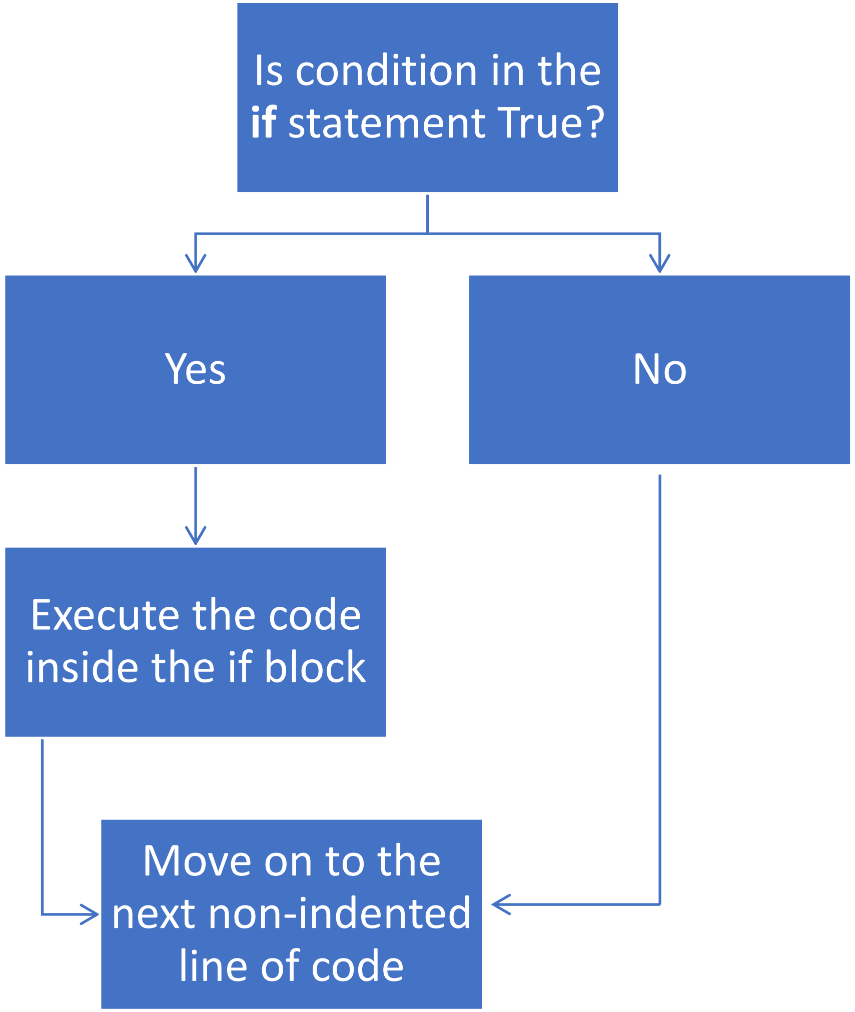
- **How it works:**
  - If the Boolean condition specified for the **while** statement is **true**, then execute the block of code inside the **while** statement.
  - If the Boolean condition is **false**, ignore the block of code in the **while** statement.
  - Once we are done executing the code in **while**, **move back to the while statement, and repeat until the condition is no longer True.**



```
let x = 1;
console.log("Counting from 1 to 10.");
while (x < 10) {
      console.log(x);
      x++;
}
console.log("Done!");
```

```
Counting from 1 to 10.
1
2
3
4
5
6
7
8
9
Done!
```

Architectures: if vs. while



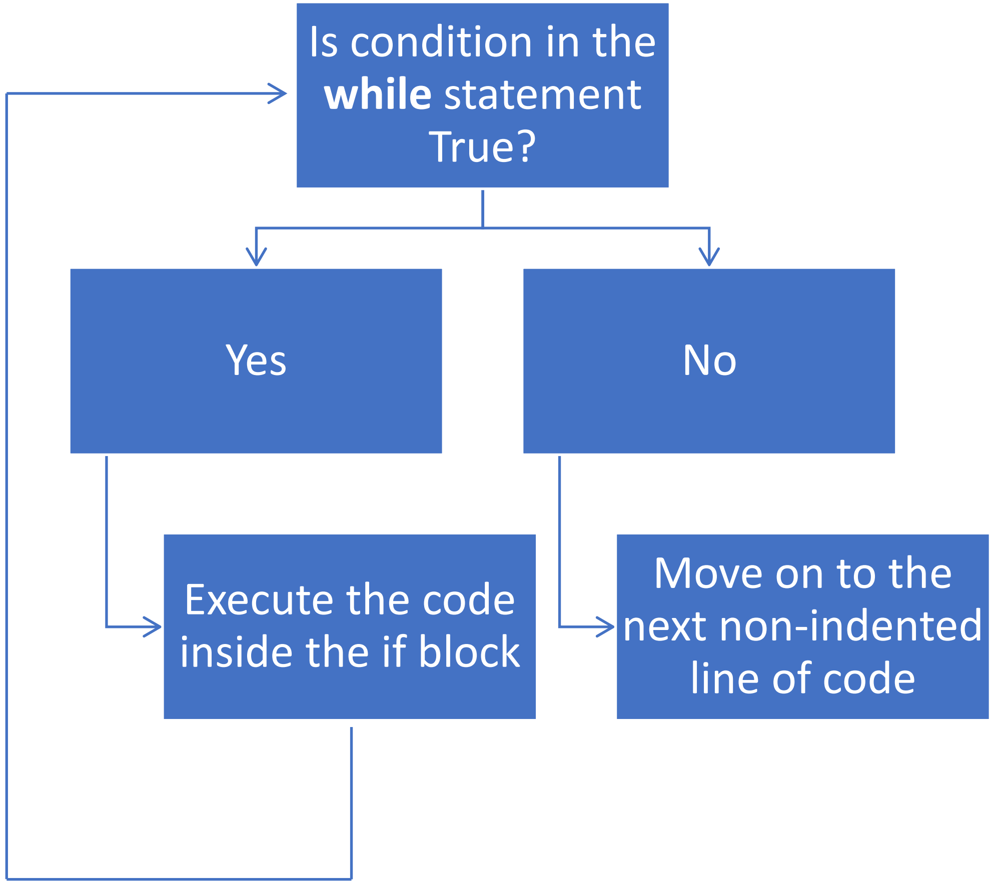
Is condition in the **if** statement True?

Yes

No

Execute the code inside the if block

Move on to the next non-indented line of code



Is condition in the **while** statement True?

Yes

No

Execute the code inside the if block

Move on to the next non-indented line of code

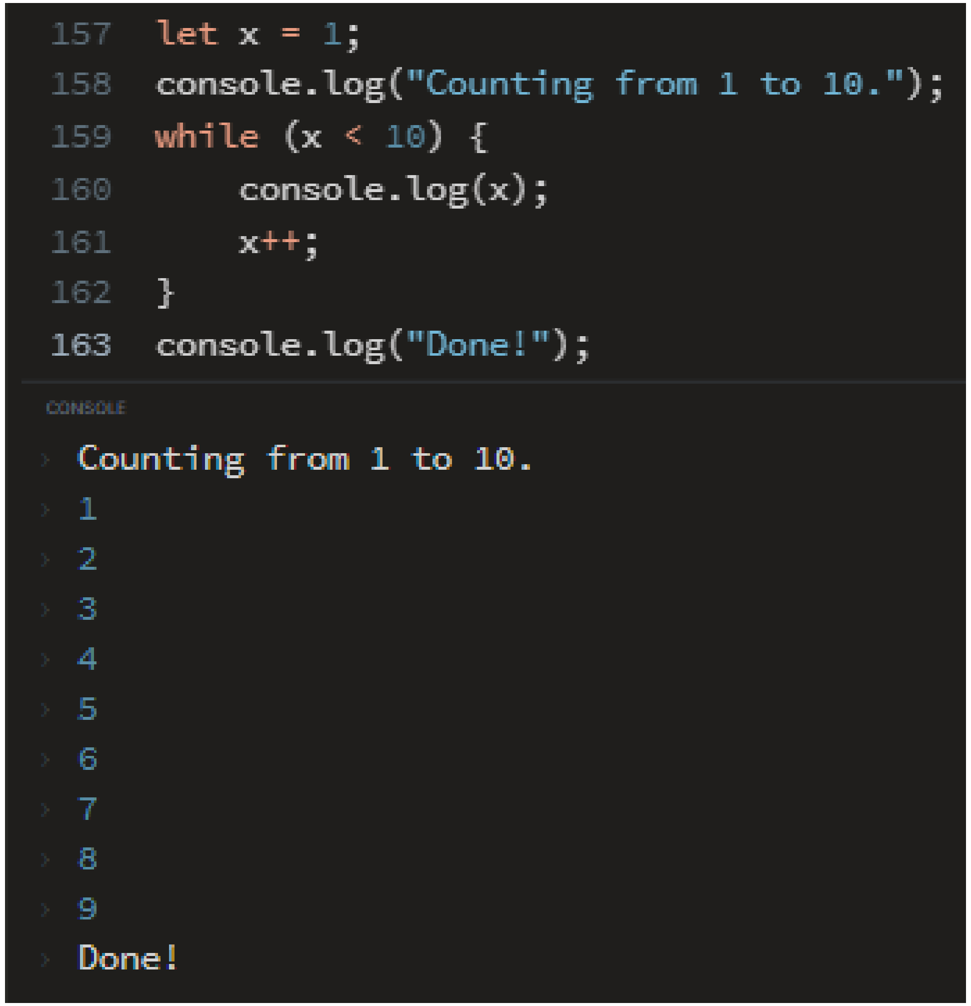
Infinite loops

The **while** statement repeats a condition until it is no longer **True**.

Infinite loops

The **while** statement repeats a condition until it is no longer **true**.

This means that there should be a clear process that **makes your condition no longer true**, at some point.



```
let x = 1;
console.log("Counting from 1 to 10.");
while (x < 10) {
    console.log(x);
    x++;
}
console.log("Done!");
```

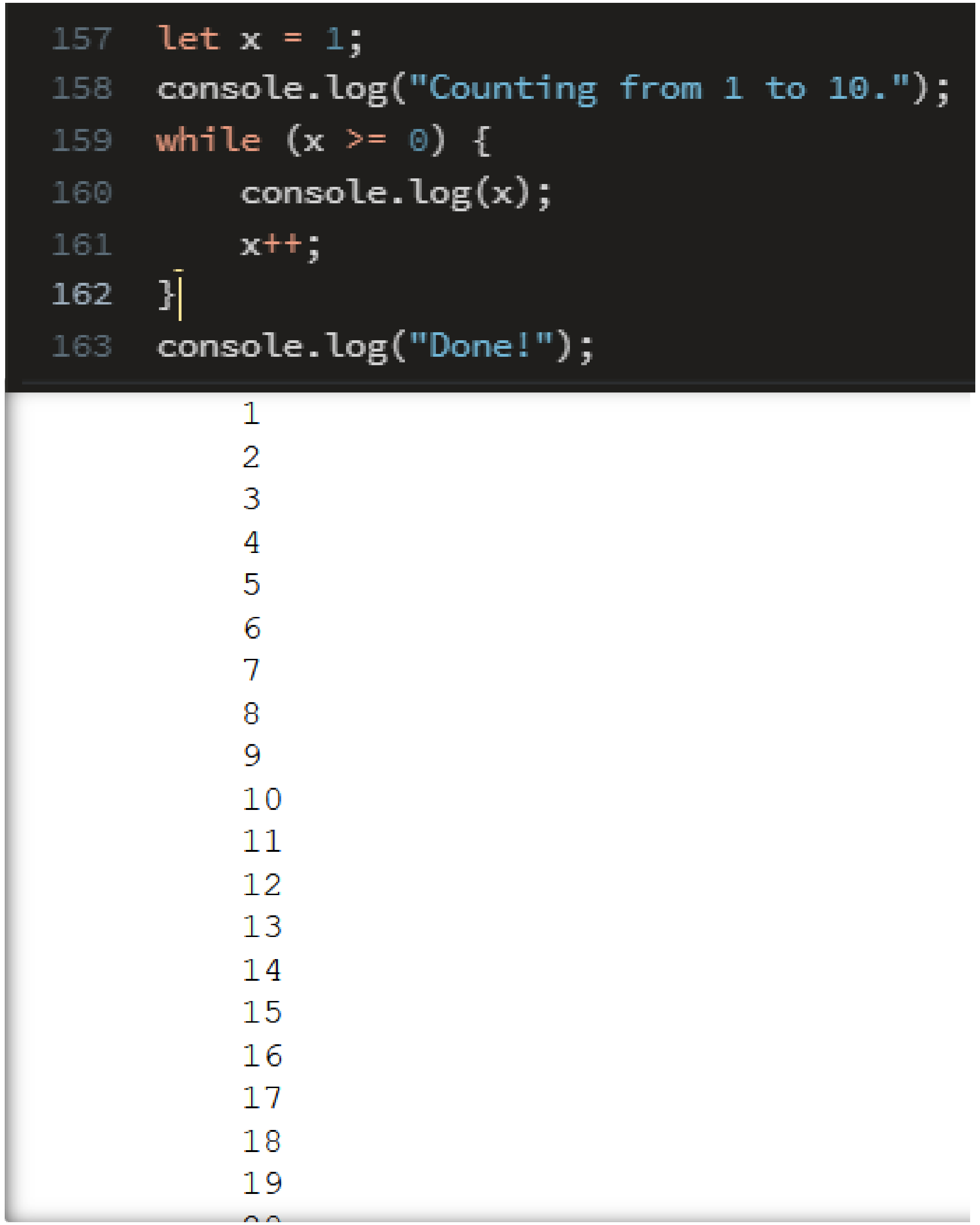
```
Counting from 1 to 10.
1
2
3
4
5
6
7
8
9
Done!
```

Infinite loops

The **while** statement repeats a condition until it is no longer **true**.

This means that there should be a clear process that **makes your condition no longer true**, at some point.

Otherwise, the **while** block will keep on repeating indefinitely… This is called an **infinite loop**.



```
let x = 1;
console.log("Counting from 1 to 10.");
while (x >= 0) {
    console.log(x);
    x++;
}
console.log("Done!");
```

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
```

Infinite loops and how to kill them

Infinite loops will keep on executing forever, unless

1. Your computer runs out of resources (bad thing to do),

Infinite loops and how to kill them

Infinite loops will keep on executing forever, unless

1. ~~Your computer runs out of resources (bad thing to do),~~

Infinite loops and how to kill them

Infinite loops will keep on executing forever, unless

1. You decide to crash the program on purpose and kill the loop manually.

Infinite loops and how to kill them

Infinite loops will keep on executing forever, unless

1. You decide to crash the program on purpose and kill the loop manually.

This is called a **keyboard interrupt.** It is done with **CTRL+C** (or **CMD+C** on mac), if in console mode and most IDEs.

Infinite loops and how to kill them

Infinite loops will keep on executing forever, unless

1. You decide to crash the program on purpose and kill the loop manually.

This is called a **keyboard interrupt.** It is done with **CTRL+C** (or **CMD+C** on mac), if in console mode and most IDEs.

Great advice #7

Great Advice #7: Avoid the infinite loops and dead code, by drawing structural diagrams.

Infinite loops and **dead code**, unless created on purpose, usually follow from a **poor design** in your code.

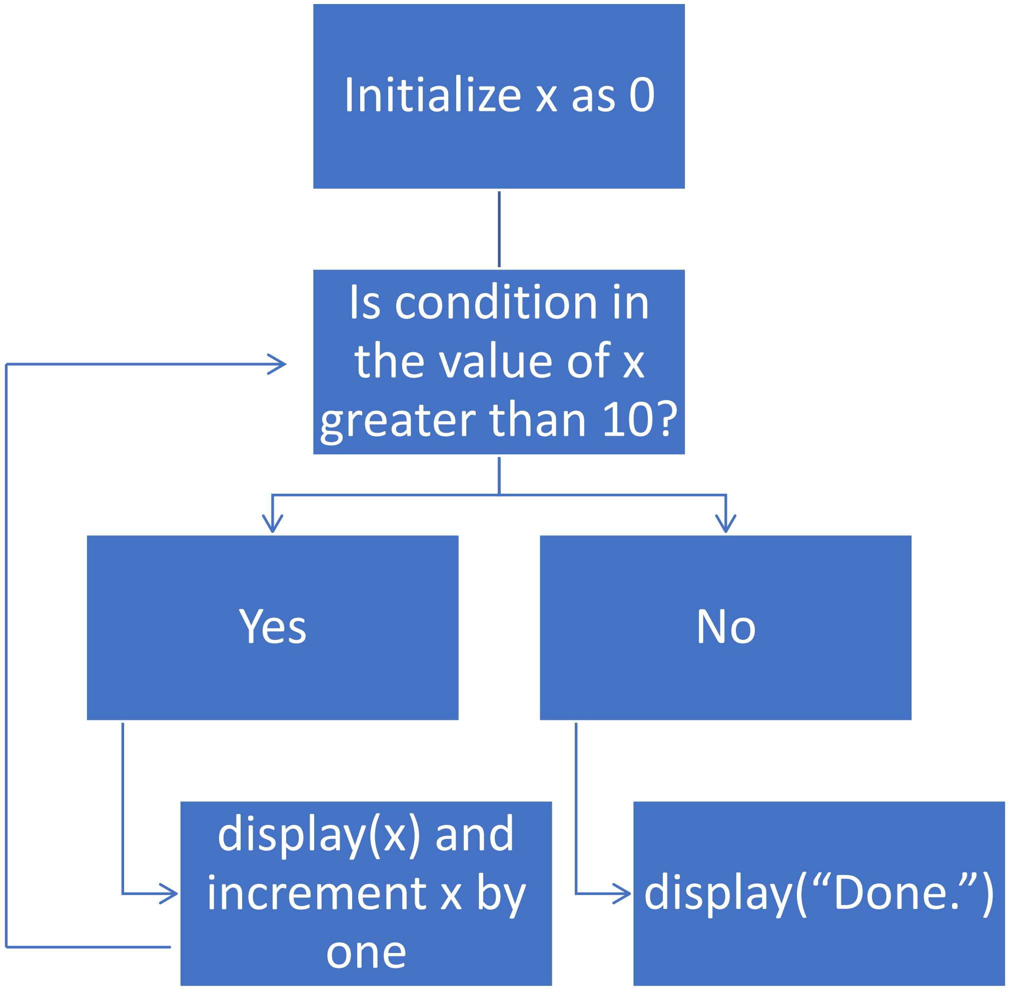
Drawing a **structural diagram**, **before coding**, greatly helps figuring out the right structure for your code.

Great advice #7

Great Advice #7: Avoid the infinite loops and dead code, by drawing structural diagrams.

Infinite loops and **dead code**, unless created on purpose, usually follow from a **poor design** in your code.

Drawing a **structural diagram**, **<u>before coding</u>**, greatly helps figuring out the right structure for your code.


Initialize x as 0
Is condition in the value of x greater than 10?
Yes
No
display(x) and increment x by one
display("Done.")

Example: diagram for our while loop, counting from 1 to 10.

Infinite loops and how to kill them

Infinite loops will keep on executing forever, unless

1. You decide to crash the program on purpose and kill the loop manually.

Infinite loops: the break statement

Infinite loops will keep on executing forever, unless

2. You use a **break** statement.

Infinite loops: the break statement

Infinite loops will keep on executing forever, unless

2. You use a **break** statement.

When encountered, the **break** statement will immediately end the current **while** loop.

Infinite loops: the break statement

Infinite loops will keep on executing forever, unless

2. You use a **break** statement.

When encountered, the **break** statement will immediately end the current **while** loop.

The code then resumes its execution with the next line outside of the **while** block.

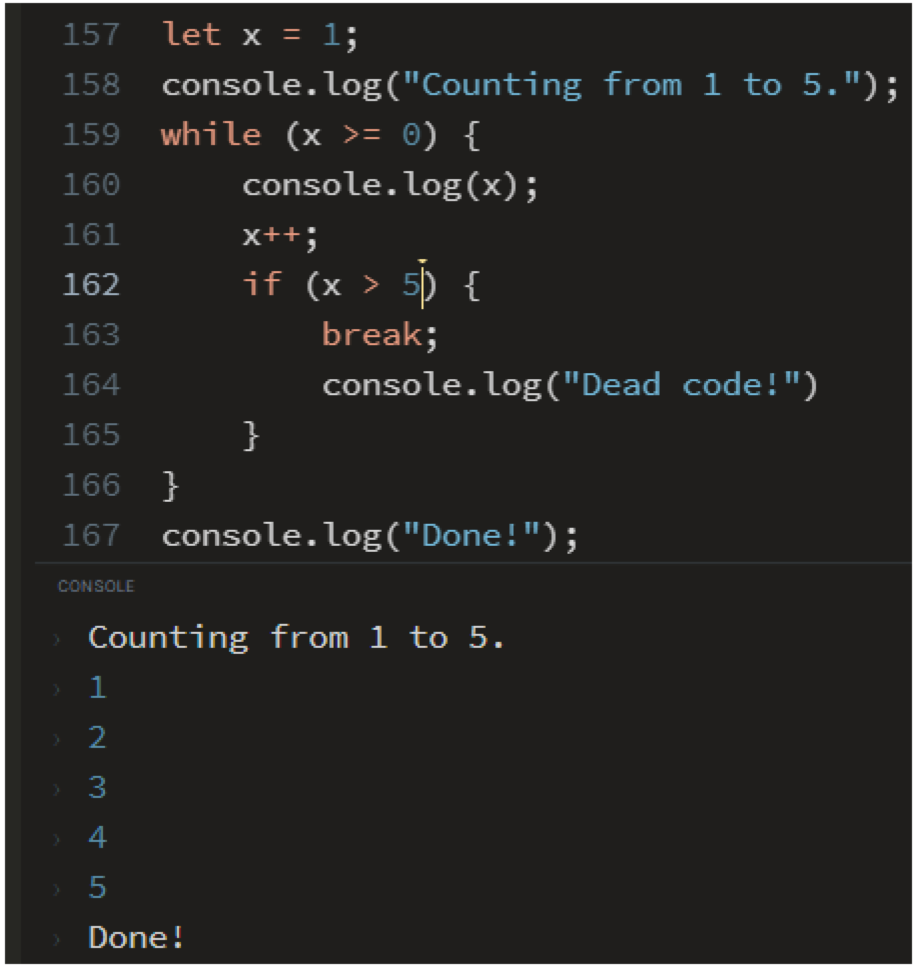
Infinite loops: the **break** statement

Infinite loops will keep on executing forever, unless

2. You use a **break** statement.

When encountered, the **break** statement will immediately end the current **while** loop.

The code then resumes its execution with the next line outside of the **while** block.



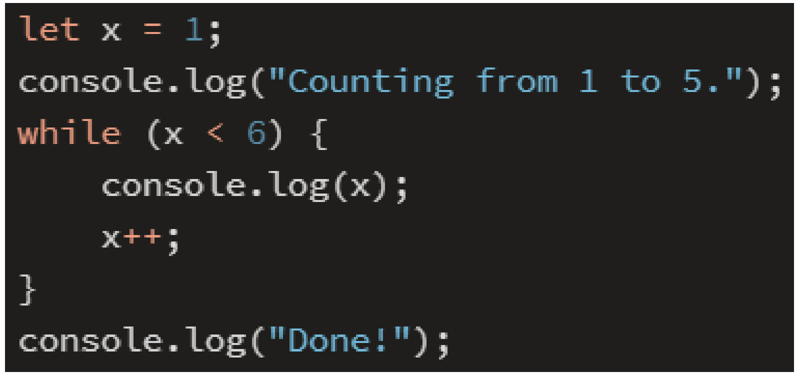
```
let x = 1;
console.log("Counting from 1 to 5.");
while (x >= 0) {
    console.log(x);
    x++;
    if (x > 5) {
        break;
        console.log("Dead code!")
    }
}
console.log("Done!");
```

CONSOLE

```
Counting from 1 to 5.
1
2
3
4
5
Done!
```

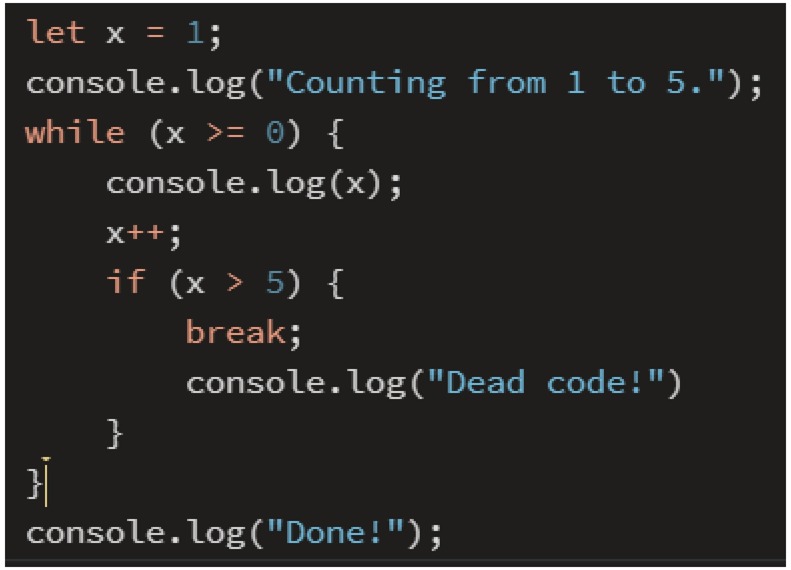
Standard **while** vs. infinite **while** + **break**

1. Standard **while** loop with condition in the while statement.



```
let x = 1;
console.log("Counting from 1 to 5.");
while (x < 6) {
    console.log(x);
    x++;
}
console.log("Done!");
```

2. Infinite **while** loop with condition in an **if** statement, and **break** in the **if** block.



```
let x = 1;
console.log("Counting from 1 to 5.");
while (x >= 0) {
    console.log(x);
    x++;
    if (x > 5) {
        break;
        console.log("Dead code!")
    }
}
console.log("Done!");
```

→ Both loops work and do the job, which one is better though?

Great advice #8

Great Advice #8: Avoid the infinite loops, if possible.

Relying on an **infinite** **while** loop with a **break** is **risky**, and should be avoided when possible.

It is often easily avoided, by using the Boolean expression of the **if** statement used for **break**, as the condition in the **while** statement.

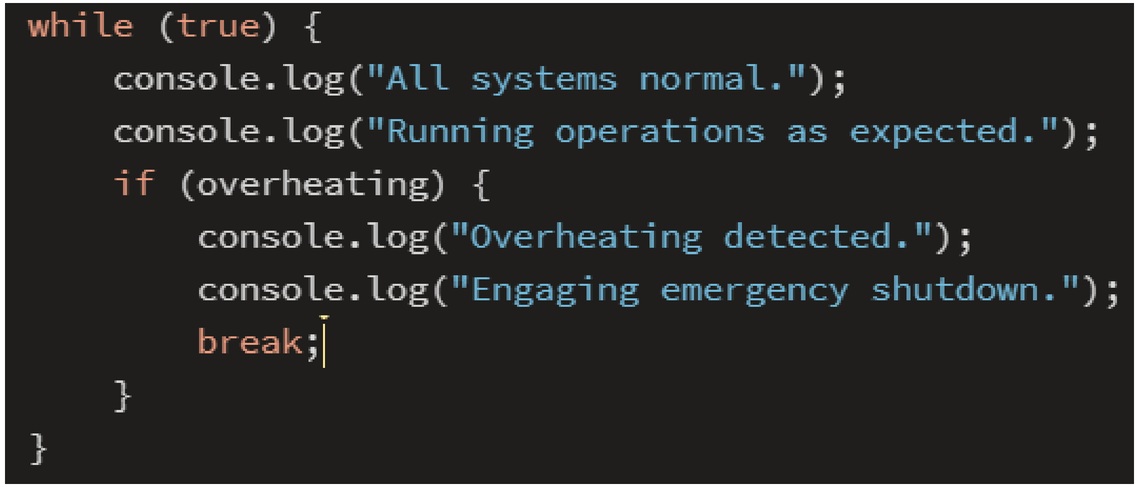
Great advice #8

Great Advice #8: Avoid the infinite loops, if possible.

Relying on an **infinite while** loop with a **break** is **risky** and should be avoided when possible.

It is often easily avoided, by using the Boolean expression of the **if** statement used for **break**, as the condition in the **while** statement.

Note: a few cases, however, require the use of a **break** statement.
For instance, **emergency shutdowns**.



```
while (true) {
    console.log("All systems normal.");
    console.log("Running operations as expected.");
    if (overheating) {
        console.log("Overheating detected.");
        console.log("Engaging emergency shutdown.");
        break;
    }
}
```

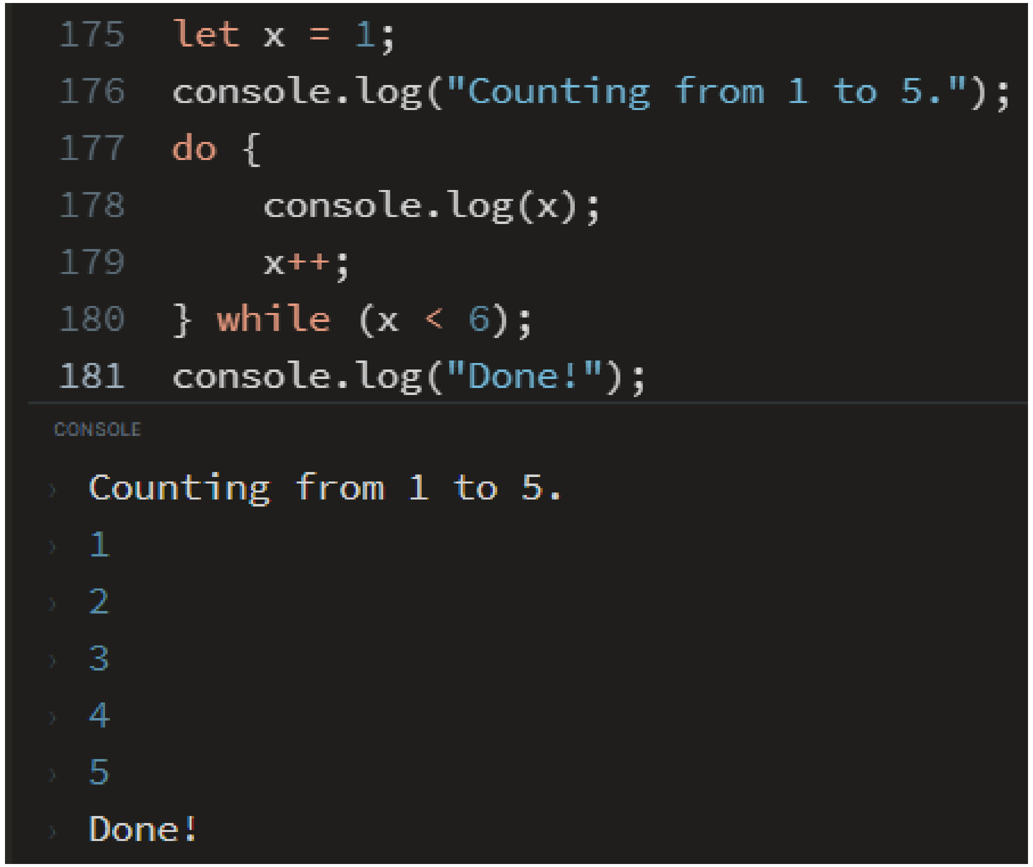
Activity: Collatz or Syracuse conjecture

- Let n be an integer >= 1. If n is even, we divide n by 2, otherwise we multiply n by 3 and afterwards, we add 1.
- We repeat this process until n = 1.
- Write a function that, given n, displays on-the-fly the values of this sequence until we reach 1. For example, starting with n = 10, we should display on the screen the values: 10, 5, 16, 8, 4, 2, 1.
- You might want to add a break statement in case the function is taking too much time to terminate.

The do-while statement

The **do-while** statement is similar to a while except that the Boolean comparison is checked each time afterward to determine whether or not it should repeat.

- **How it works:**
  - The loop is introduced by the keyword **do** and is guaranteed to be executed at least once.
  - After the **do** block of code, if the Boolean condition specified after the **while** keyword is **true**, then the block of code inside the **do** statement is repeated.
  - If the Boolean condition is **false**, we exist the **do-while** statement.
  - The procedure is repeated until the while comparison is no longer **true**.



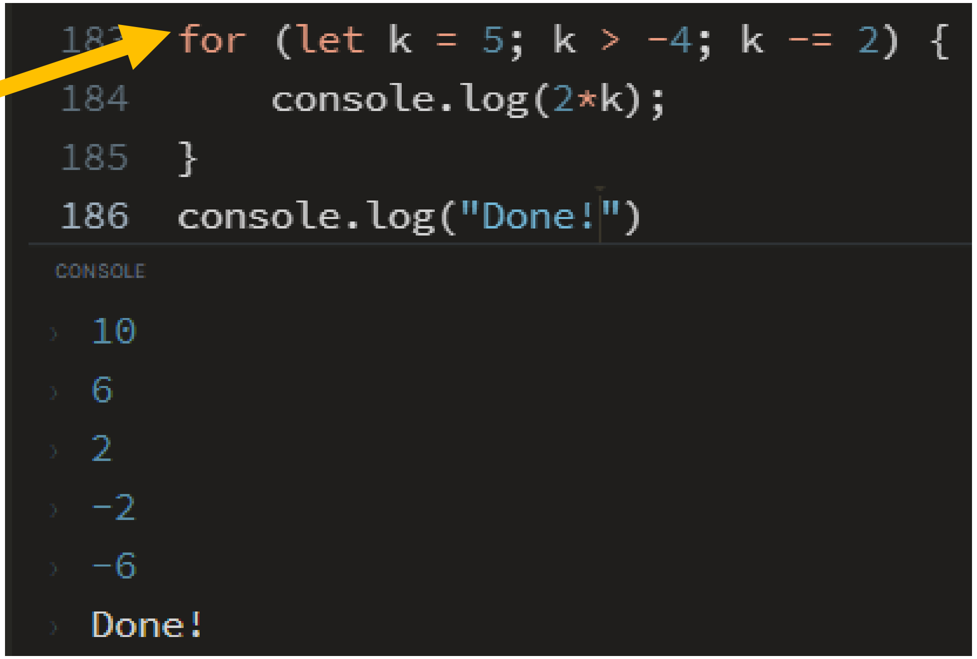
```
let x = 1;
console.log("Counting from 1 to 5.");
do {
    console.log(x);
    x++;
} while (x < 6);
console.log("Done!");
```

```
Counting from 1 to 5.
1
2
3
4
5
Done!
```

The for loop

The **for** loop is another to repeat a block of instructions.

- **Structure:**
  - Use the keyword **for**,
  - And immediately after, a set of parenthesis with 3 statements inside, separated by **;**.
  - The 1st statement declares (with **let**) a variable and assigns it an initial value. This initial value can be any value and is used as a starting point for the number of times the loop will repeat (e.g., **let k = 5**).
  - The 2nd statement is a Boolean expression that tells the loop to stop when it is no longer true (e.g., the loop is taken until **k > -4** becomes false).
  - The 3rd statement determines at which rate the loop variable is updated. (e.g., **k -= 2**).
  - Afterwards, enclose a block of instructions within curly brackets to be executed each time the loop is taken.



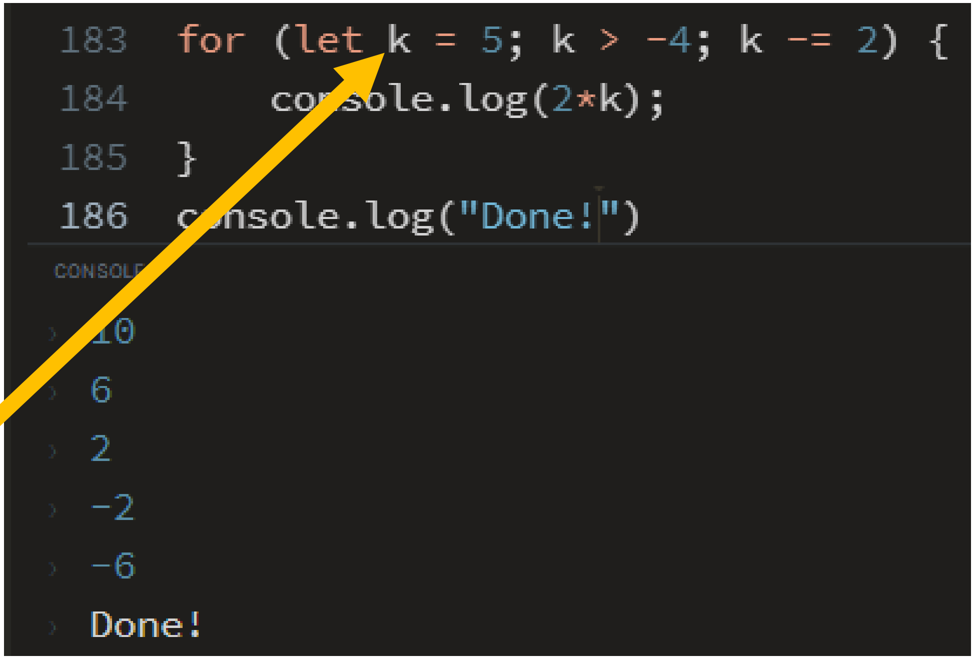
```
for (let k = 5; k > -4; k -= 2) {
    console.log(2*k);
}
console.log("Done!")
```

```
10
6
2
-2
-6
Done!
```

The for loop

The for loop is another to repeat a block of instructions.

- **Structure:**
  - Use the keyword for,
  - And immediately after, a set of parenthesis with 3 statements inside, separated by ;.
  - The 1st statement declares (with let) a loop variable and assigns it an initial value. This initial value can be any value and is used as a starting point for the number of times the loop will repeat (e.g., **let k = 5**).
  - The 2nd statement is a Boolean expression that tells the loop to stop when it is no longer true (e.g., the loop is taken until **k > -4** becomes false).
  - The 3rd statement determines at which rate the loop variable is updated. (e.g., **k -= 2**).
  - Afterwards, enclose a block of instructions within curly brackets to be executed each time the loop is taken.



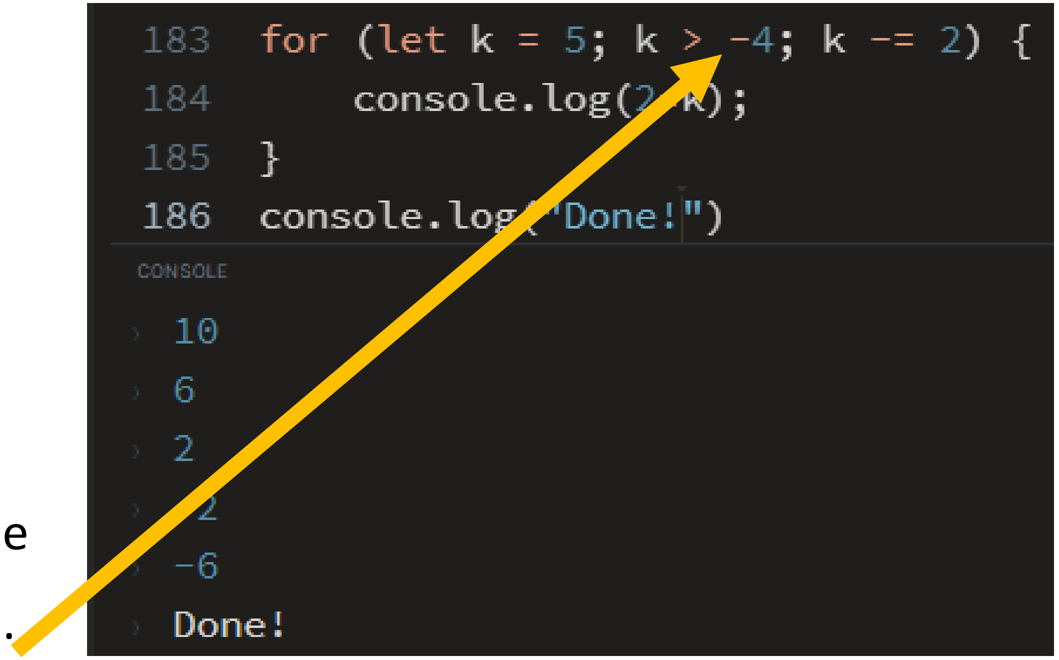
```
for (let k = 5; k > -4; k -= 2) {
    console.log(2*k);
}
console.log("Done!")
```

```
10
6
2
-2
-6
Done!
```

The for loop

The for loop is another to repeat a block of instructions.

- **Structure:**
  - Use the keyword for,
  - And immediately after, a set of parenthesis with 3 statements inside, separated by ;.
  - The 1st statement declares (with let) a loop variable and assigns it an initial value. This initial value can be any value and is used as a starting point for the number of times the loop will repeat (e.g., **let k = 5**).
  - The 2nd statement is a Boolean expression that tells the loop to stop when it is no longer true (e.g., the loop is taken until **k > -4** becomes false).
  - The 3rd statement determines at which rate the loop variable is updated. (e.g., **k -= 2**).
  - Afterwards, enclose a block of instructions within curly brackets to be executed each time the loop is taken.



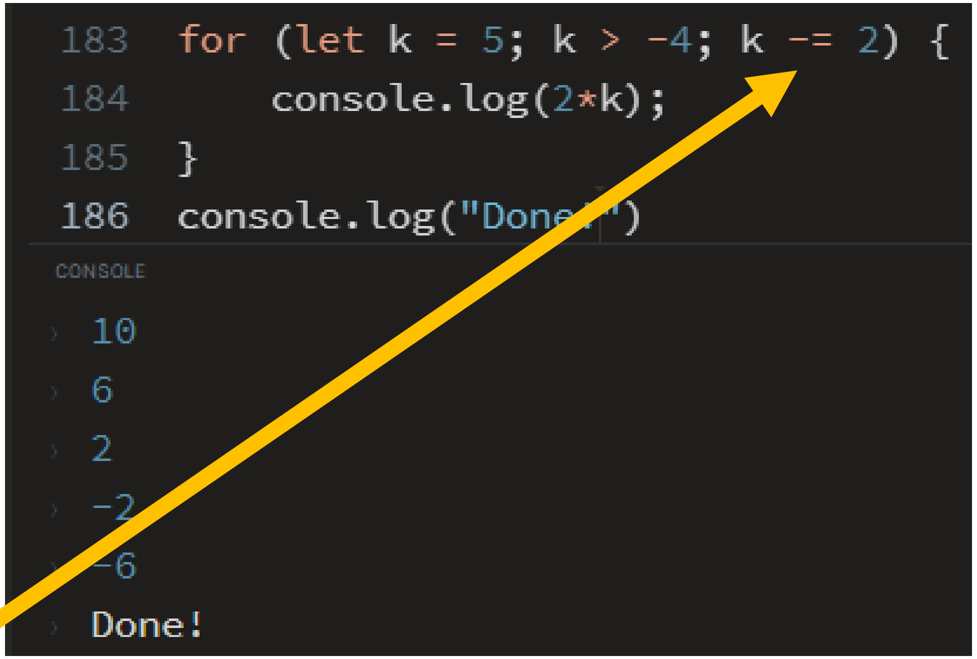
```
for (let k = 5; k > -4; k -= 2) {
    console.log(2 * k);
}
console.log("Done!")
```

```
10
6
2
-2
-6
Done!
```

The for loop

The **for** loop is another to repeat a block of instructions.

- **Structure:**
  - Use the keyword **for**,
  - And immediately after, a set of parenthesis with 3 statements inside, separated by **;**.
  - The 1st statement declares (with **let**) a loop variable and assigns it an initial value. This initial value can be any value and is used as a starting point for the number of times the loop will repeat (e.g., **let k = 5**).
  - The 2nd statement is a Boolean expression that tells the loop to stop when it is no longer true (e.g., the loop is taken until **k > -4** becomes false).
  - The 3rd statement determines at which rate the loop variable is updated. (e.g., **k -= 2**).
  - Afterwards, enclose a block of instructions within curly brackets to be executed each time the loop is taken.



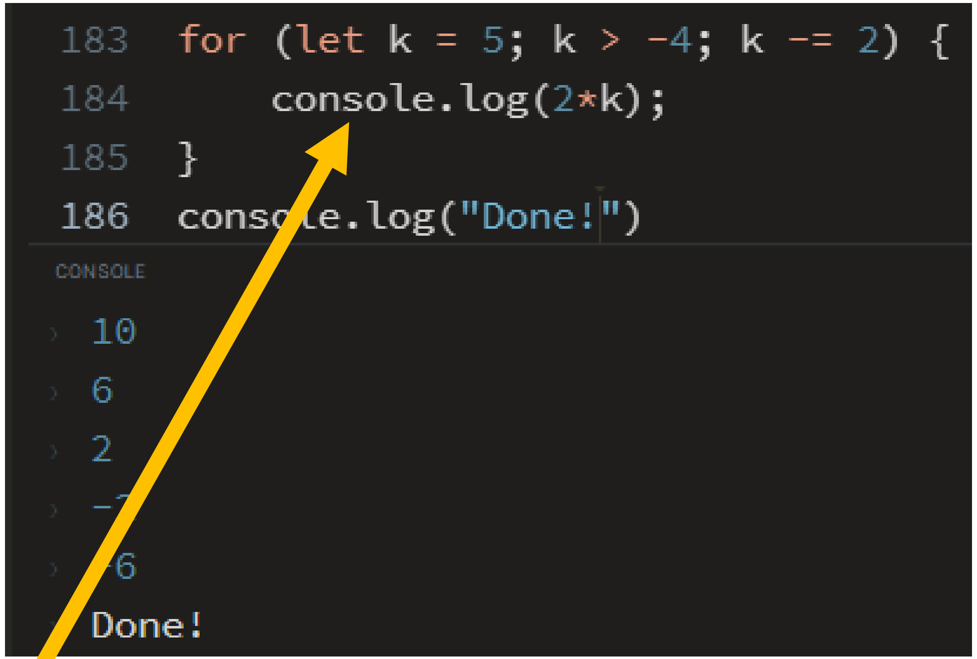
```
for (let k = 5; k > -4; k -= 2) {
    console.log(2*k);
}
console.log("Done!")
```

```
10
6
2
-2
-6
Done!
```

The for loop

The **for** loop is another to repeat a block of instructions.

- **Structure:**
  - Use the keyword **for**,
  - And immediately after, a set of parenthesis with 3 statements inside, separated by **;**.
  - The 1st statement declares (with **let**) a loop variable and assigns it an initial value. This initial value can be any value and is used as a starting point for the number of times the loop will repeat (e.g., **let k = 5**).
  - The 2nd statement is a Boolean expression that tells the loop to stop when it is no longer true (e.g., the loop is taken until **k > -4** becomes false).
  - The 3rd statement determines at which rate the loop variable is updated. (e.g., **k -= 2**).
  - Afterwards, enclose a block of instructions within curly brackets to be executed each time the loop is taken.



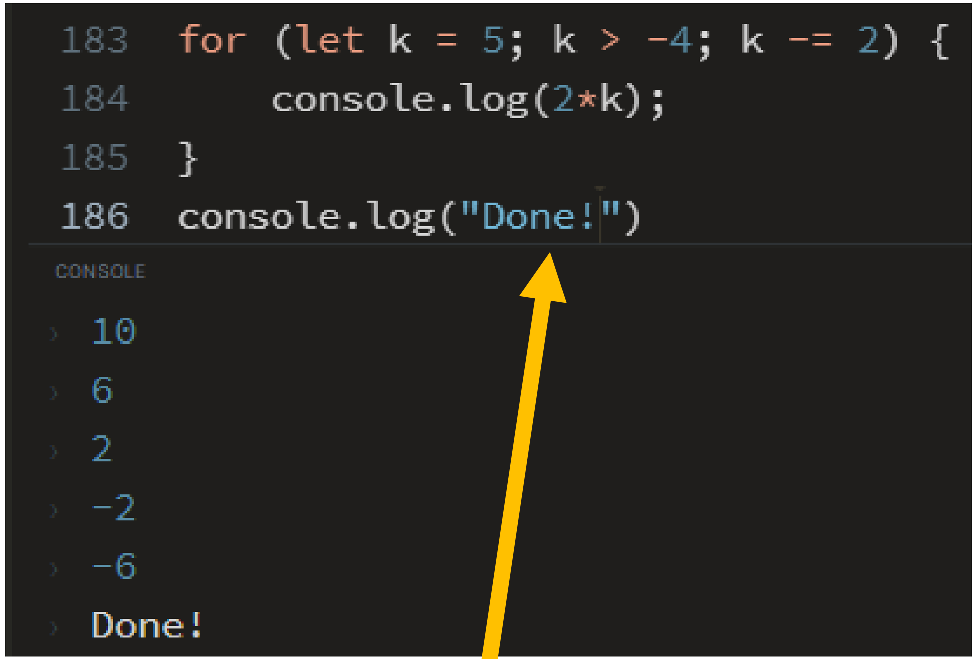
```
for (let k = 5; k > -4; k -= 2) {
    console.log(2*k);
}
console.log("Done!")
```

```
10
6
2
-2
-6
Done!
```

The for loop

The **for** loop is another to repeat a block of instructions.

- **Structure:**
  - Use the keyword **for**,
  - And immediately after, a set of parenthesis with 3 statements inside, separated by **;**.
  - The 1st statement declares (with **let**) a loop variable and assigns it an initial value. This initial value can be any value and is used as a starting point for the number of times the loop will repeat (e.g., **let k = 5**).
  - The 2nd statement is a Boolean expression that tells the loop to stop when it is no longer true (e.g., the loop is taken until **k > -4** becomes false).
  - The 3rd statement determines at which rate the loop variable is updated. (e.g., **k -= 2**).
  - Afterwards, enclose a block of instructions within curly brackets to be executed each time the loop is taken.



```
for (let k = 5; k > -4; k -= 2) {
    console.log(2*k);
}
console.log("Done!")
```

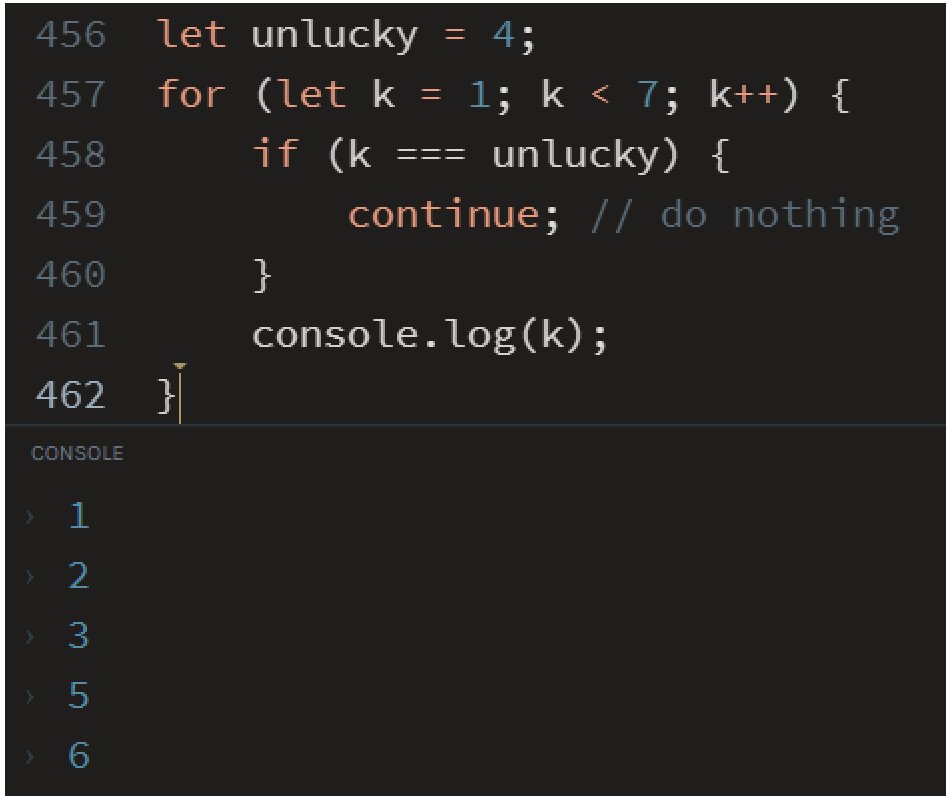
CONSOLE

```
10
6
2
-2
-6
Done!
```

- Once the program exits the loop, it moves on to the first line outside of the loop.

The continue statement

- Like the **break** statement, **continue** allows you to stop what a loop is currently doing but in a different way.
- Instead of immediately exiting the loop, **continue** immediately exits the current iteration.
- It goes back to the beginning of the loop and picks up where it left off, rather than exiting the loop entirely.
- In the example, when k === 4, the conditional statement is taken and reaches a **continue**. The iteration is immediately stopped, 4 is not displayed on the screen and the next iteration is executed with k =5.



```
let unlucky = 4;
for (let k = 1; k < 7; k++) {
    if (k === unlucky) {
        continue; // do nothing
    }
    console.log(k);
}
```

CONSOLE

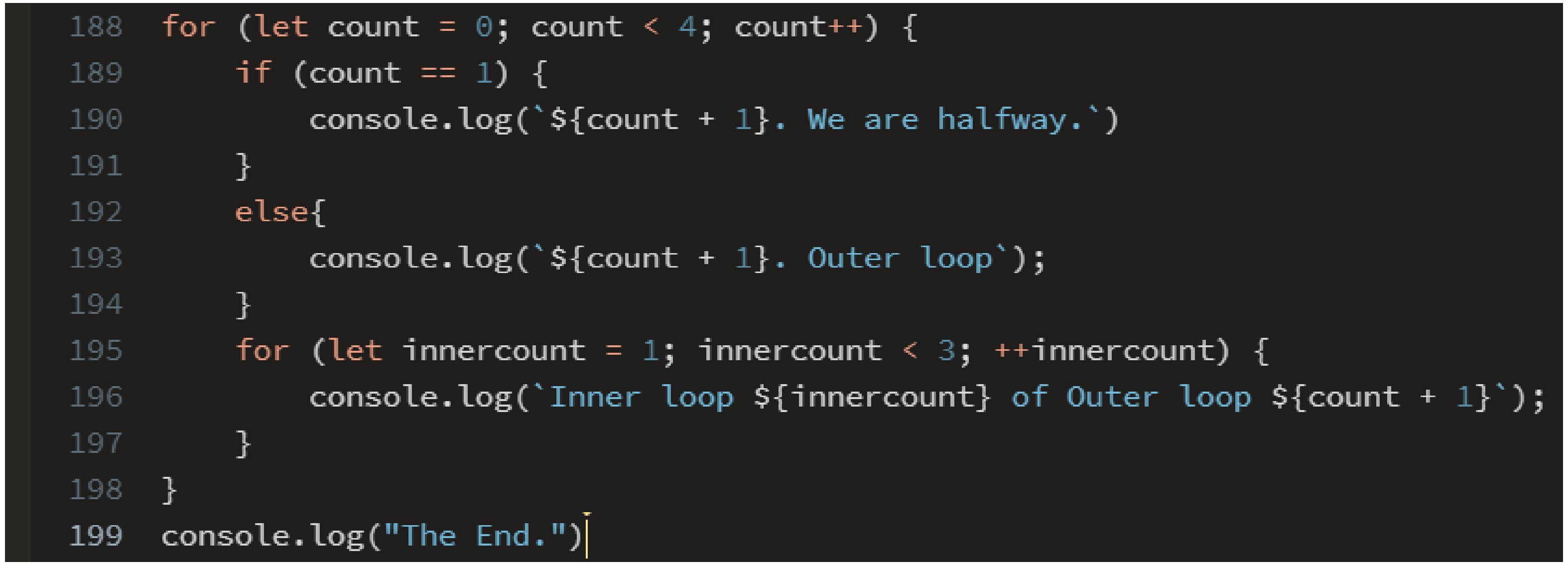
```
1
2
3
5
6
```

Exercise: prime number

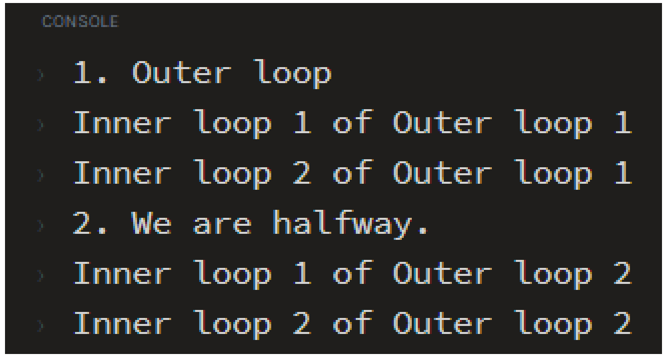
- A prime number is a positive integer n, different than 1, which can be only divided (with zero remainder) by n and 1.
- Write a simple function isPrime() that, given n, returns true whether n is prime or false otherwise.

Nested loops

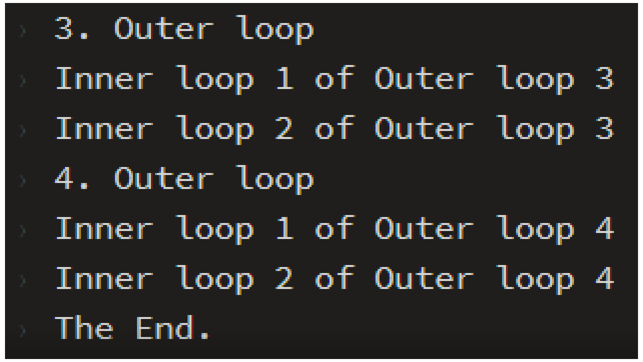
- It is possible to define inner loops inside outer loops, as well as conditional statements and so on.
- Note: In the 3rd statement of the **for** loop, count++ and ++count are equivalent because this statement is run after all the other code.



```
for (let count = 0; count < 4; count++) {
    if (count == 1) {
        console.log(`${count + 1}. We are halfway.`)
    }
    else{
        console.log(`${count + 1}. Outer loop`);
    }
    for (let innercount = 1; innercount < 3; ++innercount) {
        console.log(`Inner loop ${innercount} of Outer loop ${count + 1}`);
    }
}
console.log("The End.")
```

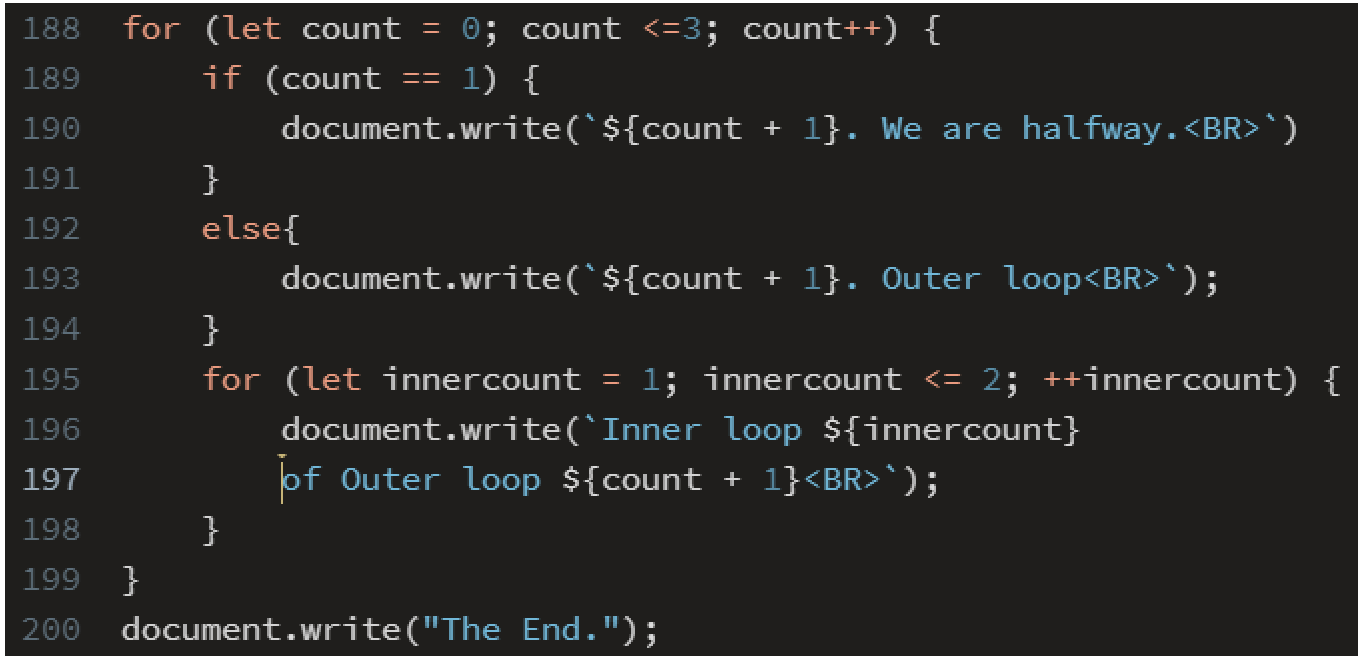


```
1. Outer loop
Inner loop 1 of Outer loop 1
Inner loop 2 of Outer loop 1
2. We are halfway.
Inner loop 1 of Outer loop 2
Inner loop 2 of Outer loop 2
```

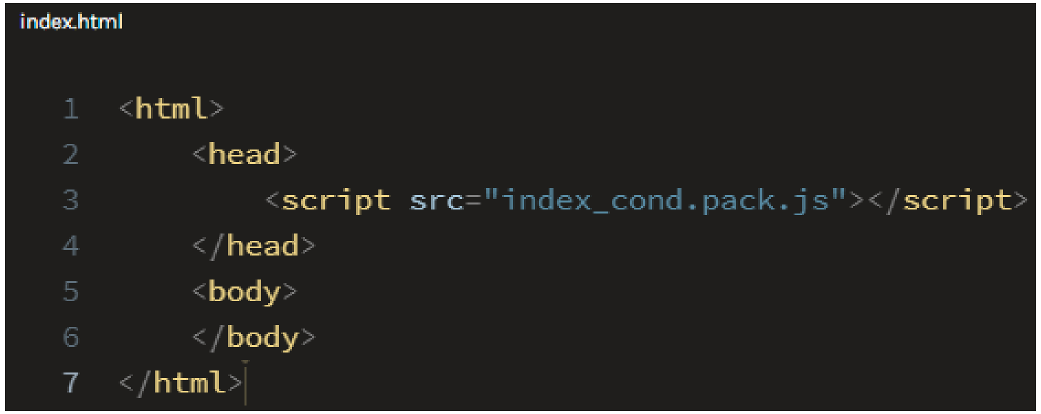


```
3. Outer loop
Inner loop 1 of Outer loop 3
Inner loop 2 of Outer loop 3
4. Outer loop
Inner loop 1 of Outer loop 4
Inner loop 2 of Outer loop 4
The End.
```

Use of JS loops in an HTML document.

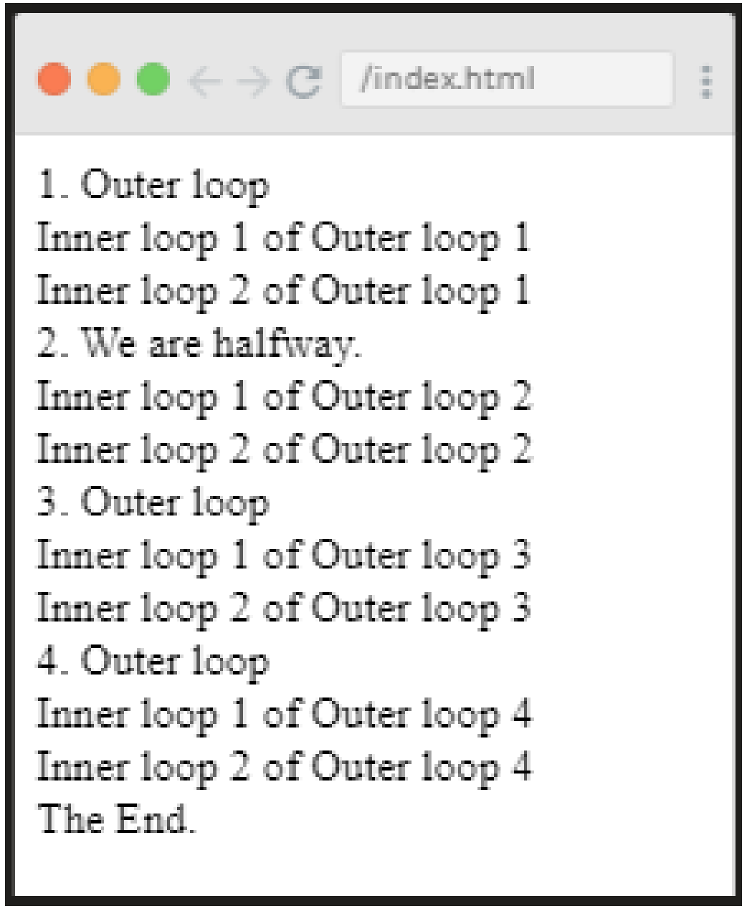


```javascript
for (let count = 0; count <=3; count++) {
    if (count == 1) {
        document.write(`${count + 1}. We are halfway.<BR>`)
    }
    else{
        document.write(`${count + 1}. Outer loop<BR>`);
    }
    for (let innercount = 1; innercount <= 2; ++innercount) {
        document.write(`Inner loop ${innercount}
        of Outer loop ${count + 1}<BR>`);
    }
}
document.write("The End.");
```



index.html

```html
<html>
    <head>
        <script src="index_cond.pack.js"></script>
    </head>
    <body>
    </body>
</html>
```



```
1. Outer loop
Inner loop 1 of Outer loop 1
Inner loop 2 of Outer loop 1
2. We are halfway.
Inner loop 1 of Outer loop 2
Inner loop 2 of Outer loop 2
3. Outer loop
Inner loop 1 of Outer loop 3
Inner loop 2 of Outer loop 3
4. Outer loop
Inner loop 1 of Outer loop 4
Inner loop 2 of Outer loop 4
The End.
```

Conclusion

- The if statement
- The else and else if statements
- Dead code and code structure
- Nested ifs
- For, while and do-while statements
- Infinite looks and how to kill them
- The break and continue statements