PSCS

Problem Solving for Computer Science

IS51021C

Goldsmiths Computing

January 18, 2020

Wk 2

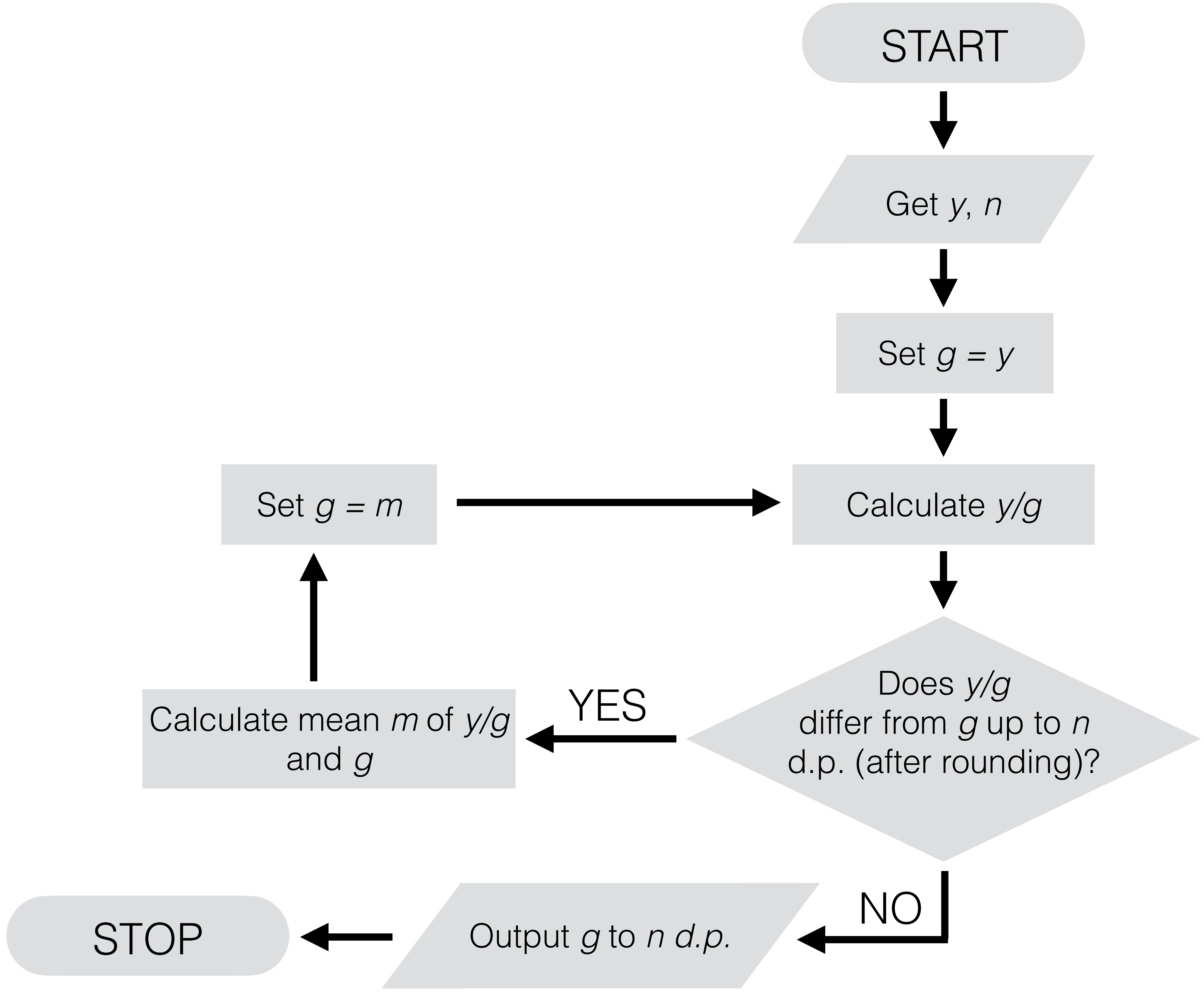
Recap of Lecture 1

- What is **a problem**?
- What is **an algorithm**?
  - Multiple algorithms for same problem
  - Predate emergence of digital computers
- Flowcharts
  - Diagrammatic representation of algorithms
- Problem 1

Recap of Lecture 1

- What is **a problem**?
- What is **an algorithm**?
  - Multiple algorithms for same problem
  - Predate emergence of digital computers
- **Flowcharts**
  - Diagrammatic representation of algorithms
- Problem 1

g	*y/g*	*?*	*m=(g+y/g)/2*


START
Get *y*, *n*
Set *g* = *y*
Calculate *y/g*
Does *y/g* differ from *g* up to *n* d.p. (after rounding)?
YES
Calculate mean *m* of *y/g* and *g*
Set *g* = *m*
NO
Output *g* to *n d.p.*
STOP

If $x^2 = n$, is x an integer?

Solution method 1: calculate $\sqrt{n}$

Solution method 2: $1^2, 2^2, 3^2, ...$

Your task:

Draw a flowchart for Solution method 2

Recap of Lecture 1

- What is **a problem**?
- What is **an algorithm**?
  - Multiple algorithms for same problem
  - Predate emergence of digital computers
- Flowcharts
  - Diagrammatic representation of algorithms
- **Problem 1**

Problem 1:

Part 1 (birthday party)

Given 48 toys and 42 sweets. What is the highest number of guests invited such that all toys and sweets are distributed equally?

Part 2 (general version)

Given X toys and Y sweets. What is the highest number of guests invited such that all toys and sweets are distributed equally?

Give a flowchart!

Problem 1:

Part 2 (general version)

Given X toys and Y sweets. What is the highest number of guests invited such that all toys and sweets are distributed equally?

N is number of guests

We want: $X = N \times W$

$Y = N \times Z$

What is the largest value of N?

This is the greatest common divisor of X and Y

$X / N = W$ $Y / N = Z$

Today:

Going from problems, algorithms and flowcharts to JavaScript

Today

1. Problems and Data Types
2. Algorithms and Functions
3. While Loops
4. From Flowcharts to Functions

Today

1. **Problems and Data Types**
2. Algorithms and Functions
3. While Loops
4. From Flowcharts to Functions

A general problem

data type?

data

Type	literal (value)
Integer	*-1, 0, 1, 2*
Boolean	*true, false*
Floating point	*11 x* 10^{-2}
Strings of characters	a$
…	

question

defined in terms of operations? ⋯

answer

data type?

Comparison: $x > y$?
Solution to expression: $x^2 = ?$
Set membership: *x in A ?*
String comparison: *$=£?*
String concatenation: *$£*
…

A general problem

data type?

data

question

defined in terms of operations? …

answer

data type?

Type: literal (value)

Integer: *-1, 0, 1, 2*
Boolean: *true, false*
Floating point: 11×10^{-2}
Strings of characters: a$
…

Comparison: $x > y$?
Solution to expression: $x^2 = ?$
Set membership: *x in A* ?

We need to be able to translate into JavaScript

Primitive Data Types in JavaScript

Type	Literals (Values)	
Boolean	`true` `false`	
Number (float)	`1` `3.14` `NaN`	
String	`"ps_for_cs"` `""`	← Empty string
Undefined	`undefined`	Variables without values
Null	`null`	Nothing

All other* data types are **objects**

*Technically Null an object in JavaScript
(one of its “oddities”)

Types

Buffalo buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo

↑ Noun ↑ Verb

Types are not always obvious in human language

Types

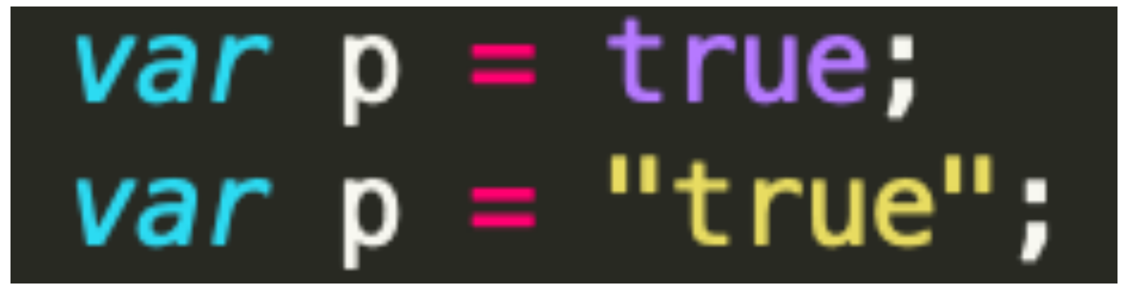
Buffalo buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo

Noun

Verb

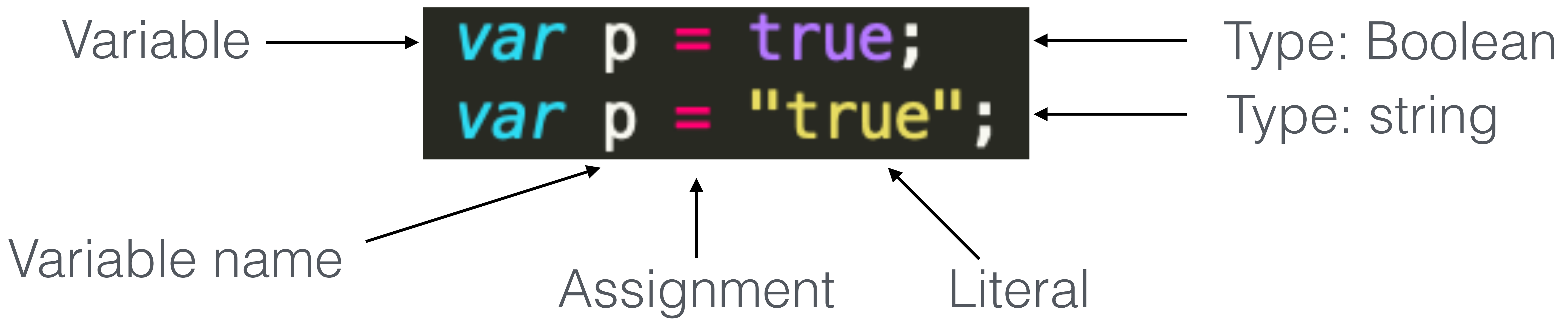
Types are not always obvious in human language

The same is true in JavaScript



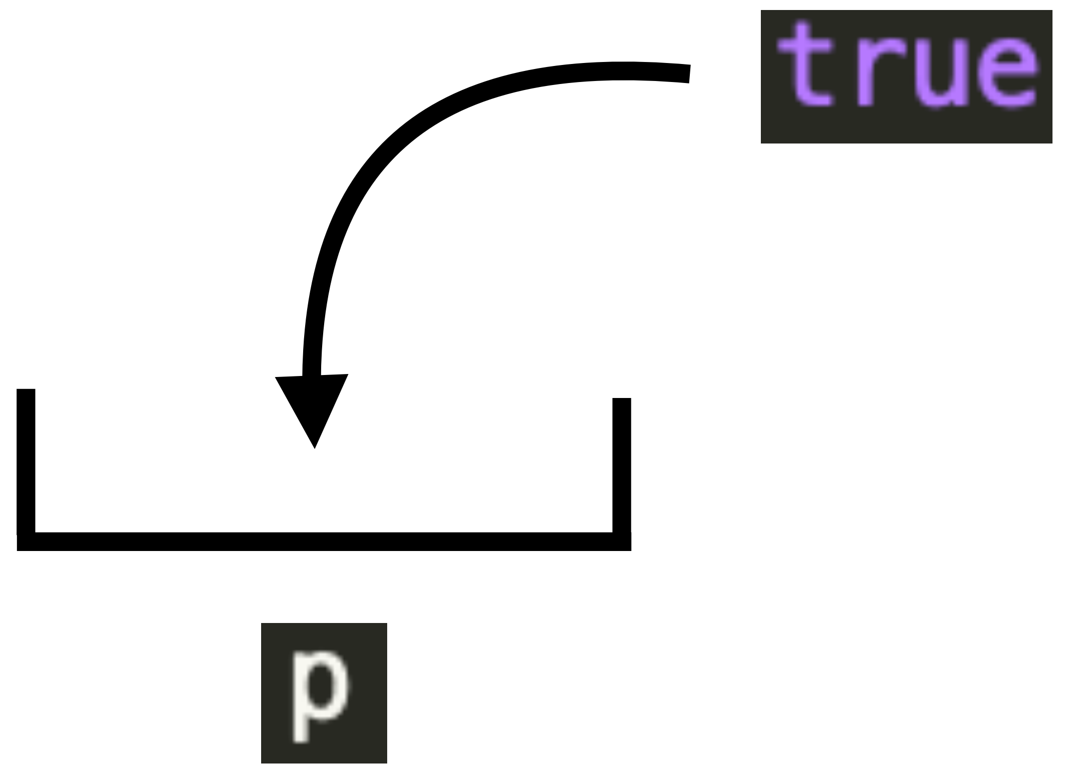
```
var p = true;
var p = "true";
```

Types

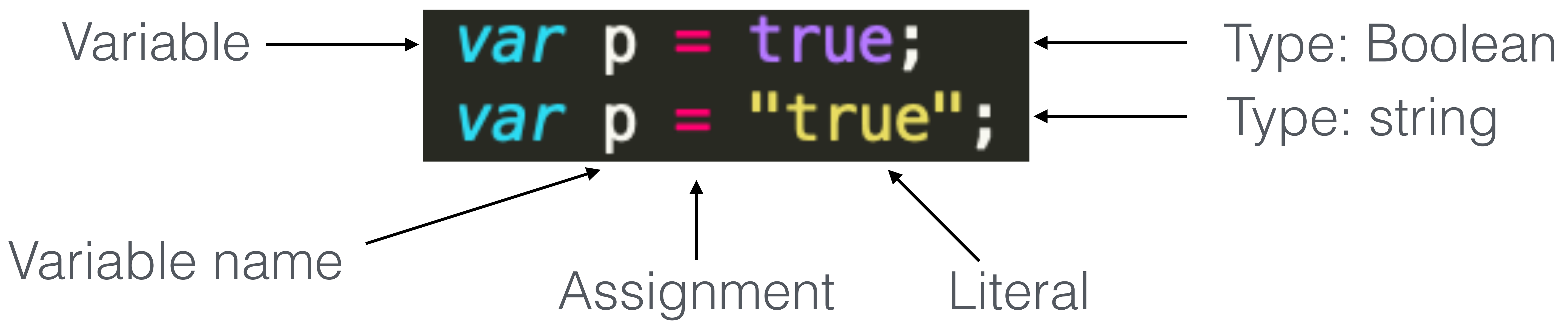


```
var p = true;
var p = "true";
```

Variables are containers for data



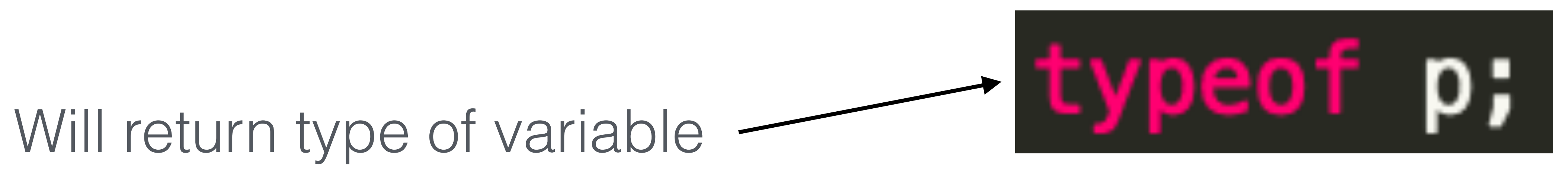
Types



```
var p = true;
var p = "true";
```

Variables:

- Not only can literals (values) change, but **so can type**
- Types are *inferred* from the values
- This is why we need "undefined"



```
typeof p;
```

NB: Not all languages are like this. Why?

A general problem

data type?

data

question

answer

data type?

defined in terms of **operations**? · · ·

Comparison: ***x > y ?***
Solution to expression: ***x^2 = ?***
Set membership: ***x in A ?***
String comparison: ***$=£?***
String concatenation: ***$£***
…

Type: literal (value)

Integer: *-1, 0, 1, 2*
Boolean: *true, false*
Floating point: *11 x 10^{-2}*
Strings of characters: a$
…

Operations

Type	Operations						Example
Number	+ Add	- Subtract	* Multiply	/ Divide	% Modulo	** Exponential	`(3*2)%2`
Boolean	|| OR	&& AND	! NOT				`true||(!true)`
String	+ Concatenate						`"$" + "1m"`

Operations

Type	Operations						Example
Number	`+` Add	`-` Subtract	`*` Multiply	`/` Divide	`%` Modulo	`**` Exponential	`(3*2)%2` $\longrightarrow$ `0`
Boolean	`||` OR	`&&` AND	`!` NOT				`true||(!true)` $\longrightarrow$ `true`
String	`+` Concatenate						`"$" + "1m"` $\longrightarrow$ `"$1m"`

Operations

Type	Operations						Example
Number	`+` Add	`-` Subtract	`*` Multiply	`/` Divide	`%` Modulo	`**` Exponential	`(3*2)%2` $\longrightarrow$ `0`
Boolean	`||` OR	`&&` AND	`!` NOT				`true||(!true)` $\longrightarrow$ `true`
String	`+` Concatenate						`"$" + "1m"` $\longrightarrow$ `"$1m"`

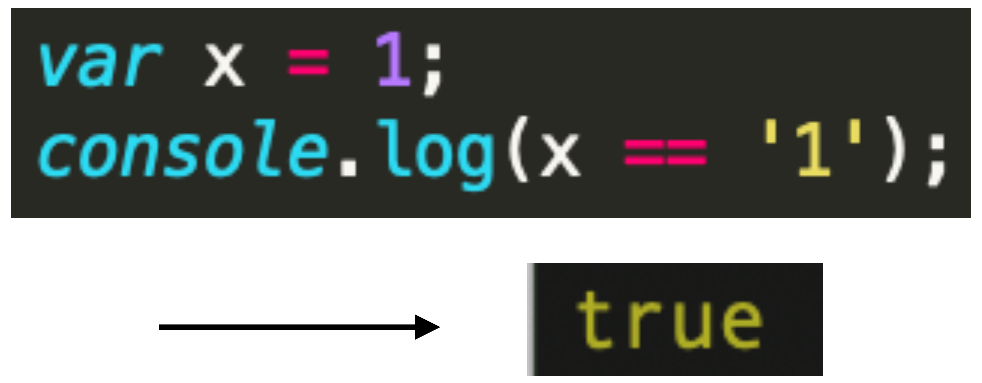
Comparisons

`==`	`!=`	`>`	`<`	`<=`	`>=`
Equal to	Not equal to	Greater than	Less than	Less than or equal to	Greater than or equal to

Comparisons

==	!=	>	<	<=	>=
Equal to	Not equal to	Greater than	Less than	Less than or equal to	Greater than or equal to

JavaScript can compare numbers and strings!



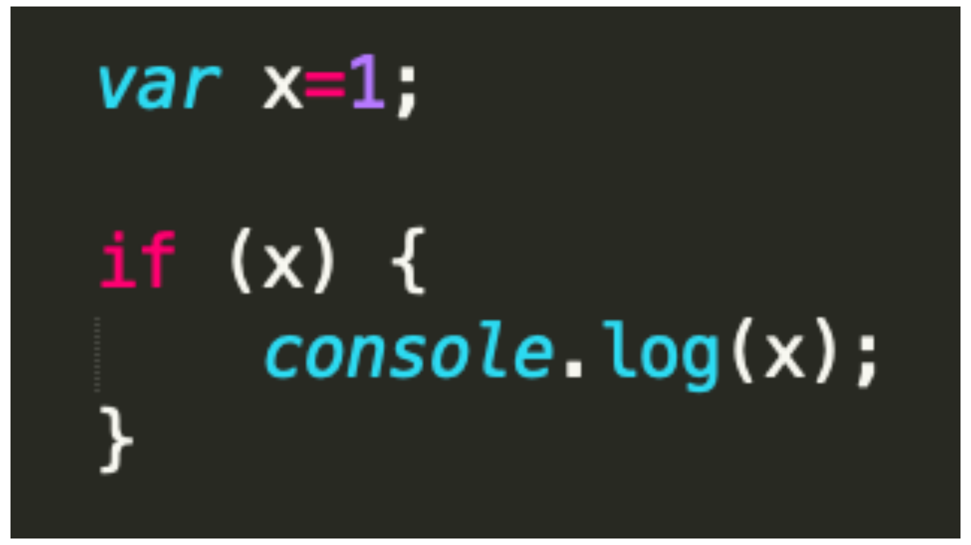
```
var x = 1;
console.log(x == '1');
```

```
true
```

Caution: Truthy & Falsy

JavaScript associates the values of true and false to non-Boolean data types

Used in context of Booleans e.g.

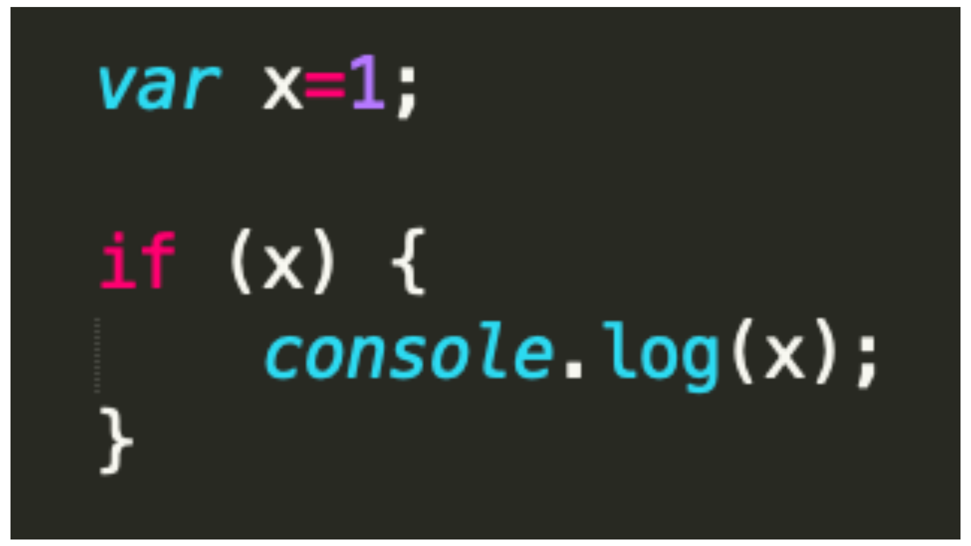


```
var x=1;

if (x) {
    console.log(x);
}
```

Caution: Truthy & Falsy

JavaScript associates the values of true and false to non-Boolean data types

Used in context of Booleans e.g.



```
var x=1;

if (x) {
    console.log(x);
}
```

→ Prints 1 in console

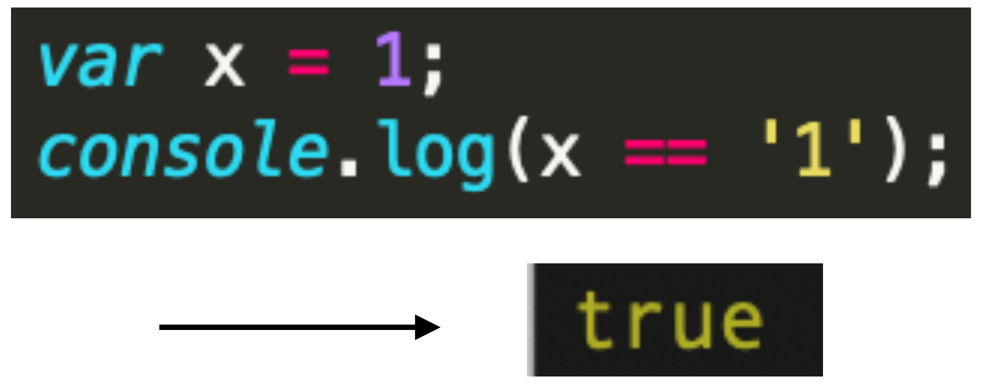
Everything is truthy, except these (and variants), which are falsy:

`NaN` `false` `""` `undefined` `null`

Comparisons

`==`	`!=`	`>`	`<`	`<=`	`>=`
Equal to	Not equal to	Greater than	Less than	Less than or equal to	Greater than or equal to

JavaScript can compare numbers and strings!



```
var x = 1;
console.log(x == '1');
```

```
true
```

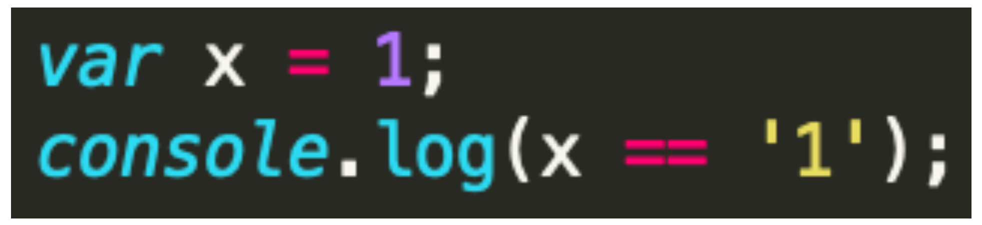
If you want to be strict about types

`===`	`!==`
Equal to and has same type	Not equal to OR not same type

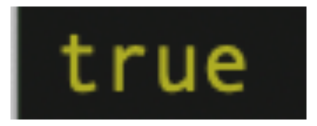
Comparisons

==	!=	>	<	<=	>=
Equal to	Not equal to	Greater than	Less than	Less than or equal to	Greater than or equal to

JavaScript can compare numbers and strings!



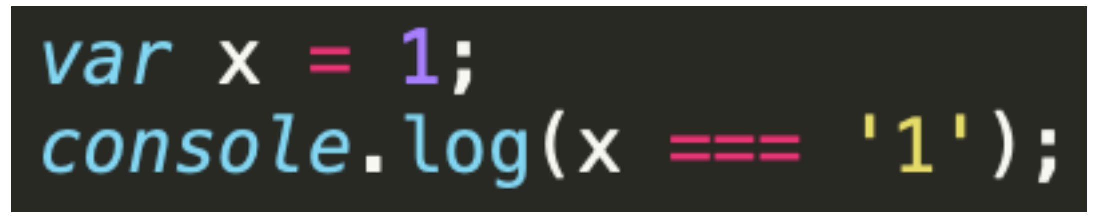
```
var x = 1;
console.log(x == '1');
```



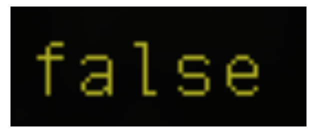
```
true
```

If you want to be strict about types

===	!==
Equal to and has same type	Not equal to OR not same type



```
var x = 1;
console.log(x === '1');
```

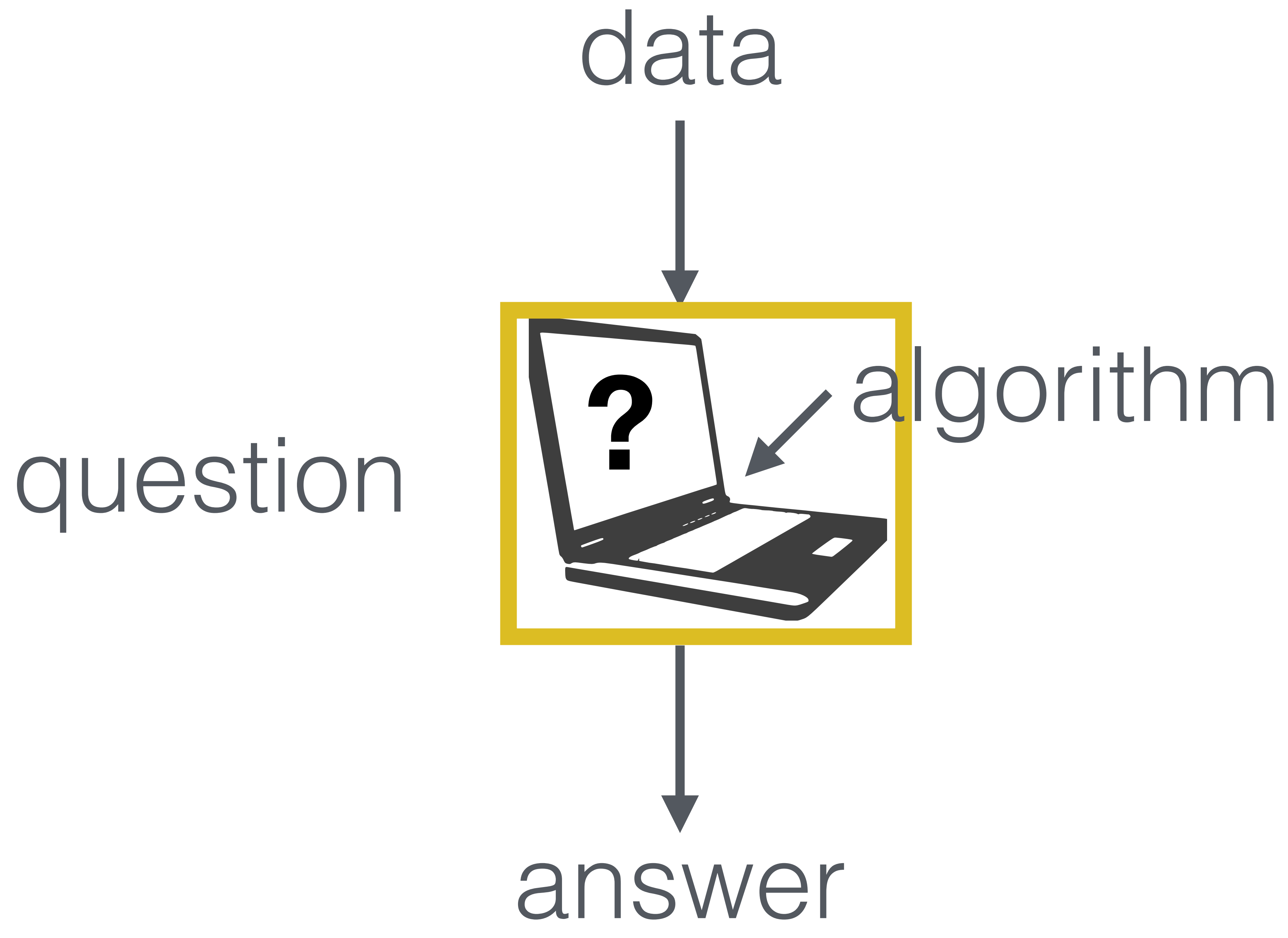


```
false
```

Today

1. Problems and Data Types
2. **Algorithms and Functions**
3. While Loops
4. From Flowcharts to Functions

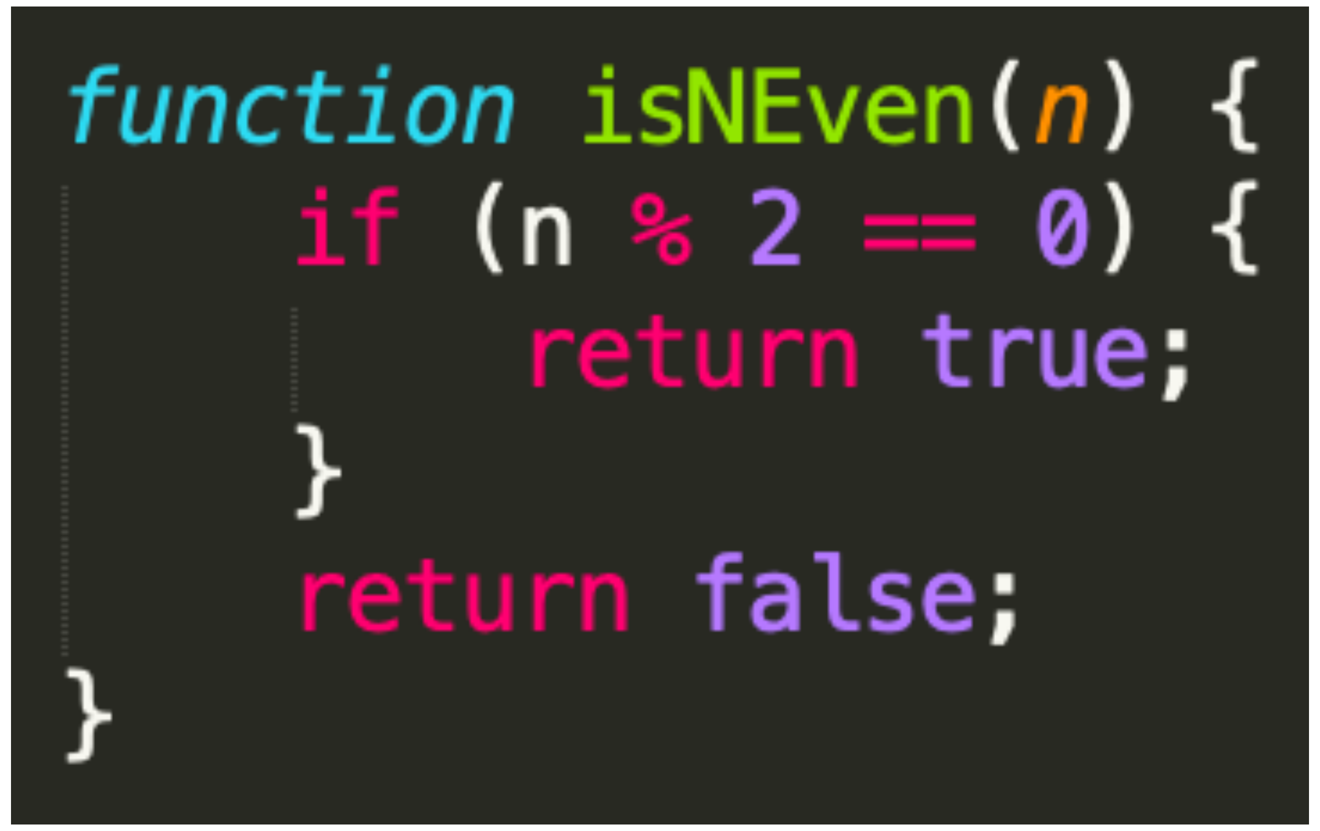
Solving a problem



Needs to be described as an **algorithm**

Which can be *implemented* as a computer program

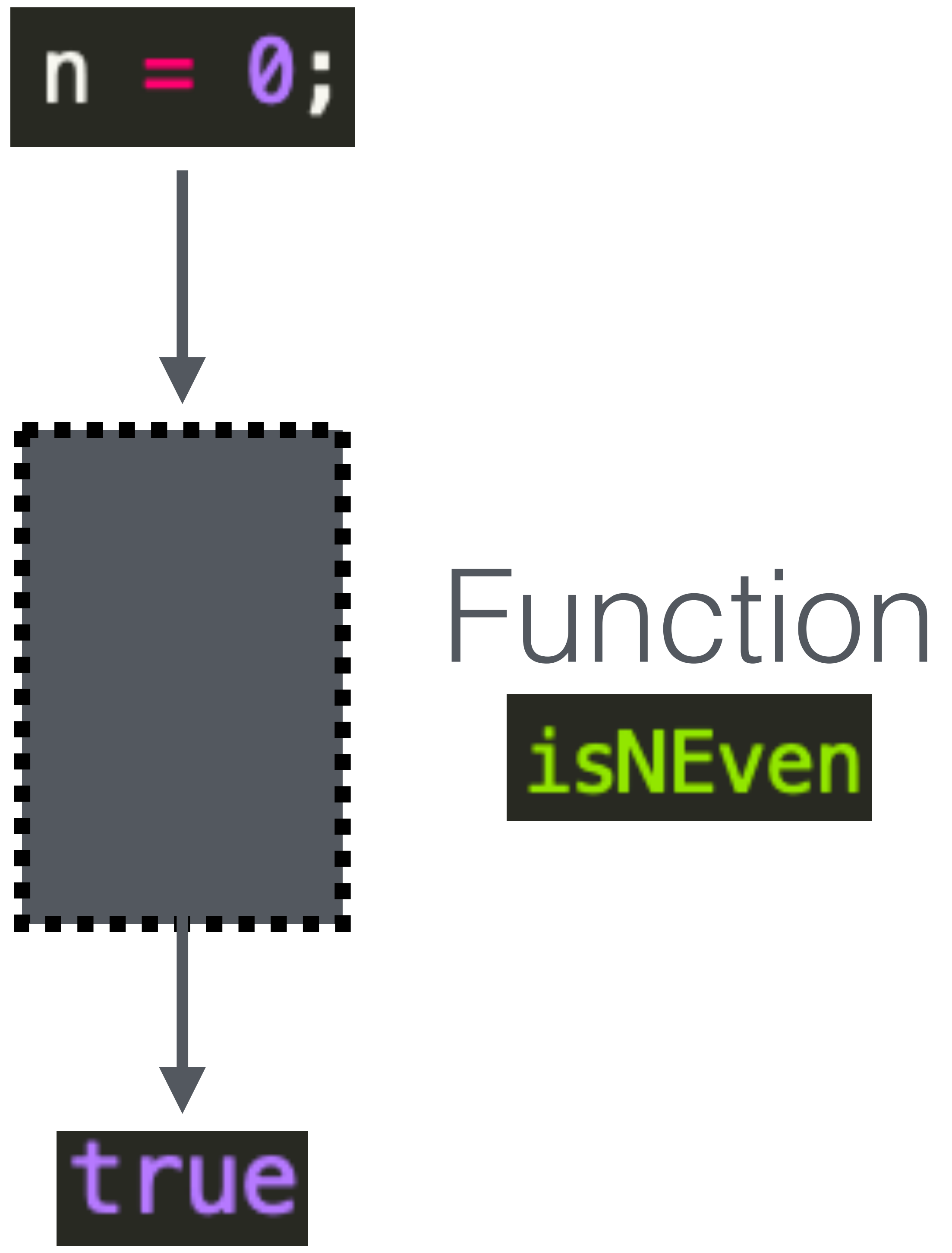
JavaScript Functions



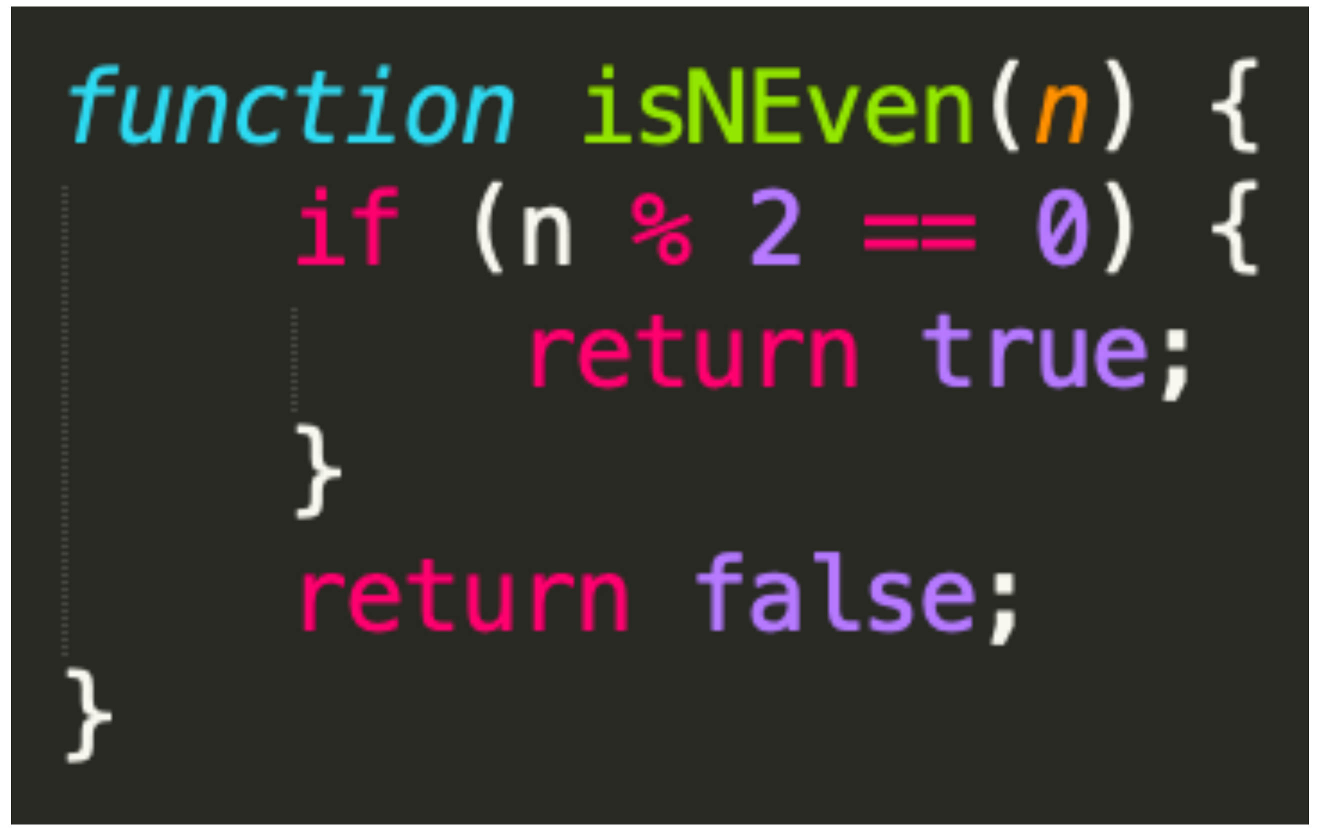
```
function isNEven(n) {
    if (n % 2 == 0) {
        return true;
    }
    return false;
}
```

JavaScript Functions

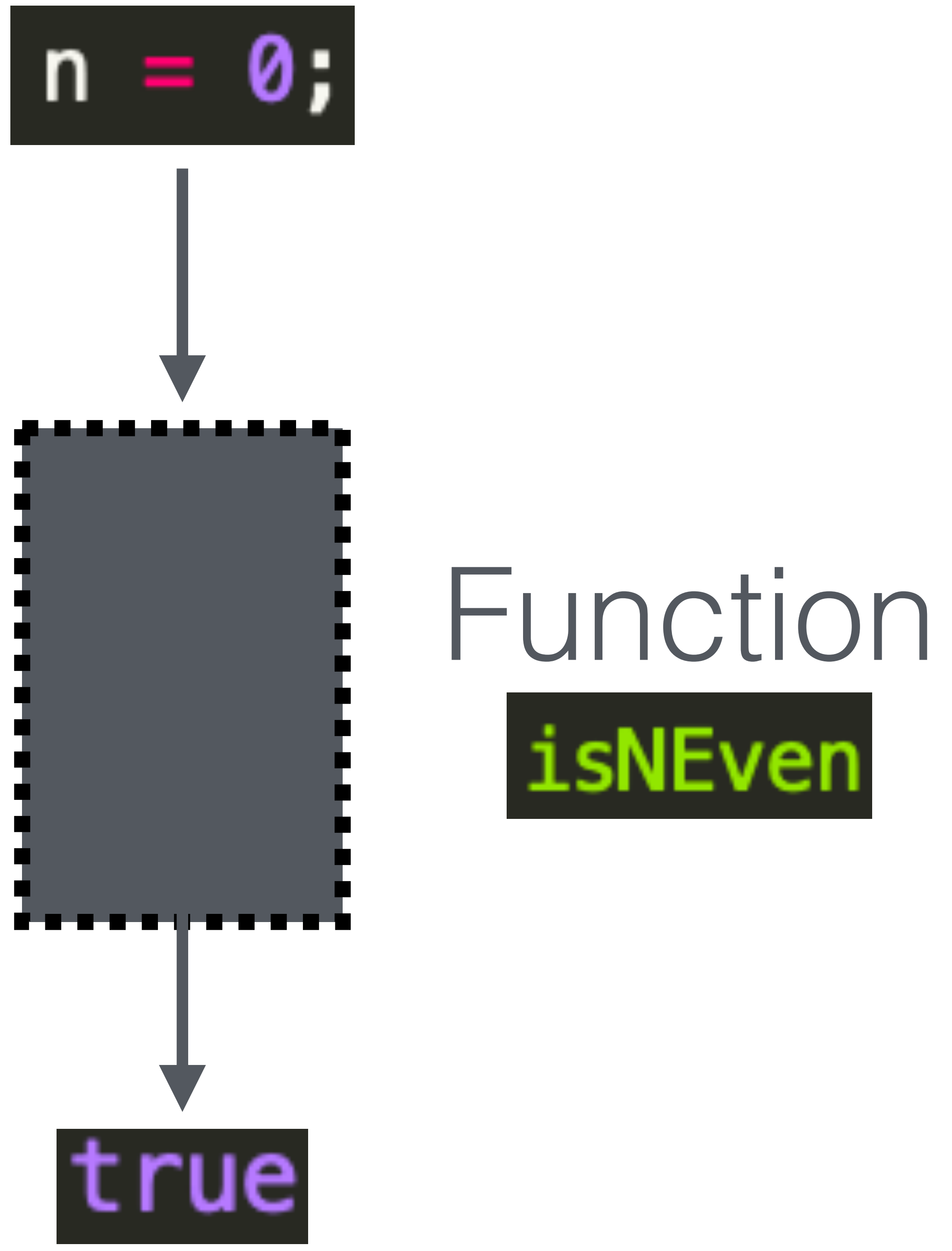
```
function isNEven(n) {
    if (n % 2 == 0) {
        return true;
    }
    return false;
}
```


```
n = 0;
```

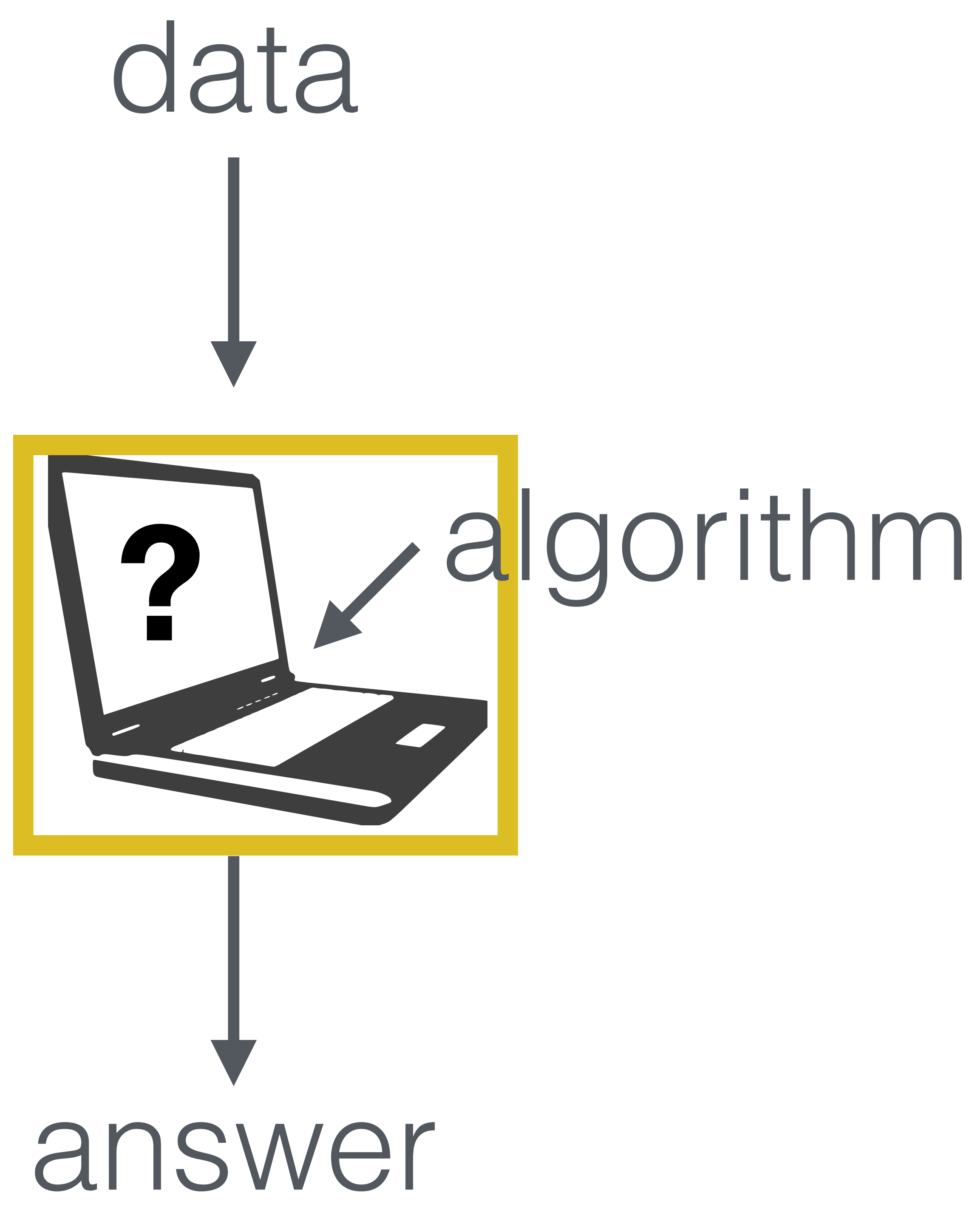
JavaScript Functions



```
function isNEven(n) {
    if (n % 2 == 0) {
        return true;
    }
    return false;
}
```



```
n = 0;
```



Functions will be the main method for implementing (general) algorithms

Admin

- This module is a “learning by doing” module: problem solving is an active process and not just a list of facts to learn for an exam

- **First week of Virtual Contact Hours (VCH)**
  - Meeting in calendar - short group discussion at beginning then classmates
  - Worksheet 1 released after this lecture (not assessed)
  - Start working on tasks as soon as you want - get help in VCH
  - My experience: people who attempt worksheets and ask questions tend to perform better

- **First quiz available today from 6pm - 2.5% of your final grade**
  - My experience: people who attempt quiz twice tend to do better(plan your time!)

Worksheet 1

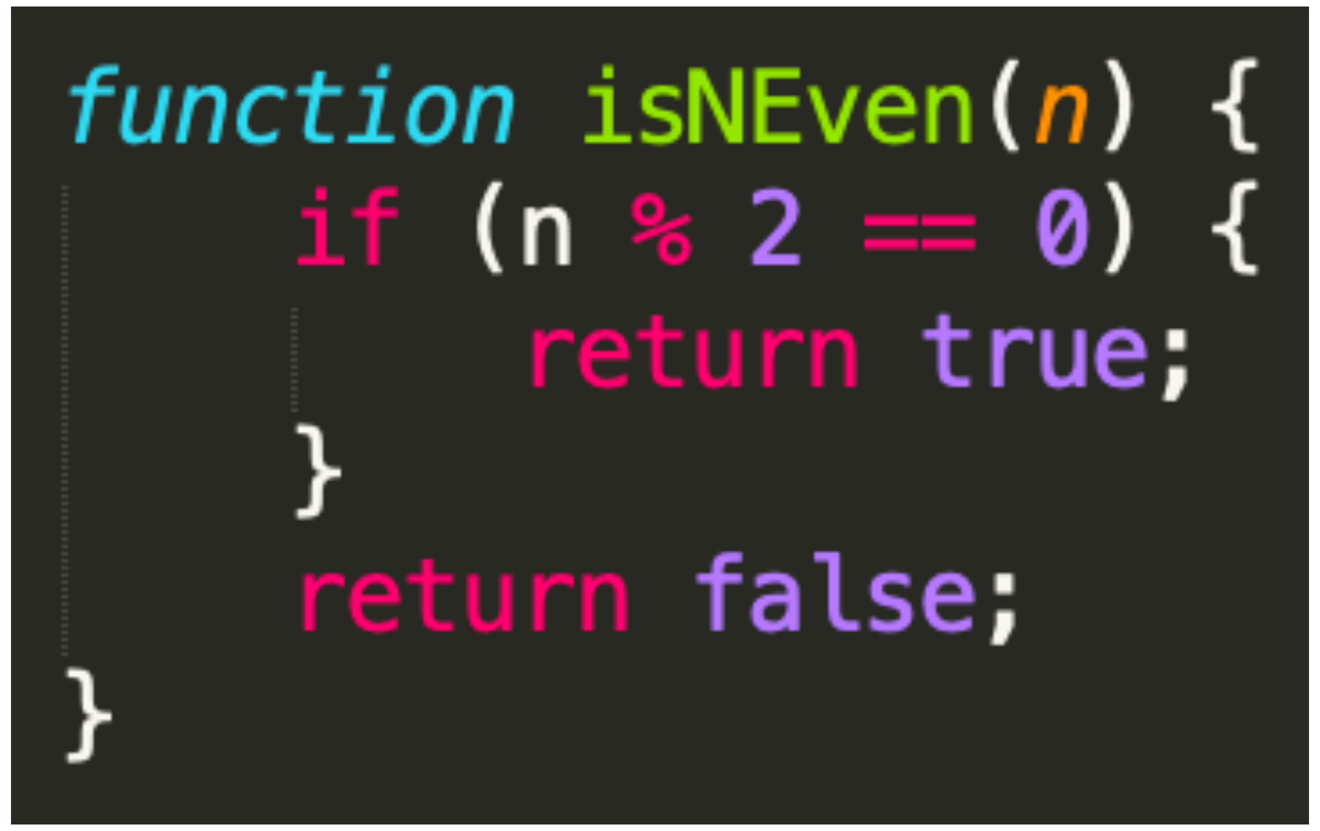
ADVICE: USE PEN AND PAPER TO DRAW PICTURES

e.g. draw arrays with their indices
write down the values of variables in each iteration of a loop
sketch examples

Functions will be the main method for implementing (general) algorithms

What kind of ingredients do we have?

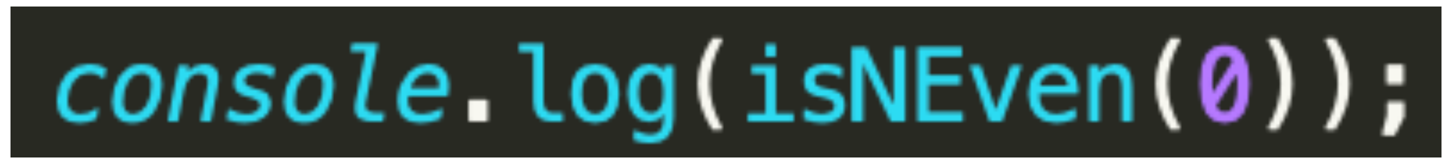
Function ingredients



```
function isNEven(n) {
    if (n % 2 == 0) {
        return true;
    }
    return false;
}
```

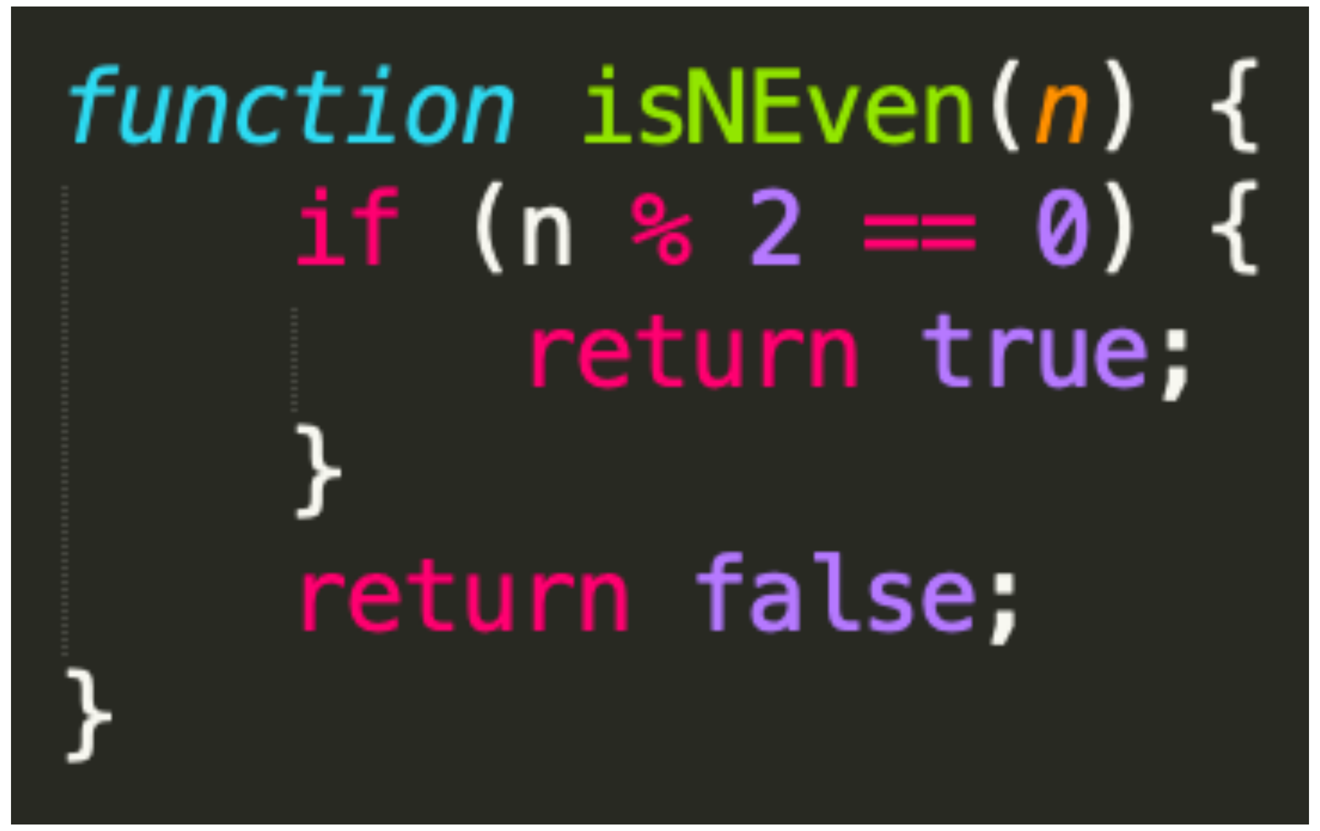
Name - what we use to call the function

Argument - input data to function



```
console.log(isNEven(0));
```

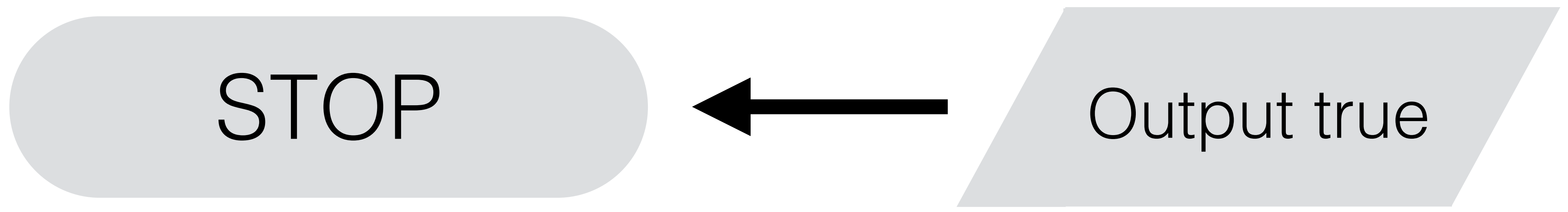
Function ingredients



```
function isNEven(n) {
    if (n % 2 == 0) {
        return true;
    }
    return false;
}
```

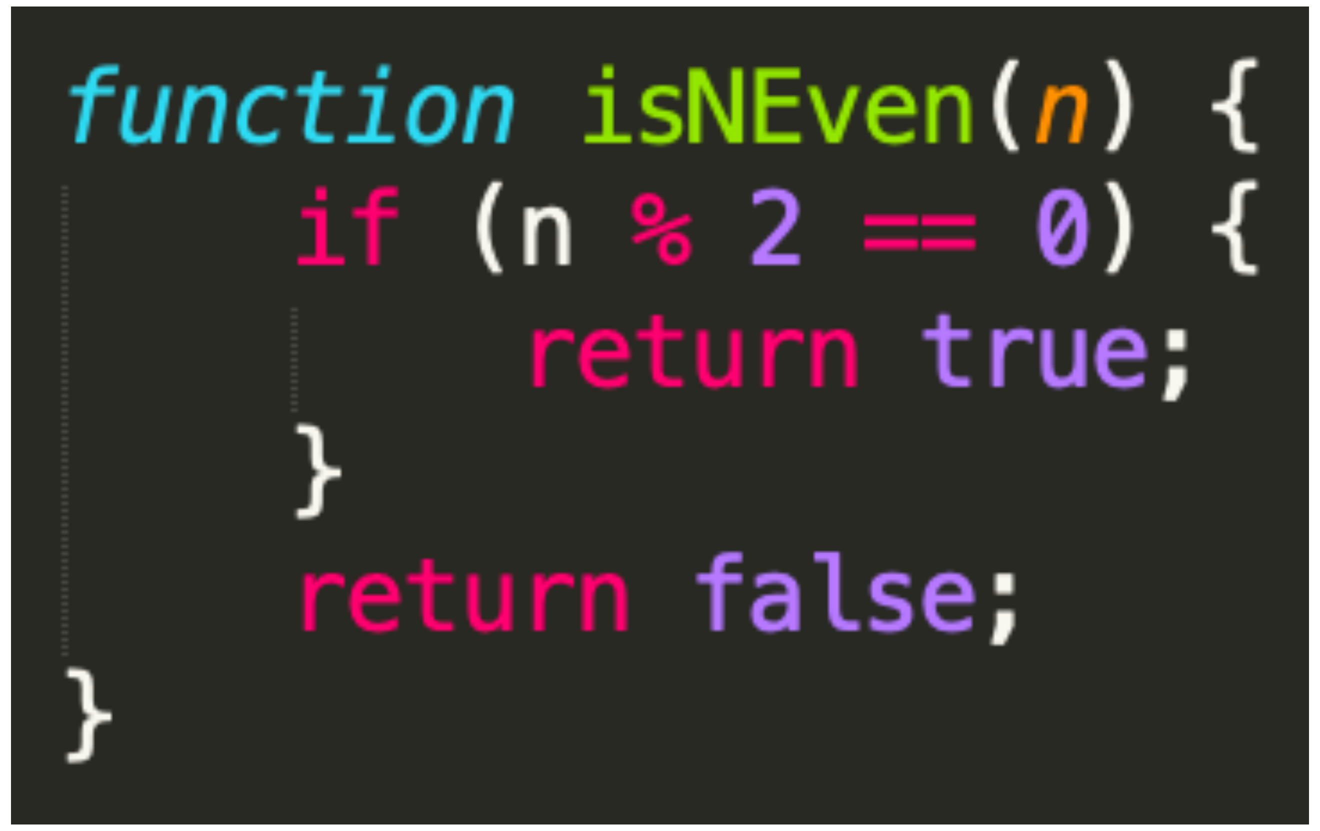
Return - output answer

When a function returns it stops being executed



What happens if we do not specify a return value?

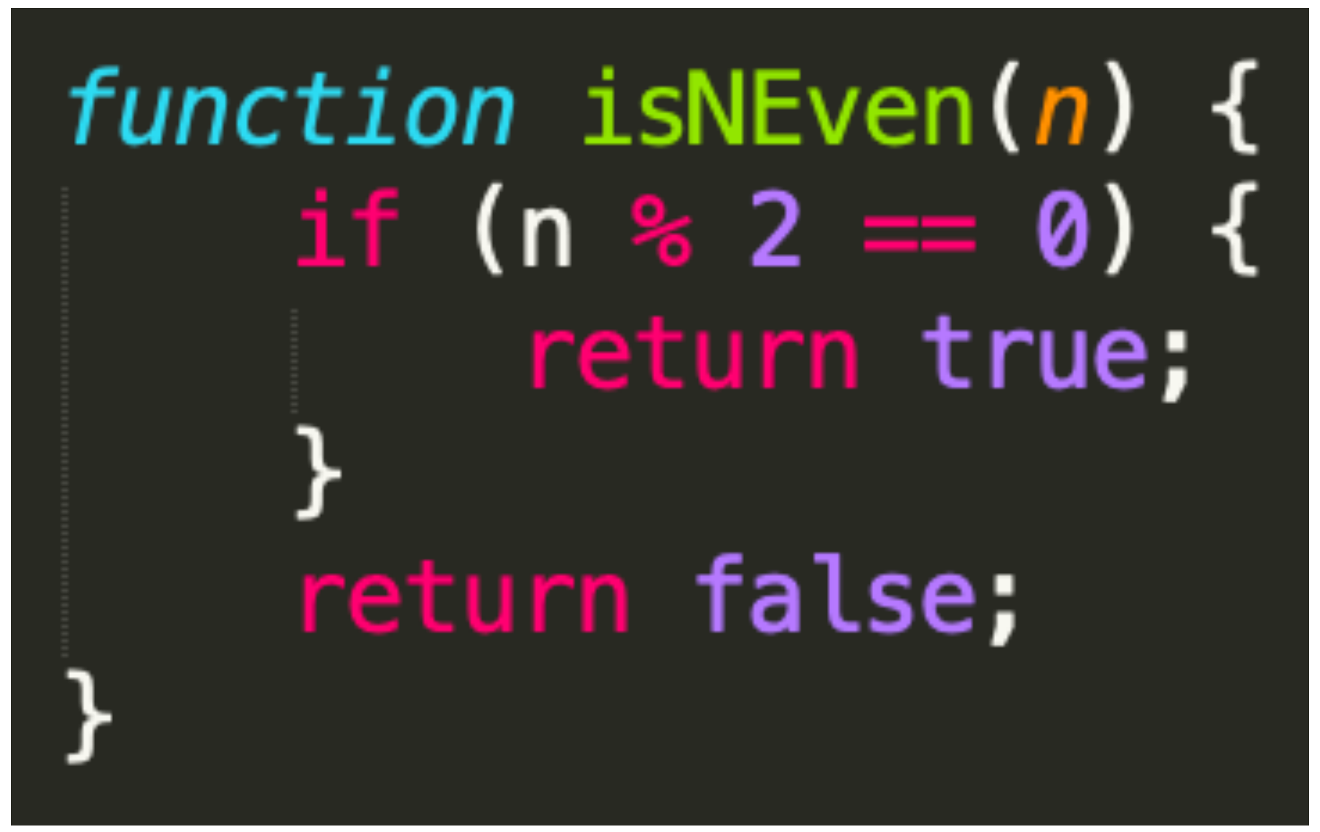
Function ingredients



```
function isNEven(n) {
    if (n % 2 == 0) {
        return true;
    }
    return false;
}
```

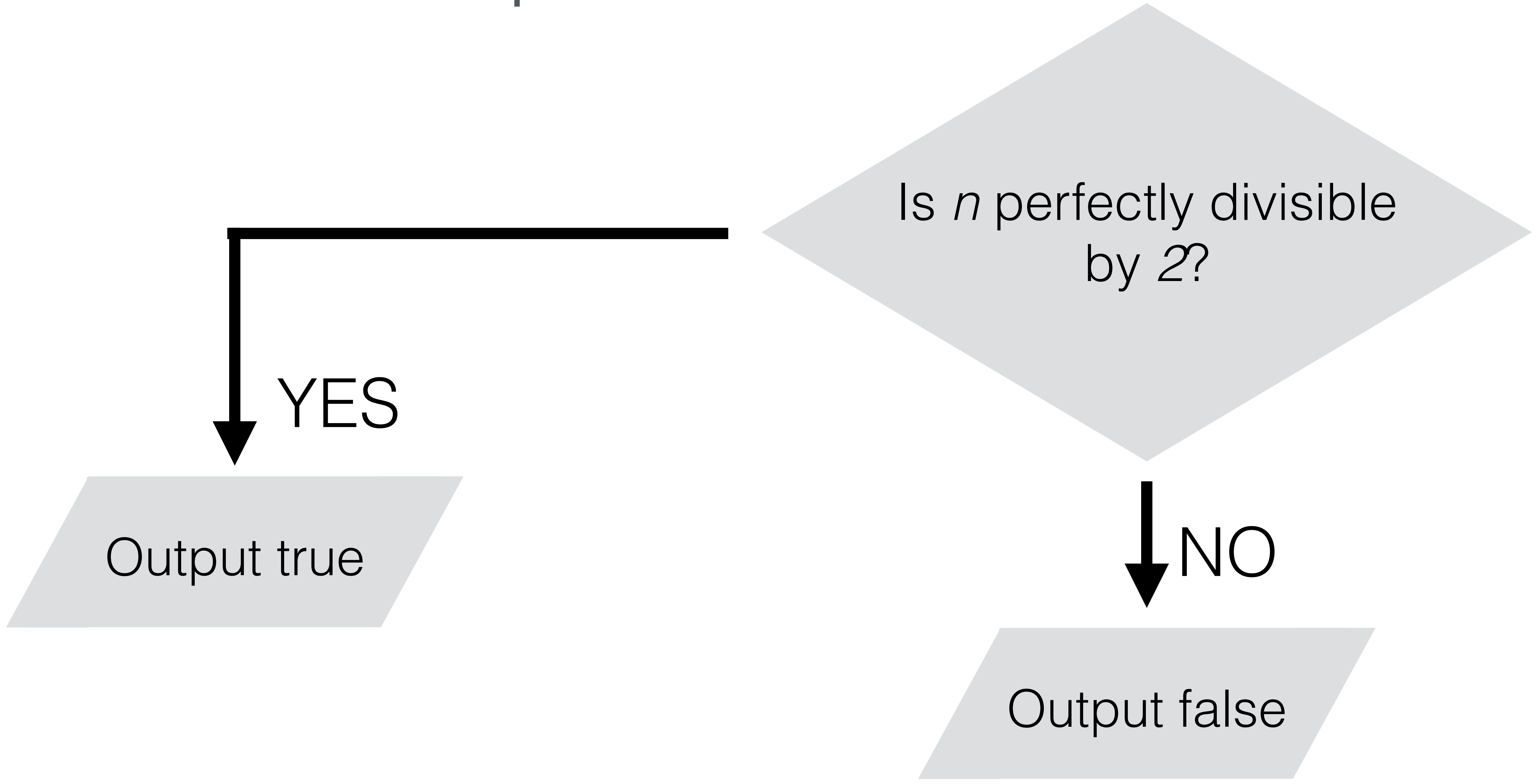
If… then - conditional operations

Function ingredients

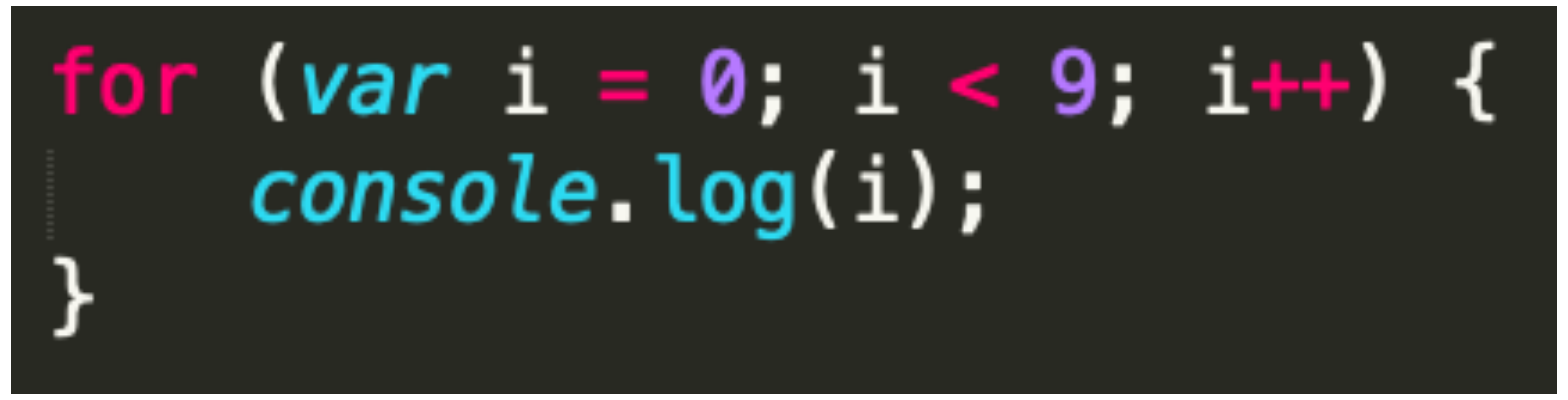


```
function isNEven(n) {
    if (n % 2 == 0) {
        return true;
    }
    return false;
}
```

If… then - conditional operations



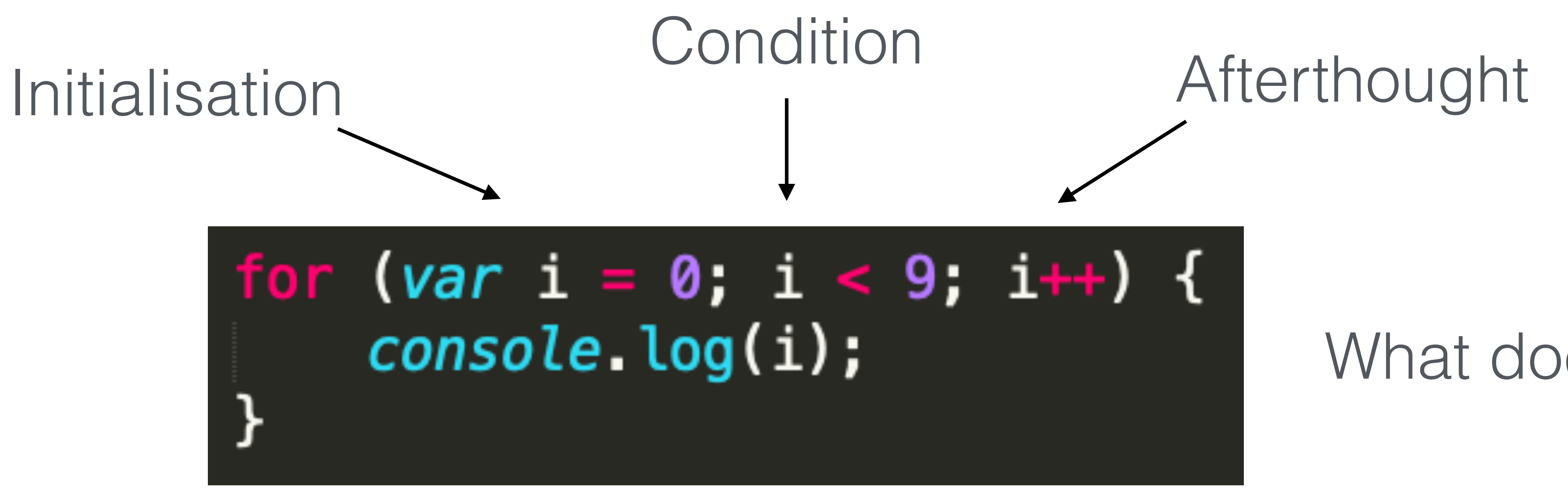
Function ingredients



```
for (var i = 0; i < 9; i++) {
    console.log(i);
}
```

For loops - for iterating some code

Function ingredients

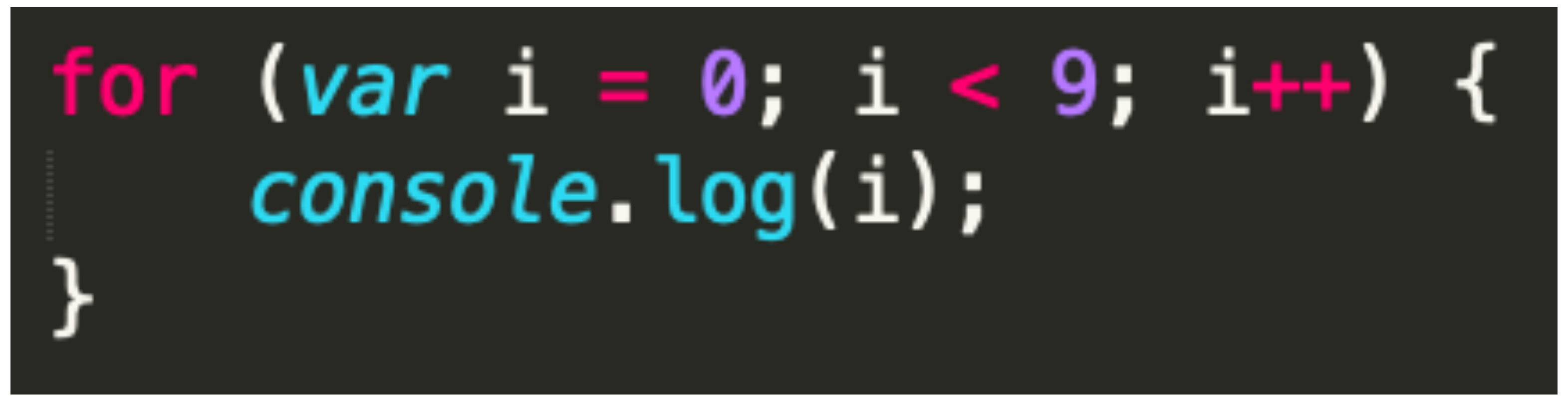


```
for (var i = 0; i < 9; i++) {
    console.log(i);
}
```

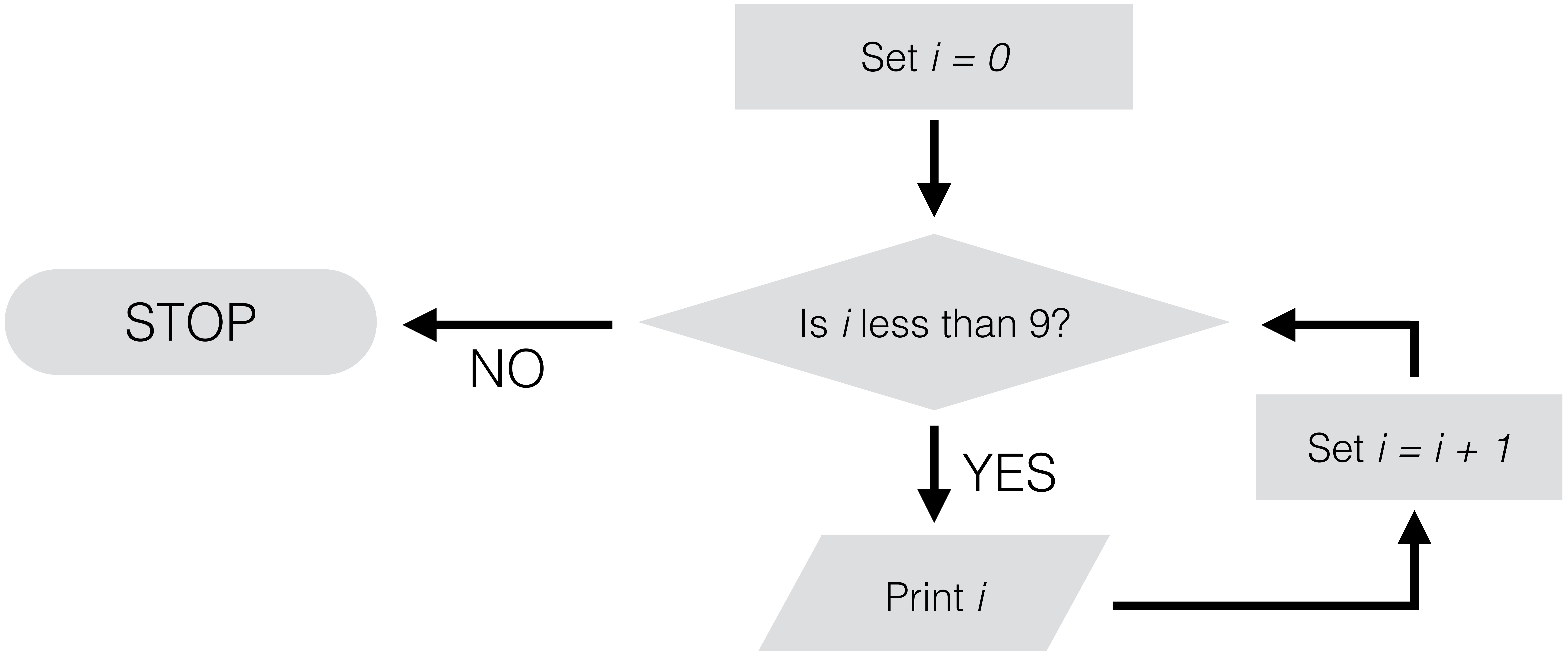
What does this do?

For loops - for iterating some code

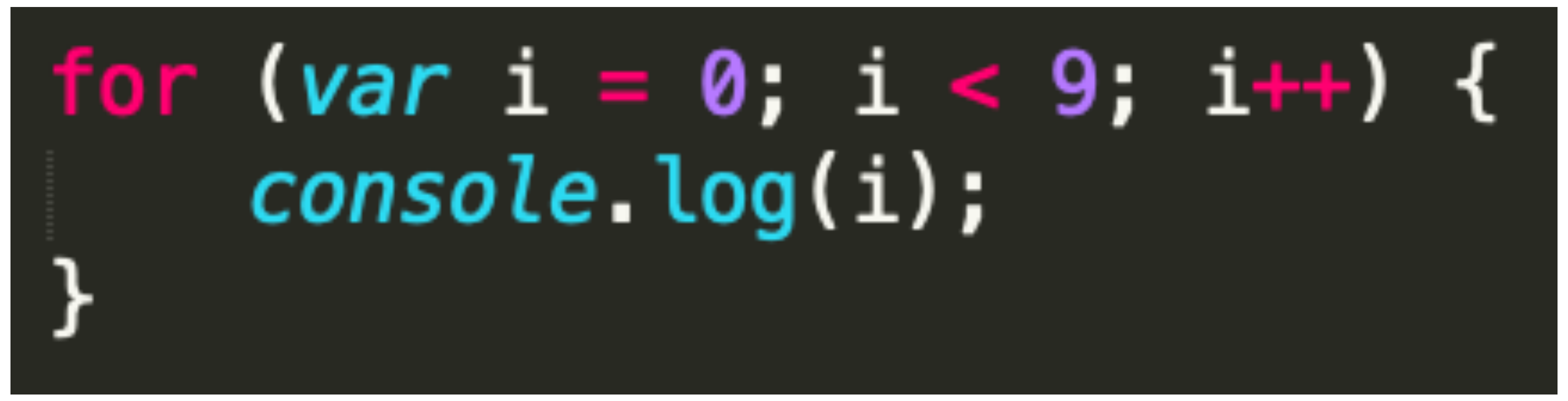
Function ingredients


```
for (var i = 0; i < 9; i++) {
    console.log(i);
}
```

For loops - for iterating some code

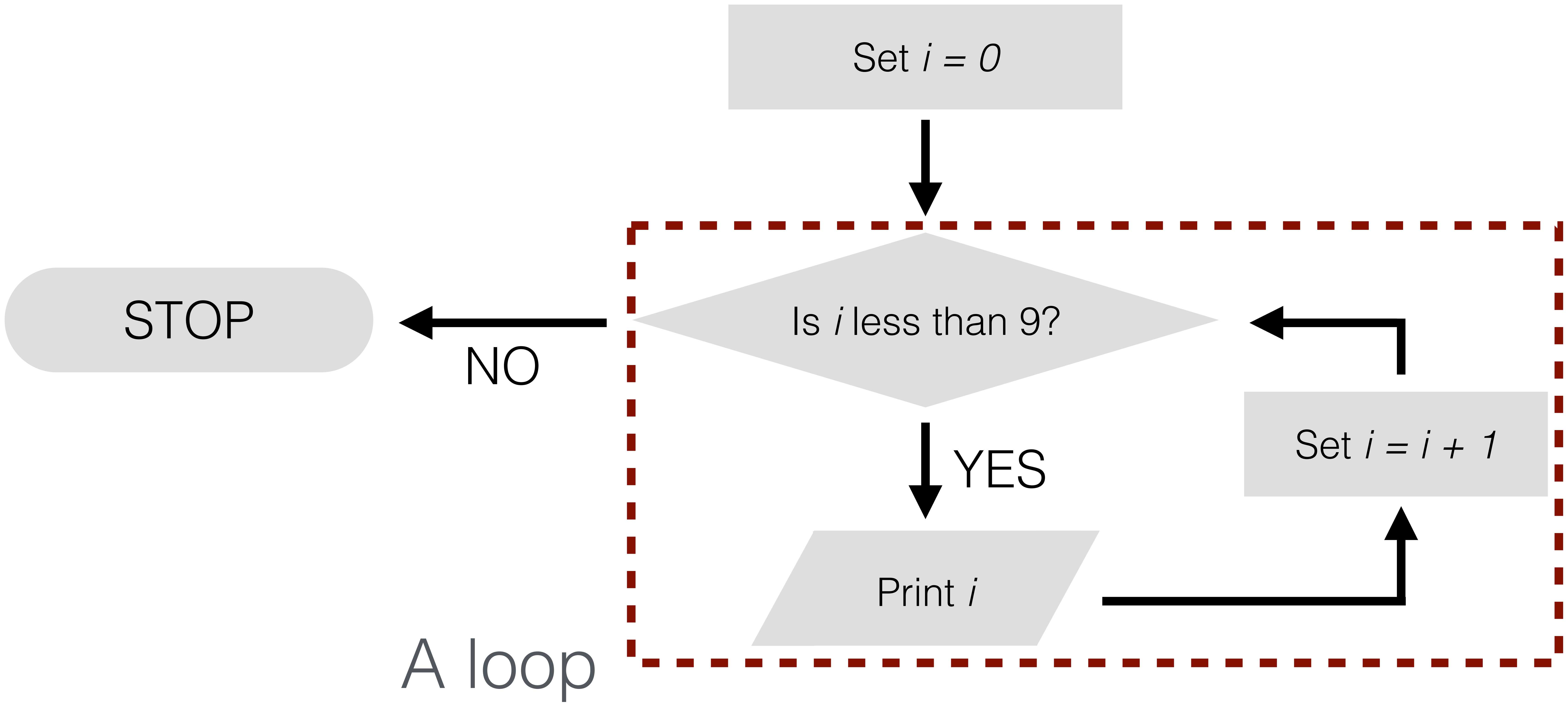


Function ingredients



```
for (var i = 0; i < 9; i++) {
    console.log(i);
}
```

For loops - for iterating some code



Today

1. Problems and Data Types
2. Algorithms and Functions
3. **While Loops**
4. From Flowcharts to Functions

Function ingredients

While loops

When should we use them?

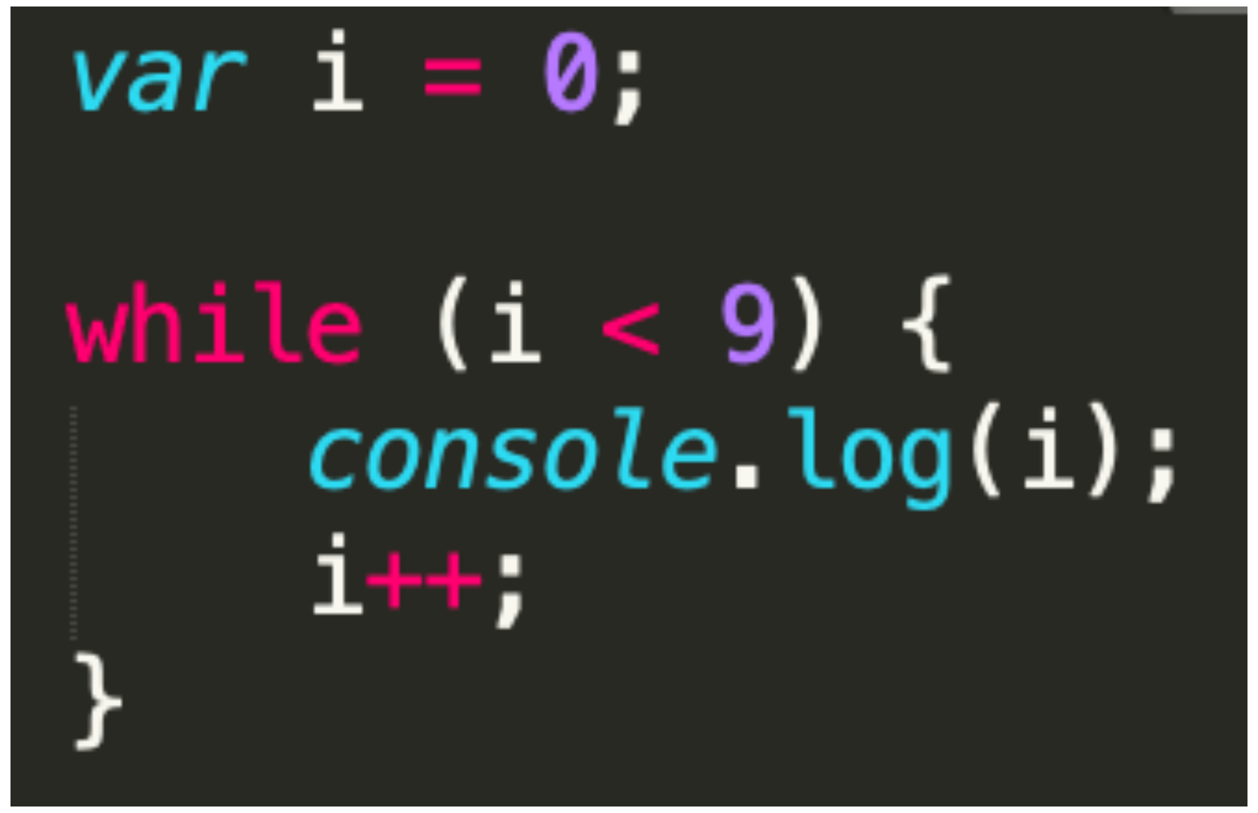
Function ingredients

While loops

When should we use them?

When we do not obviously know the number of iterations

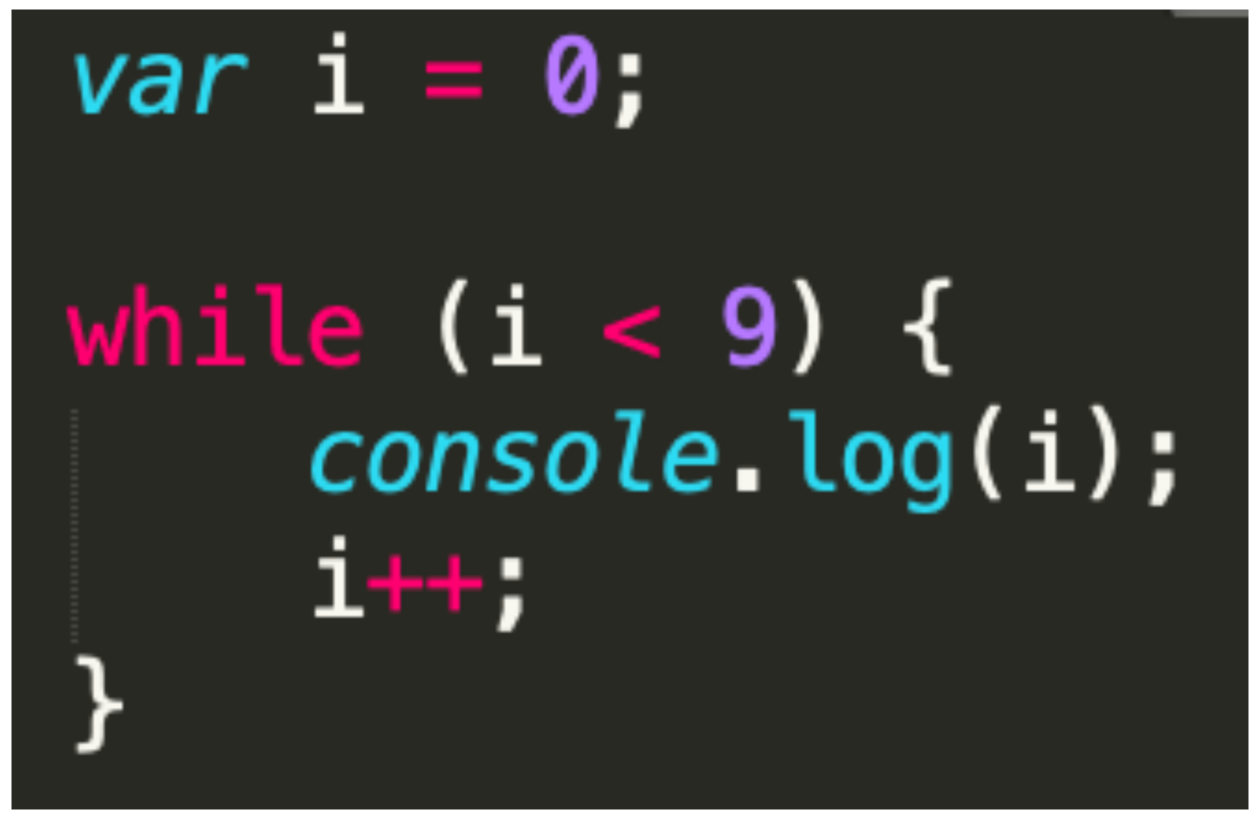
Function ingredients


```
var i = 0;

while (i < 9) {
    console.log(i);
    i++;
}
```

While loop - repeat while condition in round brackets is true

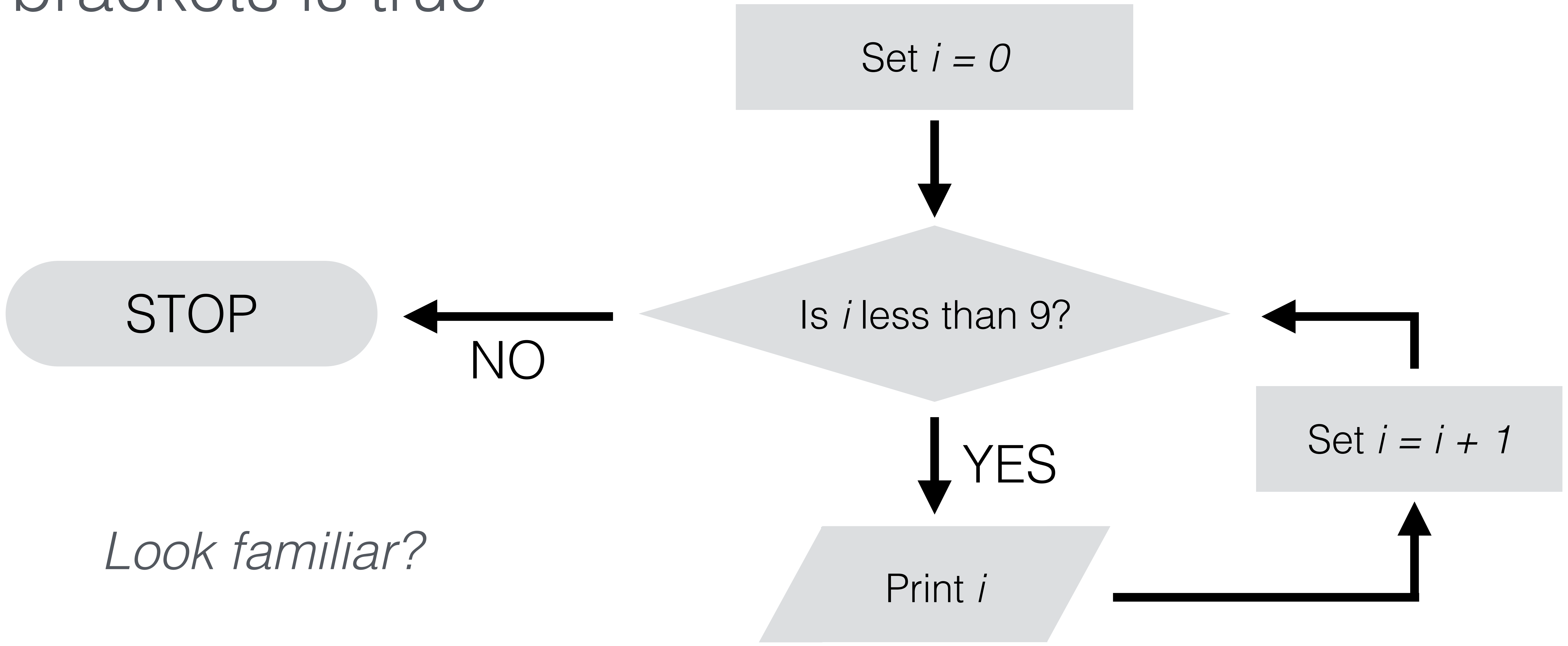
Has similar form to if conditional

Function ingredients



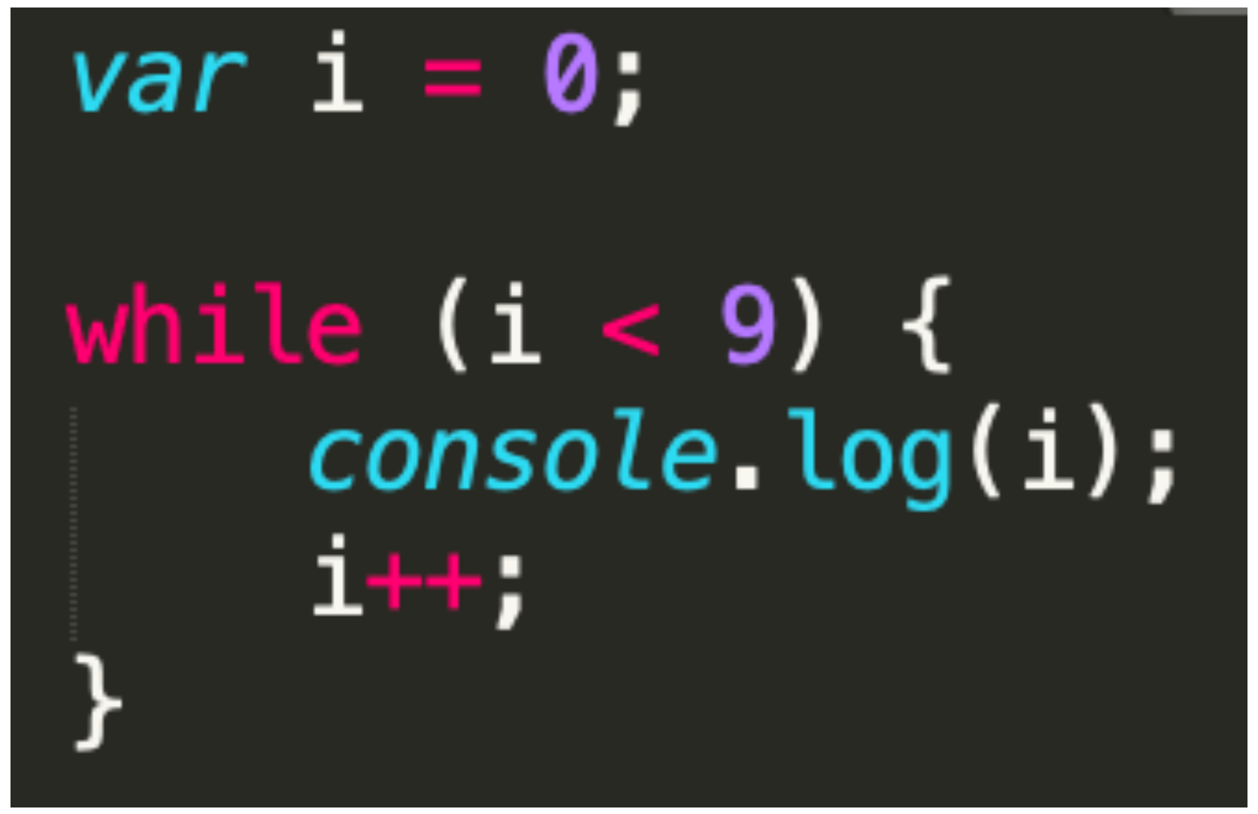
```
var i = 0;

while (i < 9) {
    console.log(i);
    i++;
}
```

While loop - repeat while condition in round brackets is true

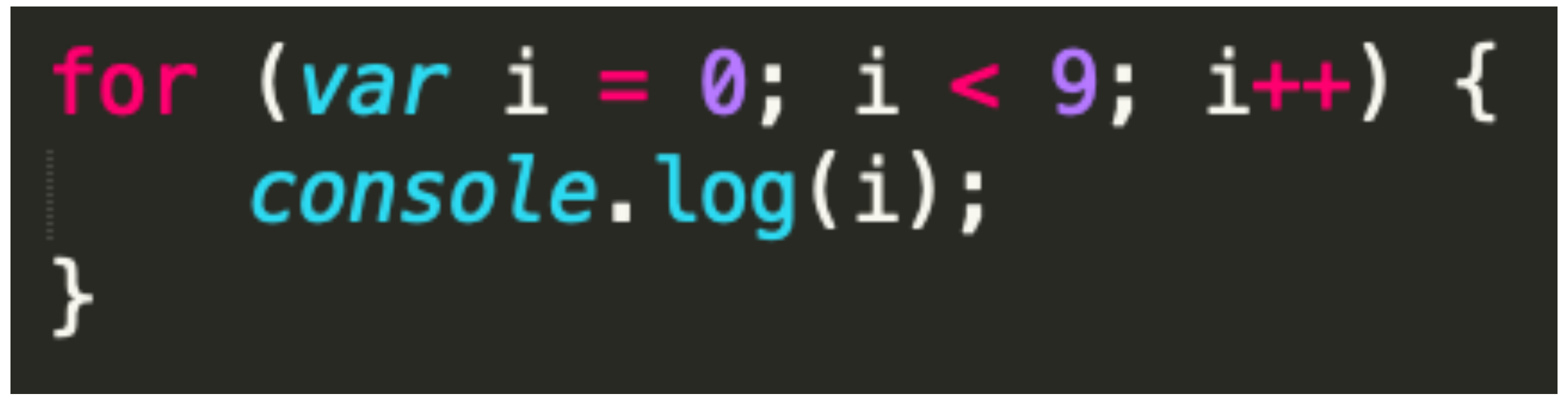


Look familiar?

Function ingredients

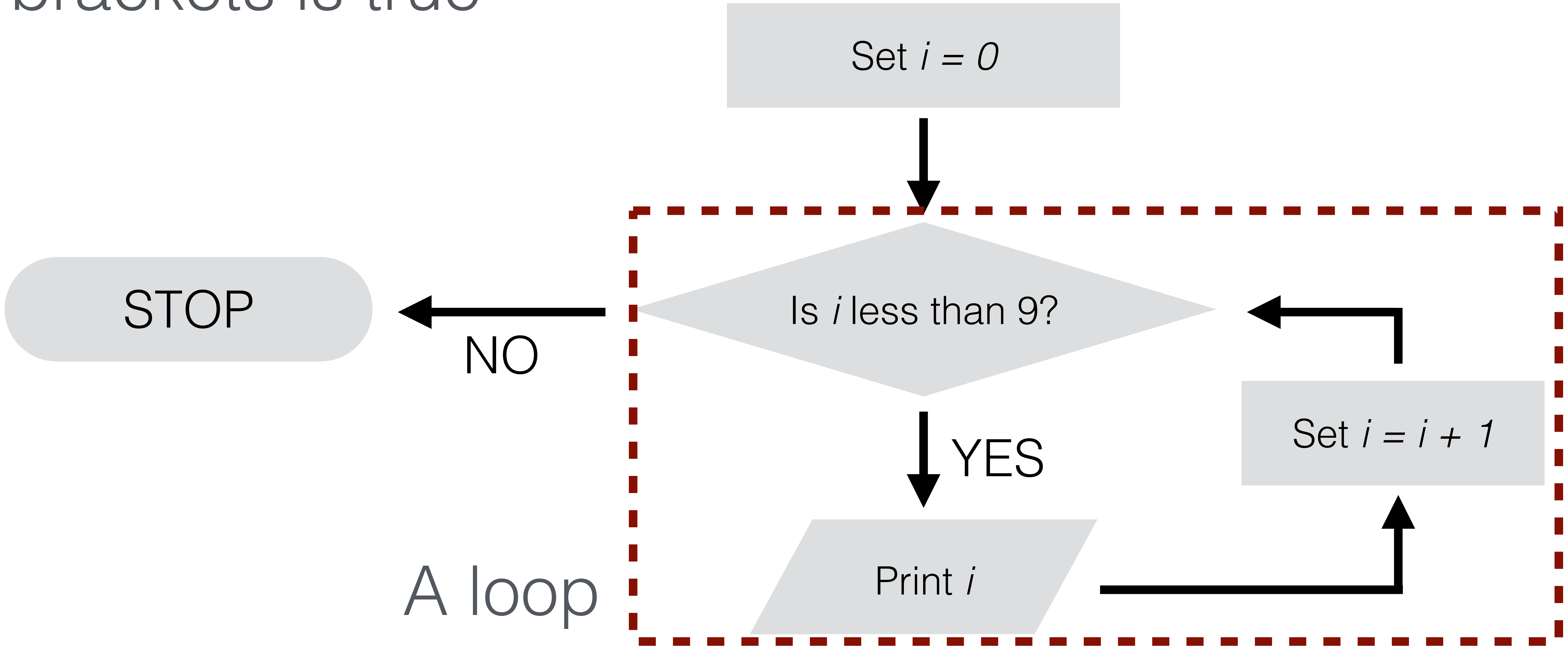


```
var i = 0;

while (i < 9) {
    console.log(i);
    i++;
}
```



```
for (var i = 0; i < 9; i++) {
    console.log(i);
}
```

While loop - repeat while condition in round brackets is true

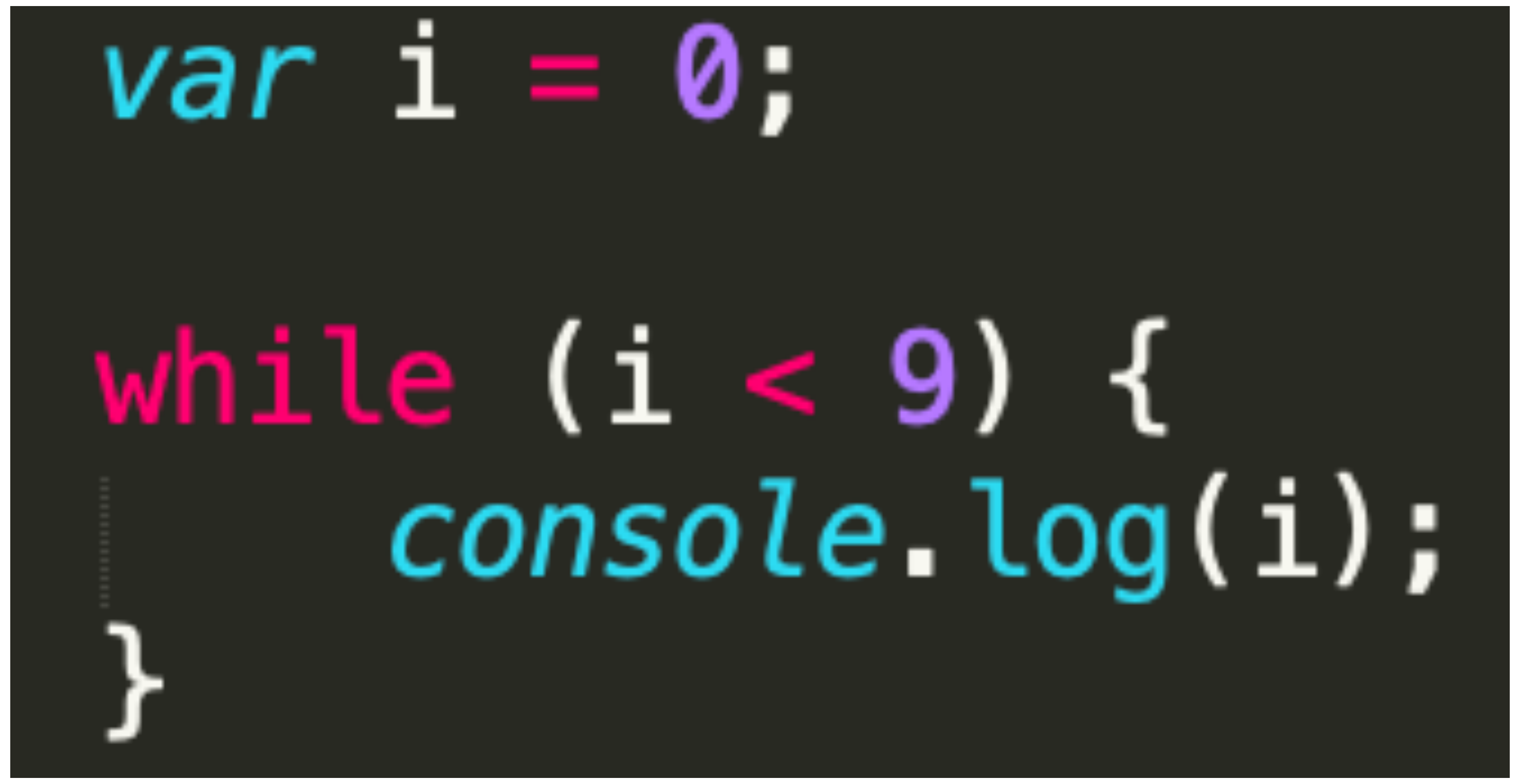


While loops

Everything you can do with a for loop, you can do with a while loop…

… and vice versa, but you need to calculate the number of iterations needed

Caution: while loops can easily lead to **infinite loops**

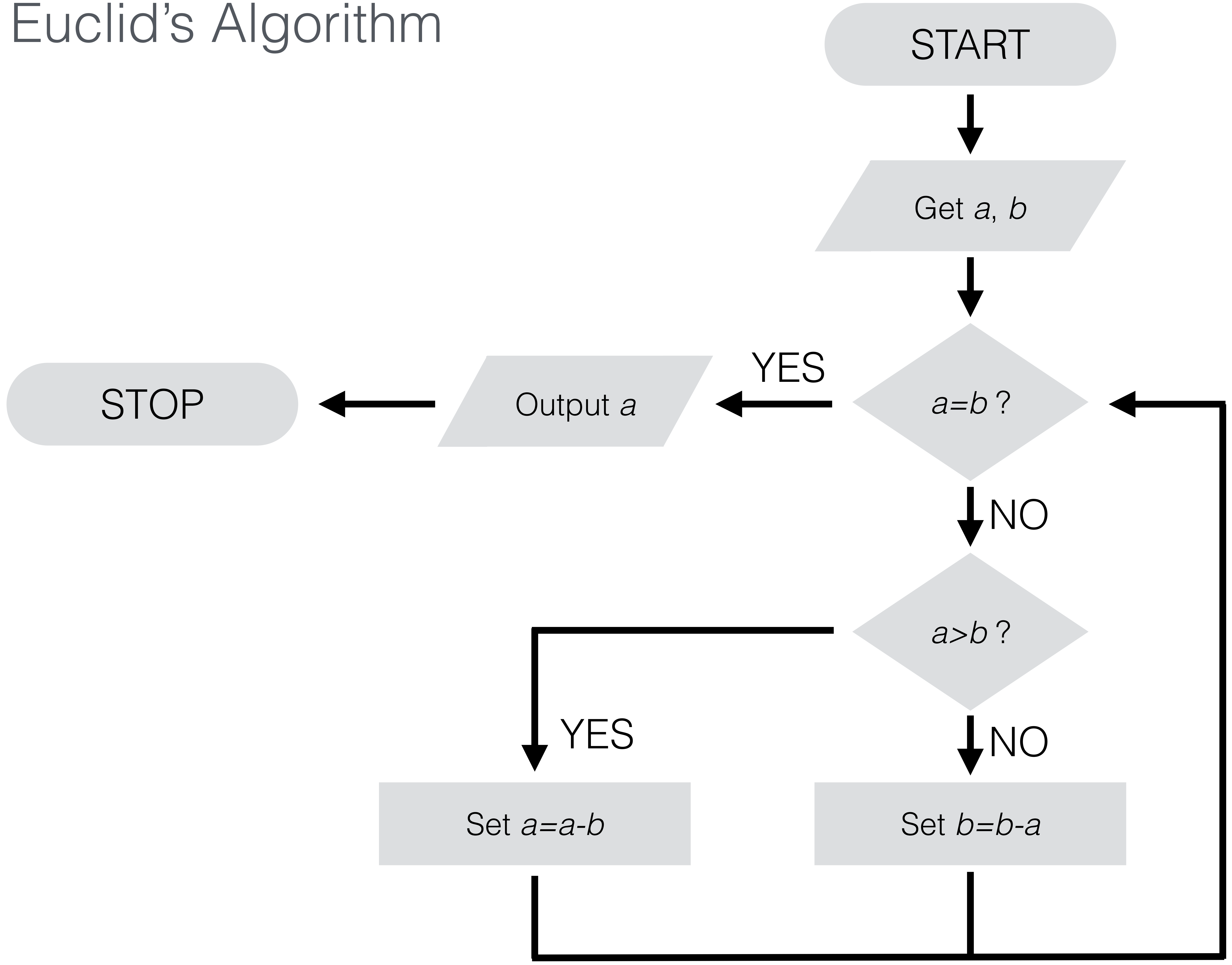


```
var i = 0;

while (i < 9) {
    console.log(i);
}
```

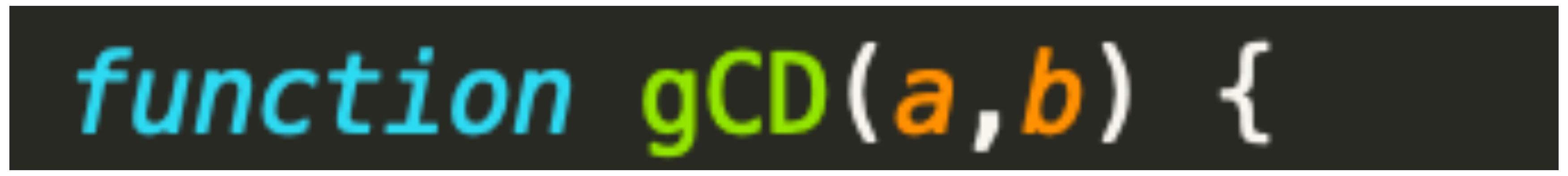
Today

1. Problems and Data Types
2. Algorithms and Functions
3. While Loops
4. **From Flowcharts to Functions**

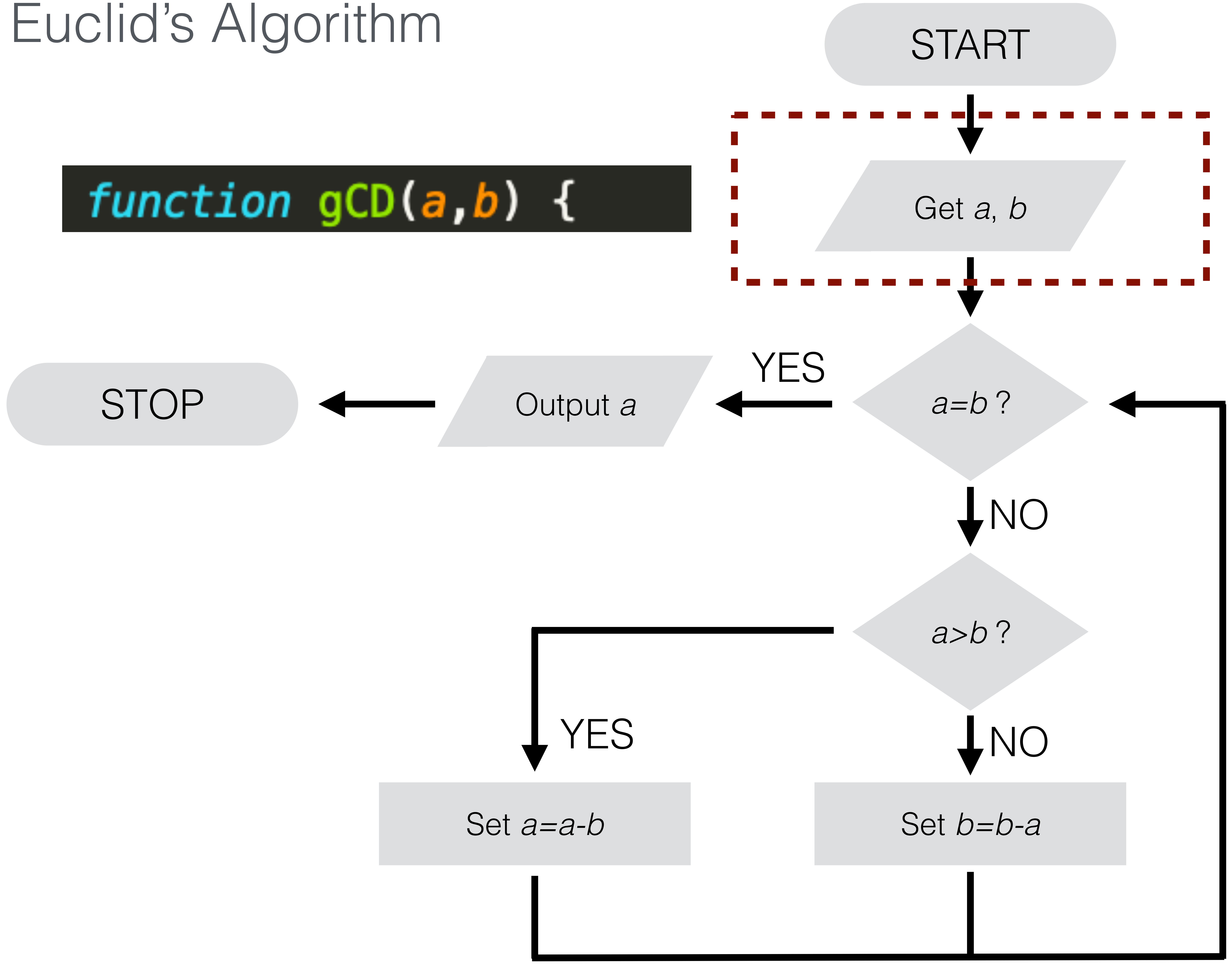
Euclid's Algorithm



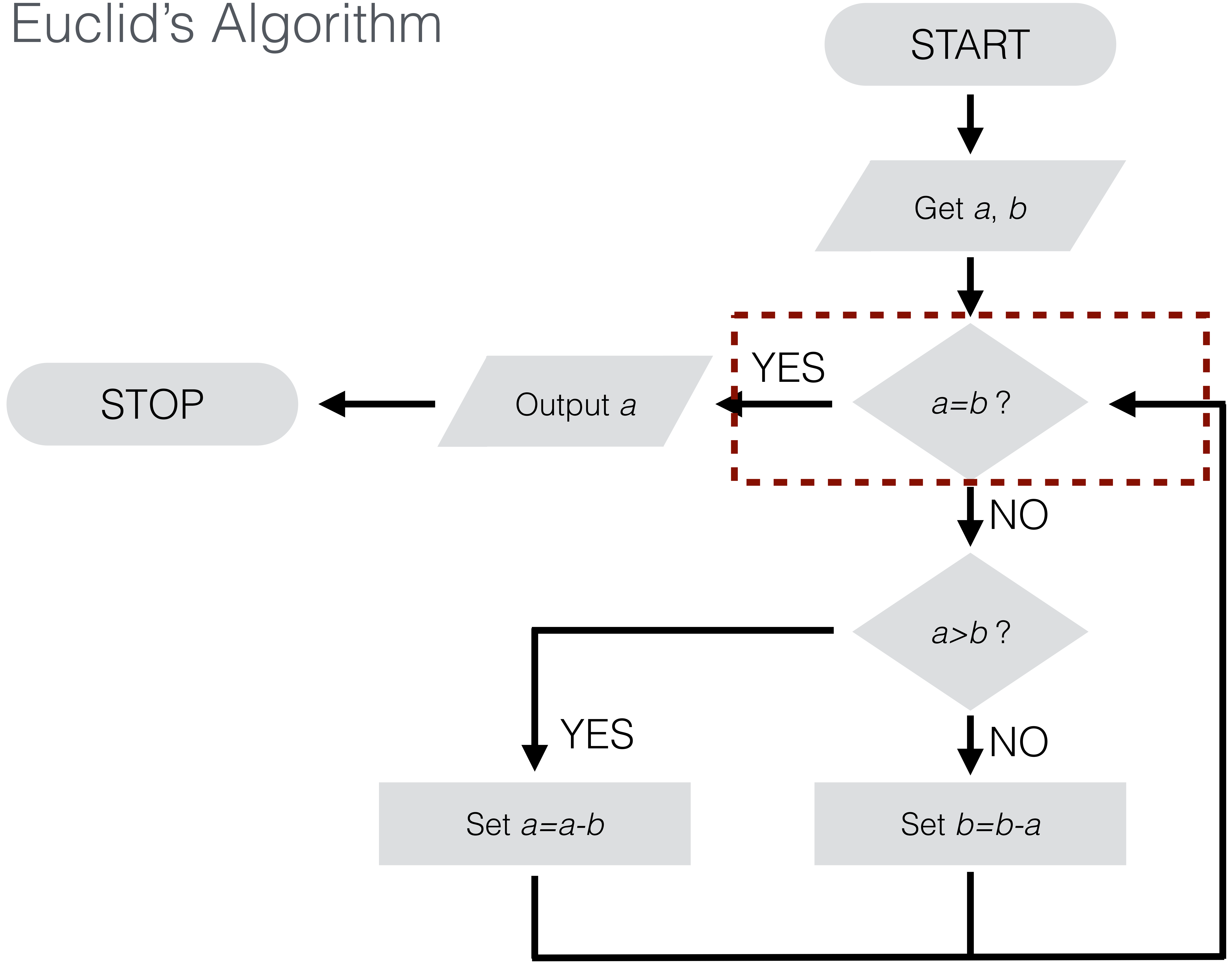
Euclid's Algorithm



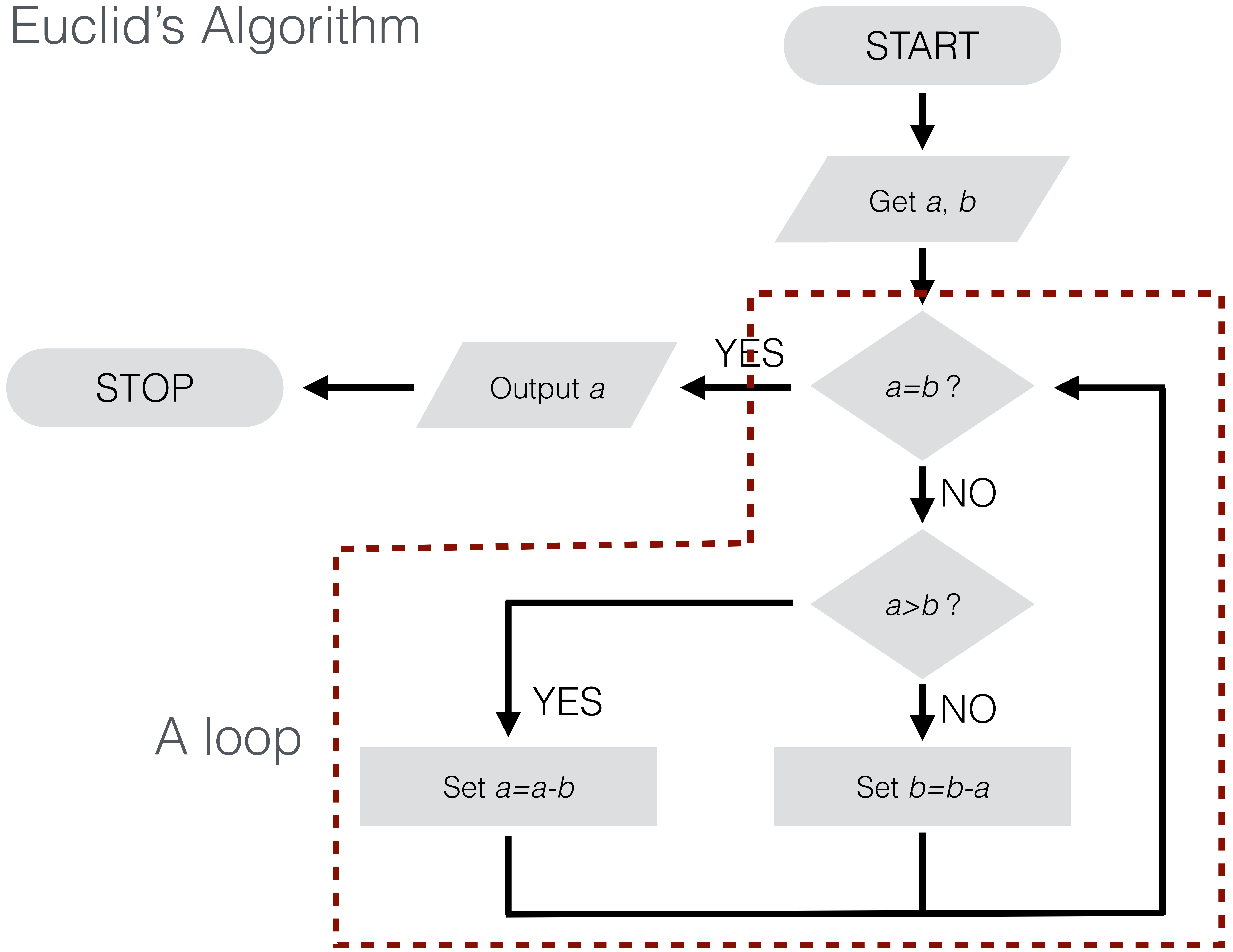
```
function gCD(a,b) {
```



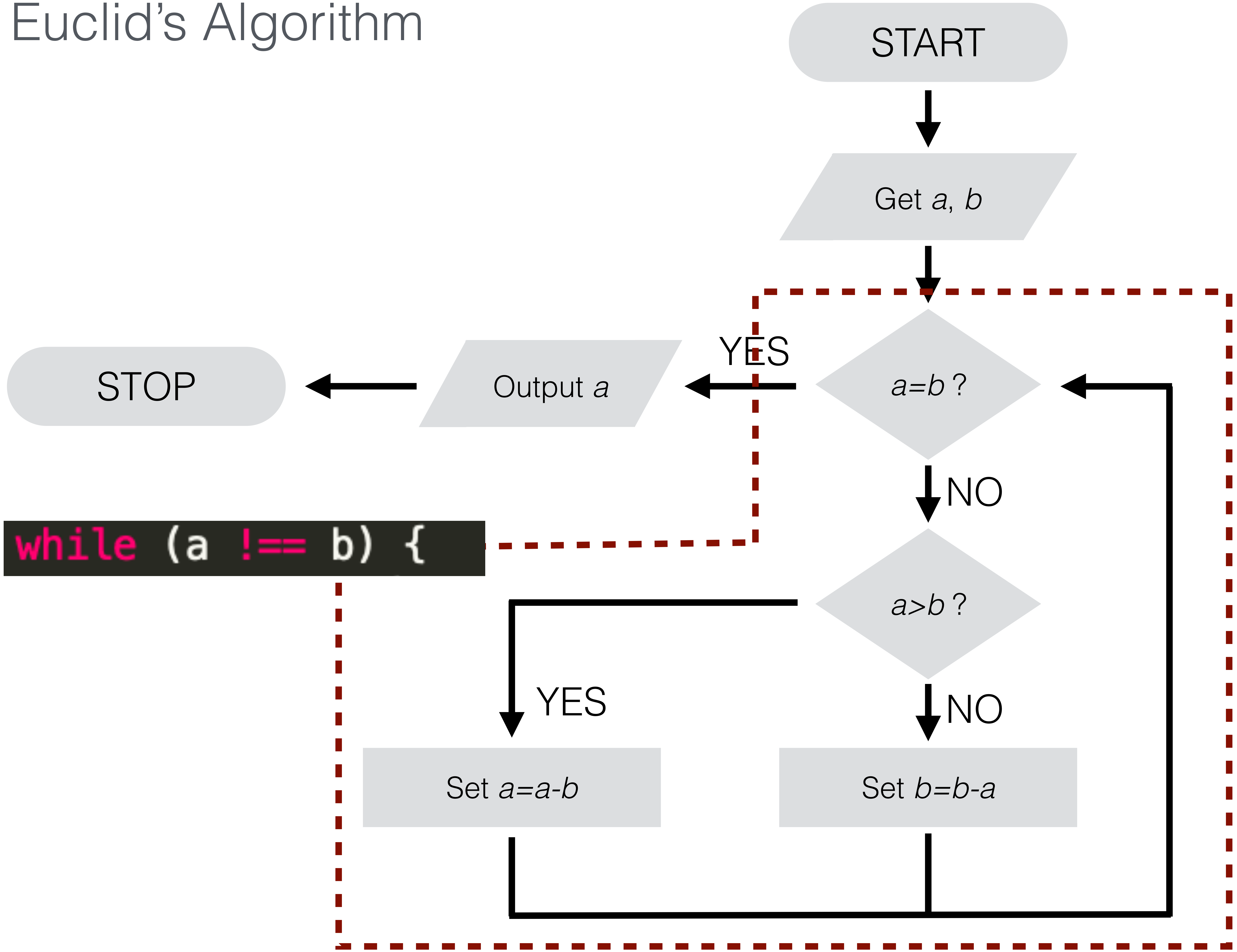
Euclid's Algorithm



Euclid's Algorithm

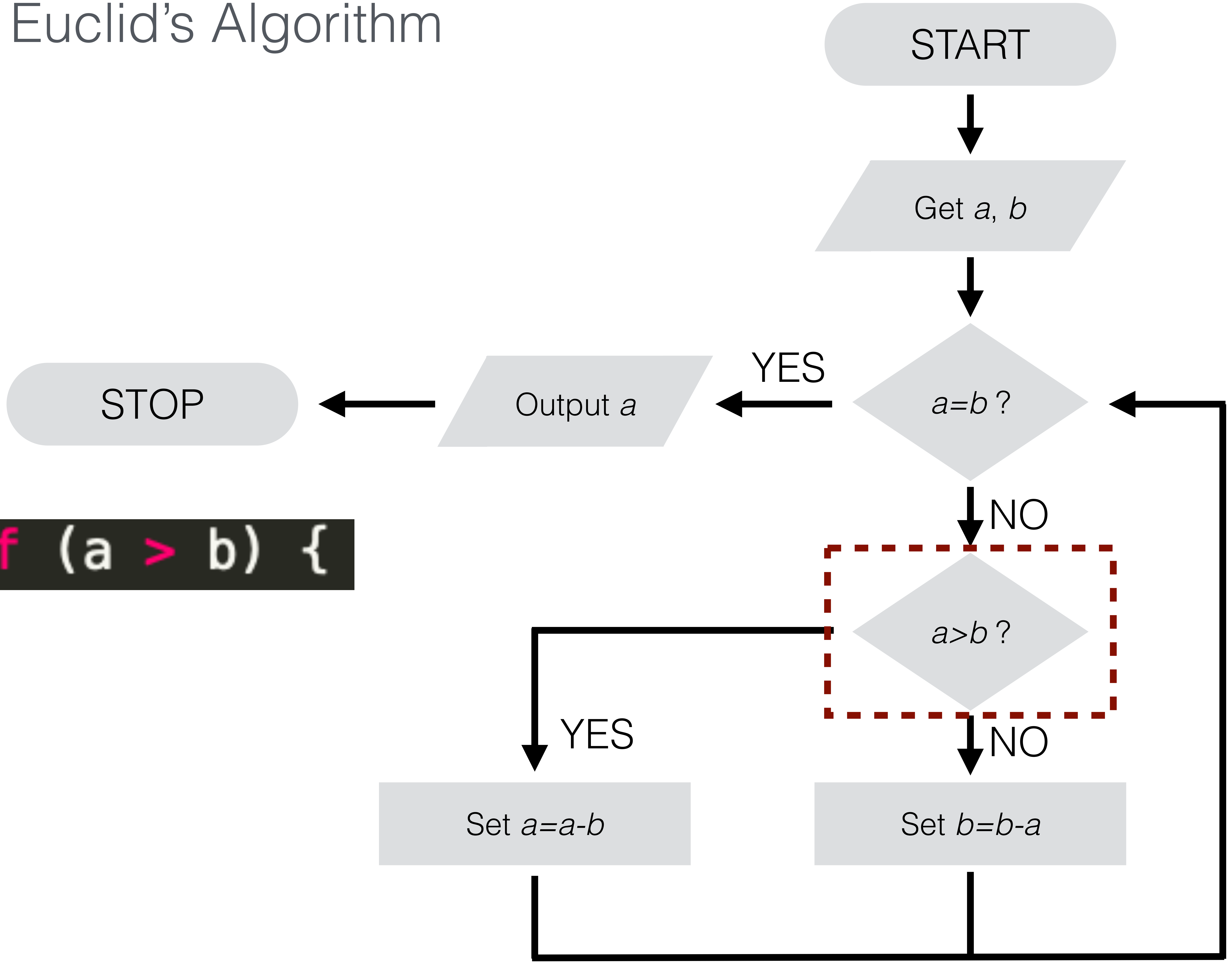


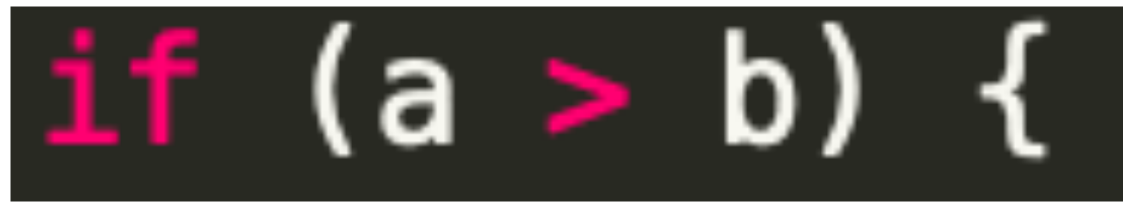
Euclid's Algorithm



```
while (a !== b) {
```

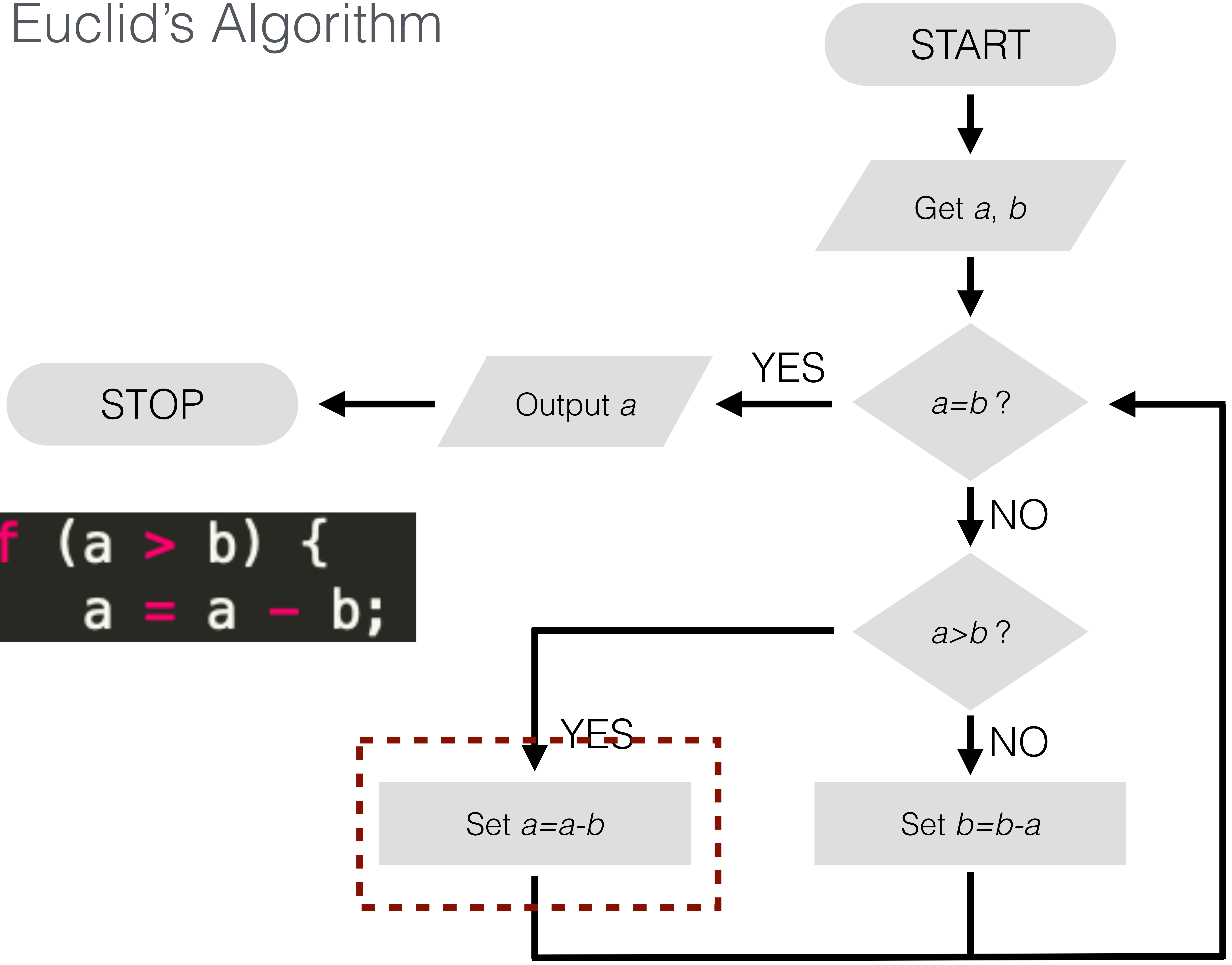
Euclid's Algorithm

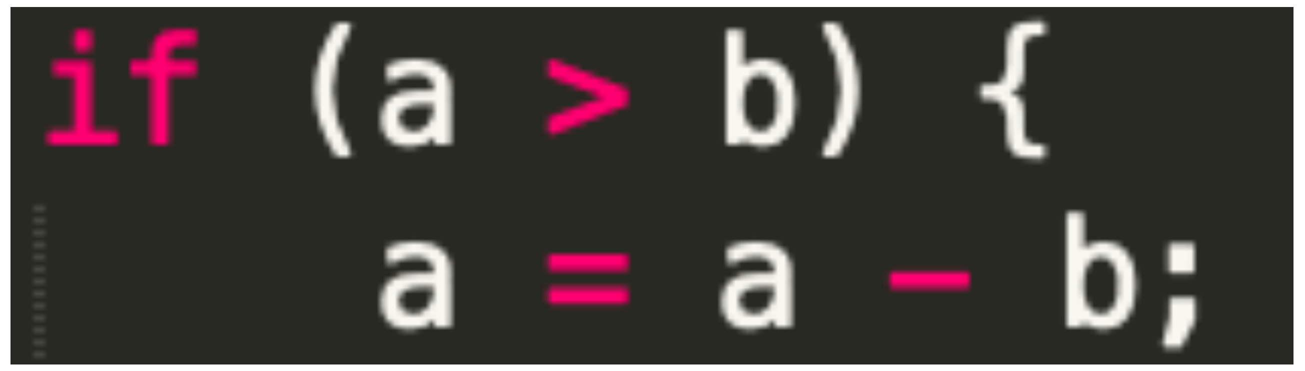




```
if (a > b) {
```

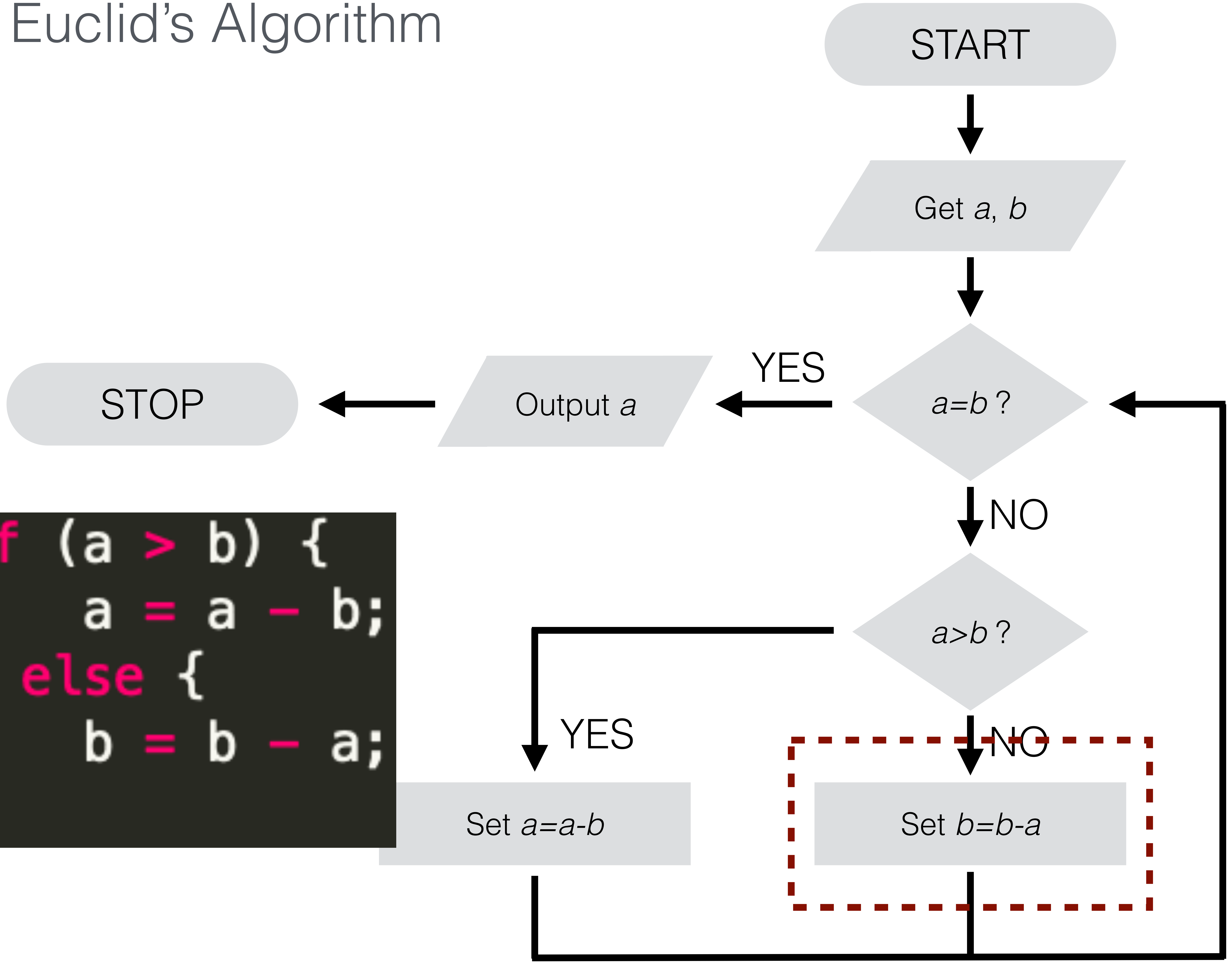
Euclid's Algorithm

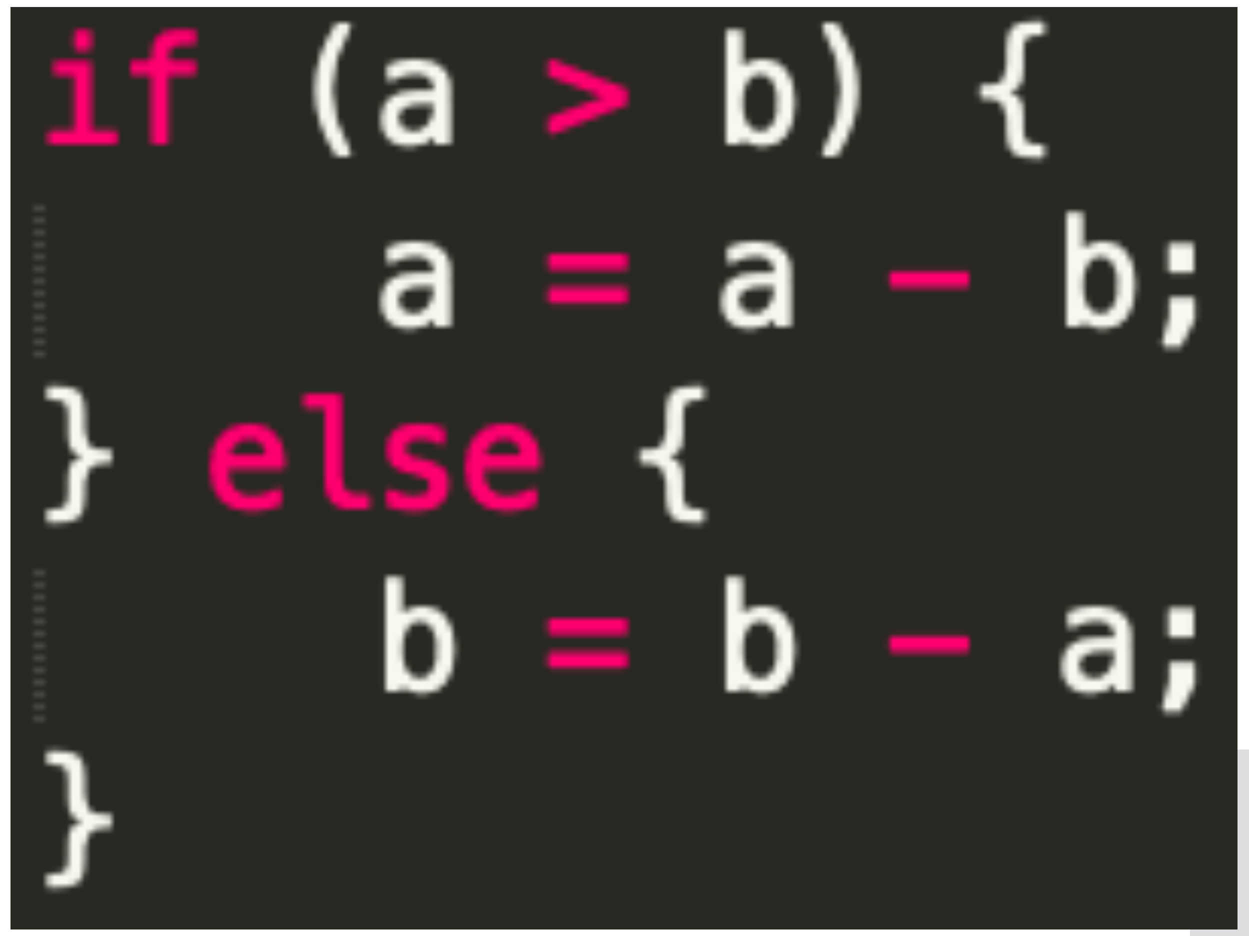




```
if (a > b) {
    a = a - b;
```

Euclid's Algorithm

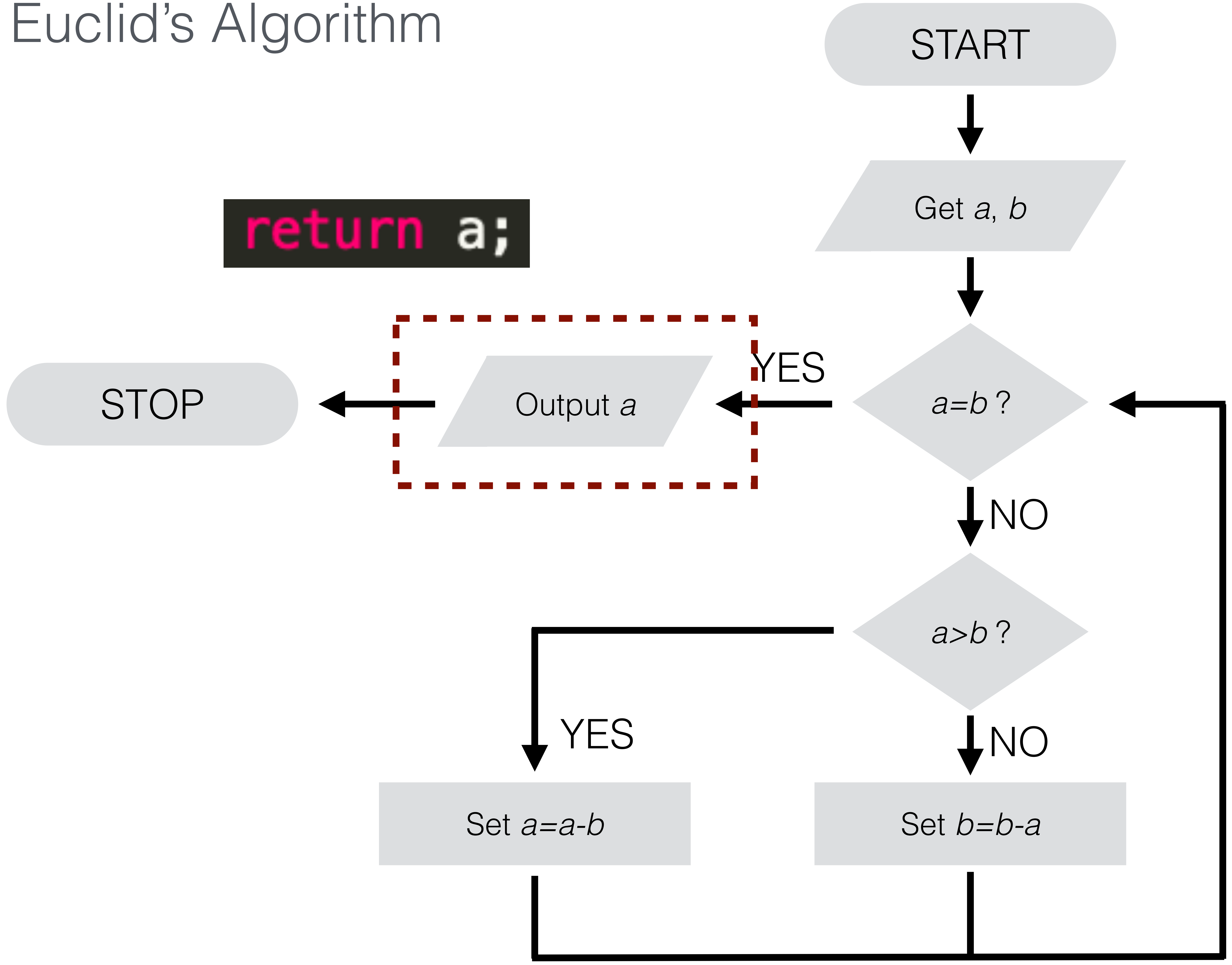




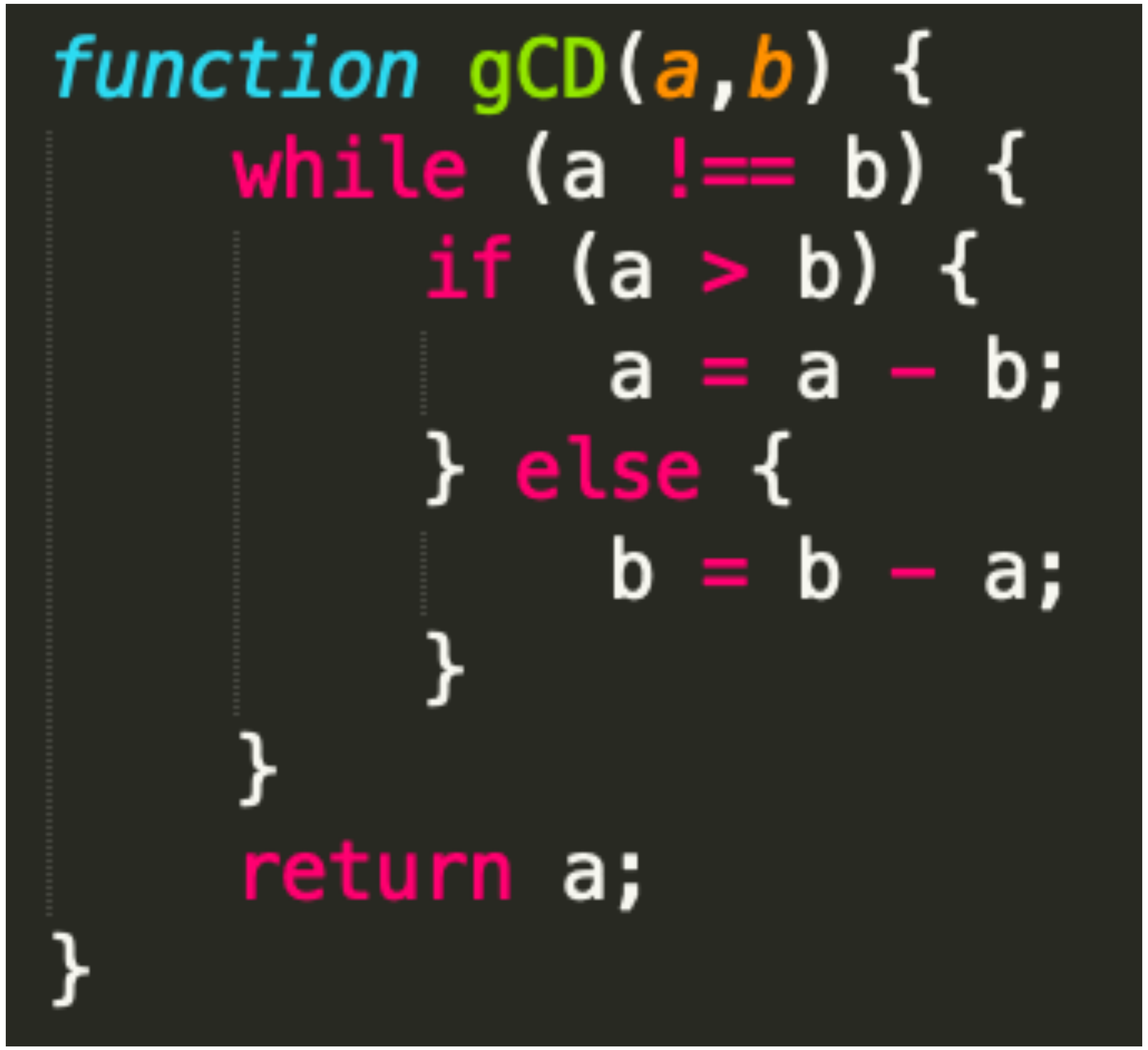
```
if (a > b) {
    a = a - b;
} else {
    b = b - a;
}
```

Euclid's Algorithm

```
return a;
```



Euclid’s Algorithm



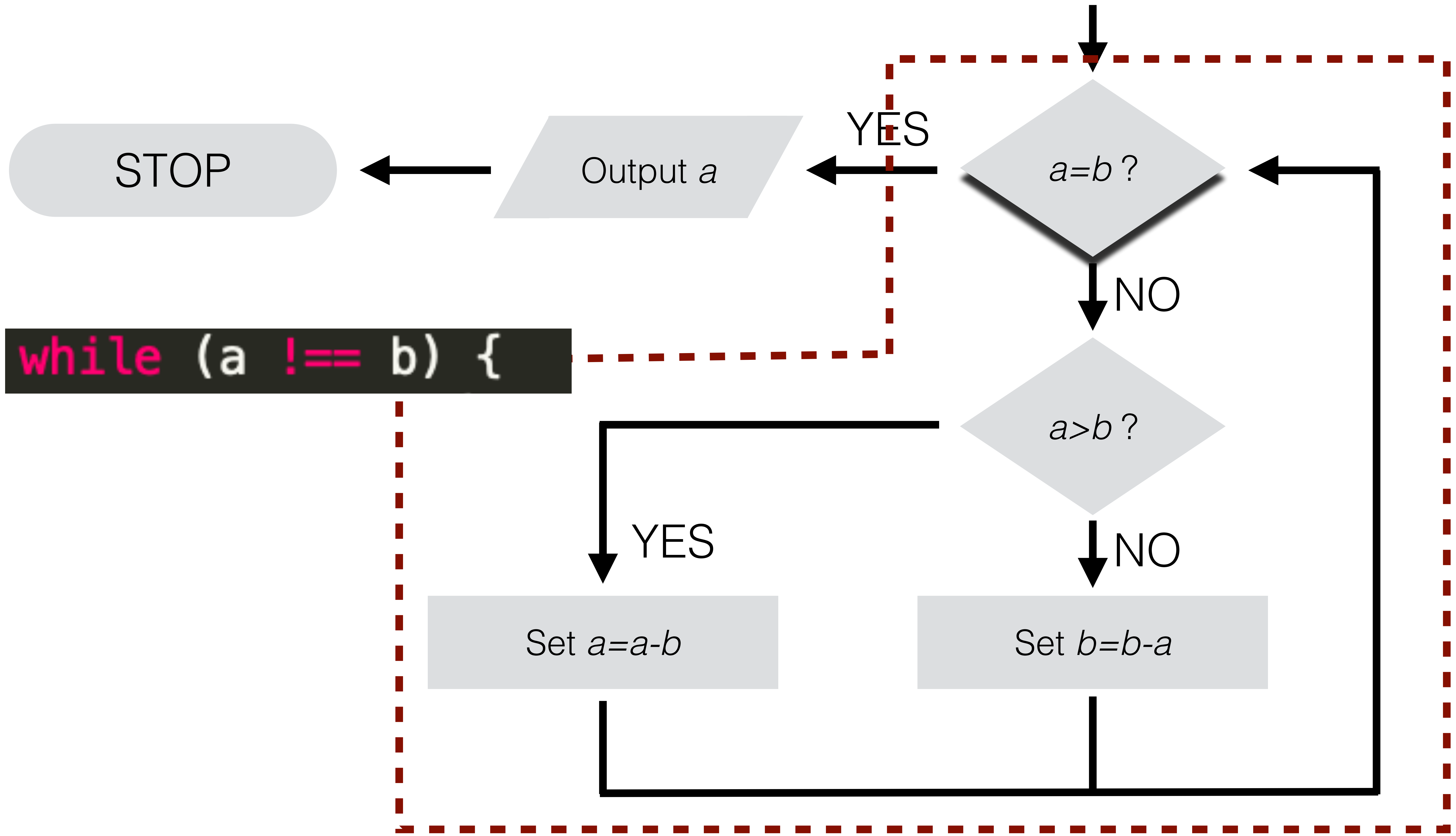
```
function gCD(a,b) {
    while (a !== b) {
        if (a > b) {
            a = a - b;
        } else {
            b = b - a;
        }
    }
    return a;
}
```

Look for decision where loops “start" in the flowchart

Good indicator of the loop condition

Look for decision where loops “start" in the flowchart

Good indicator of the loop condition



```
while (a !== b) {
```

Problem 2:

Now you need to organise a joint birthday party for two people

You are given a list from each person, each list is actually a list of friends and a list of enemies

Combine both lists into a single list of people to invite:

- Friends of both should be at the top of list
- A friend of one and enemy of the other should be at the bottom
- Enemy of both should not be on the list

Try writing some JavaScript code!
e.g. a function with two arrays as arguments