Web Developer Bootcamp

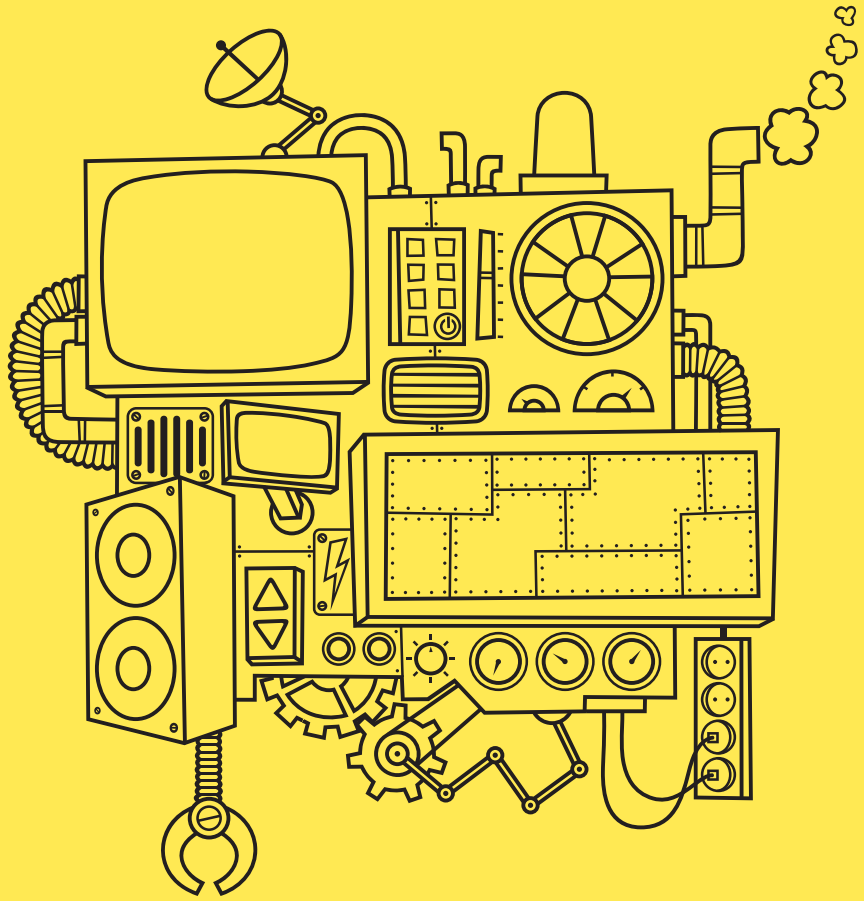
# JavaScript Functions

THE LAST "BIG" TOPIC!

# FUNCTIONS

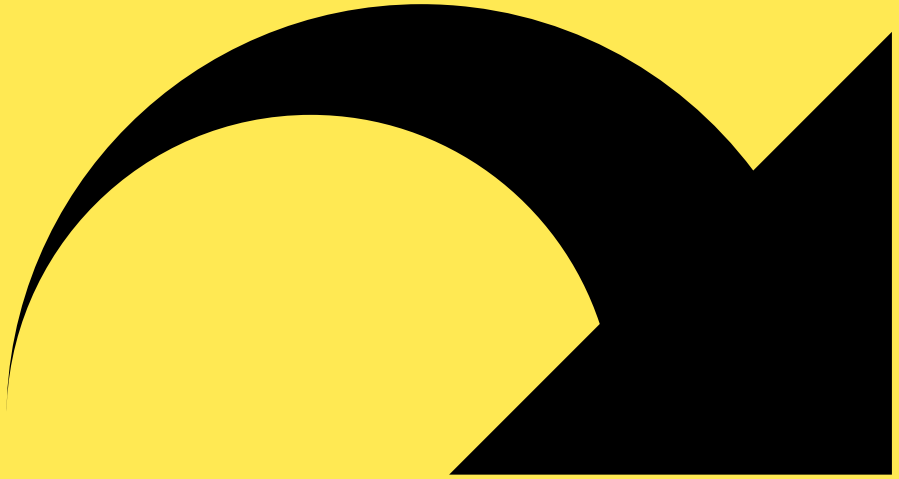
## Reusable procedures

- Functions allow us to write reusable, modular code
- We define a "chunk" of code that we can then execute at a later point.
- We use them ALL THE TIME



# 2 STEP PROCESS

DEFINE

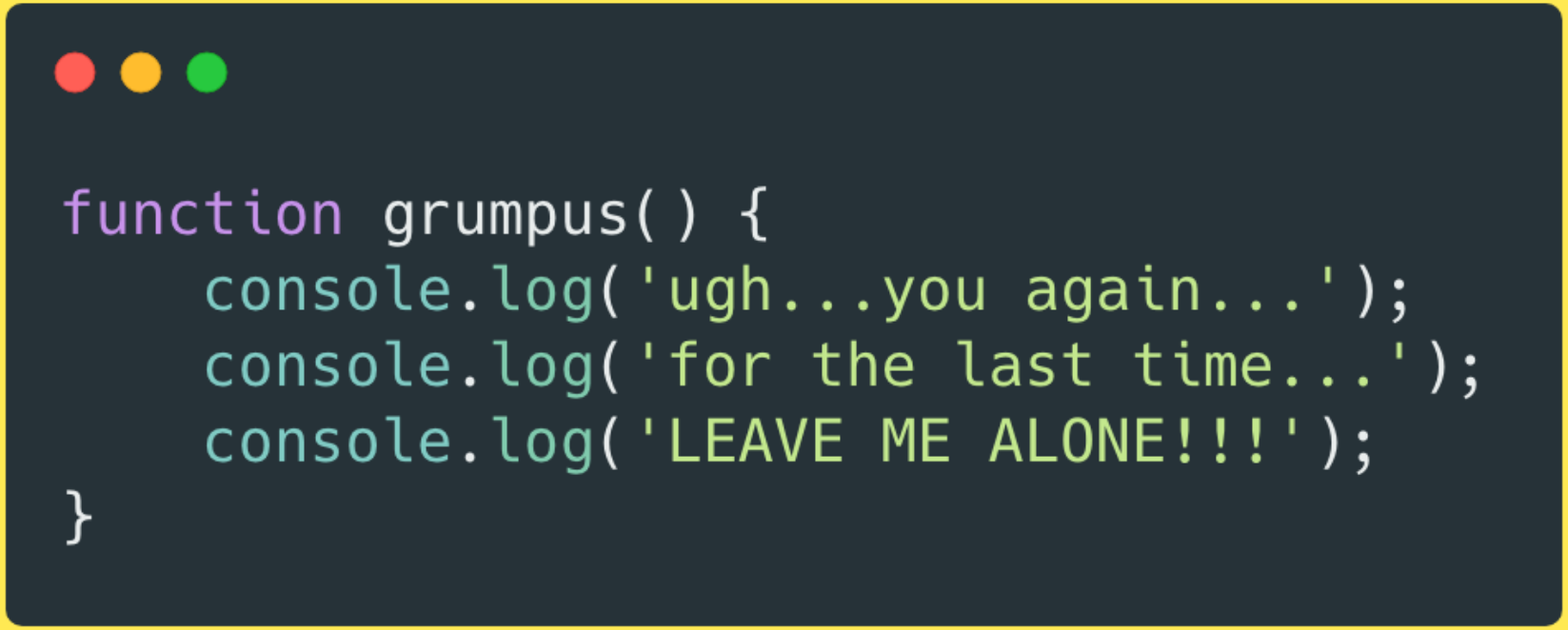


RUN

# DEFINE

```
function funcName(){
  //do something
}
```

# DEFINE



```
function grumpus() {
    console.log('ugh...you again...');
    console.log('for the last time...');
    console.log('LEAVE ME ALONE!!!');
}
```

# RUN

```
funcName();  //run once

funcName(); //run again!
```

# RUN

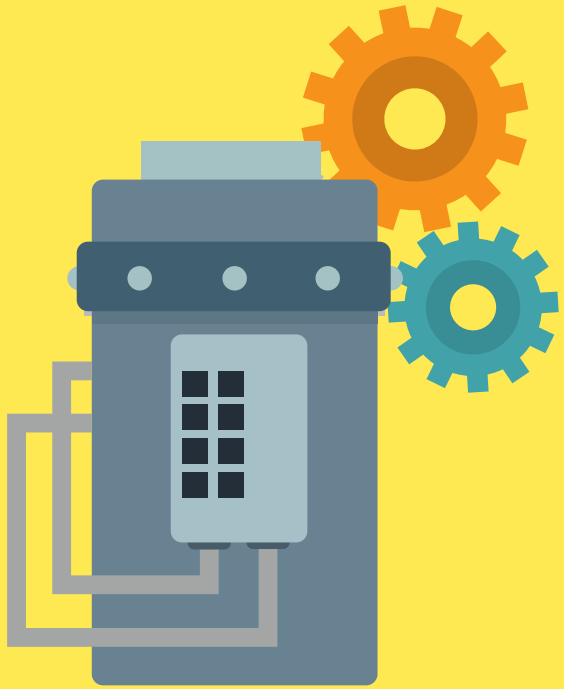
```
grumpus();
//ugh...you again...
//for the last time...
//LEAVE ME ALONE!!!

grumpus();
//ugh...you again...
//for the last time...
//LEAVE ME ALONE!!!
```

# ARGUMENTS

# INPUTS

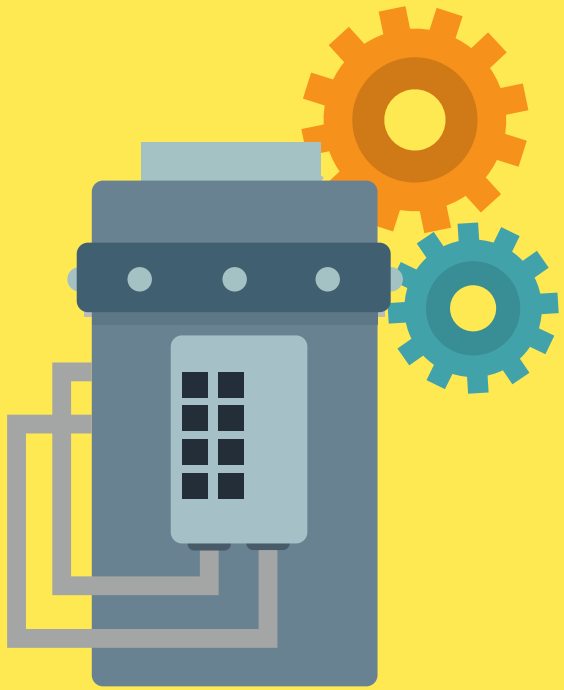
Right now, our simple functions accept zero inputs. They behave the same way every time.

# NO INPUTS

`grumpus()` →  → 

`grumpus()` → → 

# NO INPUTS

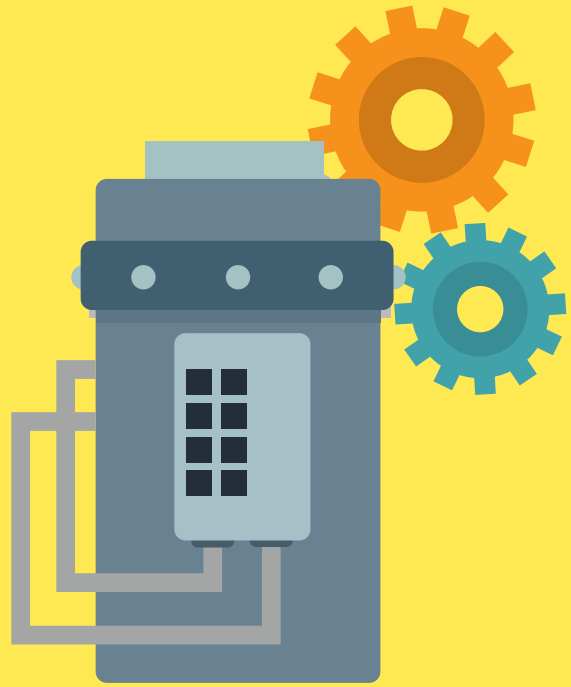
`greet()` →  → `"Hi!"`

`greet()` → → `"Hi!"`

# ARGUMENTS

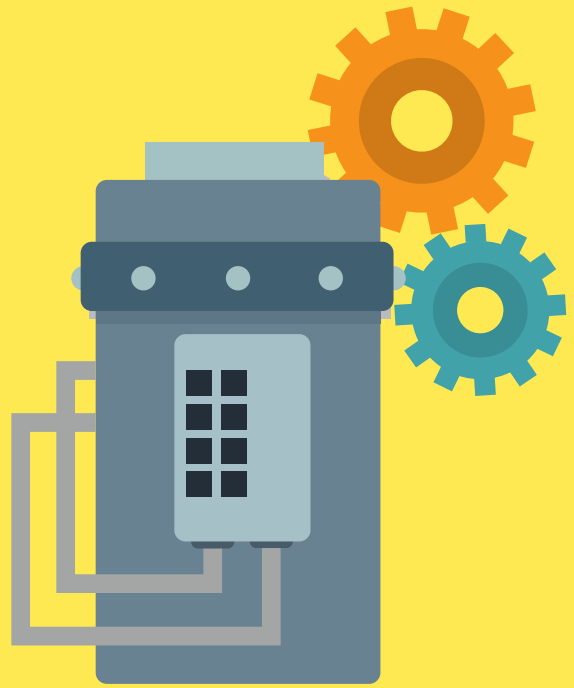
We can also write functions that accept inputs, called arguments

# ARGUMENTS

`greet('Tim')` →  → `"Hi Tim!"`

`greet('Anya')` → → `"Hi Anya!"`

# ONE MORE

`avg(20,25)` →  → 22.5

`avg(3,2,5,6)` → → 4

# We've seen this before

No inputs

```
//No input
"hello".toUpperCase();
```

Arguments!

```
//Different inputs...
"hello".indexOf('h'); //0
//Different outputs...
"hello".indexOf('o'); //4
```

# GREET TAKE 2

```
function greet(person) {
  console.log(`Hi, ${person}!`);
}
```

```
greet('Arya');
```

→ "Hi, Arya!"

```
greet('Ned');
```

→ "Hi, Ned!"

# 2 ARGUMENTS!

```
function findLargest(x, y) {
    if (x > y) {
        console.log(`${x} is larger!`);
    }
    else if (x < y) {
        console.log(`${y} is larger!`);
    }
    else {
        console.log(`${x} and ${y} are equal!`);
    }
}
```

```
findLargest(-2,77)
```

"77 is larger!"

```
findLargest(33,33);
```

"33 and 33 are equal"

# RETURN

Built-in methods **return** values
when we call them.
We can store those values:

```
const yell = "I will end you".toUpperCase();

yell; //"I WILL END YOU"

const idx = ['a','b','c'].indexOf('c');

idx; //2
```

# NO RETURN!

Our functions print values out, but do NOT return anything

```
function add(x, y) {
    console.log(x + y);
}

const sum = add(10, 16);
sum; //undefined
```

# FIRST RETURN!

Now we can capture a return value in a variable!

```
function add(x, y) {
    return x + y; //RETURN!
}

const sum = add(10, 16);
sum; //26

const answer = add(100, 200);
answer; //300
```

# RETURN

The return statement ends function execution AND specifies the value to be returned by that function