FUNCTIONS

AGENDA

- Functions 101
  - Body, argument, return
  - Scope
  - Olika funktions deklarationer
    - Function declaration, function expression, arrow function
  - Callstack
  - Optional arguments
- Övningar

FUNCTIONS

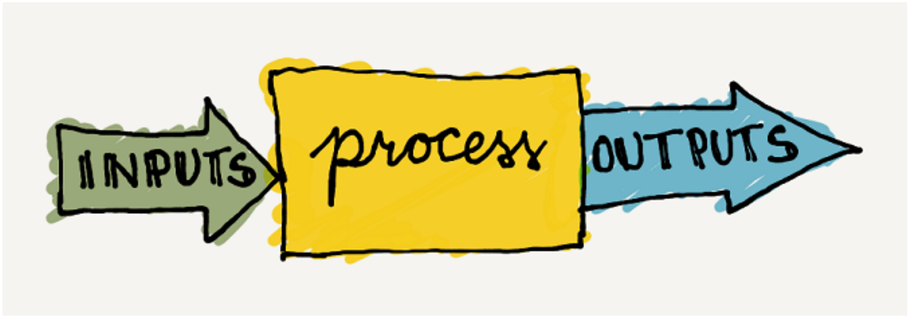
- Funktioner är den viktigaste beståndsdelen i JavaScript
- Funktionen är det som möjliggör **abstraktion**
  - Abstraktion är det som möjliggör att vi kan tänka i större perspektiv
  - Exempel på abstraktion:
    - Dagar, Veckor, År,
    - Lärare, studenter, Skola, Kurser
    - Länder, Världsdelar

FUNCTIONS

- En funktion

```
const square = function(x) {
  return x * x;
};

console.log(square(12));
// → 144
```



FUNCTIONS

- En funktion

```
const square = function(x) {
  return x * x;
};

console.log(square(12));
// → 144
```

  - **square** är funktionen
  - **{ ... }** är funktions kropp (body)
  - **x** anger input (argument)
    - fungerar som variabel
  - **return** anger output

FUNCTIONS

- Vi kan anropa funktionen flera gånger med olika input
- Vi kan ha funktioner utan input (inga argument)
- Vi kan ha funktioner utan output (ingen return)
  - Dessa funktioner brukar då istället ha en sidoeffekt (ex: console.log)

FUNCTIONS

- Uppgift
  - Skapa en funktion (hello) som tar ett argument (name)
  - Och returnerar strängen `Hi ${name}, how are you?`
  - Exekvera funktionen med ditt namn och printa ut resultatet
  - Skriv om funktionen så att den printar ut direkt istället för returnera sträng
  - Exekvera funktionen

FUNCTIONS

- Uppgift
  - Skriv en funktion (sayMyName) som tar 2 argument (name) och (nr)
    - Funktionen ska i en for-loop printa ut (name) ett antal (nr) gånger

FUNCTIONS

- Skriv en funktion ”calculate” som inte tar argument. Funktionen ska
  - använda prompt(”Ange första talet”) 1 gång
  - använda prompt(”Ange operatorn”) 1 gång
    - Använd **if else** för att utröna operator
  - använda prompt(”Ange andra talet”) 1 gång
- Sedan ska ”calculate” returnera svaret
  - med input ”1”, ”+”, ”3” ska funktionen returnera 4
  - med input ”2”, ”*”, ”4” ska funktionen returnera 8

FUNCTIONS

- Uppgift
  - Skriv funktionen ”calculate” som inte tar argument. Funktionen ska
    - använda prompt(”Ange första talet”) 1 gång
    - använda prompt(”Ange operatorn”) 1 gång
      - Använd **switch** istället för **if else** för att utröna operator
    - använda prompt(”Ange andra talet”) 1 gång
  - Sedan ska ”calculate” returnera svaret
    - med input ”1”, ”+”, ”3” ska funktionen returnera 4
    - med input ”2”, ”*”, ”4” ska funktionen returnera 8

FUNCTIONS

- Frågor
  - Vad är ”function body”?
  - Vad är argument?
  - Vad gör return?
  - Vad är en sidoeffekt?

FUNCTIONS

- Scope
  - Varje variabel (**var**, **let**, **const**) har ett ”scope”, som anger synligheten
    - En variabel i funktion har ***lokalt scope*** inuti funktion (syns endast där)
    - En variabel utanför funktion har ***globalt scope*** (syns överallt)
  - En funktion har sin egna värld med variabler (local environment)

FUNCTIONS

- Scope
  - Det är inte bara i funktioner vi har lokalt scope, också i while, for, och if-else
  - Ett scope kan titta ut
  - Ett scope kan inte titta in
  - För **var** variabler gäller dock inte detta

```
let x = 10;
if (true) {
  let y = 20;
  var z = 30;
  console.log(x + y + z);
  // → 60
}
// y is not visible here
console.log(x + z);
// → 40
```

FUNCTIONS

- Vi vill göra en funktion som ger hummus recept…
- Vad gör funktionen, och hur påverkar factor?

```
const hummus = function(factor) {
  const ingredient = function(amount, unit, name) {
    let ingredientAmount = amount * factor;
    if (ingredientAmount > 1) {
      unit += "s";
    }
    console.log(`${ingredientAmount} ${unit} ${name}`);
  };
  ingredient(1, "can", "chickpeas");
  ingredient(0.25, "cup", "tahini");
  ingredient(0.25, "cup", "lemon juice");
  ingredient(1, "clove", "garlic");
  ingredient(2, "tablespoon", "olive oil");
  ingredient(0.5, "teaspoon", "cumin");
};
```

- Hjälpfunktion ”ingredient” kan se (factor)
- Yttre funktion ”hummus” kan inte se (ingredientAmount)

FUNCTIONS

- Uppgift
  - Skapa en funktion ”thiefControl” som tar argument boolean (isThief)
  - Lägg I funktion **if else** som kontrollerar värdet på argument
  - I if body ({ … }) skapa en let variabel (message) ”your a thief”
  - I else body ({ … }) skapa en let variabel (message) ”your fine”
  - Försök efter if else att printa ut message, vad händer? Hur fixar vi detta?

FUNCTIONS

- Frågor
  - Vad är **global scope**?
  - Vad är **local scope**?
  - Vad är regeln kring variabler inuti ett scope?
  - Vad särskiljer **var** variabler från **let** och **const**?

FUNCTIONS

- Functions as values
  - let variabler med funktioner kan bytas ut

```
let launchMissiles = function() {
  missileSystem.launch("now");
};
if (safeMode) {
  launchMissiles = function() {/* do nothing */};
}
```

  - Dock inte const

FUNCTIONS

- Uppgift
  - Skriv en booleansk let variabel “isMyLuckyDay” och tilldela den true
  - Skriv en if else som tilldelar 2 olika godtyckliga funktioner till en variabel mysteryFunction
  - Exekvera mysteryFunction
  - Ändra “isMyLuckyDay” till false
  - Copy/pasta if else från tidigare
  - Exekvera mysteryFunction igen

FUNCTIONS

👉 **Function declaration**

Function that can be used before it's declared

👉 **Function expression**

Essentially a function *value* stored in a variable

👉 **Arrow function**

Great for a quick one-line functions. Has no `this` keyword (more later...)

```
function calcAge(birthYear) {
  return 2037 - birthYear;
}

const calcAge = function (birthYear) {
  return 2037 - birthYear;
};

const calcAge = birthYear ⟹ 2037 - birthYear;
```

☝️ Three different ways of writing functions, but they all work in a similar way: receive **input** data, **transform** data, and then **output** data.

FUNCTIONS

```
function calcAge(birthYear) {
  return 2037 - birthYear;
}
```

- Function declaration
  - Vi behöver ingen variabel för att hålla
  - Funktionen kan definieras efter att den har deklarerats (!)

FUNCTIONS

- Function declaration

```
console.log("The future says:", future());

function future() {
  return "You'll never have flying cars";
}
```

- Dessa funktioner är inte del av “top-to-bottom” flow av kod exekvering
- Dessa funktioner flyttas automatiskt upp längst upp i scopet när JS motor exekverar kod, så “future” hamnar ovanför console.log instruktionen

FUNCTIONS

```
const calcAge = function (birthYear) {
  return 2037 - birthYear;
};
```

- Function expression
  - Det funktionsuttryck vi hittills har arbetat med
  - Tilldelas till variabel, och har semikolon

FUNCTIONS

```
const calcAge = birthYear ⇒ 2037 - birthYear;
```

- Arrow function
  - Kan skippa () om argument är endast 1
  - Kan skippa { } om "function body" har en rad

```
const square1 = (x) => { return x * x; };
const square2 = x => x * x;
```

  - Annars fungerar som function expression

FUNCTIONS

- Arrow function
  - Börja med function deklaration -> function expression -> arrow function
  - Men kan definieras med { } och () som vi såg

```
const power = (base, exponent) => {
  let result = 1;
  for (let count = 0; count < exponent; count++) {
    result *= base;
  }
  return result;
};
```

FUNCTIONS

- Uppgift
  - Definiera en “function declaration” (hoursBeforeNewYear)
    - Argument: 2 variabler, (weeks) och (days)
    - Returnerar: försök själv… (på en rad)
  - Skriv om funktionen som “function expression”
  - Skriv om funktionen som “arrow function” med { }
  - Skriv om funktionen som “arrow function” utan { }

```
function calcAge(birthYear) {
  return 2037 - birthYear;
}

const calcAge = function (birthYear) {
  return 2037 - birthYear;
};

const calcAge = birthYear ⇒ 2037 - birthYear;
```

FUNCTIONS

- The call stack

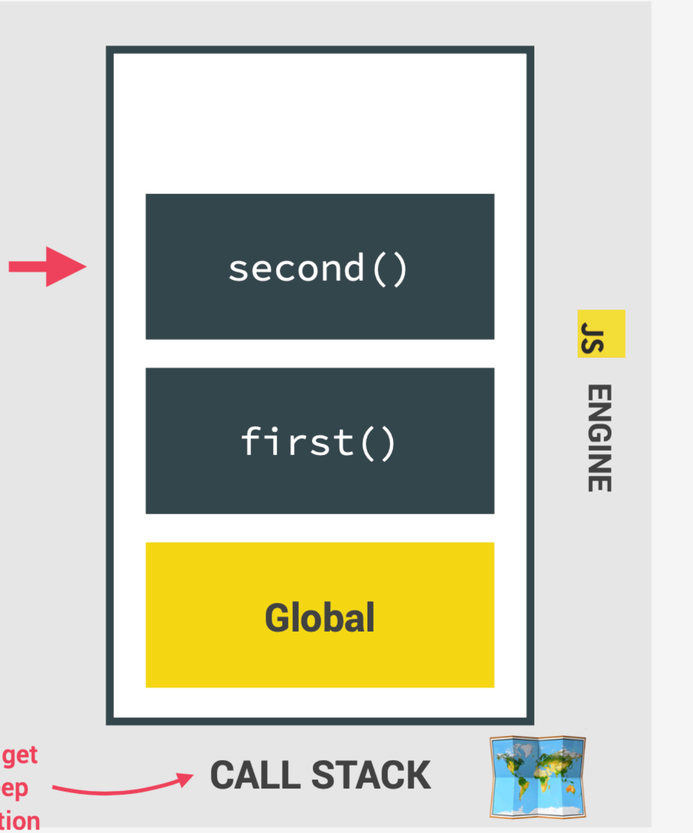
👉 Compiled code starts execution

```
const name = 'Jonas';

const first = () => {
  let a = 1;
  const b = second(7, 9);
  a = a + b;
  return a;
};

function second(x, y) {
  var c = 2;
  return c;
}

const x = first();
```



- Visa exempel i chrome debugger

FUNCTIONS

- Följande kod

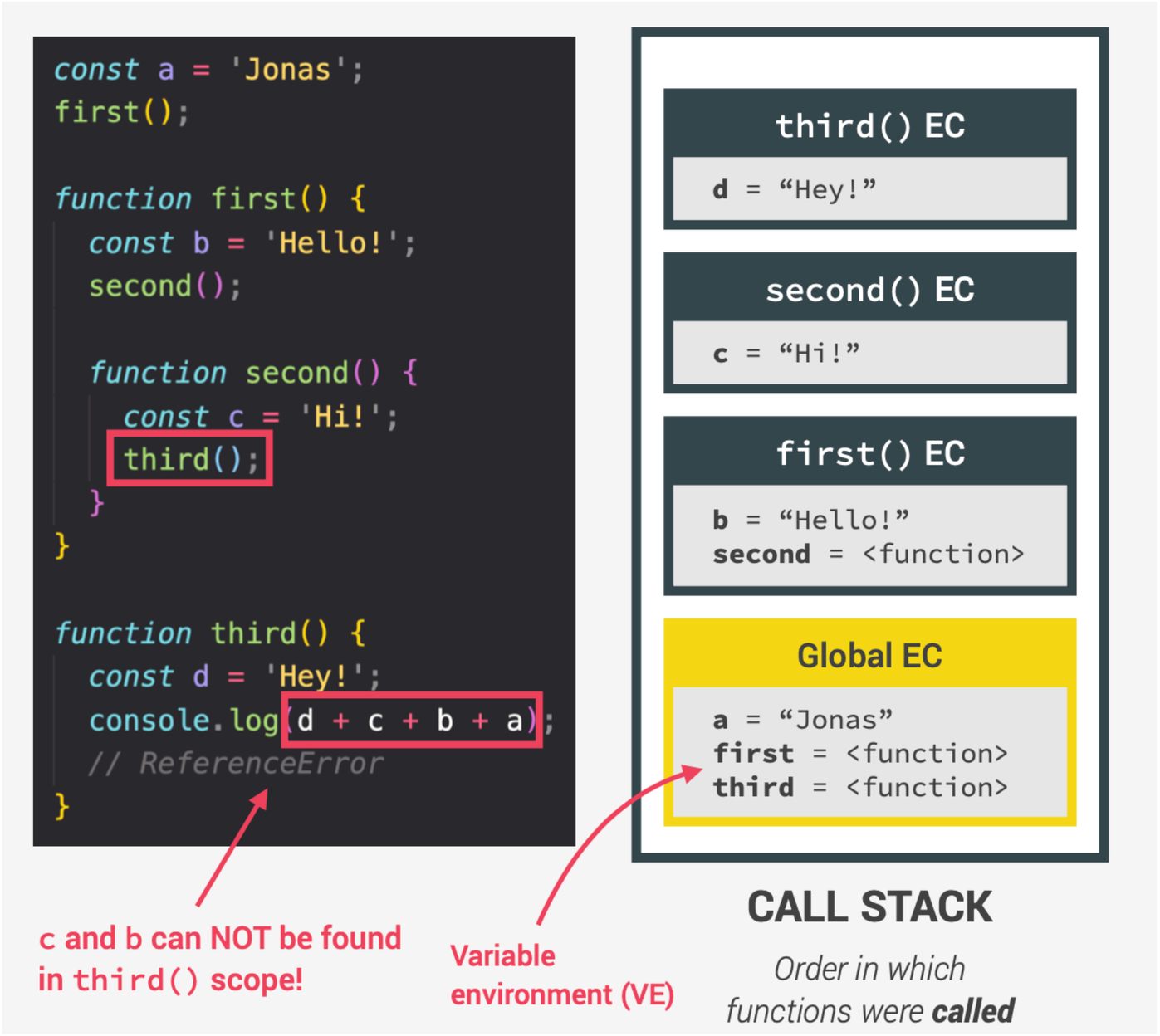
```
function greet(who) {
  console.log("Hello " + who);
}
greet("Harry");
console.log("Bye");
```

- Exekvering

```
not in function
   in greet
        in console.log
   in greet
not in function
   in console.log
not in function
```

FUNCTIONS

- The call stacks



FUNCTIONS

- Fråga
  - Vad händer i följande kod

```
function chicken() {
  return egg();
}
function egg() {
  return chicken();
}
console.log(chicken() + " came first.");
// → ??
```

  - Försök rita upp Exekvering

FUNCTIONS

- Optional arguments
  - Följande kod fungerar fint

```
function square(x) { return x * x; }
console.log(square(4, true, "hedgehog"));
// → 16
```

  - Vilket medför att vi kan skriva

```
function minus(a, b) {
  if (b === undefined) return -a;
  else return a - b;
}

console.log(minus(10));
// → -10
console.log(minus(10, 5));
// → 5
```

FUNCTIONS

- Optional arguments
  - Vi kan introducera ”optional arguments” (default argument)

```
function power(base, exponent = 2) {
  let result = 1;
  for (let count = 0; count < exponent; count++) {
    result *= base;
  }
  return result;
}

console.log(power(4));
// → 16
console.log(power(2, 6));
// → 64
```

FUNCTIONS

- Uppgift
  - Skriv en funktion som tar 2 argument (varav 1 optional argument)
  - Denna gång får du uttrycka ditt eget innehåll.
  - Kontrollera och använd optional argumentet i funktionen på något sätt

SUMMERING (VAD HAR NI LÄRT ER?)

- Body, argument, return (Vad är dessa?)
- Scope (Vilka finns? Vad gäller?)
- Olika funktions deklarationer (Hur ser dessa ut?)
  - Function declaration
  - Function expression
  - Arrow function
- Callstack (Hur fungerar detta?)
- Optional arguments

ÖVNINGAR (REPETITION)

Mark and John are trying to compare their BMI (Body Mass Index), which is calculated using the formula:
`BMI = mass / height ** 2 = mass / (height * height)` (mass in kg and height in meter).

Your tasks:

1. Store Mark's and John's mass and height in variables
2. Calculate both their BMIs using the formula (you can even implement both versions)
3. Create a Boolean variable `'markHigherBMI'` containing information about whether Mark has a higher BMI than John.

Test data:

- Data 1: Marks weights 78 kg and is 1.69 m tall. John weights 92 kg and is 1.95 m tall.
- Data 2: Marks weights 95 kg and is 1.88 m tall. John weights 85 kg and is 1.76 m tall.

ÖVNINGAR (REPETITION)

There are two gymnastics teams, **Dolphins** and **Koalas**. They compete against each other 3 times. The winner with the highest average score wins a trophy!

Your tasks:

1. Calculate the average score for each team, using the test data below
2. Compare the team's average scores to determine the winner of the competition, and print it to the console. Don't forget that there can be a draw, so test for that as well (draw means they have the same average score)
3. **Bonus 1**: Include a requirement for a minimum score of 100. With this rule, a team only wins if it has a higher score than the other team, and the same time a score of at least 100 points. **Hint:** Use a logical operator to test for minimum score, as well as multiple else-if blocks 😉
4. **Bonus 2:** Minimum score also applies to a draw! So a draw only happens when both teams have the same score and both have a score greater or equal 100 points. Otherwise, no team wins the trophy

ÖVNINGAR

1. Write a function called `'describeCountry'` which takes three parameters: `'country'`, `'population'` and `'capitalCity'`. Based on this input, the function returns a string with this format: *'Finland has 6 million people and its capital city is Helsinki'*
2. Call this function 3 times, with input data for 3 different countries. Store the returned values in 3 different variables, and log them to the console

ÖVNINGAR

1. The world population is 7900 million people. Create a **function declaration** called `'percentageOfWorld1'` which receives a `'population'` value, and returns the percentage of the world population that the given population represents. For example, China has 1441 million people, so it's about 18.2% of the world population
2. To calculate the percentage, divide the given `'population'` value by 7900 and then multiply by 100
3. Call `'percentageOfWorld1'` for 3 populations of countries of your choice, store the results into variables, and log them to the console
4. Create a **function expression** which does the exact same thing, called `'percentageOfWorld2'`, and also call it with 3 country populations (can be the same populations)

ÖVNINGAR

1. Recreate the last assignment, but this time create an **arrow function** called `'percentageOfWorld3'`

ÖVNINGAR

1. Create a function called `'describePopulation'`. Use the function type you like the most. This function takes in two arguments: `'country'` and `'population'`, and returns a string like this: *'China has 1441 million people, which is about 18.2% of the world.'*
2. To calculate the percentage, `'describePopulation'` call the `'percentageOfWorld1'` you created earlier
3. Call `'describePopulation'` with data for 3 countries of your choice

FUNCTIONS

- Läxa
  - Eloquent JavaScript
    - https://eloquentjavascript.net/03_functions.html
    - F.o.m ”Closure"