{POWER.CODERS}

JS repetition with loops

Agenda

Today we will look into

- Group work
- Repetition Loops
- More about objects
- Exercises

But first ... remember ternary conditions?

TERNARY CONDITION

```
condition ? expr1 : expr2
```

If the condition is true, provide expr1 else provide expr2.

It is like shorthand for an `if` `else` statement.

EXAMPLE

```
function isUserValid(bool){
  return bool;
}

var answer = isUserValid(true) ? "You may enter" : "Access denied";

var automatedAnswer = "You account number is "
  + ( isUserValid(false) ? "1234" : "not available") + "."
```

CHANGE THIS FUNCTION INTO A TERNARY

... and assign it to variable called experiencePoints

```
function experiencePoints() {
    if (winBattle()) {
        return 10;
    } else {
        return 1;
    }
}
```

Group work

Let's build a very basic to-do list.

- Define the key features in your group.
- Start creating the JavaScript code.
- Ignore HTML for now.

Create a to-do list in JavaScript

What does it need? Which data structure? Where would you use functions?

Some ideas:

- **user input** of a new to-do.
- **list all to-dos** in an alphabetically order in the DOM.
- your to-do list should be only visible by you. Make it **password protected**.
- if you get stuck with the code, try to put in **comments as pseudo-code** first or instead.

1. Understand the problem

- **inputs:** First the Password. Then the To-do.

1. Understand the problem

- **inputs:** First the Password. Then the To-do.
- **process:** After password check show to-dos and make it possible to add new ones.

1. Understand the problem

- **inputs:** First the Password. Then the To-do.
- **process:** After password check show to-dos and make it possible to add new ones.
- **outputs:** Error message if password is incorrect. To-do list sorted alphabetically in the DOM if password is correct.

1. Understand the problem

- **inputs:** First the Password. Then the To-do.
- **process:** After password check show to-dos and make it possible to add new ones.
- **outputs:** Error message if password is incorrect. To-do list sorted alphabetically in the DOM if password is correct.

The word DOM indicates that this time we need HTML tags for our input as well as our output.

2. Plan your solution

What is the procedure?

1. Ask for password
2. Check if password is correct.
3. No: Show error message and ask again (go back to step 1).
4. Yes: Show sorted to-do list and go to the next step.
5. Show input to add new item.
6. Add new to-do item to list.
7. Sort alphabetically.
8. Show updated list.

3. Divide and conquer

Each step in the process is one **small and solvable problem**.
Solve each small problem one by one.

3. Divide and conquer

Each step in the process is one **small and solvable problem**.
Solve each small problem one by one.

Every step symbolizes one task. Use **functions**.

3. Divide and conquer

Each step in the process is one **small and solvable problem**. Solve each small problem one by one.

Every step symbolizes one task. Use **functions**.

Start with **minimal functionality**. Don't think about features not asked for.

Step 1: Inputs and Outputs

Write the HTML and declare global variables

Step 1: Inputs and Outputs

Write the HTML and declare global variables

- password:

```
const password = "string";
```

Step 1: Inputs and Outputs

Write the HTML and declare global variables

- password:

```
const password = "string";
```

- prompt for password

```
let userPassword = prompt("What is your
password?");
```

Step 1: Inputs and Outputs

Write the HTML and declare global variables

- password:

```
const password = "string";
```

- prompt for password

```
let userPassword = prompt("What is your
password?");
```

- add item:

```
let input = document.querySelector("input");
```

Step 1: Inputs and Outputs

Write the HTML and declare global variables

- password:

```
const password = "string";
```

- prompt for password

```
let userPassword = prompt("What is your
password?");
```

- add item:

```
let input = document.querySelector("input");
```

- button:

```
let btn = document.querySelector("button");
```

Step 1: Inputs and Outputs

Write the HTML and declare global variables

- password:

```
const password = "string";
```

- prompt for password

```
let userPassword = prompt("What is your
password?");
```

- add item:

```
let input = document.querySelector("input");
```

- button:

```
let btn = document.querySelector("button");
```

- to-do-list:

```
let ul = document.querySelector("ul");
```

Step 2: Password protection

isUserValid()

- check if password is correct
- yes: return true
- no: return false

Step 3: Password incorrect

while() loop

Ask for password until it is correct.

Step 4: Password correct

addClass(elm,cls)

Add class to DOM elements to make them visible.

Step 5: Get user input

addEventListener(eventType, function)

On the JavaScript event **click** add new to-do to the list.

Step 6: Add item to list

addItemToList()

- create new li-tag
- add user input as content
- append li to to-do list
- clear user input

Step 7: Sort list alphabetically

sortList()

- declare temp array variable
- loop through list items to get content in temp array
- sort the array
- remove the list
- create a new list
- loop through array and create li-tags
- add value of list item as content
- append each li to the ul-tag

4. Debug and test

5. Optimize your code

- add validation to user input of to-do
- add event listener for the return key
- change the to-do list from being invisible to not being there and being newly created
- store the items long-term
- add functionality: update a to-do item, e.g. set it to done
- add functionality: remove a to-do item

Repetition loops

Loops

There are four types of loops:

Loops

There are four types of loops:

- `while`
- `do while`
- `for`
- `forEach`

`while`

The `while` loop is the easiest. It repeats through a block of code, as long as a specified **condition** is **true**.

```
while(statement){
  //codeblock this gets looped until the statment is false
}
```

`while`

The `while` loop is the easiest. It repeats through a block of code, as long as a specified **condition** is **true**.

```
while(statement){
  //codeblock this gets looped until the statment is false
}
```

If a condition is always **true**, the loop will run **forever**.
Make sure that the condition eventually becomes **false**.

`do while`

The `do while` loop is rarely used. The loop will always be executed **at least once**, even if the **condition** is false.

```
do {
  //code block here
} while(statment);
```

`for`

The `for` loop is most commonly used.

```
for(initialization; condition; iterator){
  //codeblock this gets looped until the condition is false
}
```

`for`

The `for` loop is most commonly used.

```
for(initialization; condition; iterator){
  //codeblock this gets looped until the condition is false
}
```

The `for` loop has a specific pre-defined amount of loops

forEach

The `forEach` loop came with ES5.

```
array.forEach(function(value, index) {
  //more code here
});
```

`forEach`

The `forEach` loop came with ES5.

```
array.forEach(function(value, index) {
  //more code here
});
```

The `forEach` loop is for looping through arrays, the length of the array defined how many loops there will be.

`forEach` in ES6

New notation in ES6: `forEach` with arrow function

```
array.forEach(element => {
  //more code here
});
```

Looping through an array

The `for of` loop is intended for iterating over arrays. An alternative to `forEach`.

```
let list = ["doors", "windows", "rooms"];
for(let x of list){
  console.log(x);
}
```

It works for all iterable objects, including strings.

```
for(let ch of "hello"){
  console.log(ch);
}
```

Looping through an object

The `for in` loop is intended for iterating over the keys of an object.

```
let obj = {doors: 2, windows: 8, rooms: 5};
for(let x in obj){
  console.log(x);
}
```

- Do not use this loop for arrays.
- The iterating variable `x` is always a string.

Looping through an object

Alternatively you can use `Object.keys().forEach()` or `Object.values().forEach` to loop through an object.

```
let obj = {doors: 2, windows: 8, rooms: 5};
Object.keys(obj).forEach(function(property_name){
  console.log(obj[property_name]);
});
```

```
let obj = {doors: 2, windows: 8, rooms: 5};
Object.values(obj).forEach(function(property_value){
  console.log(property_value);
});
```

More about objects

Remember this?

Object constructor

```
function Person(name, age, married) {
  this.name = name;
  this.age = age;
  this.isMarried = married;
  this.hello = function() {
    return "Hello " + this.name;
  }
};
```

Classes

New in ES 6

```
class Person {
  constructor(name, age, married) {
    this.name = name;
    this.age = age;
    this.isMarried = married;
    this.hello = function() {
      return "Hello " + this.name;
    }
  }
}
```

Classes

New in ES 6

```
class Person {
  constructor(name, age, married) {
    this.name = name;
    this.age = age;
    this.isMarried = married;
    this.hello = function() {
      return "Hello " + this.name;
    }
  }
}
```

- Classes are **templates** for JavaScript objects

Classes

New in ES 6

```
class Person {
  constructor(name, age, married) {
    this.name = name;
    this.age = age;
    this.isMarried = married;
    this.hello = function() {
      return "Hello " + this.name;
    }
  }
}
```

- Classes are **templates** for JavaScript objects
- A newer notation for the object constructor

Destructuring

Faster and easier way to access object properties. New in ES6. The name of the variable and the property name have to be identical.

```
const obj = {
  person: "Susanne",
  age: 38,
  experience: 13
}

/** the way we know */
const person = obj.person;
let age = obj.age;
let experience = obj.experience;

/** destructured */
const { person } = obj;
let { age, experience } = obj;
```

DESTRUCTURE THIS

```
const person = {
    firstName : "John",
    lastName  : "Doe",
    age       : 50,
    eyeColor  : "blue"
};

let firstName = person.firstName;
let lastName = person.lastName;
let age = person.age;
let eyeColor = person.eyeColor;
```

Dynamic object properties

With `[]` in your property name, you can put in dynamic values, like another variable or a calculation.

```
const name = "first name";

const obj = {
  [name]: "Susanne",
  [5 + 13]: 38,
  experience: 13
}
```

Symbol

The one datatype we did not talk about yet.

```
const s1 = Symbol();
const s2 = Symbol();
```

- Mainly used as **unique identifier** aka keys in objects
- It is hidden
- It is unique

Use case

Using symbols as unique values

```
const possibleStatus = {
    OPEN: Symbol('Open'),
    IN_PROGRESS: Symbol('In progress'),
    COMPLETED: Symbol('Completed'),
    HOLD: Symbol('On hold'),
    CANCELED: Symbol('Canceled')
};
// complete a task
task.setStatus(possibleStatus.COMPLETED);
```

Use case

Using symbols as unique values

```
const possibleStatus = {
    OPEN: Symbol('Open'),
    IN_PROGRESS: Symbol('In progress'),
    COMPLETED: Symbol('Completed'),
    HOLD: Symbol('On hold'),
    CANCELED: Symbol('Canceled')
};
// complete a task
task.setStatus(possibleStatus.COMPLETED);
```

You would have used strings before. Now the status is unique for each task.

Another use case

Using symbols as computed property name

```
let status = Symbol('status');
const task = {
    [status]: possibleStatus.OPEN,
    description: 'Learn ES6 Symbol'
};
console.log(task);
```

Group exercises (15 min each)

Arrays

```
// using this array,
// var array = ["Banana", "Apples", "Oranges", "Blueberries"];
```

1. Access and output Oranges.
2. Remove the Banana from the array.
3. Sort the array in order.
4. Put "Kiwi" at the end of the array.
5. Remove "Apples" from the array.
6. Sort the array in reverse order, i.e. ['a', 'c', 'b'] becomes ['b', 'c', 'a'])

You should at the end have

```
["Kiwi", "Oranges", "Blueberries"]
```

Exercise: Your top choices

- Create an array to hold your top choices (colors, presidents, whatever).
- For each choice, log to the screen a string like: "My #1 choice is blue."
- Change it to log "My 1st choice, "My 2nd choice", "My 3rd choice", picking the right suffix for the number based on what it is.

Exercise: Recipe card

- Create an object to hold information on your favorite recipe. It should have properties for title (a string), servings (a number), and ingredients (an array of strings).
- On separate lines (one console.log statement for each), log the recipe information so it looks like:
    - Soup
    - Serves: 2
    - Ingredients: cinnamon, cumin, cocoa

Work on your project