Gameplan compound interest
Goal: An app that calculates compound interest

1. What's compound interest?

Wikipedia. Einstein: "Compound interest is the eighth wonder of the world. He who understands it, earns it ... he who doesn't ... pays it."
If you invest right, getting rich is possible. Creating an app to calculate how easy/hard it will be to get rich.
https://www.youtube.com/watch?v=gvZSpET11ZY

2. Input variables:
- Investment amount (start capital)
- Investment amount (monthly amount)
- Interest rate
- Age (start)
- Age (retire)

3. Workflow:
- ~~Make a folder~~ (mkdir)
- ~~Make a git repo out of it~~ (git init)
- ~~.gitignore setup~~ (touch .gitignore, add ds-store to it, git add .gitignor, git commit -am"added ds-store to gitignore")
- ~~Make node app~~ (touch app.js)
- ~~Import data~~ (make a module that imports data)
    - ~~Create customer json~~ (touch customers.json, if you want to rename: mv customers.json customer.json)
    - ~~Populate customer json~~ (object: key has to have quotes!
        - {
        - "name": "Ineke",
        - "age": 25,
        - "finances": {
        - "startcapital": 1000,
        - "monthlyadd": 100
        - },
        - "pension": {
        - "age": 65,
        - "interest": {
        - "pessimistic": 2,
        - "average": 4,
        - "optimistic": 8
        - }
        - }
        - }
        -
    - ~~Read the customer json~~
        - //importing necessary modules

        - const fs = require('fs');
        -
        - //read the customer data json
        - // () => {} short way to write a function
        - // (err, data) => {} same as function(err, data){}
        - fs.readFile(__dirname + '/customer.json', 'utf-8', (err, data) => {
        -         // let = another var: The difference is scoping. var is scoped to the nearest function block and let is scoped to the nearest enclosing block (both are global if outside any block), which can be smaller than a function block. Also, variables declared with let are not accessible before they are declared in their enclosing block. As seen in the demo, this will throw an exception.
        -         //parse the file to a readable object
        -         let parsedData = JSON.parse(data);
            - //check if it’s working
        -         console.log(parsedData);
        - })
        -
- Compound math
    - ~~Var end amount~~
        - customer.pension.endamount = 0;
    - ~~Little loop (every year happens something)~~
        - ~~Add money~~
        - ~~Add interest~~
            - for (var i = customer.pension.duration - 1; i >= 0; i--) {
            -                 // check if it loops 39 times 38-0
            -                 //console.log("I looped " + i + " times");
            -                 //calculate monthly add
            -                 customer.pension.endamount.pessimistic += (customer.finances.monthlyadd * 12);
            -                 customer.pension.endamount.average += (customer.finances.monthlyadd * 12);
            -                 customer.pension.endamount.optimistic += (customer.finances.monthlyadd * 12);
            -                 //calculate added interest after a year
            -                 customer.pension.endamount.pessimistic *= customer.pension.interest.pessimistic;
            -                 customer.pension.endamount.average *= customer.pension.interest.average;
            -                 customer.pension.endamount.optimistic *= customer.pension.interest.optimistic;
            -         }
            -
    - ~~Var start amount = var end amount~~
        - customer.pension.endamount = {
        -                 pessimistic: customer.finances.startcapital,

- average: customer.finances.startcapital,
- optimistic: customer.finances.startcapital
- }

- ~~output data to user~~
    - //output our data
    - console.log("Welcome " + customer.name + " to our advanced pension planner!");
    - console.log("You're starting with: " + customer.finances.startcapital + " and add a monthly amount of " + customer.finances.monthlyadd);
    - console.log("When you retire at age: " + customer.pension.age + " you will have the following: ")
    - //output calculation stuff
    - console.log("In a pessimistic scenario: €" + customer.pension.endamount.pessimistic);
    - console.log("In a average scenario: €" + customer.pension.endamount.average);
    - console.log("In a optimistic scenario: €" + customer.pension.endamount.optimistic);
    -

```
//Welcome Ineke to our advanced pension planner!
You're starting with: 1000 and add a monthly amount of 100
When you retire at age: 65 you will have the following:
In a pessimistic scenario: €73447.124585141
In a average scenario: €117446.98482650022
In a optimistic scenario: €329783.12013422046
```

- Extra: we want not so much numbers behind the dot
  Google: round to at most 2 decimals after dot
  Math.round(number * 100) / 100
  ```
          // get at most 2 decimals behind dot
  var roundDecimal = (number) => {
          return Math.round(number * 100) / 100;
  }
  ```
  (add it to the console.logs)

- Extra add comma's to number at every 1000
  Google: how to print a number with commas as thousands separators in js
  ```
  function numberWithCommas(x) { return
  x.toString().replace(/\B(?=(\d{3})+(?!\d))/g, ","); }
  ```
  ```
          //get commas at every 1000
  var addCommas = (number) => {
     return number.toString().replace(/\B(?=(\d{3})+(?!\d))/g, ",");
  }
  ```

- extra : two functions for one number: wrap that

```
                //helper funtion for prettyNr
// get at most 2 decimals behind dot
let roundDecimal = (number) => {
        return Math.round(number * 100) / 100;
}

//helper funtion for prettyNr
//get commas at every 1000
let addCommas = (number) => {
   return number.toString().replace(/\B(?=(\d{3})+(?!\d))/g, ",");
}

//add commas and round to twoe decimals
let prettyNr = (number) => {
        return addCommas(roundDecimal(number))
}
```

```
//Welcome Ineke to our advanced pension planner!
You're starting with: 1000 and add a monthly amount of 100
When you retire at age: 65 you will have the following:
In a pessimistic scenario: €73,447.12
In a average scenario: €117,446.98
In a optimistic scenario: €329,783.12
```

4. Nice to have features:
    - Detailed algorithm
        - Monthly additions
        - Monthly returns
    - Historical data
    - Cost structure
        - Log out amount paid to the bank