Note week 02!

1. **Arrow - I am still confuse about it!**

There’s a third notation for functions, which looks very different from the others. Instead of the function keyword, it uses an arrow (=>) made up of an equal sign and a greater-than character (not to be confused with the greater-than-or-equal operator, which is written >=).

const power = (base, exponent) => {

let result = 1;

for (let count = 0; count < exponent; count++) {

result \*= base;

}

return result;

};

The arrow comes *after* the list of parameters and is followed by the function’s body. It expresses something like “this input (the parameters) produces this result (the body)”.

1. **This is a kind of code that I am still confuse with!!**

**() => I am return a function without any parameter. What is the propose of the arrow and the local variable.**

function wrapValue(n) {

let local = n;

return () => local;

let wrap1 = wrapValue(1);

let wrap2 = wrapValue(2);

console.log(wrap1());

// → 1

console.log(wrap2());

// → 2

## 

## Growing functions

## That I need a concept that I need to study a little more.

function printFarmInventory(cows, chickens) {

let cowString = String(cows);

while (cowString.length < 3) {

cowString = "0" + cowString;

}

console.log(`${cowString} Cows`);

let chickenString = String(chickens);

while (chickenString.length < 3) {

chickenString = "0" + chickenString;

}

console.log(`${chickenString} Chickens`);

}

printFarmInventory(7, 11);

### **Recursion**

**That is a concept that I want to study more!!!!!!!**

We’ve seen that % (the remainder operator) can be used to test whether a number is even or odd by using % 2 to see whether it’s divisible by two. Here’s another way to define whether a positive whole number is even or odd:

* Zero is even.
* One is odd.
* For any other number N, its evenness is the same as N - 2.

Define a recursive function isEven corresponding to this description. The function should accept a single parameter (a positive, whole number) and return a Boolean.

## Breaking Out of a Loop – Cool Stuff!!

Having the looping condition produce false is not the only way a loop can finish. There is a special statement called break that has the effect of immediately jumping out of the enclosing loop.

This program illustrates the break statement. It finds the first number that is both greater than or equal to 20 and divisible by 7.

for (let current = 20; ; current = current + 1) {

if (current % 7 == 0) {

console.log(current);

break;

}

}

// → 21

1. **INEQUALITY!!**

**Cool stuff!!!!**

16 != '16'; // type coercion makes these equal

<< false

16 !== '16';

<< true

1. I really like the way how JS works with variables and store the data!

## Quiz Ninja Project

Now that we have come to the end of the chapter, it’s time to put what we’ve learned into practice in our Quiz Ninja project.

Since we’ve been learning all about JavaScript in this chapter, we’re going to add some code in the main.js file. Open that file and add the following lines:

const question = "What is Superman's real name?"

const answer = prompt(question);

alert(`You answered ${answer}`);

Now let’s go through this code line by line to see what is happening:

const question = 'What is Superman's real name?';

This declares a variable called question and assigns the string 'What is Superman's real name?' to it.

Next, we need to ask the question stored in the question variable, using a prompt dialog:

const answer = prompt(question);

A prompt dialog allows the player to type in a response, which is then stored in the variable it is assigned to, which is answer in this case.

Finally, we use an alert dialog to display the player's answer using string interpolation to insert the value of answer into the template literal that is displayed in an alert box:

alert(`You answered ${answer}`);

1. RANDOM!!!

random numbers: write a function that will return a random number in a range from 0 -> num-1

      const random = function(num){

        return Math.floor(Math.random() \* num);