Life is Possible - 生命教育 手機程式工作坊
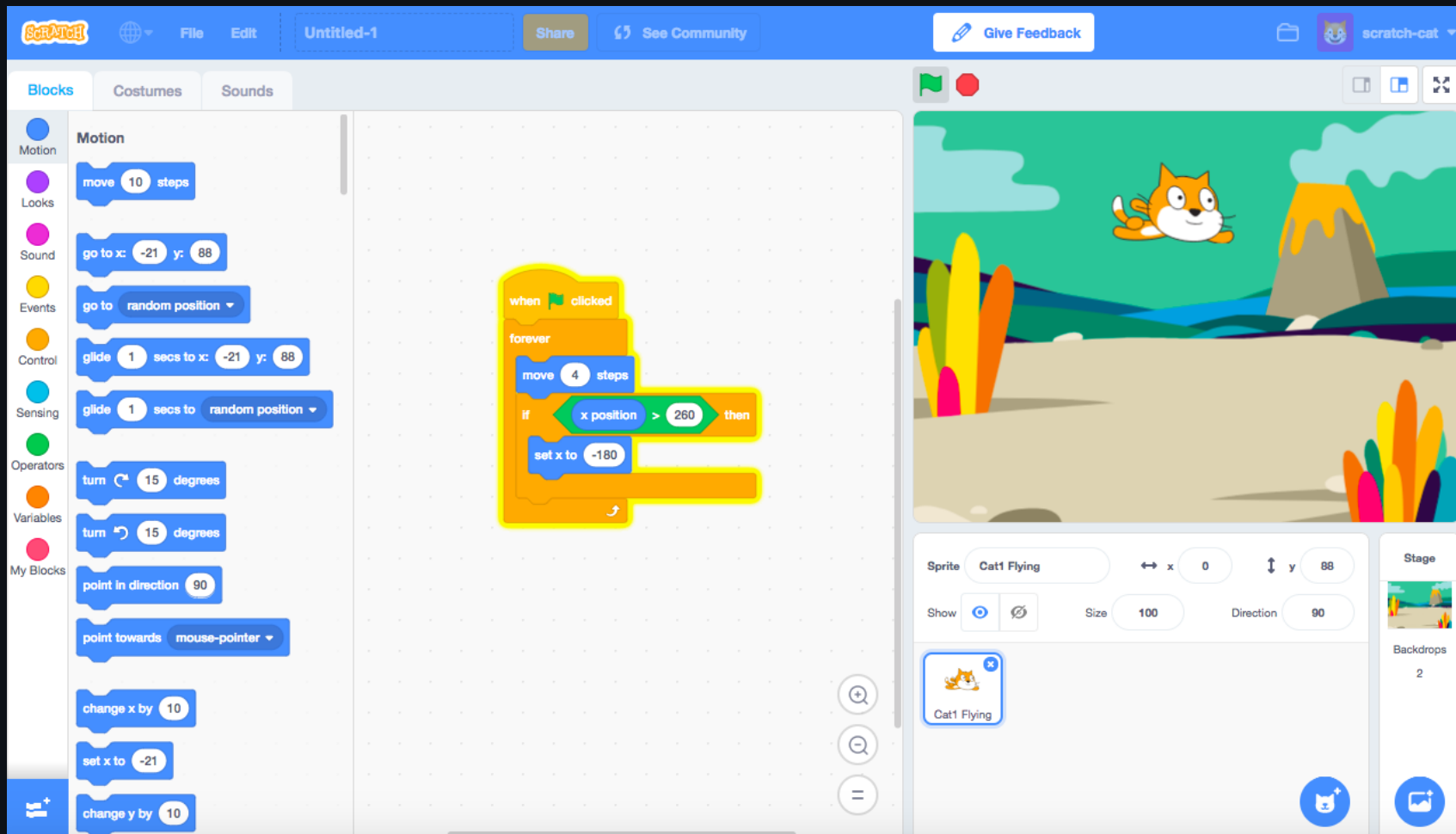
Lecture 04 - Introduce to JavaScript

Menu

- Deep into programming

- Introduce to real programming with Javascript
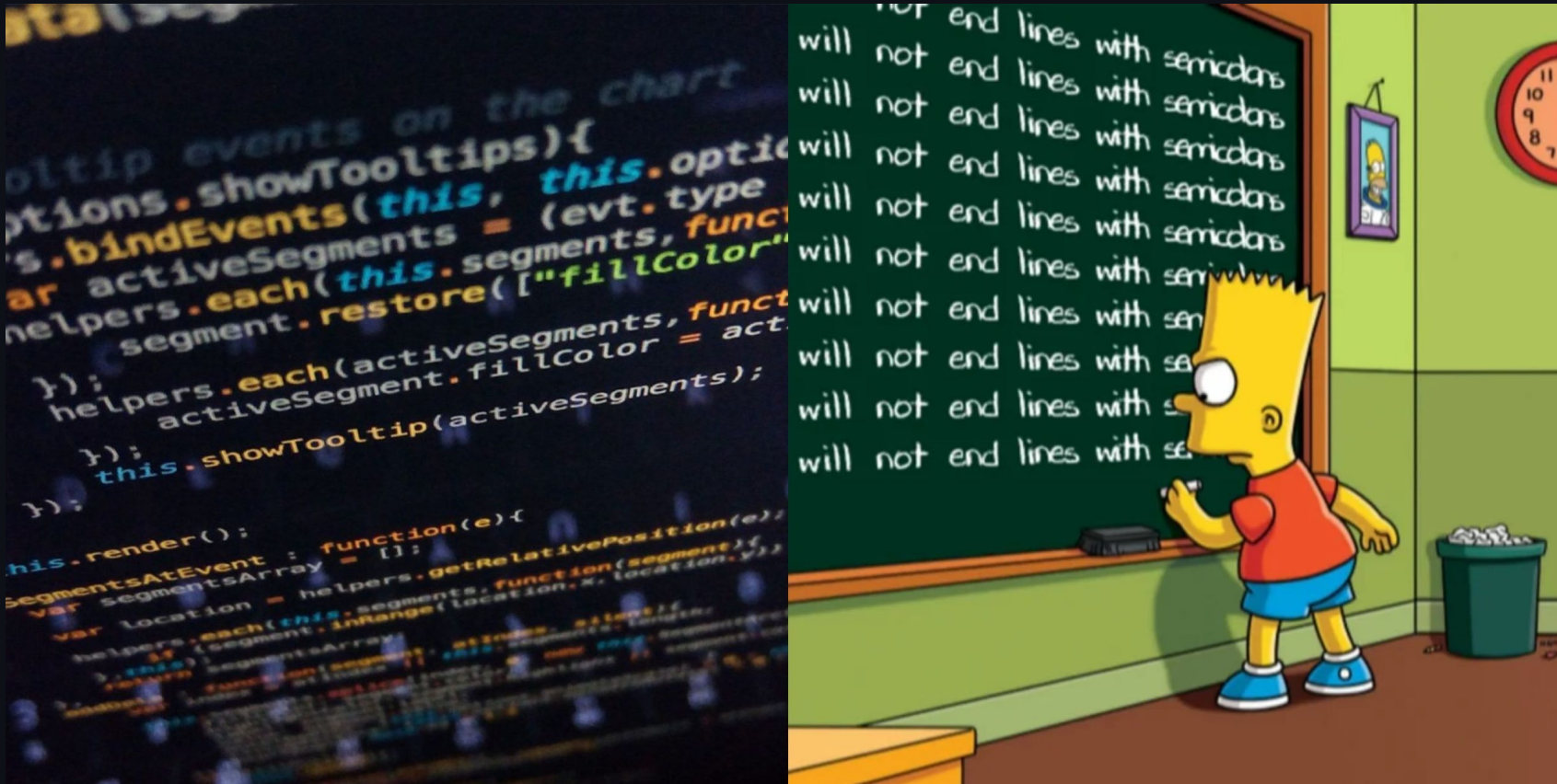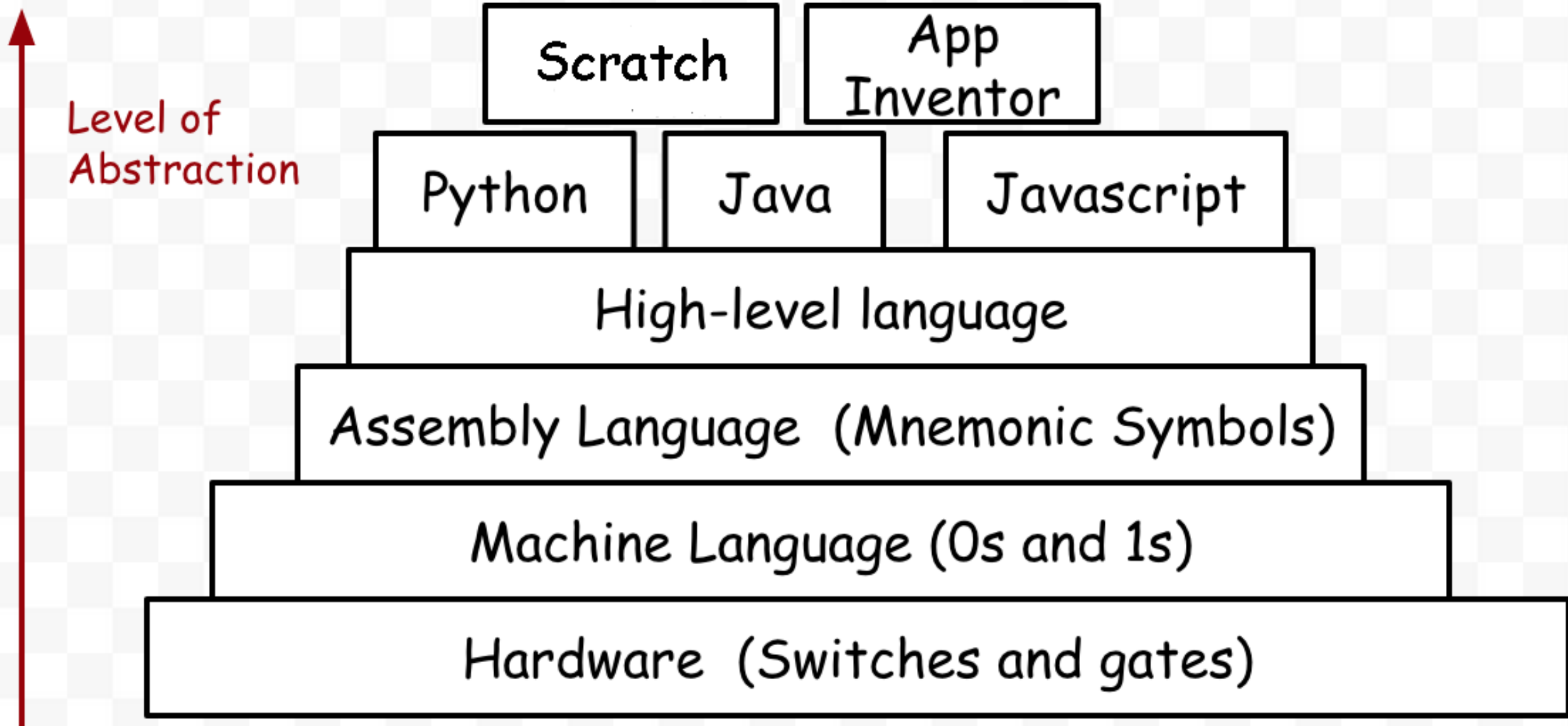
Coding with `logic block`?

Real programmer dont do this...

Real programmer code with text / words only

Programming languages

Level of Abstraction

Scratch

App Inventor

Python   Java   Javascript

High-level language

Assembly Language (Mnemonic Symbols)

Machine Language (0s and 1s)

Hardware (Switches and gates)

Why we learn Javascript?

- Relative Easy to learn as a beginner

- Learn once, use everywhere (web / desktop / linux)

- High popularity across the world
  https://www.stackscale.com/blog/most-popular-programming-languages/

- Great community support and large library support `npm`

Javascript can run on

- 🖥️ Web browser (V8 engine base)
- 📱 Android (V8 engine with browser)
- 📲 IOS (V8 engine with browser)
- 🖥️ Windows (Nodejs)
- 💻 MacOS (Nodejs)
- 🖱️ Linux (Nodejs)

Almost everything you can seen from daily life can run javascript.

Write a welcoming program

`app.js`

```
let str = "reemo"
console.log("Hello all");
console.log(str);
console.log("This is " + str);
```

Then type `node app.js` in the terminal to run the script.

Notices: If you did not install the `nodejs` , install it from https://nodejs.org/en/ now.

`console.log();`

This is the fundamental function for javascript to print stuff

Usage

```javascript
console.log("Hello world"); // "Hello world"
console.log("Hello " + "Peter"); // "Hello Peter"
console.log("Hello", "Peter"); // "Hello Peter"

console.log(100); // 100
console.log(100 + 45); // 145
```

Javascript menu today

- Data Types

- Arithmetic Operators

- Assignment Operators

- Define variables

- Conditional Statements

- Compare Statements

- Functions

JavaScript Comments

> Used to explain codes, and to make it more readable.
>
> Codes inside comments will not run / execute / complie

Single line comments start with `//` .

```js
// I will not run
// console.log("Hello")
// Hello mate

console.log("mate")
```

JavaScript Comments

Multi-line comments start with `/*` and end with `*/`.

```
/*
Hello
This is multi line comments
I can contains a lot of stuff
in the same block
console.log("Yoooo")
*/


console.log("mate")
```

General Data Types in js

- `string` e.g. `"Hello"`, `"Good day"`, `"I go to school by bus"`
- `number` e.g. `12`, `-3`, `32.476`, `0x012`, `11010010`
- `boolean` e.g. `true`, `false`

`string`

The common string / char that we want to shows

- `"Hello world"`, `"a"`, `"😀"` (Using `"`)
- `'Hello mate'`, `'b'`, `'😍'` (Using `'`)
- `` `Hello mom` ``, `` `c` ``, `` `😎` `` (Using `` ` ``)

`number`

A data type to represent a number for math computations. (`+` , `-` , `*` , `/` , `%`)

- Integer: `1` , `384`

- Signed Integer: `1` , `384` , `-43`

- Float / Double: `1.23` , `-34.3423` , `0.001`
  (Float = single-precision, Double = double-precision)

- Infinity: `infinity` , `-infinity`

`number` Basic Arithmetic Operators

- `+` : Addition (e.g. `3 + 5 = 8`)
- `-` : Subtraction (e.g. `8 - 2 = 6`)
- `*` : Multiplication (e.g. `3 * 4 = 12`)
- `/` : Division (e.g. `10 / 5 = 2`, `14 / 5 = 2.8`)

More `number` Arithmetic Operators

Commons in general

- `**` : Exponentiation (`2^3` => `2**3`)
- `%` : Modulus / Remainder (`5 % 2 == 1` , `10 % 2 == 0`)

Commons in loop

(will cover more in next lesson)

- `++` : Increment (`i++`)
- `--` : Decrement (`i--`)

`boolean`

A data type to determine the **conditions** will happends or not. ONLY `true` and `false` will be appares on this data type.

- `true` : Means the conditions will be happend / is match.
- `false` : Means the conditions will be NOT BE happend / is NOT match.

boolean example

```
13 == 100 // Is 13 equal to 100 ? false
100 > 50 // Is 100 bigger than 50 ? true
0 < 2 // Is 0 smaller than 2 ? true

"tom" == "hello tom" // Is "tom" equal to "hello tom" ? false
"apple" == "apple" // Is "apple" equal to "apple" ? true
```

`boolean` `and &&`, `or ||`, `not !`

In all programming languages, several symbo will be represent the logics.

- `and` : Using `&&`

- `or` : Using `||`

- `not` : Using `!`

Boolean Operators

AND

A	B	A AND B
True	True	True
True	False	False
False	True	False
False	False	False

OR

A	B	A OR B
True	True	True
True	False	True
False	True	True
False	False	False

NOT

A	NOT A
True	False
False	True

`boolean` with `and &&`, `or ||`, `not !`

- `and` : Means **two conditions** are true, then will return true.

- `or` : Means **either one conditions** is true, then will return true.

- `not` : Reverse the conditions (`!true => false`, `!false => true`)

Why `and &&`, `or ||`, `not !`

Actually this is an simple question or logic sentence in real life.

If we want to represent a conditions in english like this:

```
If today is monday and the weather is not raining, then I will go to Peter's home.
```

In programming, we code like this:

```
if(today.day == "monday" && today.weather != "raining"){
    goToPeterHome();
}
```

Why `and`, `or`, `not`

A more complex conditions will be like this:

```
This is Tom. Hay Peter, I want to play a game with you.
Make a guess between 1 to 6. And I start roll the dice.

If the number is a even and a one or a six, you will win the game.
If the number is a even but the number is 4, you will lose.

Yet, if the number is a odd, you will lose.
But, if the number is a odd and the number is 3, you will win.
```

If you are Tom and want to make a game to Peter, how's the login in programming languages?

```js
// let assume this function will return a random number of 1 to 6
let diceNumber = randomDiceNumber();

if(diceNumber % 2 == 0 && diceNumber == 4){
    lose()
}
else if(diceNumber % 2 == 0 && (diceNumber == 1 || diceNumber == 6)){
    win()
}
else if(diceNumber % 2 == 1 && diceNumber == 3){
    win()
}
else if(diceNumber % 2 == 1){
    lose()
}
```

Advance Data Types in js

Common (Will cover more on lecture)

- `Object` and `Array`
- `Null` and `Undefined`

Advance (Will NOT cover on lecture, but important for future)

- `Error`
- `Date`
- `Map` and `Set`
- `Class`

Break

Assignment Operators

Operator Short hand	Example	Same As
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
**=	x **= y	x = x ** y

Define a variables

```
var names = "reemo";
let age = 123;
const isMale = true;
```

In general, we can define a variable with `var`, `let` or `const`.

In most of the case (99.99%), we **DO NOT RECOMMENDED** using `var` due to the legacy reasons. Know more

Define a variables

```
// <declare_words> <declare_variables_name> = <data_values>
let names = "reemo";
const age = 10;
```

- `let` is a define words to tell the computer to define a variable that can be changes.

- `const` also, but we assume that variable define with `const` is the variable that *CAN NOT* be changed after it defined.

`let`

1. Declare variables with `let` can be re-define

2. Is block-scope

```javascript
let names = "reemo";
console.log(names) // reemo

names = "tom"
console.log(names) // tom
```

`const`

1. In commons, declare variables with `const` can NOT be re-define.

2. Is block-scope

```
const names = "reemo";
console.log(names) // reemo

names = "tom" // Cannot assign to 'names' because it is a constant.
console.log(names) // error: Uncaught TypeError: Assignment to constant variable.
```

There is exceptions of using `const` in data type like `array` and `object` , but we assume all `const` variables are can not be re-assign first.

Conditional Statements

- Compare: `==`, `===`, `!=`, `!==`
- For Maths: `>`, `<`, `>=`, `<=`
- Logics: `if`, `else if`, `else`

Compare Statements `==`, `===`, `!=`, `!==`

- `==` : Weak Equals to
- `===` : Strong Equals to
- `!=` : Weak Not Equals to
- `!==` : Strong Not Equals to

`Strong` compare and `Weak` compare?

- `Strong` means not only values matches, but the data type need to be match too.
- `Weak` means if the values matches (no matter is string or number), if the content is the same, then it will return true.

Strong and Weak examples

```
let num = 100; // type: number
let stringNum = "100" // type: string

console.log( num == stringNum ) // Weak compare: true
console.log( num === stringNum ) // Strong compare: false
```

Notices: In this examples, `num` is a data type of `number`, but `stringNum` is a data type of `string` although both represents `100`.

IF you are not sure which compare type to use, always use strong `===` `!==` compare for safely reasons.

Maths Compare Statements : `>` , `<` , `>=` , `<=`

- `>` : Bigger than

- `>=` : Bigger than or equal to

- `<` : Smaller than

- `<=` : Smaller than or equal to

```
10 > 6 // true
5 < 89 // true
10 > 10 // false, why? Since 10 not not bigger than 10, is equal to 10
10 >= 10 // true
```

Logics Compare Statements: `if` , `else if`, `else`

Inside code will run if the conditions fullfilled

- `if(){}`
- `else if(){}`
- `else{}`

if

demo.js

```js
// Only the conditions between the () is true, the under code will run

if(true){
    console.log("I will run yeah 😀")
}

if(false){
    console.log("I will NOT run oh no 😢")
}
```

`ifDemo.js`

```javascript
let nums = 100;

if(nums > 10){
    nums += 23 // this code will run since 100 > 10 is true
}

console.log(nums) // 123
```

`ifDemoTwo.js`

```javascript
let nums = 5;

if(nums > 10){
    nums += 23 // this code will NOT run since 5 > 10 is false
}

console.log(nums) // 5
```

`if` and `else`

`demo.js`

```javascript
// If the conditions in `if` is false, it will run the else code sections
if(false){
    console.log("I will NOT run oh no 😥")
}
else{
    console.log("I will run yeah 😀")
}
```

`if` and `else`

`ifElseDemo.js`

```js
let nums = 20;

if(nums > 10){
    nums += 23 // if the nums > 10 is true, this line will run
}
else{
    nums -= 10 // if the nums > 10 is false, this line will run
}

console.log(nums) // 43
```

if and else

`ifElseTwoDemo.js`

```js
let nums = 5;

if(nums > 10){
    nums += 23 // if the nums > 10 is true, this line will run
}
else{
    nums -= 10 // if the nums > 10 is false, this line will run
}

console.log(nums) // -5
```

Advance `if` and `else` usage

Following code are not the same

```javascript
if(true){
    console.log("I will run yeah 😀")
}

console.log("I will run no matter if() is true or false")
```

```javascript
if(false){
    console.log("I will run yeah 😀")
}
else{
    console.log("I will run But only if() is false")
}
```

`if` and `else` and `else if`

`ifElseDemo.js`

```javascript
let nums = 50;

if(nums == 10){
    console.log("I am a 10")
}
else if(nums == 20){
    console.log("I am a 20")
}
else{
    console.log("nope") // This line will printed
}

// "nope"
```

`if` and `else` and `else if`

`ifElseDemo.js`

```javascript
let nums = 10;

if(nums == 10){
    console.log("I am a 10") // This line will printed
}
else if(nums == 20){
    console.log("I am a 20")
}
else{
    console.log("nope")
}

// "I am a 10"
```

if and else and else if

`ifElseDemo.js`

```javascript
let nums = 20;

if(nums == 10){
    console.log("I am a 10")
}
else if(nums == 20){
    console.log("I am a 20") // This line will printed
}
else{
    console.log("nope")
}

// "I am a 20"
```

Functions

Functions

function is a block of code designed to perform a particular task.

`basic-function.js`

```js
function myFunctionName(){
    // Stuff to do when this function is called.
    console.log("Hello mate.")
}


myFunctionName() // calling the function
```

Functions with params

A function can be pass in params for the purpose of re-use.

`basic-function-params.js`

```js
function greetings(names){
    // Stuff to do when this function is called.
    console.log("Hello " + names)
}

greetings("peter") // Hello peter
greetings("tom") // Hello tom
```

Functions with more params

A function can be pass a lot of params too.

`basic-function-params.js`

```js
function greetings(names, ages){
    // Stuff to do when this function is called.
    console.log("Hello " + names)
    console.log("Are you the age of " + ages + " ?")
}

greetings("peter", 18) // Hello peter / Are you the age of 18 ?
greetings("tom", 33) // Hello tom / Are you the age of 33 ?
```

Functions return

A function can be `return` a values for other variable to use too.

`function-return.js`

```js
function returnSentences(names){
    let sentences = "Hello " + names;
    return sentences
}

let myNameSentences = returnSentences("peter"); //myNameSentences = "Hello peter"
console.log(myNameSentences); // "Hello peter"

console.log(returnSentences("tom")); // "Hello tom"
```

Advance Functions: Arrow Function (Bonus)

In javascript, a function can also code like this

```
let myFunction = (a, b) => a * b;
```

Break

Lab 01 - Define variables and types

Peter want's to define some variables but he don't know how to do. Can you help him?

1. Define two `let` variables `income` and `outcome`
   with values `10000` and `4000`.

2. Define two `const` variables `names` and `date`
   with values `"Peter"` and `"2022-03-06"`.

3. Define a `let` variable `total` that base on `income - outcome` to get the total
   count.

Lab 01 - Define variables and types

`lab02-template.js`

```javascript
let someStuff = ...
const someStuffAlso = ...
```

Lab 02 - Check student stages

Write a program to check the student education stages by the following requirement:

If the student age lower than 5, he / she will be in `Not educated`
If the student age between 6 to 12, he / she will be in `Primary school`
If the student age between 13 to 18, he / she will be in `Secondary school`
If the student age between 19 to 22, he / she will be in `University`
If the student age bigger 23, he / she will be in `Working`

`console.log()` the status if the age match that status.

Lab 02 - Check student stages (Cont)

`lab02-template.js`

```js
function calculateAgeStatus(age){
    console.log(age)
    // your code
}

// Testing
calculateAgeStatus(4)  // Not educated
calculateAgeStatus(59) // Working
calculateAgeStatus(16) // Secondary school
calculateAgeStatus(8)  // Primary school
calculateAgeStatus(21) // University
```

Summary

We have learn the:

- Run a `app.js`

- Print variables with `console.log`

- Data Types `string`, `number`, `boolean`

- Arithmetic Operators `++`, `*`, `/` ...

- Assignment Operators `=`, `+=`, `*=`, `/=` ...

- Define variables `let`, `const`, `var`

- Conditional Statements `if`, `else`

- Compare Statements `==`, `===`, `<=` ...

- Functions `function add(a,b){ return a + b }`

More pratices

https://code.tecky.io/

https://js.checkio.org/

More materials

We skipper these concepts in this course

- Object, Array, for loop

- Array Looping (for, while)

- Scope

But you can read it by your own if interested in `lecture-ex1`

End