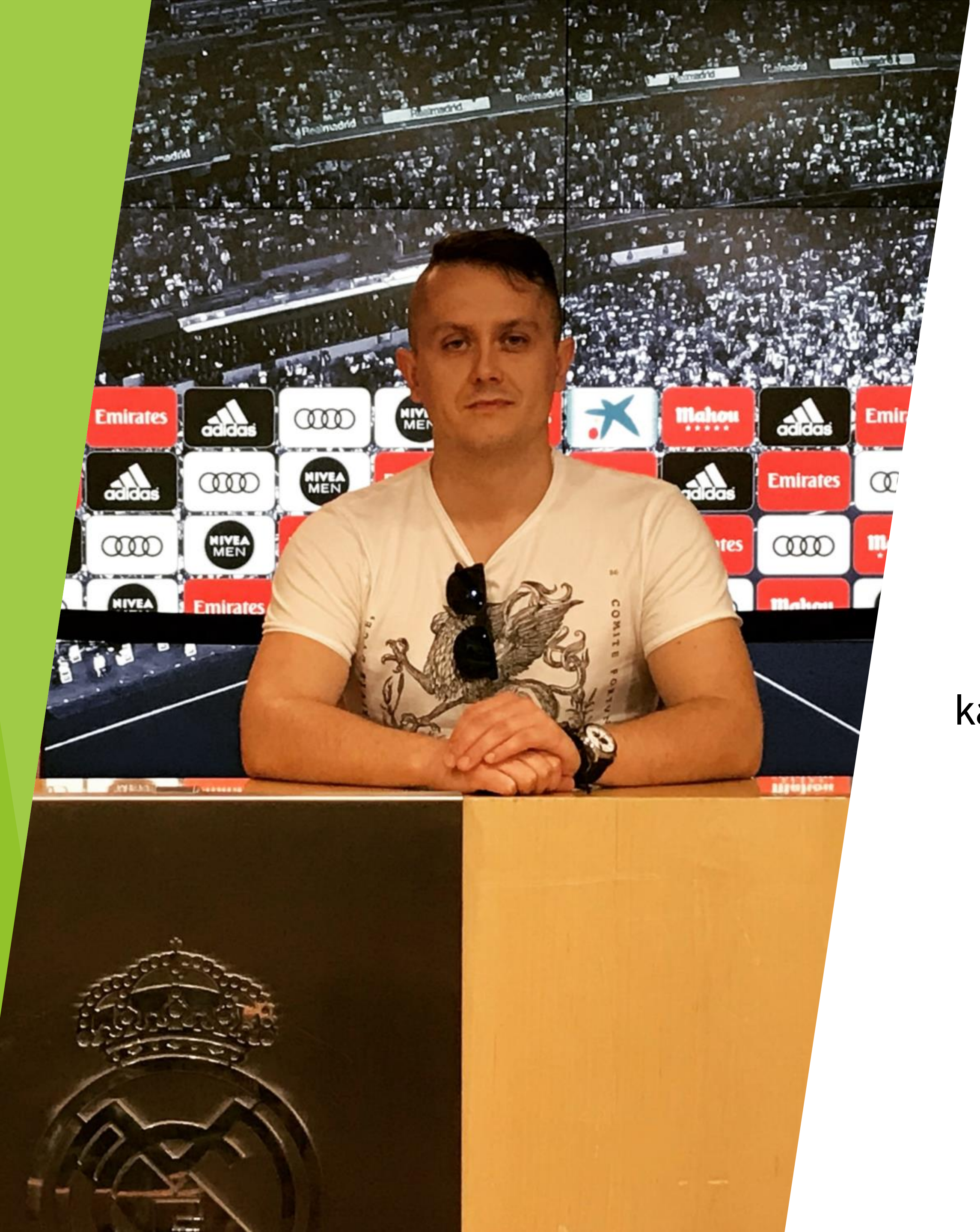
JavaScript

Karol Rogowski

IT'S ALL
ABOUT YOU



About me

karol.rogowski@gmail.com

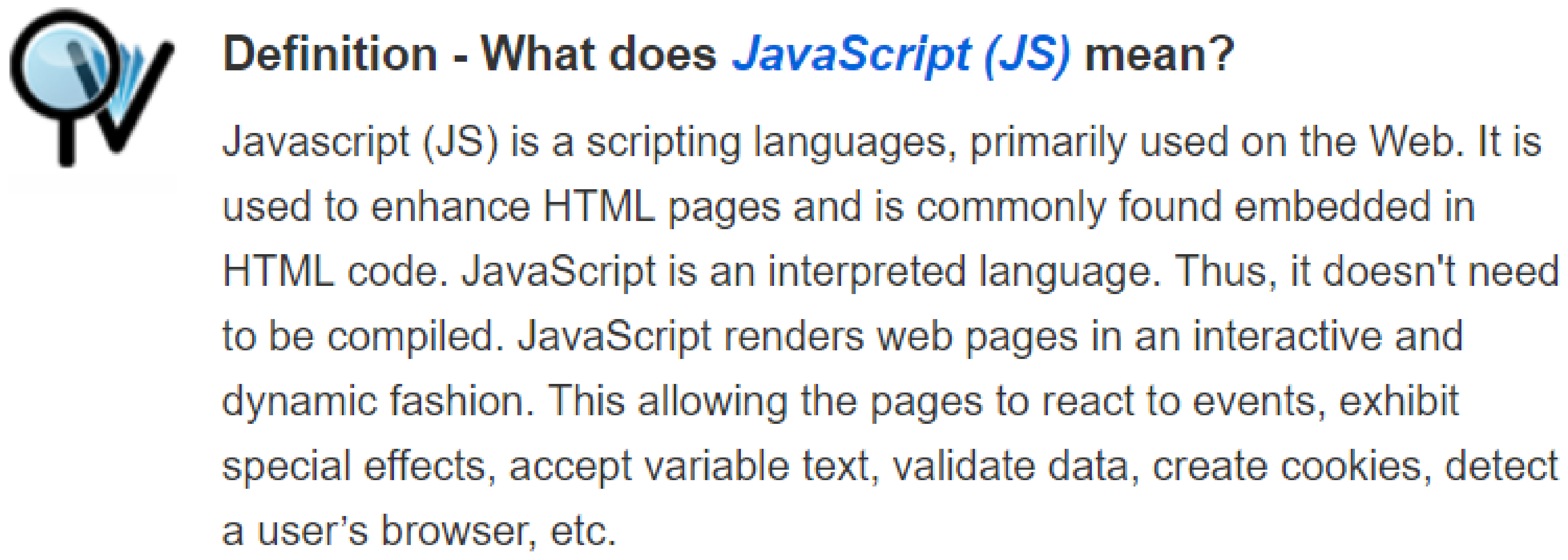
GOAL

PLAN

Why?

Why?

What is JavaScript?



Definition - What does *JavaScript (JS)* mean?

Javascript (JS) is a scripting languages, primarily used on the Web. It is used to enhance HTML pages and is commonly found embedded in HTML code. JavaScript is an interpreted language. Thus, it doesn't need to be compiled. JavaScript renders web pages in an interactive and dynamic fashion. This allowing the pages to react to events, exhibit special effects, accept variable text, validate data, create cookies, detect a user's browser, etc.

Why js?

- Beginner Friendliness
- JavaScript Is In The Browser
- Most Popular Programming Language In The World
- It’s Everywhere
- An abundance of JavaScript Jobs
- Community

History

History

- 1995 – Brendan Erich Creates JavaScript
- 1997 – ECMAScript (European Computer Manufacturers Association)
- 1999 – ECMAScript 3
- 2000~ - WAR
- 2009 – ECMAScript 5 (ES5)
- 2015 – ECMAScript 2018 (ES6)
- > 2015 - yearly updates

Tools

Tools

- Text Editor – VS Code (https://code.visualstudio.com)
- Node.js (https://nodejs.org)
- NPM (https://www.npmjs.com)
- Webpack (https://webpack.js.org)
- Git (https://git-scm.com)
- Brain (https://you.are.awesome)

Start

Hello

LIVE DEMO

Variables

Variables

- Example applications
- Naming
- Best practices

LIVE DEMO

Error

LIVE DEMO

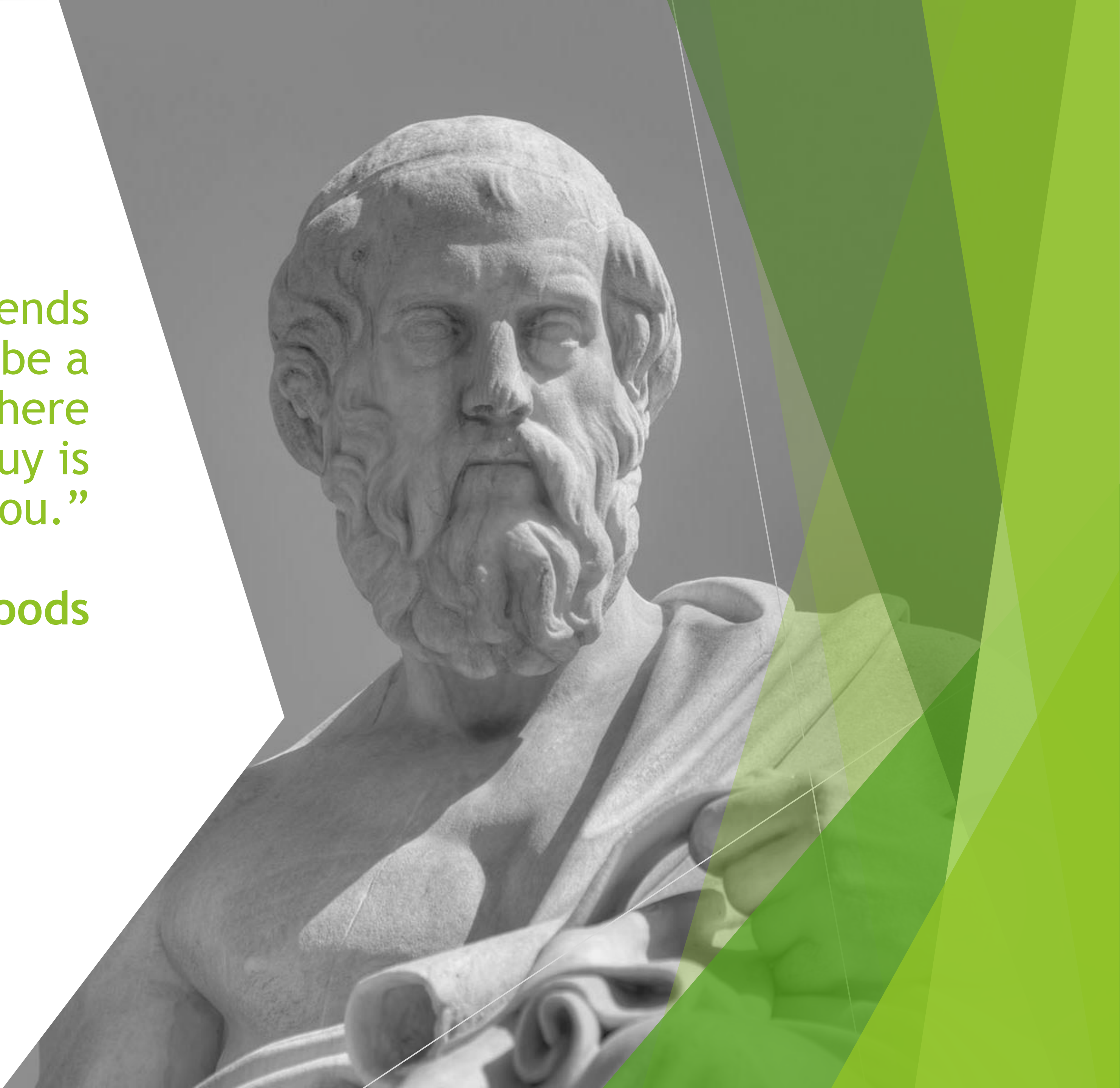
Operators (arithmetic)

- + Addition
- - Subtraction
- * Multiplication
- / Division
- % Modules
- ++ Increment by one
- -- Decrement by one

LIVE DEMO

“Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live. Because that guy is probably going to be you.”

— **John Woods**



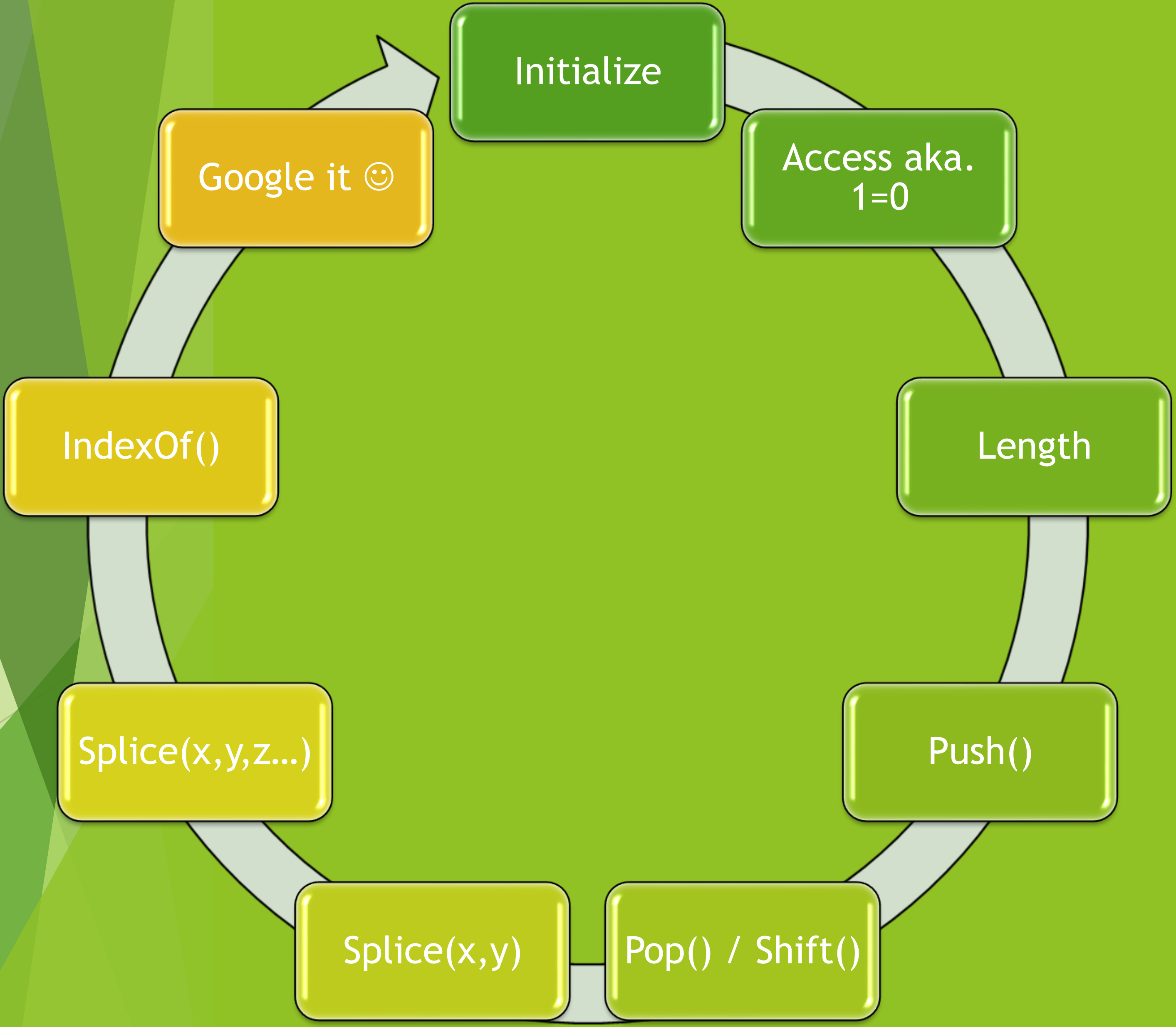
Types

Types

- String
- Number
- Boolean
- Undefined / null
- Array

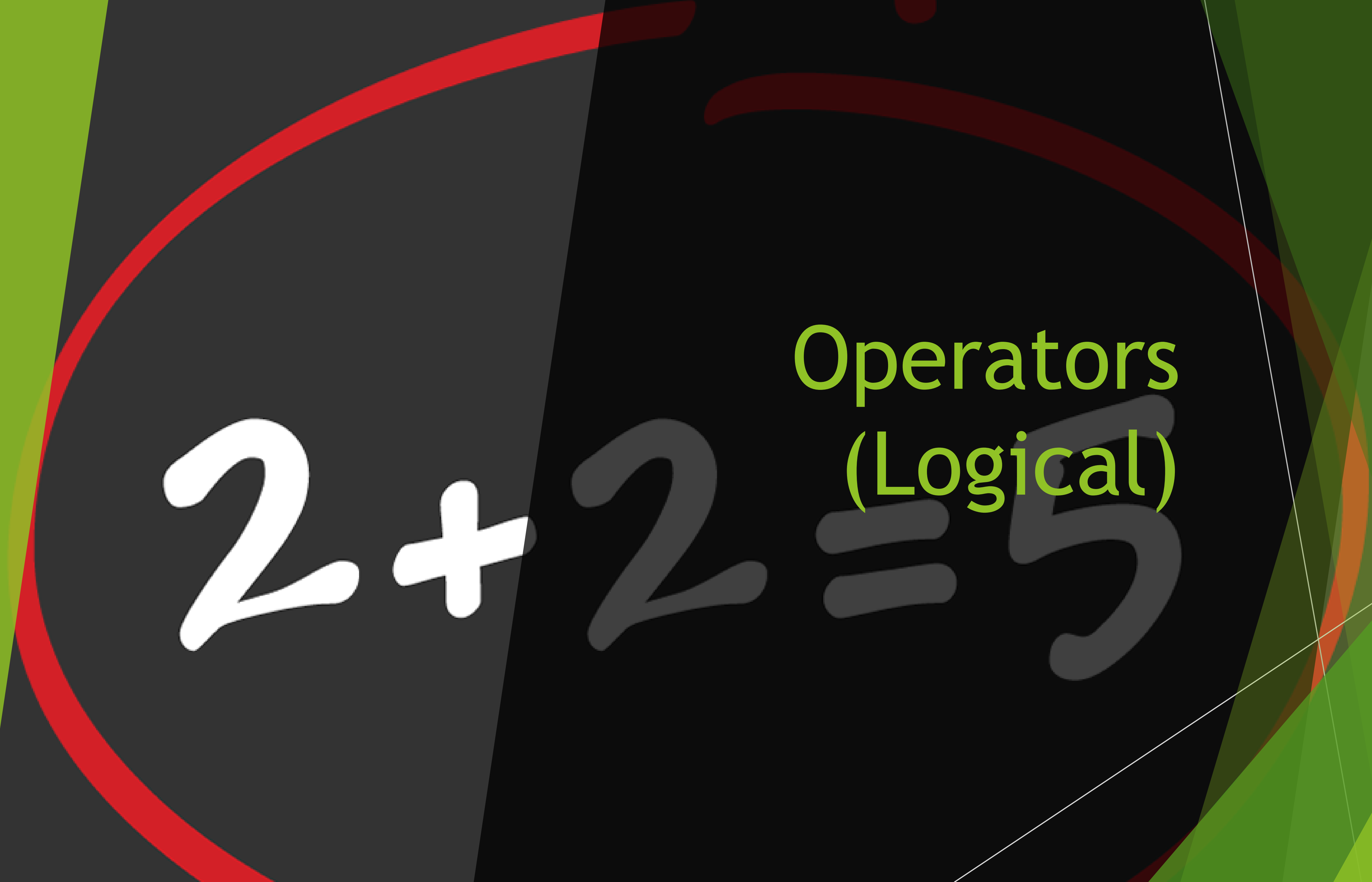
LIVE DEMO

Array


Initialize
Access aka. 1=0
Length
Push()
Pop() / Shift()
Splice(x,y)
Splice(x,y,z…)
IndexOf()
Google it ☺

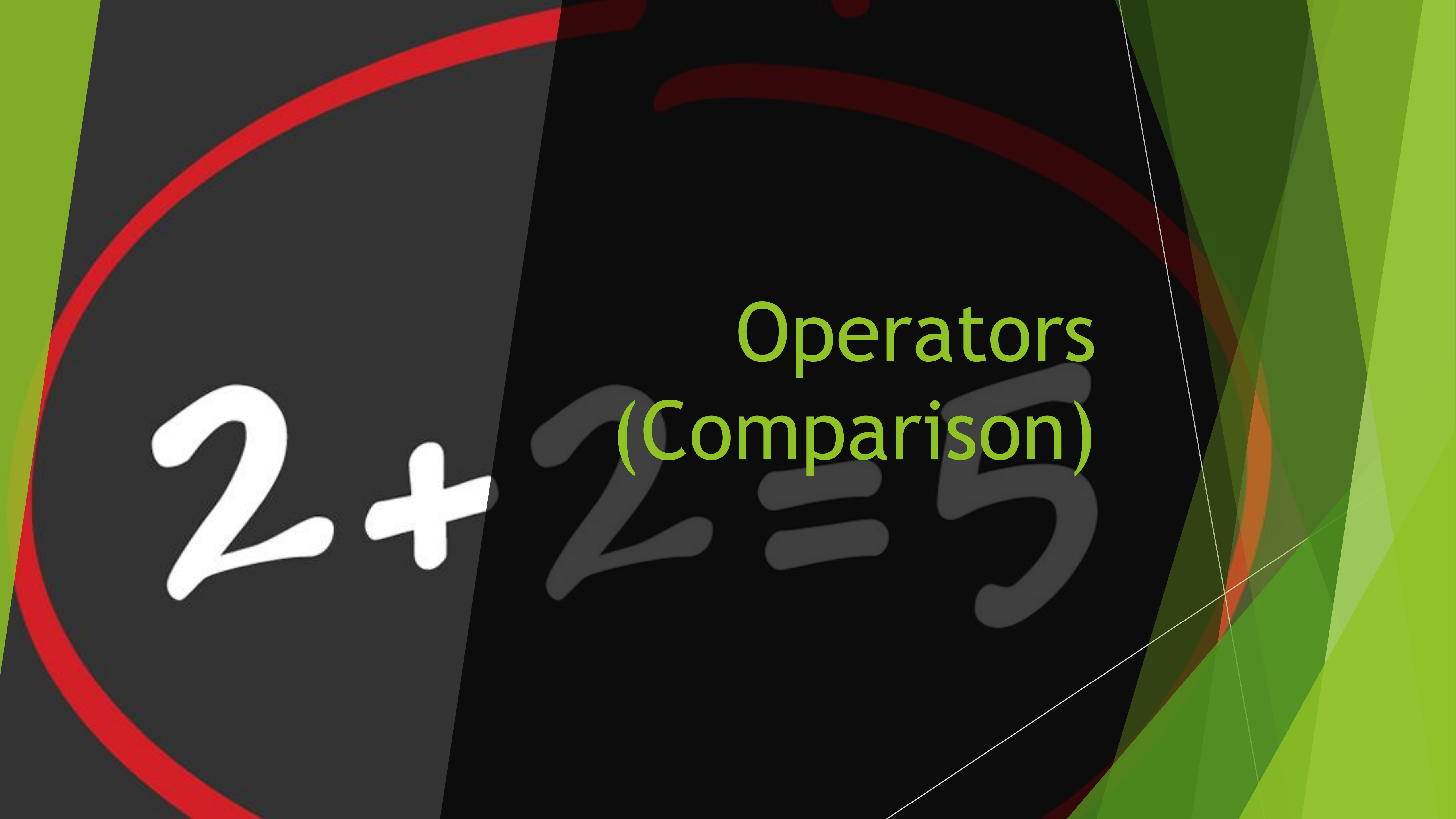
LIVE DEMO

Operators (Logical)



Operators (Logical)

OPERATOR	NAME
&&	AND
||	OR
!	NOT



Operators (Comparison)

Operators (Comparison)

OPERATOR	NAME
==	Equal
===	Strict Equal
!=	Not Equal
<	Less than
<=	Less than or equal
>	Greater than
>=	Greater than or equal

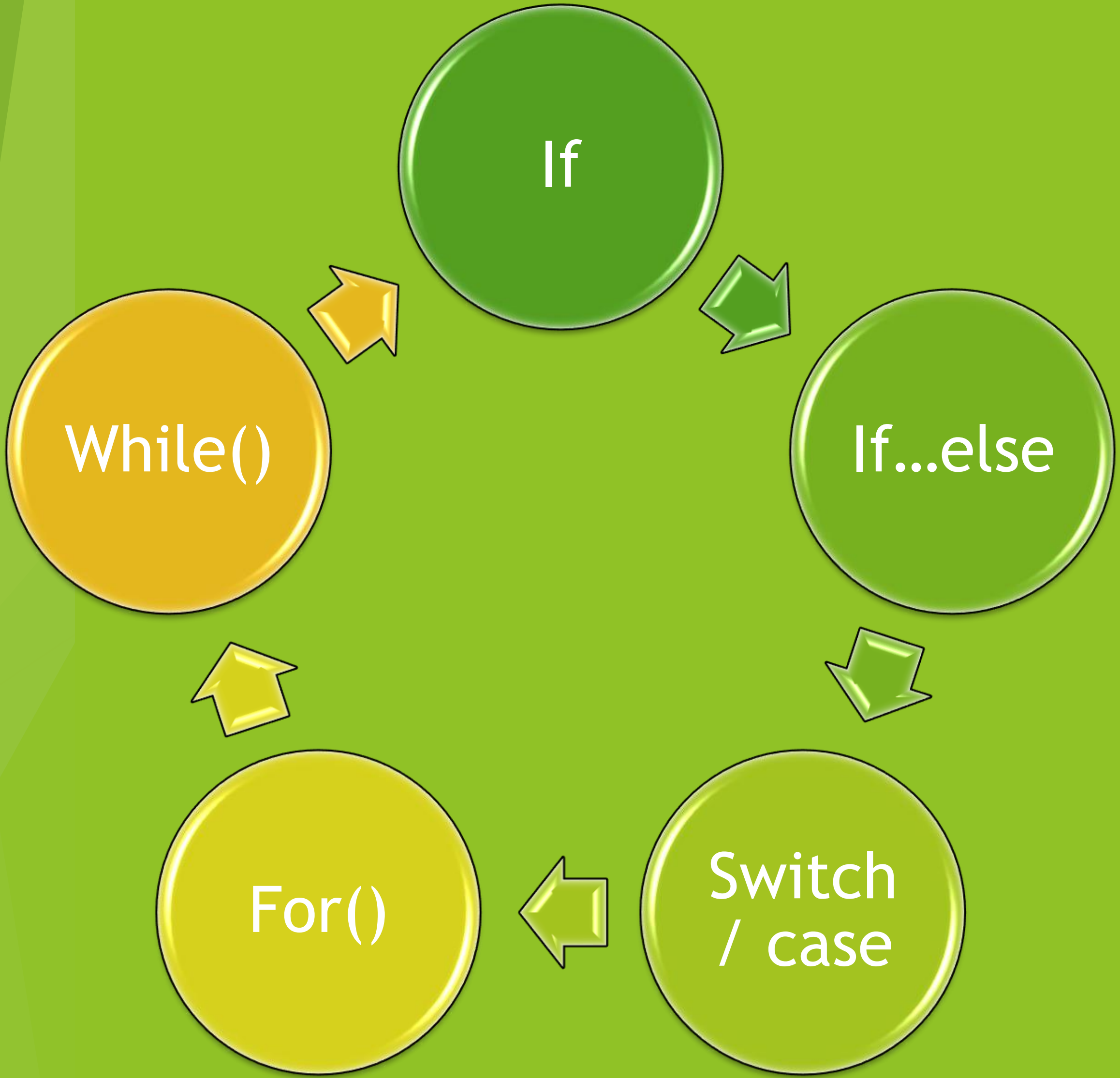
Truthy vs Falsy

Truthy	Falsy
True	False
‘0’	0
‘false’	‘’ / “”
[]	Null
{}	Undefined
function(){}	NaN

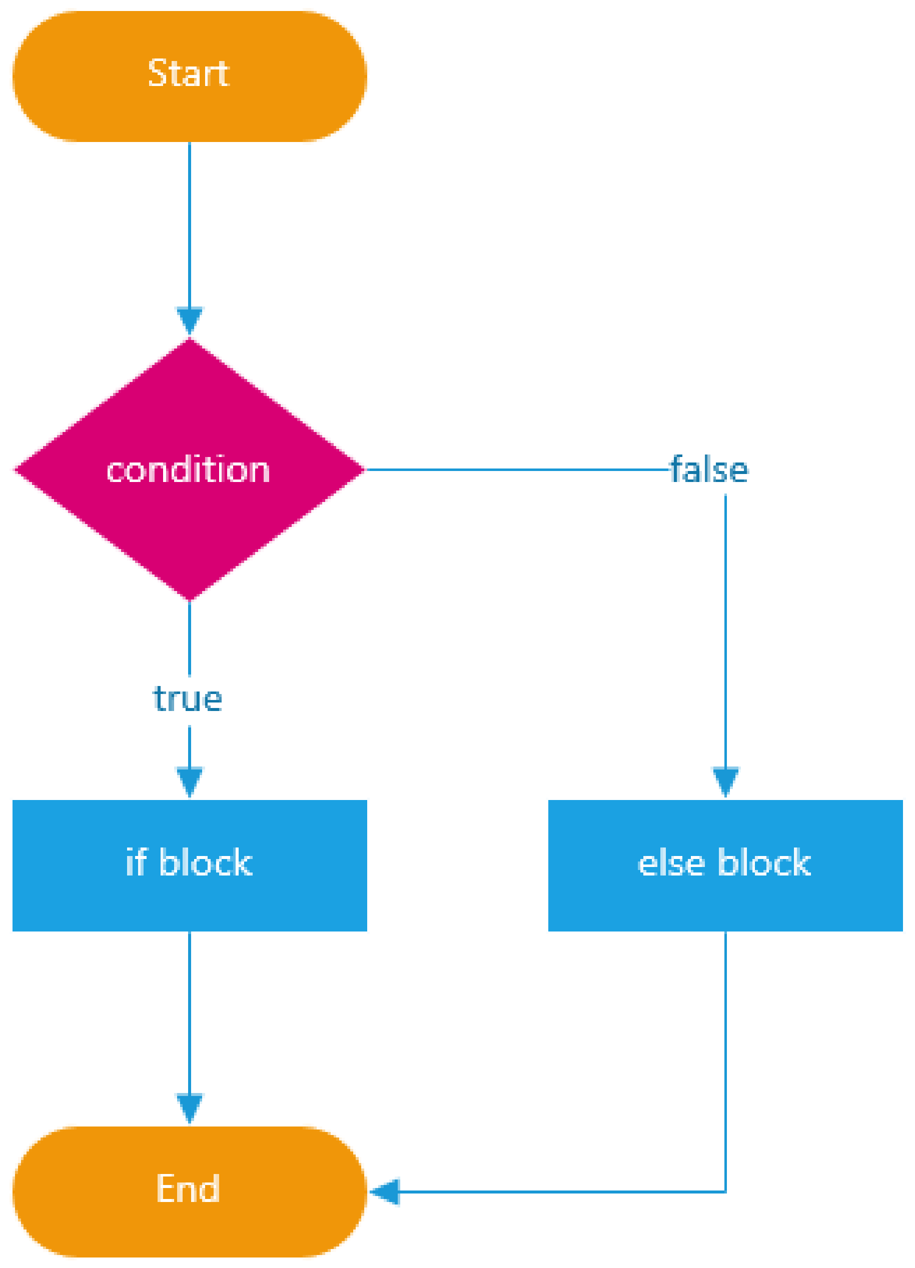
LIVE DEMO

Flow

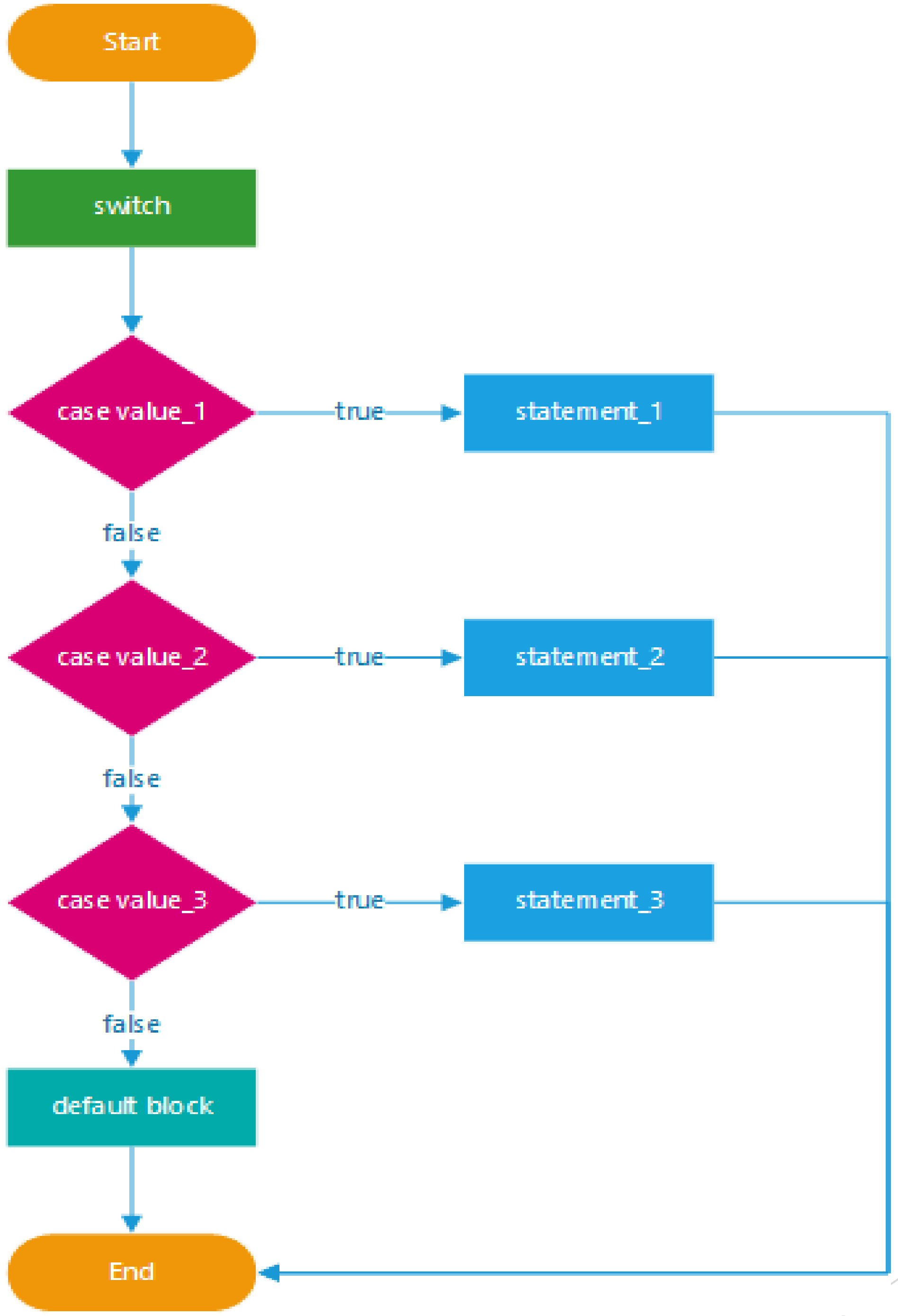
Flow



If…else

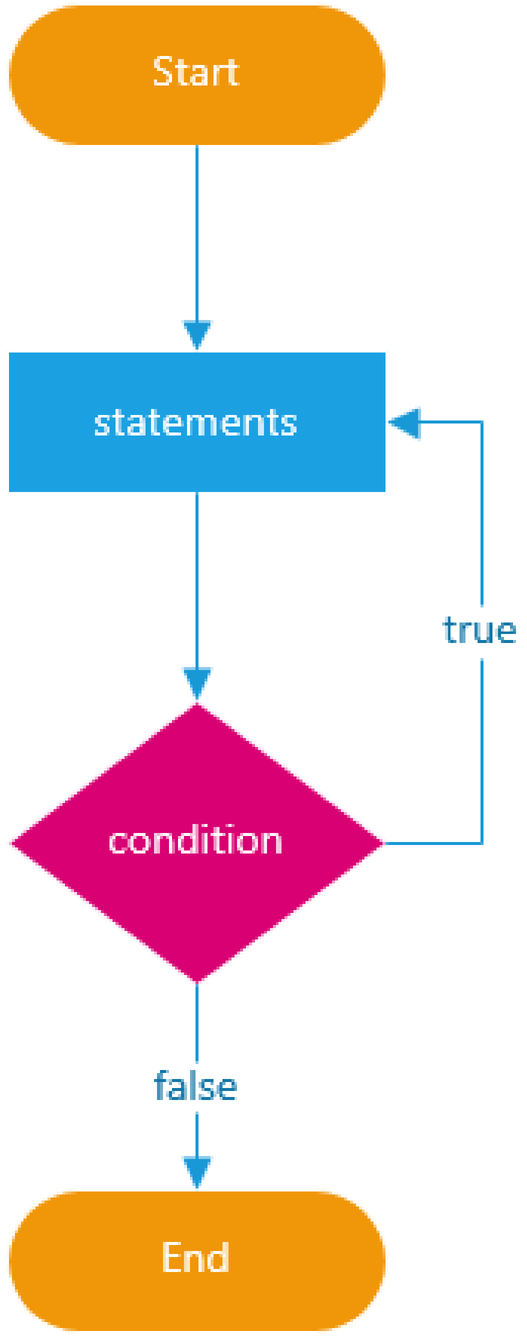


Switch...case



Start
switch
case value_1
true
statement_1
false
case value_2
true
statement_2
false
case value_3
true
statement_3
false
default block
End

For...while

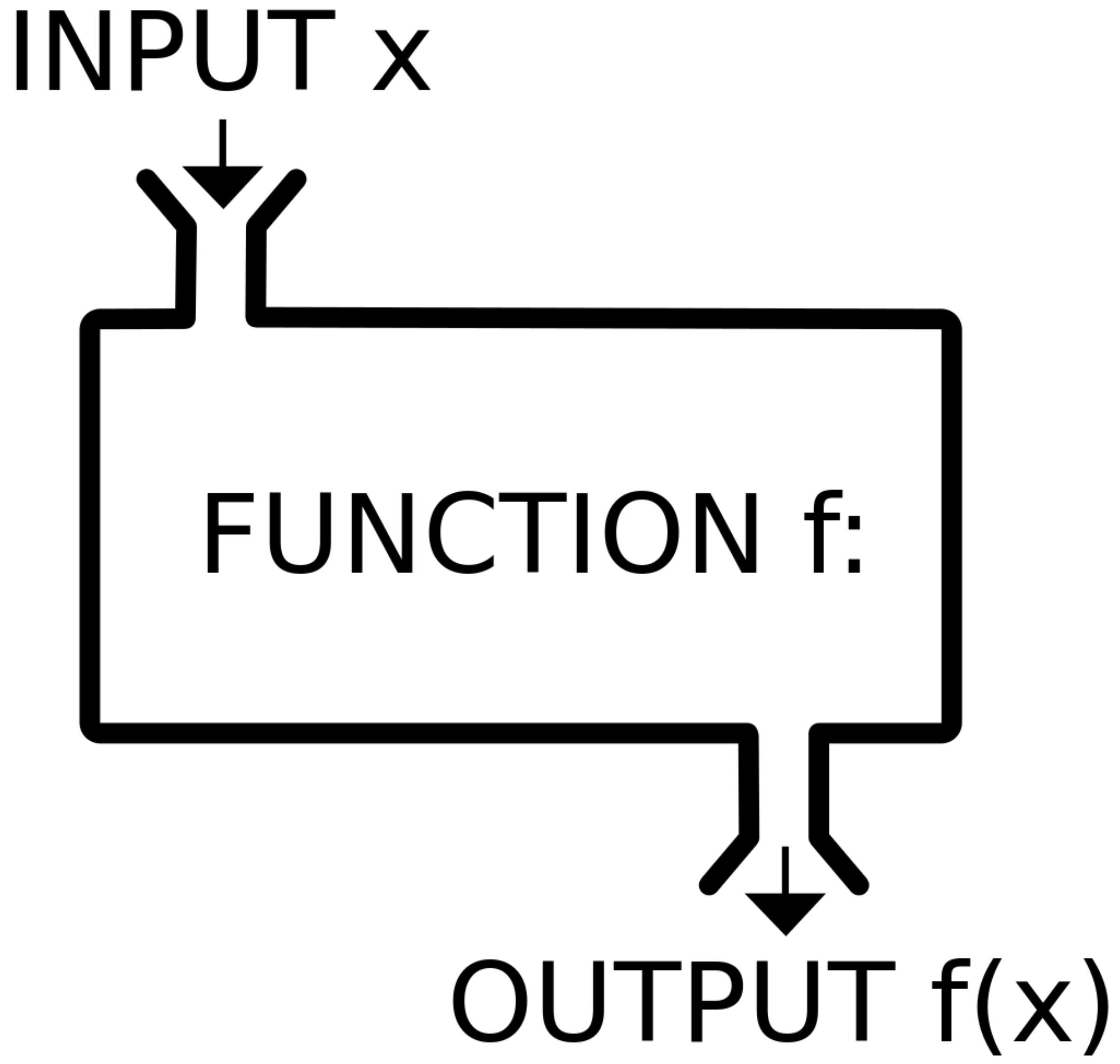


LIVE DEMO

Best practices

Avoid	Avoid direct comparisons • (x === false) --> (!x)
Use	Use === aka. Strict equality • (x == y) -> (x === y)
Convert	Convert to real boolean • (x === y) -> (!!x === !!y)

Functions


INPUT x
FUNCTION f:
OUTPUT f(x)

Functions

- Basics
- Parameters
- Return

Functions

```
function sayHello() {

}
```

Functions

```
function sayHello() {

   console.log('Hello there’);

}
```

Functions

```
function sayHello() {

console.log('Hello there');

}

sayHello();
```

Functions

```
function showValue(x){
   console.log('Value is: '+x);
}

showValue(2);
showValue('Karol');
```

Functions

```
function showSum(x,y){
   let sum = x + y;
   console.log('Sum equels :' + sum);
   console.log('Is of type :'+typeof(sum));
}

showSum(2,3);
showSum("karol",2);
showSum(2,"karol");
showSum("karol","rogowski");
```

Functions

```
let var1 = 2;
let var2 = 3;

function showSum2(x,y){
   let sum = x + y;
   console.log('Sum equels :' + sum);
   console.log('Is of type :'+typeof(sum));
   y = y+x;
   console.log(y);
}

showSum2(var1, var2);
console.log(var2);
```

Functions

```
function getSum(x,y){
    let result = x + y;
    return result;
}

let var1 = getSum(2,3);
console.log('Sum equels :' + var1);
console.log('Is of type :'+typeof(var1));

let var2 = getSum(2,'Karol');
console.log('Sum equels :' + var2);
console.log('Is of type :'+typeof(var2));

let var3 = getSum('Karol','Rogowski');
console.log('Sum equels :' + var3);
console.log('Is of type :'+typeof(var3));
```

Functions

```
function exampleFunction(){
   console.log("exampleFunction executed");
   let x = 10;
}

exampleFunction();
console.log(x);
```

Functions

```
let x =5;

function exampleFunction(){
   console.log("exampleFunction executed");
   let x = 10;
   console.log(x);
}

exampleFunction();
console.log(x);
```

Functions

```
let x =5;

function exampleFunction(){
   console.log("exampleFunction executed");
   x = 10;
   console.log(x);
}

exampleFunction();
console.log(x);
```

Functions

```
let x =5;

function exampleFunction(){
    let x =1;
    console.log("exampleFunction executed");
    x = 10;
    console.log(x);
}

exampleFunction();
console.log(x);
```

Objects

Objects

- Basics
- Objects + Functions
- Grouped Objects
- Out of the box

Objects

```
let book = {
   title: 'LOTR',
   pages: 2745,
   hardcover: true
}
```

Objects

```
let book = {
   title: 'LOTR',
   pages: 2745,
   hardCover: true
};

console.log(book.title);
console.log(book.pages);
console.log(book.hardCover);
```

Objects

```
let book = {
   title: 'LOTR',
   pages: 2745,
   hardCover: true
};
function showBookInfo(bookObject){
   console.log(bookObject.title);
   console.log(bookObject.pages);
   console.log(bookObject.hardCover);
}

showBookInfo(book);
```

Objects

```
let book = {
   title: 'LOTR',
   pages: 2745,
   hardCover: true
};

function changeCover(bookObject){
   bookObject.hardCover = !bookObject.hardCover;
   console.log('Cover changed');
}

changeCover(book);
showBookInfo(book);
```

Objects

```
let books = [
    {
        title: 'LOTR',
        pages: 2745,
        hardCover: true
    },
    {
        title: 'Witcher',
        pages: 1266,
        hardCover: false
    },
    {
        title: 'Sherlock Holmes',
        pages: 1950,
        hardCover: true
    }
];
```

Objects

```
for(let i = 0; i < books.length; i++){
   showBookInfo(books[i]);
}

books.forEach(function(book) {
   showBookInfo(book);
});
```

Out of the box

Math

Date

String

Number

Error

Function

Language Features

Language Features

- Constants
- Let and Var
- Rest Parameters
- Destructing Array
- Destructing Object
- Spread

Constants

```
const constVar =2;
console.log(constVar);
```

Constants

```
const constVar;
console.log(constVar);
```

Constants

```
const constVar =2;

constVar =3;

console.log(constVar);
```

Let and var

```
console.log(varLet);
let varLet = 'varLet’;

console.log(varVar);
var varVar = 'varVar';
console.log(varVar);
```

Let and var

```
if(true){
    let varLet =1;
}
console.log(varLet);

if(true){
    var varVar =1;
}
console.log(varVar);
```

Let and var

```
if(true){
   var varVar =1;
}

console.log(varVar);
varVar =2;
console.log(varVar);

var varVar ='varVar';
console.log(varVar);
```

Rest parameters

```
function ShowData(a,b,...c){
   console.log(a);
   console.log(b);
   console.log(c);
}

ShowData(1,2,3,4,5,6);
ShowData(1);
ShowData(1,2);
ShowData(1,2,3,'four','5',6);
```

Destructing array

```
let ids = [1,2,3,4];
let [id1, id2, id3] = ids;
console.log(id1);
console.log(id2);
console.log(id3);
```

Destructing array

```
let ids = [1,2,3,4];

let [mainId, ...remainingIds] = ids;

console.log(mainId);
console.log(remainingIds);
```

Destructing array

```
let ids = [1,2,3,4];

let mainId;
let [, ...remainingIds] = ids;

console.log(mainId);
console.log(remainingIds);
```

Destructing array

```
let ids = [1,2,3,4];

let [mainId,, ...remainingIds] = ids;

console.log(mainId);
console.log(remainingIds);
```

Destructing objects

```
var person = {
   id : 1,
   name : 'Karol'
}

let { id, name } = person;
console.log(id,name);
```

Destructing objects

```
var person = {
   id : 1,
   name : 'Karol'
}

let id, name;
{id, name} = person;
console.log(id,name);

({id, name} = person);
console.log(id,name);
```

Destructing objects

```
var person = {
   id : 1,
   name : 'Karol'
}

let id, name, year;
({id, name,year} = person);
console.log(id,name,year);
```

Spread

```
function ShowData(a,b){
   console.log(a,b);
}

let values = [1,2];
ShowData(...values);
```

Spread

```
function ShowData(a,b){
   console.log(a,b);
}

let text1 = 'ab';
ShowData(...text1);

let text2 = 'a';
ShowData(...text2);

let text3 = 'abc';
ShowData(...text3);
```

Functions (in depth)

Functions (in depth)

- Function Scope
- Block Scope
- IIFE (Immediately Invoked Function Expression)
- Closure
- this
- Call / Apply
- Bind
- Arrow function
- Default values

Function Scope

```
function outerFunction(param1){
   let variable1 = 'variable1';
}

outerFunction('example data');
console.log(variable1);
```

Function Scope

```
function outerFunction(param1){
    let variable1 = 'variable1';
    let innerFunction = function innerFunctionDefinition(){
        console.log(variable1, param1);
    }
    innerFunction();
}

outerFunction('example data');
```

Function Scope

```
function outerFunction(param1){
    let variable1 = 'variable1';
    let innerFunction = function innerFunctionDefinition(){
        let variable1 = 'variable inner version';
        console.log(variable1);
    }
    innerFunction();
    console.log(variable1);
}

outerFunction('example data');
```

Block Scope

```
if(true){
   let var1 = 'var1';
}

console.log(var1);
```

Block Scope

```
let var1 = 'outer vaue'
if(true){
   let var1 = 'inner value';
   console.log(var1);
}

console.log(var1);
```

IIFE

```
function one(){
   console.log('one');
};

(function(){
   console.log('two');
})();

one();
```

IIFE

```
let iife = (function(){
   let var1 = 'iife value';
   console.log(var1);
   return {};
})();

console.log(iife);
```

Closure

```
let iife = (function(){
   let var1 = 'inner';
   let getValue = function(){
      return var1;
   };
   return {
      innerData: getValue
   };
})();

console.log(iife.innerData());
```

this

```
(function(){
   console.log(this);
})();
```

this

```
let obj = {
   id:1,
   getThisId: function(){
      let id =2;
      return this.id;
   },
   getId: function(){
      let id =2;
      return id;
   }
}
```

Call

```
let obj = {
    id:1,
    getId: function(){
        return this.id;
    }
}

let contextObject = {id:2};

console.log(obj.getId());
console.log(obj.getId.call(contextObject));
```

Apply

```
let obj = {
    id:1,
    getId: function(par1, par2){
        return par1+ this.id+par2;
    }
}

let contextObject = {id:2};

console.log(obj.getId('p','s'));
console.log(obj.getId.apply(contextObject,['prefix ',' sufix']));
```

Bind

```
let obj = {
    id:1,
    getId: function(){
        return this.id;
    }
}

let contextObject = {id:2};
let newGetId = obj.getId.bind(contextObject);

console.log(newGetId());
```

Arrow function

```
let fun1 = () => 'fun1';
console.log(fun1());
```

Arrow function

```
let fun2 = prefix => prefix + 'fun1';
console.log(fun2('p'));
```

Arrow function

```
let fun3 = (prefix,sufix) => prefix + 'fun1' + sufix;
console.log(fun3('p','s'));
```

Arrow function

```
let funSum = (x, y)=>{
   let result = x+y;
   return result
};
console.log(funSum(4,7));
```

Default values

```
let showInfo = function(main, prefix='P', sufix = 'S’){
    console.log(prefix, main, sufix);
};

showInfo();
showInfo('example');
showInfo('example','My Prefix');
showInfo('example','My Prefix','My Sufix');
```

Iffe (question ☺)

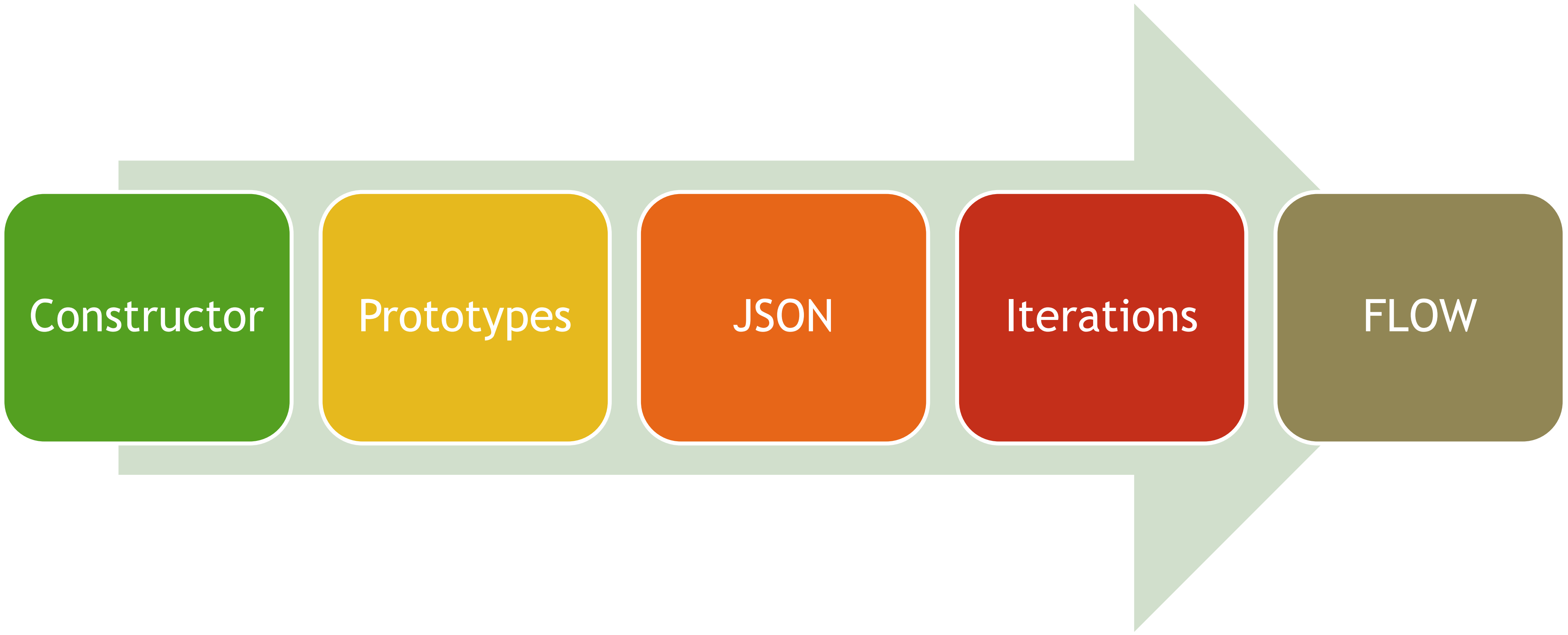
```
let dataObject = {
    id:1,
    data: 'example data'
}

var proxy = (function(foo){
    return {
        getData: function(){
            return foo;
        },
        setData: function(val) {
            foo.data = val;
        }
    }
})(dataObject);

console.log(proxy.getData());
proxy.setData('changed data');
console.log(proxy.getData());

console.log(dataObject);
```

Iffe (question ☺)

```
let dataObject = {
    id:1,
    data: 'example data'
}

var proxy = (function(foo){
    return {
        getData: function(){
            return foo;
        },
        setData: function(val) {
            foo.data = val;
        }
    }
})(Object.assign({},dataObject));

console.log(proxy.getData());
proxy.setData('changed data');
console.log(proxy.getData());

console.log(dataObject);
```

Arrays and Objects

Arrays and Objects



Constructor

```
function Person(){
}

let karol = new Person();

console.log(karol);
```

Constructor

```
function Person(){
    console.log(this);
}

let karol = new Person();
let adam = Person();
```

Constructors

```
function Person(firstName, lastName){
    this.firstName = firstName;
    this.lastName = lastName;
}

let karol = new Person('Karol','Rogowski');

console.log(karol);
```

Constructor

```
function Person(firstName, lastName){
    this.firstName = firstName;
    this.lastName = lastName;
    this.sayHello = () => console.log( 'Hello from ' + this.firstName +
                                                    ' ' + this.lastName);
}

let karol = new Person('Karol','Rogowski');

console.log(karol);
karol.sayHello();
```

Prototype

```
function Person(firstName, lastName){
    this.firstName = firstName;
    this.lastName = lastName;
}

Person.prototype.sayHello = function() {
    console.log( 'Hello from ' + this.firstName + ' ' + this.lastName);
}

let karol = new Person('Karol','Rogowski');

console.log(karol);
karol.sayHello();
```

Prototype

```
function Person(firstName, lastName){
    this.firstName = firstName;
    this.lastName = lastName;
}

let karol = new Person('Karol','Rogowski');

Person.prototype.sayHello = function() {
    console.log( 'Hello from ' + this.firstName + ' ' + this.lastName);
}

console.log(karol);
karol.sayHello();
```

Prototype

```
function Person(firstName, lastName){
    this.firstName = firstName;
    this.lastName = lastName;
}

let karol = new Person('Karol','Rogowski');

console.log(karol);
karol.sayHello();

Person.prototype.sayHello = function() {
    console.log( 'Hello from ' + this.firstName + ' ' + this.lastName);
}
```

Prototype

```
function Person(firstName, lastName){
    this.firstName = firstName;
    this.lastName = lastName;
}

let karol = new Person('Karol','Rogowski');

Person.prototype.sayHello = () => {
    console.log( 'Hello from ' + this.firstName + ' ' + this.lastName);
    console.log(this);
}

console.log(karol);
karol.sayHello();
```

Prototype

```
String.prototype.showMe = function(){
    console.log('Hello world from '+this);
}

'Karol Rogowski'.showMe();
```

Prototype

```
Number.prototype.getValueDescription = function(){
    return "My value is: " + this
}

console.log((4).getValueDescription());
```

Prototype

```
function Demo(){
    console.log('Demo function result');
}

Function.prototype.customRun = function(){
    console.log('Custom run begin');
    this();
    console.log('Custom run end');
}

Demo.customRun();
```

JSON

```
let person = {
    id: 1,
    name:'Karol Rogowski'
}

console.log(person);
console.log(JSON.stringify(person));
```

JSON

```
let people = [{
        id: 1,
        name:'Karol Rogowski'
    },{
        id: 2,
        name:'Jan Kowalski'
    },{
        id: 3,
        name:'Robert Lewandowski’
    }]

console.log(people);
console.log(JSON.stringify(people));
```

JSON

```
let personJSON = `{
    "id":1,
    "name":"Karol Rogowski"
}`;

let person = JSON.parse(personJSON);
console.log(person);
```

JSON

```
let peopleJSON = `[
       {
           "id":1,
           "name":"Karol Rogowski"
       },
       {
           "id":2,
           "name":"Jan Kowalski"
       },
       {
           "id":3,
           "name":"Robert Lewandowski"
       }
   ]`

let people = JSON.parse(peopleJSON);
console.log(people);
```

Array Iteration

```
let people = [
    {
        innerId: 'dfr458hj',
        name: 'Karol Rogowski',
        birthYear: 1985,
        sayHello: function(){console.log(this.name+ ' says hello')}
    },
    {
        innerId: 'plo745as',
        name: 'Jan Kowalski',
        birthYear:1980,
        sayHello: function(){console.log(this.name+ ' says hello')}
    },
    {
        innerId: 'qaz390pl',
        name: 'Robert Lewandowski',
        birthYear: 1988,
        sayHello: function(){console.log(this.name+ ' says hello')}
    }
]
```

Array Iteration

```
people.forEach(p => console.log(p));

people.forEach((p,i)=>console.log(i+':'+ p.name));

people.forEach(p => p.sayHello());
```

Array Iteration

```
console.log(people.filter(p=> p.birthYear > 1980));

console.log(people.every(p=> p.birthYear > 1980));

console.log(people.every(p=> p.birthYear >= 1980));
```

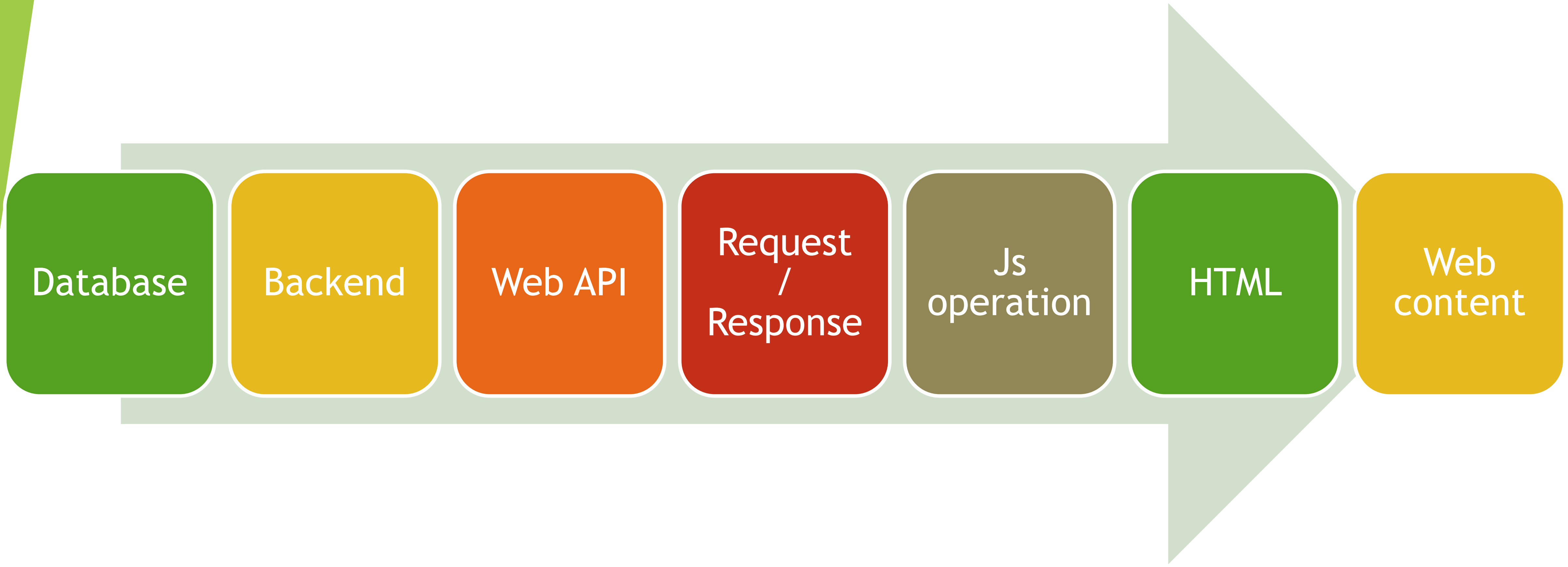
Array Iteration

```
console.log(people.map((p,i)=>i + ':' + p.name));

console.log(people.find(p=>p.birthYear != 1985));
```

Array Iteration

```
function Person(firstName, lastName, id){
    this.id = id;
    this.firstName = firstName;
    this.lastName = lastName;
}

console.log(people.map((p,i)=> new Person(p.name.split(' ')[0],
                                          p.name.split(' ')[1],
                                          i)));
```

Flow – big picture



Flow “API”

```
let apiObject = {
    getPeople: ()=>`[
        {
            "id":1231,
            "name":"Karol Rogowski"
        },
        {
            "id":2123,
            "name":"Jan Kowalski"
        },
        {
            "id":3111,
            "name":"Robert Lewandowski"
        }
    ]`
}
```

FLOW "mappings"

```
function Person(firstName, lastName, id){
    this.id = id;
    this.firstName = firstName;
    this.lastName = lastName;
}

let result = JSON.parse(apiObject.getPeople());

console.log(result.map((p)=> new Person(p.name.split('')[0],
                                        p.name.split(' ')[1],p.id)));
```

Classes and Modules

01

Basic

02

Constructor

03

Methods

04

Inheritance

05

Modules

Classes and Modules

Basic

```
class Person {

}

let me = new Person();
console.log(me);
```

Constructor

```
class Person {
   constructor(id){
      this.id = id;
   }
}

let me = new Person(1);
console.log(me);
```

Constructor

```
class Person {
   constructor(id){
      personId = id;
   }
}

let me = new Person(1);
console.log(me);
```

Constructor

```
class Person {
   constructor(id){
      let personId = id;
   }
}

let me = new Person(1);
console.log(me);
```

Methods

```
class Person {
    constructor(id){
        this.id = id;
    }
    showInfo(){
        return `Person Id: ${this.id}`
    }
}

let me = new Person(1);
console.log(me);
console.log(me.showInfo());
```

Methods

```
class Person {
    constructor(id, firstName, lastName){
        this.id = id;
        this.firstName = firstName;
        this.lastName = lastName;
    }

    showInfo(){
        return `${this.firstName + ' ' + this.lastName} Id: ${this.id}`
    }
}

let me = new Person(1,'Karol','Rogowski');
console.log(me);
console.log(me.showInfo());
```

Inheritance

```
class Person {
    constructor(){
        this.type = 'basic person';
    }
    showInfo(){
        return `Of type ${this.type}`
    }
}

class JsDeveloper extends Person{
}

let jsDev = new JsDeveloper();
console.log(jsDev);
console.log(jsDev.showInfo());
```

Inheritance

```
class Person {
    constructor(id){
        this.id = id;
        this.type = 'basic person';
    }
    showInfo(){
    return `Of type ${this.type} and id ${this.id}`
    }
}

class JsDeveloper extends Person{
    constructor(id){
        super(id);
    }
}

let jsDev = new JsDeveloper(5);
console.log(jsDev);
console.log(jsDev.showInfo());
```

Inheritance

```
class Person {
    constructor(id){
        this.id = id;
        this.type = 'basic person';
    }
    showInfo(){
        return `Of type ${this.type} and id ${this.id}`
    }
}

class JsDeveloper extends Person{
    constructor(id){
        super(id);
        this.type = 'JS Developer'
    }
}

let jsDev = new JsDeveloper(5);
console.log(jsDev);
console.log(jsDev.showInfo());
```

Inheritance

```
class Person {
    constructor(id){
        this.id = id;
        this.type = 'basic person';
    }
    showInfo(){
        return `Of type ${this.type} and id ${this.id}`
    }
}

class JsDeveloper extends Person{
    constructor(id,framework){
        super(id);
        this.type = 'JS Developer';
        this.framework = framework;
    }
}

let jsDev = new JsDeveloper(5,'React');
console.log(jsDev);
console.log(jsDev.showInfo());
```

Inheritance

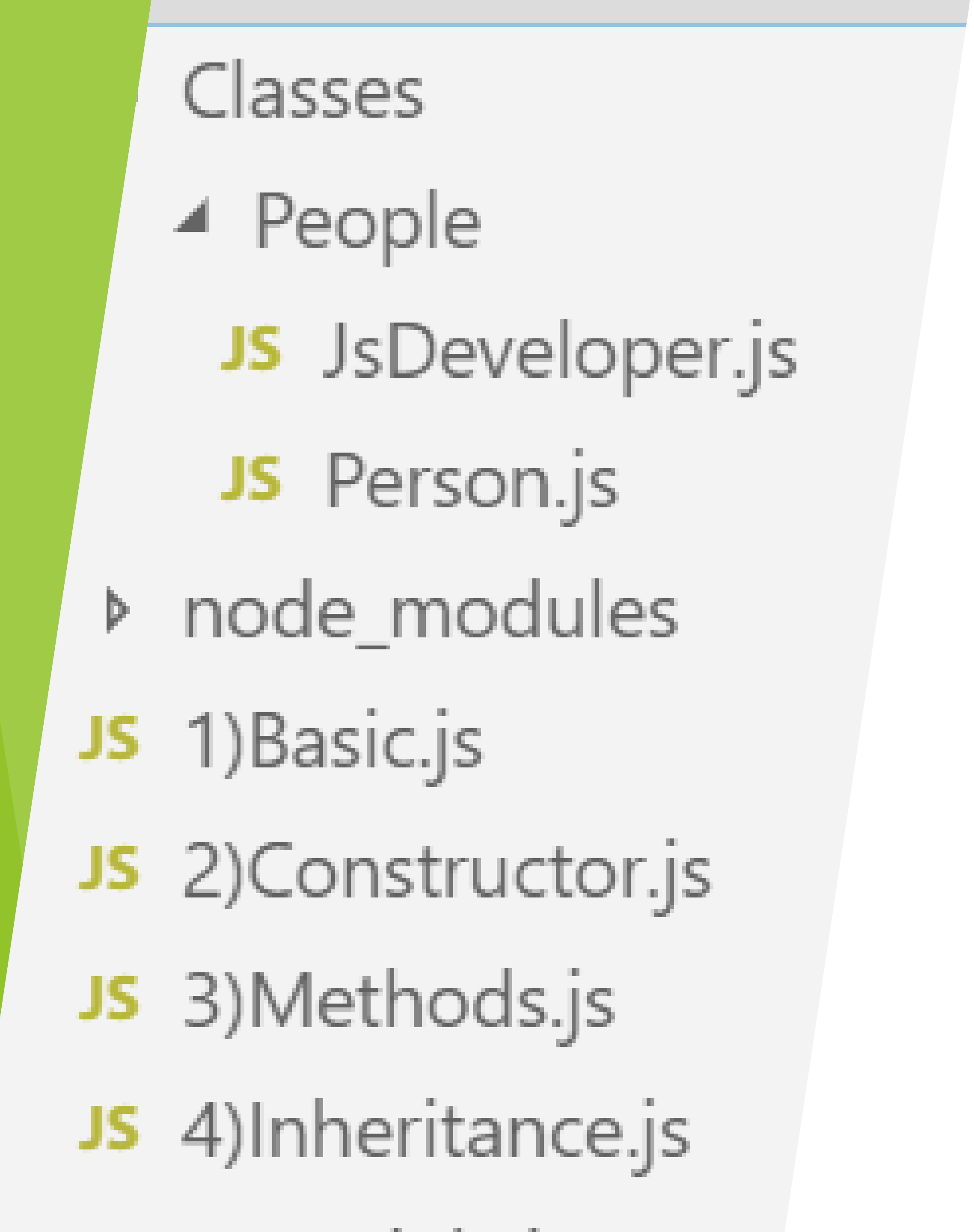
```
class Person {
    constructor(id){
        this.id = id;
        this.type = 'basic person';
    }
    showInfo(){
        return `Of type ${this.type} and id ${this.id}`
    }
}

class JsDeveloper extends Person{
    constructor(id,framework){
        super(id);
        this.type = 'JS Developer';
        this.framework = framework;
    }

    showDeveloperInfo(){
        return `Of type ${this.type}, id ${this.id} and favourite frameworks is ${this.framework}`
    }
}

let jsDev = new JsDeveloper(5,'React');
console.log(jsDev);
console.log(jsDev.showDeveloperInfo());
```

Inheritance

```
class Person {
    constructor(id){
        this.id = id;
        this.type = 'basic person';
    }
    showInfo(){
        return `Of type ${this.type} and id ${this.id}`
    }
}

class JsDeveloper extends Person{
    constructor(id,framework){
        super(id);
        this.type = 'JS Developer';
        this.framework = framework;
    }

    showInfo(){
        return `Of type ${this.type}, id ${this.id} and favourite frameworks is ${this.framework}`
    }
}

let jsDev = new JsDeveloper(5,'React');
console.log(jsDev);
console.log(jsDev.showInfo());
```

Inheritance

```
class Person {
    constructor(id){
        this.id = id;
        this.type = 'basic person';
    }
    showInfo(){
        return `Of type ${this.type} and id ${this.id}`
    }
}

class JsDeveloper extends Person{
    constructor(id,framework){
        super(id);
        this.type = 'JS Developer';
        this.framework = framework;
    }

    showInfo(){
        return super.showInfo() + ` and favourite frameworks is ${this.framework}`
    }
}

let jsDev = new JsDeveloper(5,'React');
console.log(jsDev);
console.log(jsDev.showInfo());
```



Classes
- People
  - JsDeveloper.js
  - Person.js
- node_modules
- 1)Basic.js
- 2)Constructor.js
- 3)Methods.js
- 4)Inheritance.js

Module 1/4

Module 2/4 (*Person.js*)

```
class Person {
   constructor(id){
      this.id = id;
      this.type = 'basic person’;
   }

   showInfo(){
   return `Of type ${this.type} and id ${this.id}`
   }
}

module.exports = Person;
```

Module 3/4 (*JsDeveloper.js*)

```
var Person = require("./Person");

class JsDeveloper extends Person{
    constructor(id,framework){
        super(id);
        this.type = 'JS Developer';
        this.framework = framework;
    }

    showInfo(){
        return super.showInfo() + ` and favourite frameworks is ${this.framework}`
    }
}

module.exports = JsDeveloper;
```

Module 4/4 (*Module.js*)

```
let JsDeveloper = require("./Classes/People/JsDeveloper");

let jsDev = new JsDeveloper(5,'React');
console.log(jsDev);
console.log(jsDev.showInfo());
```

Errors

1

Error

2

Try / Catch

3

Finally

4

Custom Error

Errors

Error

```
let person = Karol
console.log(person);
```

Try / Catch

```
try {
   let person = Karol
} catch (error) {
   console.log('error: ', error);
}

console.log('done');
```

Finally

```
try {
   let person = Karol
} catch (error) {
   console.log('error: ', error);
} finally {
   console.log('finally block reasech’)
}

console.log('done');
```

Custom Error

```
try {
   throw new Error('Custom application error')
} catch (error) {
   console.log('error: ', error);
} finally {
   console.log('finally block reasech’)
}

console.log('done');
```

Promises

01
Basic

02
Static Result

03
Generator

04
Many Promises

05
Chaining

06
All

07
Race

Promises

Basic

```
let promise = new Promise(
    function(resolve, reject){
        console.log('promise code executed');
        setTimeout(resolve, 500, 'Karol Rogowski');
    }
);
```

Basic

```
let promise = new Promise(
    function(resolve, reject){
        console.log('promise code executed');
        setTimeout(resolve, 500, 'Karol Rogowski');
    }
);

promise.then(
    value => console.log('fullfilled: ' + value),
    error => console.log('rejected: ' + error)
);
```

Basic

```
let promise = new Promise(
    function(resolve, reject){
        console.log('promise code executed');
        setTimeout(reject, 500, 'Karol Rogowski');
    }
);

promise.then(
    value => console.log('fullfilled: ' + value),
    error => console.log('rejected: ' + error)
);
```

Basic

```
let promise = new Promise(
    function(resolve, reject){
        console.log('promise code executed');
        setTimeout(reject, 500, 'Karol Rogowski');
    }
);

promise.then(
    value => console.log('fullfilled: ' + value)
);

promise.catch(
    error => console.log('rejected: ' + error)
);
```

Basic

```
let promise = new Promise(
    function(resolve, reject){
        console.log('promise code executed');
        setTimeout(reject, 500, 'Karol Rogowski');
    }
);

promise.catch(
    error => console.log('rejected: ' + error)
);

promise.then(
    value => console.log('fullfilled: ' + value)
);
```

Basic

```
let promise = new Promise(
    function(resolve, reject){
        console.log('promise code executed');
        setTimeout(reject, 500, 'Karol Rogowski');
    }
);

promise.catch(
    error => console.log('rejected: ' + error)
);

promise.then(
    value => console.log('fullfilled: ' + value)
);

promise.catch(
    error => console.log('rejected2: ' + error)
);
```

Basic

```
let promise = new Promise(
    function(resolve, reject){
        console.log('promise code executed');
        setTimeout(reject, 500, 'Karol Rogowski');
    }
);

promise.catch(
    error => console.log('rejected: ' + error)
);

promise.then(
    value => console.log('fullfilled: ' + value),
    error => console.log('rejected3: ' + error)
);

promise.catch(
    error => console.log('rejected2: ' + error)
);
```

Basic

```
let promise = new Promise(
    function(resolve, reject){
        setTimeout(resolve, 1000, 'Karol Rogowski');
    }
);

console.log('before handle');

promise.then(
    value => console.log('fullfilled: ' + value),
    error => console.log('rejected: ' + error)
);

console.log('after handle');
```

Basic

```
var trustworthy = false; // true
let promise = new Promise(function(resolve, reject) {
    if (trustworthy) {
        resolve("The person is trustworthy");
    } else {
        reject("The person can't be trusted");
    }
});


promise.then(
    value => console.log('fullfilled: ' + value),
    error => console.log('rejected: ' + error)
);
```

Basic

```
var trustworthy = true;
let promise = new Promise(function(resolve, reject) {
setTimeout(function() {
    if (trustworthy) {
        resolve(
            {
                value: "The person is trustworthy",
                code: "CD1_TPIT"
            });
    } else {
        reject(
        {
            value:"The person can't be trusted",
            code: "CD2_TPCNBT"
        });
    }
    }, 1000);
});

promise.then(
value => console.log('fullfilled1: ' + JSON.stringify(value)),
error => console.log('rejected1: ' + JSON.stringify(error))
)
```

Basic

```
                    value:"The person can't be trusted",
                    code: "CD2_TPCNBT"
                });
            }
        }, 1000);
});

promise.then(
    value => console.log('fullfilled1: ' + JSON.stringify(value)),
    error => console.log('rejected1: ' + JSON.stringify(error))
)

promise.then(
    value => console.log('fullfilled2: ' + JSON.stringify(value)),
    error => console.log('rejected2: ' + JSON.stringify(error))
);
```

Static Result

```
var resolvedPromise = Promise.resolve(123);

resolvedPromise.then(
    value => console.log('fullfilled: ' + value),
    error => console.log('rejected: ' + error)
);
```

Static Result

```
var rejestedPromise = Promise.reject(321);

rejestedPromise.then(
    value => console.log('fullfilled: ' + value),
    error => console.log('rejected: ' + error)
);
```

Promise Generator

```
var promiseRes = function(n = 0) {
    return new Promise(function(resolve, reject) {
        setTimeout(function() {
            resolve({
                resolvedAfterNSeconds: n
            });
        }, n * 1000);
    });
};


let promiseResolved = promiseRes(2);
promiseResolved.then(function(value) {
    console.log("Value when promise is resolved : ", value);
},function(reason) {
    console.log("Reason when promise is rejected : ", reason);
});
```

Promise Generator

```
var promiseRej = function(n = 0) {
    return new Promise(function(resolve, reject) {
        setTimeout(function() {
            reject({
                rejectedAfterNSeconds: n
            });
        }, n * 1000);
    });
};

let promiseRejected = promiseRej(2);
promiseRejected.then(function(value) {
    console.log("Value when promise is resolved : ", value);
},
function(reason) {
    console.log("Reason when promise is rejected : ", reason);
});
```

Many promises 1/2

```
var generatePromise = function(id) {
    return new Promise(function(resolve, reject) {
        let randomNumberOfSeconds = getRandomNumber(2, 10);
        setTimeout(function() {
            let randomiseResolving = getRandomNumber(1, 10);
            if (randomiseResolving > 5) {
                resolve({
                    ordernumber: id,
                    randomNumberOfSeconds: randomNumberOfSeconds,
                    randomiseResolving: randomiseResolving
                });
            } else {
                reject({
                    ordernumber: id,
                    randomNumberOfSeconds: randomNumberOfSeconds,
                    randomiseResolving: randomiseResolving
                });
            }
        }, randomNumberOfSeconds * 1000);
    });
};
```

Many promises 2/2

```
for (i=1; i<=10; i++) {
    let promise = generatePromise(i);

    promise.then(function(value) {
        console.log("Value when promise is resolved : ", value);
    },function(reason) {
        console.log("Reason when promise is rejected : ", reason);
    });
}
```

Chaining 1/2

```
var promiseRes = function(n = 0, info) {
    return new Promise(function(resolve, reject) {
        setTimeout(function() {
            resolve({
                resolvedAfterNSeconds: n,
                info: info
            });
        }, n * 1000);
    });
};
```

Chaining 2/2

```
let promise1 = promiseRes(2, 'Main level');
promise1.then(
    function(value){
        console.log(value);
        return promiseRes(1,'FirstLevel’);
    }
).then(
    function(value){
        console.log(value);
        return promiseRes(3,'Second Level’);
    }
).then(
    function(value){
        console.log(value);
        return promiseRes(1,'Final Level’);
    }
).then(
    function(value){
        console.log(value);
    }
)
```

All 1/4

```
var promiseRes = function(n = 0, info) {
    return new Promise(function(resolve, reject) {
        setTimeout(function() {
            resolve({
                resolvedAfterNSeconds: n,
                info:info
            });
        }, n * 1000);
    });
};

var promiseRej = function(n = 0, info) {
    return new Promise(function(resolve, reject) {
        setTimeout(function() {
            reject({
                rejectedAfterNSeconds: n,
                info:info
            });
        }, n * 1000);
    });
};
```

All 2/4

```
var promises = [];
promises.push(promiseRes(2, 'Promise 1'));
promises.push(promiseRes(1, 'Promise 2'));
promises.push(promiseRes(3, 'Promise 3'));
promises.push(promiseRes(4, 'Promise 4'));

var handleAllPromises = Promise.all(promises);
handleAllPromises.then(function(values) {
        console.log("All the promises are resolved", values);
    },
    function(reason) {
        console.log("One of the promises failed with the following reason", reason);
});
```

All 3/4

```
var promises = [];

var handleAllPromises = Promise.all(promises);
handleAllPromises.then(function(values) {
        console.log("All the promises are resolved", values);
    },
    function(reason) {
        console.log("One of the promises failed with the following reason", reason);
});
```

All 4/4

```
var promises = [];
promises.push(promiseRes(2, 'Promise 1'));
promises.push(promiseRes(1, 'Promise 2'));
promises.push(promiseRej(3, 'Promise 3'));
promises.push(promiseRej(4, 'Promise 4'));
promises.push(promiseRes(2, 'Promise 5'));

var handleAllPromises = Promise.all(promises);
handleAllPromises.then(function(values) {
        console.log("All the promises are resolved", values);
    },
    function(reason) {
        console.log("One of the promises failed with the following reason", reason);
});
```

Race 1/3

```
var promiseRes = function(n = 0, info) {
    return new Promise(function(resolve, reject) {
        setTimeout(function() {
            resolve({
                resolvedAfterNSeconds: n,
                info:info
            });
        }, n * 1000);
    });
};

var promiseRej = function(n = 0, info) {
    return new Promise(function(resolve, reject) {
        setTimeout(function() {
            reject({
                rejectedAfterNSeconds: n,
                info:info
            });
        }, n * 1000);
    });
};
```

Race 2/3

```
var promises = [];
promises.push(promiseRes(2, 'Promise 1'));
promises.push(promiseRes(1, 'Promise 2'));
promises.push(promiseRej(3, 'Promise 3'));
promises.push(promiseRej(4, 'Promise 4'));
promises.push(promiseRes(2, 'Promise 5'));

var handleRacePromises = Promise.race(promises);
handleRacePromises.then(function(values) {
        console.log("First resolve", values);
    },
    function(reason) {
        console.log("First reject", reason);
});
```

Race 3/3

```
var promises = [];
promises.push(promiseRes(2, 'Promise 1'));
promises.push(promiseRej(1, 'Promise 2'));
promises.push(promiseRes(3, 'Promise 3'));
promises.push(promiseRej(4, 'Promise 4'));
promises.push(promiseRes(2, 'Promise 5'));

var handleRacePromises = Promise.race(promises);
handleRacePromises.then(function(values) {
        console.log("First resolve", values);
    },
    function(reason) {
        console.log("First reject", reason);
});
```