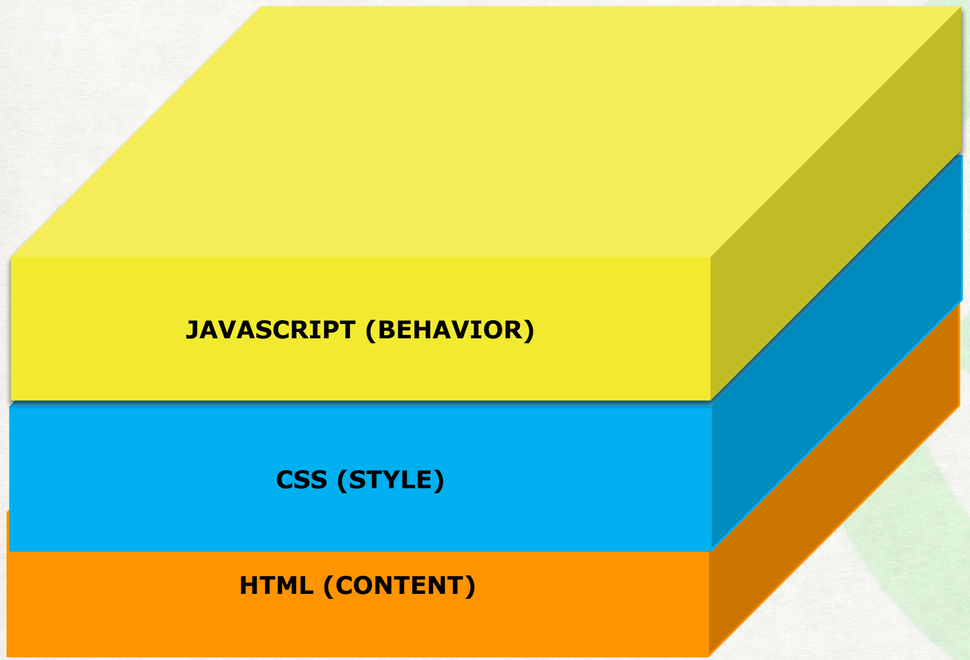
WHAT IS JAVASCRIPT?

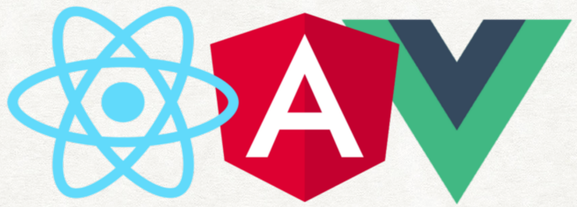


- Javascript is the language of the browser(client)
- Javascript is a scripting language.
- No need yo worry about memory management. Because JS executes directly without a compiler. JAVA is compiled languages
- One of the most popular language for developers

WHAT CAN WE DO WITH JS?

- Build web applications, mobile apps, commend line tools, games, etc.
- Built interactive user interfaces with frameworks like React
- Used in building very fast server side and full stack applications
- Used in mobile development (React native, NativeScripts, etc)
- Used in desktop application development (Electron JS).
- Makes modern web development possible

FrontEnd Development

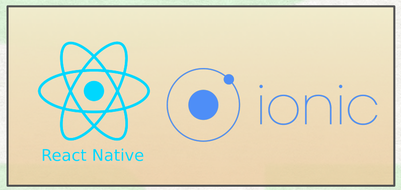


Facebook, Netflix, Dropbox, AirBnb vb.

BackEnd Development

Netflix, Uber, E-Bay vb.

Mobile Development with React Native

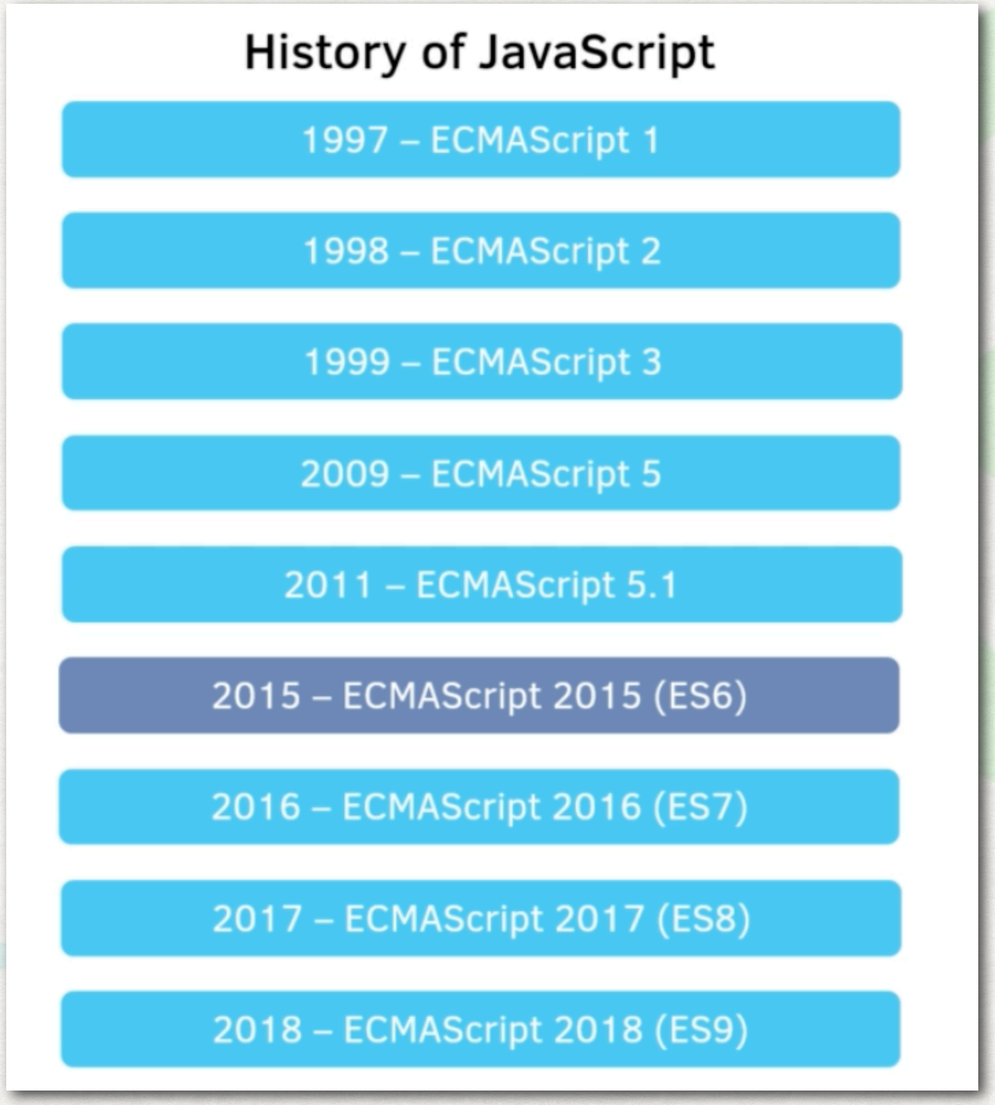


Facebook, Instagram, Skype, Uber, Pinterest

Desktop Development With electron

VSCode, Whatsapp, Slack, Skype, Twitch, Teams

HISTORY OF JS



ES6 Features:

Let and const

Templete strings

Arrow functions

Rest and Spread Operators

Destructuring

Array functions: find, findIndex,map, reduce, filter,…

BABEL: When ES6 is not working on older browsers, babel helps convert ES6 to javascript for all browsers

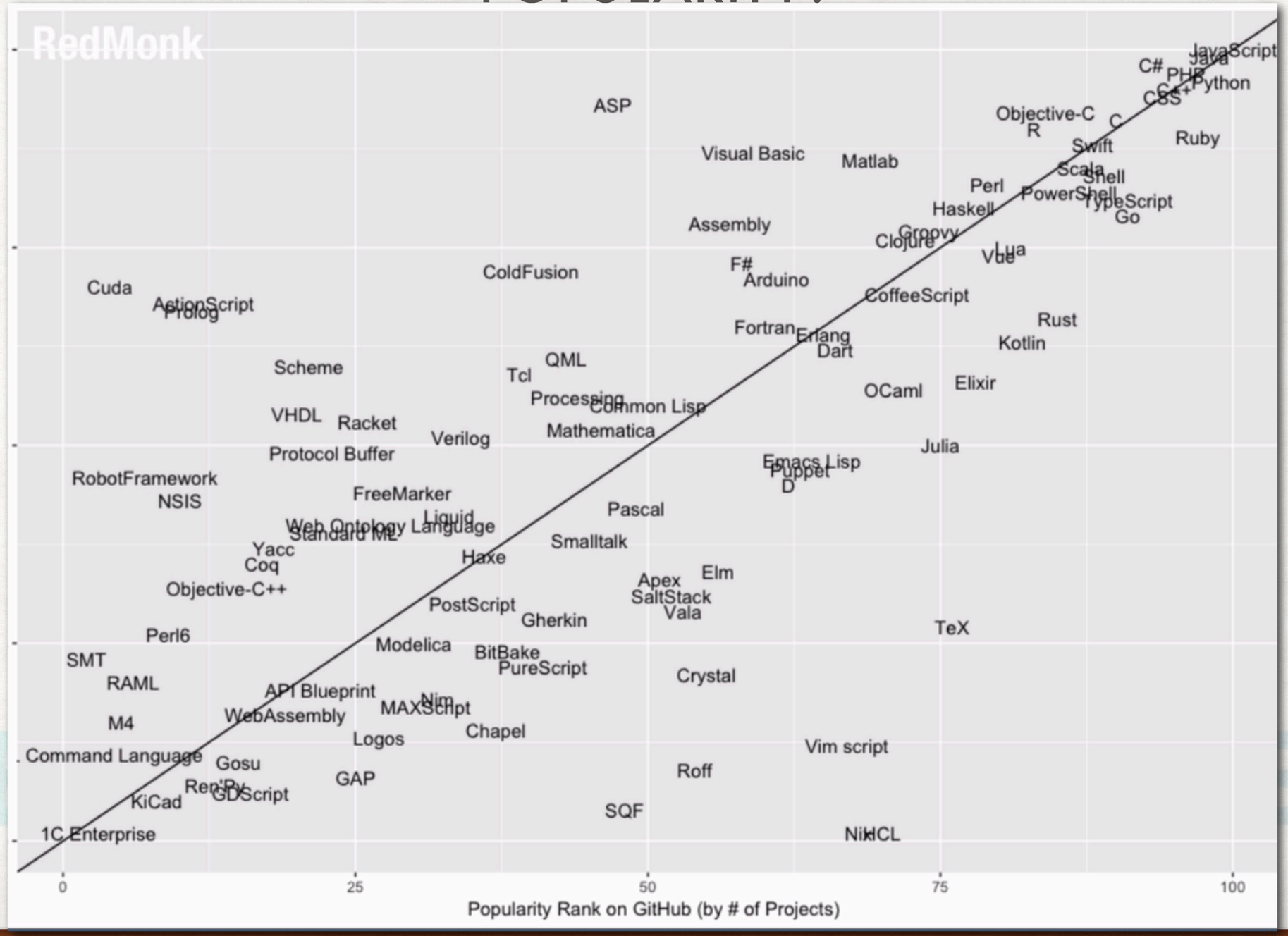
JAVASCRIPT INTRO

- Ecmascript is used to define the standards.
- Ecmascript started in 1997, but since 2015 it started releasing new version every year. It is known as ES15/ES16/et.
- Some new features are introduces such as let and const keywords with ES15

WHERE DOES JS CODE RUNS?

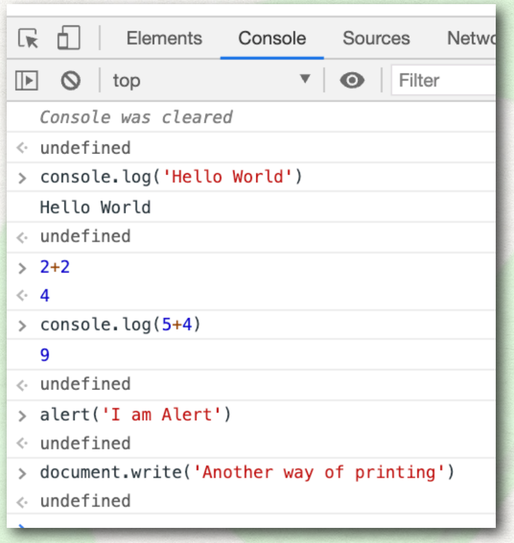
- No need a compiler or IDE to run javascript.
- It originally design to run on the browser like Firefox, chrome,..
- With Node we can run outside of browser. So we can run either on the browser or in a node. This So let us build backend development with javascript

POPULARITY?

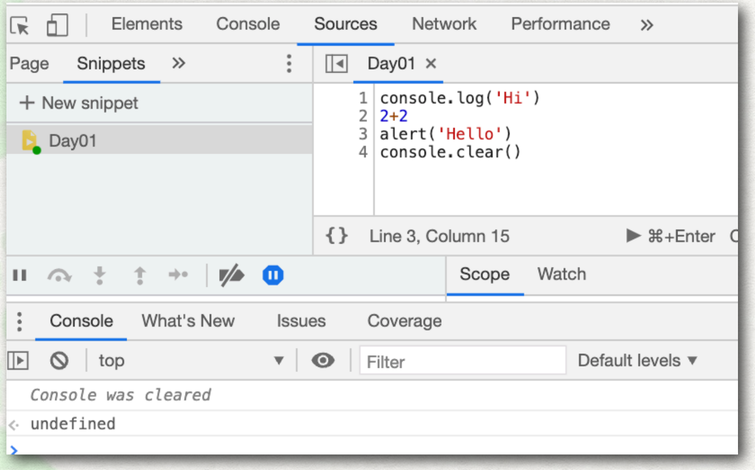

RedMonk
JavaScript
Java
C#
PHP
C++
Python
CSS
ASP
Objective-C
R
C
Ruby
Visual Basic
Matlab
Swift
Scala
Shell
Perl
PowerShell
TypeScript
Haskell
Go
Assembly
Groovy
Clojure
Lua
Vue
F#
ColdFusion
Arduino
Cuda
CoffeeScript
ActionScript
Prolog
Rust
Fortran
Erlang
Kotlin
Dart
QML
Scheme
Tcl
OCaml
Elixir
Processing
Common Lisp
VHDL
Racket
Mathematica
Verilog
Julia
Protocol Buffer
Emacs Lisp
Puppet
RobotFramework
D
FreeMarker
NSIS
Pascal
Liquid
Web Ontology Language
Standard ML
Smalltalk
Yacc
Haxe
Coq
Apex
Elm
Objective-C++
SaltStack
PostScript
Vala
Gherkin
TeX
Perl6
Modelica
BitBake
SMT
PureScript
Crystal
RAML
API Blueprint
Nim
MAXScript
WebAssembly
M4
Chapel
Logos
Vim script
Command Language
Gosu
Roff
GAP
KiCad
GDScript
SQF
1C Enterprise
0
25
50
75
100
Popularity Rank on GitHub (by # of Projects)

HOW TO RUN JAVASCRIPT CODE

1. **Browsers** => console

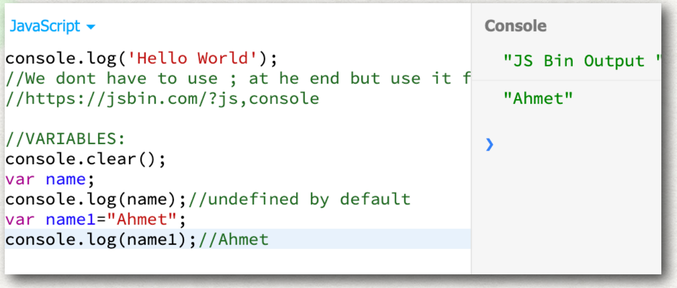


console.clear() is for clearing console

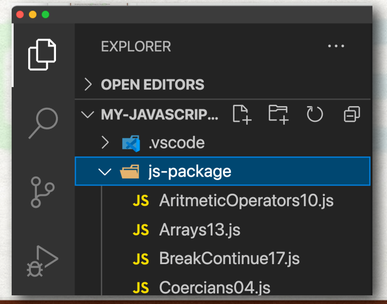


2. **We can also go Sources** >. New Slippet > create folder > write codes

3.Some websites allows run js codes => **https://jsbin.com/?js,console**



4.**ANY IDE + nodejs** => node + file name with extension is used to run the code

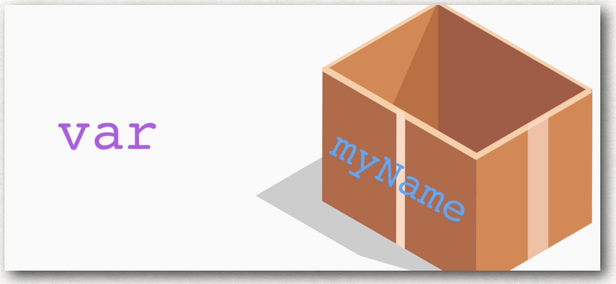


We will use NodeJS

- Create a folder: my-js-project
- Open VS Code **File** -> **Open** -> **my-js-project** ->Open

VARIABLES

```
var myName = “Ahmet”;
```



var : Keyword —> This is to declare a variable. We use this when we first declare the variable.
myName=“Bayram”; —>We are reassigning the variable. When we reassign, we don’t need to declare var again.

myName: variable name
Ahmet : Variable value

We use variable to hold the data and use it later when needed.

VARIABLES

There are other data types will learn throughout the course

Variable	Explanation	Example
String	This is a sequence of text known as a string. To signify that the value is a string, enclose it in single quote marks.	`let myVariable = 'Bob';`
Number	This is a number. Numbers don't have quotes around them.	`let myVariable = 10;`
Boolean	This is a True/False value. The words `true` and `false` are special keywords that don't need quote marks.	`let myVariable = true;`
Array	This is a structure that allows you to store multiple values in a single reference.	`let myVariable = [1,'Bob','Steve',10];` Refer to each member of the array like this: `myVariable[0]`, `myVariable[1]`, etc.
Object	This can be anything. Everything in JavaScript is an object, and can be stored in a variable. Keep this in mind as you learn.	`let myVariable = document.querySelector('h1');` All of the above examples too.

NAMING CONVENTIONS FOR JS VARIABLES

1. Give meaningful names —> firstName for first name.

2. Can’t use reserved keywords like var, let, const, alert, etc. —> var
var=“Ahmet”; Not correct

3. Can’t start with numbers —>var 1myName= “Ahmet”;

4. Can’t have space —> var my Name=“Ahmet”;

5. letters+numbers+$+_ are only allowed variables —> var a1$_=“Ahmet”;

6.Use camelCase —> var myNameAndLastName=“Ahmet Bayram”;

Note: variable names are case sensetive firstName; FirstName; are
different

DATA TYPES

String

"Hello"

Numbers

123

Boolean

true false

Primitive Data Types

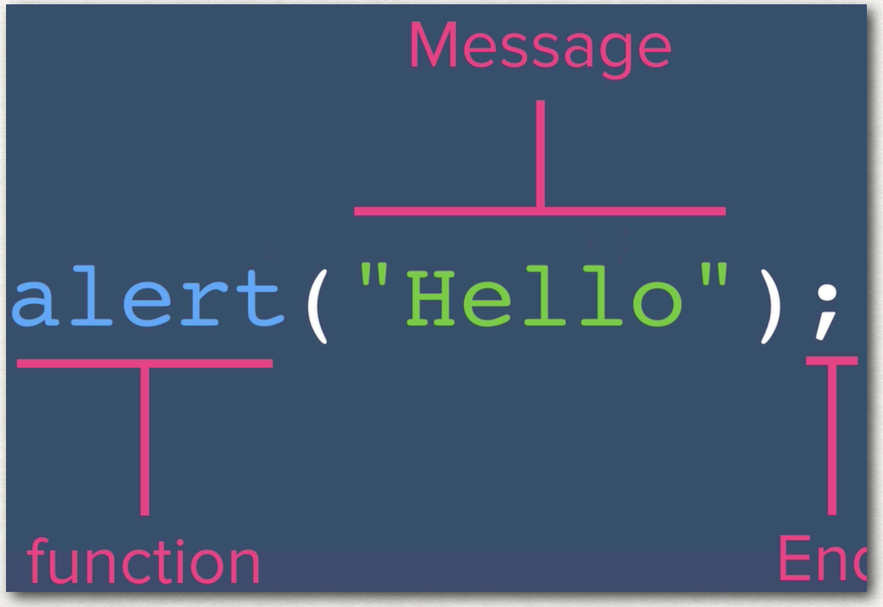
Number

String

Boolean

Undefined

Null



Data type of Hello is “String”

Use single or double quote for string

```
'Hello'
```

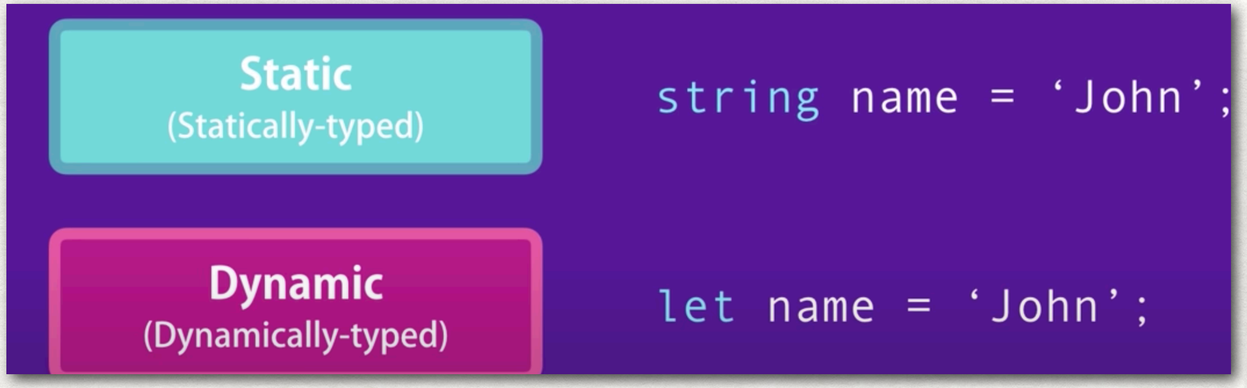
DATA TYPES

Javascript is dynamic language. It means the type of the variable can change at runtime

DYNAMIC TYPING:

Javascript is dynamic language. It means the type of the variable can change at runtime

Unlike static language, the type of the variable are determined base on the type of assign value



JavaScript is a *loosely typed* and *dynamic* language. Variables in JavaScript are not directly associated with any particular value type, and any variable can be assigned (and re-assigned) values of all types:

```
let foo = 42;    // foo is now a number
foo     = 'bar'; // foo is now a string
foo     = true;  // foo is now a boolean
```

JavaScript

```
var x = 10;
console.log(10)
console.log(typeof(x))
var y = "10";
console.log(y)
console.log(typeof(y))
var z = true;
console.log(z)
console.log(typeof(z))
```

Console

```
10
"number"
"10"
"string"
true
"boolean"
```

DEFINING VARIABLES

- **var, const, let**
- **let** ve **const, ES6 came with ES6** (Modern Javascript).

```
Type Namei = InitialValue;

const pi = 3.14;          var counter = 1;          let age = 30;
```

; is not mandatory

FUNCTIONAL VS SCOPED BLOCKS

Function scoped variable available anywhere inside the function that is declared in

Block scope function is only available only inside of the block such as if or loop

We need these terms to understand valet,const

```
> function function_name(agruments){
    var x = 10;
    let xx = 11;//defined throughout the function
    if(true){
        var y = 20; //function scoped
        let yy = 22; //yy is not defined outside
    }
    console.log(x,xx);
    //y is defined as 'var' is function scoped
    //yy is not defined as let is block scoped
    console.log(y,yy); //yy-> Reference Error
}
<- undefined
> function_name()
```

Scope of the function

Scope of the if-block

```
var a = "some value";   // functional or global scoped
let b = "some value";   // block scoped
const c = "some value"; // block scoped + cannot get new value
```

VAR

```
var price;
price = 23;
console.log(price);

price = 19.99;
console.log(“Price:" + price);

var price = "free";
console.log(“Price: " + price);
```

23

Price: 19.99

Price: free

1. **First js variable is var. It is a function scope.**

2. **Var can be reassigned to a new value**

3. **We can declare a var variable after using it. Let and const doesn’t allow this**

```
console.log(myVarVariable);//
undefined

var myVarVariable=“usa”;

console.log(myVarVariable);//usa
```

LET

```
let language = "Java";
language = "Javascript";
console.log(language);
console.log(typeof language);

language = 1;
console.log(language);
console.log(typeof language);

language = true;
console.log(language);
console.log(typeof language);

language = null;
console.log(language);
console.log(typeof language;
```

Javascript
string

1
number

true
boolean

null
object

1.Let is only defined within the block that is originally defined.Similar to const

3.We can't redeclare same let variables name using let keyword

3. We can't declare a let variable after using it not to get error.Similar to const

CONST

```
const isim = “TechPro";
console.log(isim);
console.log(typeof isim1);

const pi = 3.14;
console.log(pi);
console.log(typeof pi);

const isTrue = true;
console.log(isTrue);
console.log(typeof isTrue);
```

1. const is only defined within the block that is originally defined

2. We can’t change const variables

3. const variables must be initialized

- **Advantage:** You can’t change the value by mistake
- **Disadvantage:** You can’t use for changing values

- Use const if the value doesn’t change

Arithmetic Operators

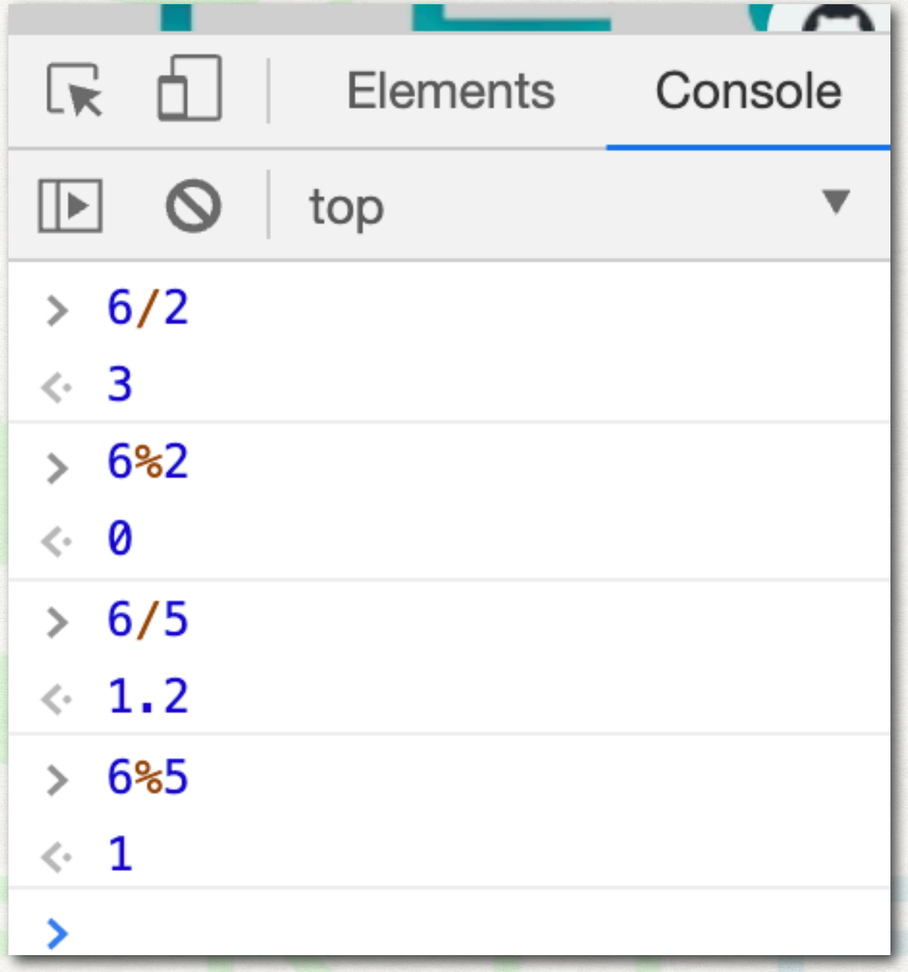
Operator	Meaning
+	Add or Concatanation
-	Subtraction
*	Multiplication
%	Mod or Remainder
++	Increment
--	Decrement
**	Exponent

BASIC ARITHMETIC AND MODULO OPERATION IN JS

Operation Symbols +,-,*,/,%,++,—

Operations follow the order of operation

So we may use parentheses to prioritize operations



+ SIGN

```
const bread = 2;
const eggs = 30;
const cheese = 40;
const total = bread + eggs + cheese;
console.log(“Spending :" + total  + "  USD");
```

Result
Spending: 72 USD

```
const name = 'John';
const lastName = 'Cash';
console.log(name + lastName);JohnCash
console.log(name + ‘ ‘ + lastName);John Cash
```

Important
+ concats the strings

```
const x = 5;
const y = "5";
const result = x + y;
console.log(result);
```

Result
55

- SIGN

```
const year = 2021;
const bDay = 1980;
const age = yıl - dogumTarihi;
console.log(“AGE :" + age);
console.log(“AGE :" + (year - bDay));
```

Result
Age : 41

Important
If we don’t use the additional parenthesis inside then js tries to concat the values and returns NaN

NaN = Not a Number

* AND **

```
const pi = 3.14;
const r = 3 ;
const area = pi*r**2;
const circumference = 2*pi*r
console.log(circumference, area);
console.log(“Circumference :" + circumference, “Area:"+area);
```

NOT: ** Exponent

(++ ,--, %) OPERATORS

```
let a = 3;
let b = ++a;
let c = --a;
console.log(a,b,c);

a += 5;
console.log(a);
```

RESULT
3 4 3

RESULT
8

```
let z = 3;
const k = z++;
```

ERROR: const value cannot be changed

```
let z = 3;
let k = z++;
```

RESULT
3.
Next time we print k will be 4

```
const num = 123;
console.log("Unit Digit: " + num%10);
```

RESULT
Unit Digit: 3

INCREMENT AND DECREMENT

x++;

x—;

x+=3;x=x+3

X-=3;x=x-3

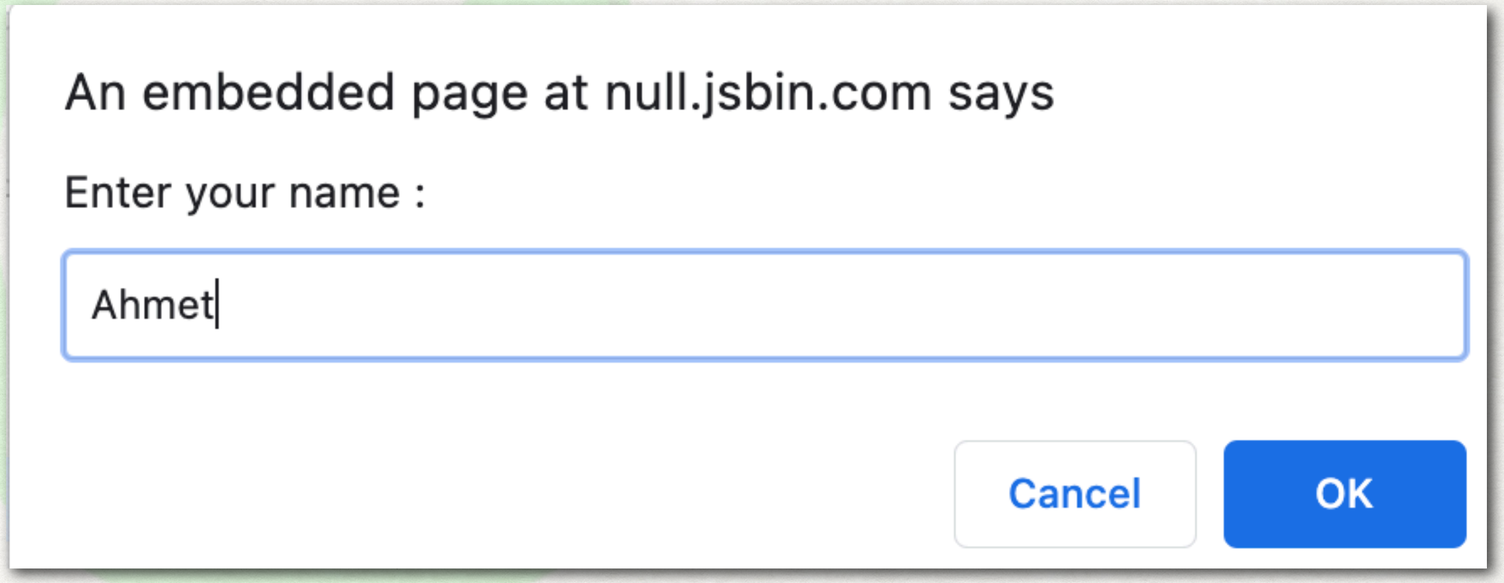
x*=3;x=3*x

x/=3;x=x/3

```
//INCREMENT AND DECREMENT
var x=4;
// x=x+1;
x++;
console.log(x)//
x+=5;
console.log(x)//10
x--;
console.log(x)//9
x*=3;
console.log(x)//25
```

PROMPT

PROMPT:
Prompt makes the browser to display a dialog to enter input text, and wait until the user either submits the text or cancels the dialog.



BASIC ARITHMETIC AND MODULO OPERATION IN JS

QUESTION:

Ask user “what is your car’s age?”

Calculate and print your own car’s age based on this:

You car age is 1 more than 2 times older than the other car

```
var carAge=prompt("how old your car")
var myCarAge=1+2*carAge
console.log("My Car is "+ myCarAge);
```

ROUNDING NUMBERS

Math.ceil(number) —>Rounds Up
Math.floor(number) —>Rounds Down
Math.round(number) —>Rounds to closest whole number
10.4->10. 10.6 -> 11

number.toFixed(numofdigit) —> Rounds to the number of digit

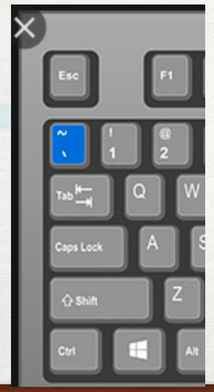
STRING CONCATENATION

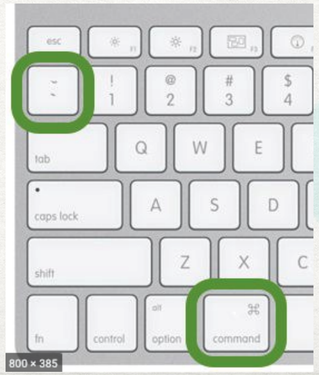
Combine values using + sign : “hello” + “world” means hello worl

```
//STRING CONCATERATION
var message="Hi!";
var firstName="Ahmet";
alert(message+" ,how are you?"+" "+firstName);
```

BACKTICK (`)

- In addition to Single (') or Double (") quotes We can use **backtick** (`)ile de **string** to create strings
  - **`string`**
  - **`string ${variable} string`**






```
console.log(`My name is ${firstName}`);

const num=15;
const result = `Spending : ${num}`;
console.log(result);
```

RESULT

Spending: $15

STRING LENGTH

let len= word.length;

```
//STRING LENGTH
var myVar="ahmet";
var varCount=myVar.length;
alert(varCount);
```

Task:

Create a prompt : “Write your tweet” —-> use prompt

Type ran

dom text and calculate the number of sent text and the remaining of sent text

Note Max character is 280.

INDEXOF

The `indexOf` method will starts index of 0.
It always return `-1` if there was *no match*.
IndexOf always returns a number.

```
const location =  'Regent Street, London, England';
```

```
if (existsInString) {
  console.log(`Yes, "London" exists in "${location}"`);
}
```

SLICING AND SUBSTRING

```
bread.slice(x,y);
```

x starting index(inclusive), y ending index(exclusive)

Index start at 0

What is the difference between slice and substring.

```
//SLICE and SUNBSTRING:
var name="javascript";
name.slice(0,1);//0 starting inclutive number,
name.slice(3);//start from 3 to the ending
//TASK: user gets alert if there are more that
alert(prompt("Enter Tweet: ").slice(0,240));
name.substring(0,1);
```

They are almost same, but there are few differences like substring swaps when star>stop. Slice returns empty strings

https://stackoverflow.com/questions/2243824/what-is-the-difference-between-string-slice-and-string-substring

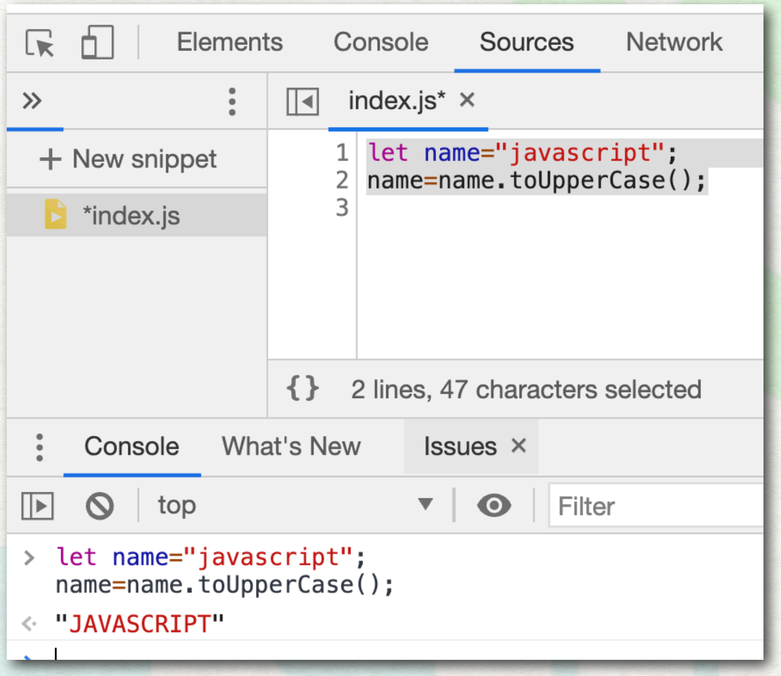
TO UPPER LOWER CASE /LOWER CASE

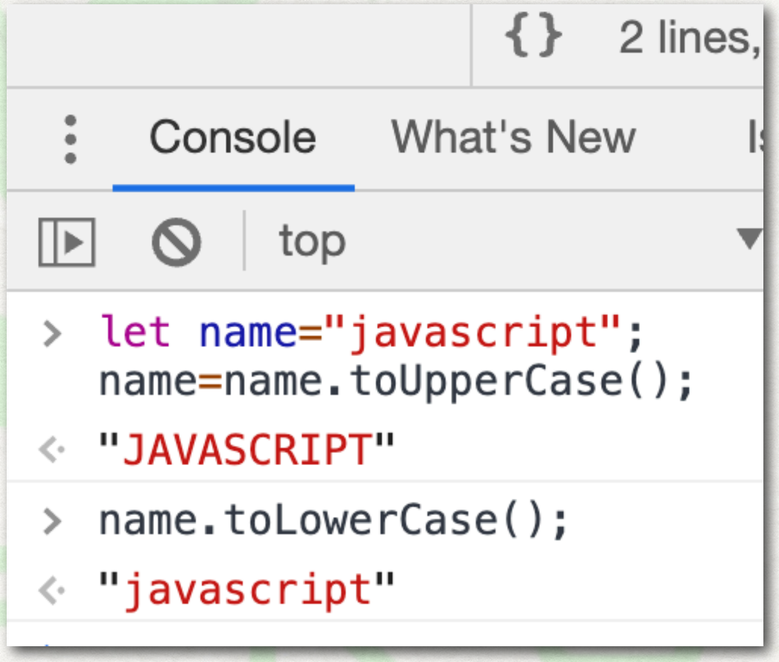
word.toUpperCase()—>Converts the values to upper case
‘abc’.toUpperCase() —> ABC

word.toLowerCase()—>Converts the values to upper case
‘ABC’.toLowerCase() —> abc

TO UPPER LOWER CASE /LOWER CASE

```
//TOUPPERCASE/ TO LOWER CASE
var name ="javascript";
name=name.toUpperCase();//We don't have to put in a variable
name=name.toLowerCase();//We don't have to put in a variable
```





PRACTICE

Ask user to enter a name: and print “Hello, Name”

Make sure only the first initial is always capital and the rest always small

Eg:

var name=ahmet; =>>>>>Ahmet

var name=aHmet; =>>>>Ahmet

var name=AHMET =>>>>Ahmet

PRACTICE QUESTIONS

1. Get one side of the square from the user and print the area of the square on the console

2. Get both sides of the rectangle form the user and print the perimeter of the rectangle on the console

3. Get the radius of the circle from user and print the area of the circle on the console

4. Get both sides of the triangle and print the hypothenuse of the triangle. Round your answer to the nearist whole number.

5.Create a prompt : “Write your tweet” —-> use prompt. Type random text and calculate the number of sent character and the remaining character.Note Max character is 280.

COMPARISON

Operator	Meaning
==	Compares only VALUES
===	Compares VALUES and DATA Types. If both true then return true, otherwise returns false
!=	Not Equal Sign. Compares only values
>	Greater Than
<	Less Than
>=	Greater than or Equal to
<=	Less than or Equal to

```
const s1 = 5;

console.log(s1 == 5);          // true
console.log(s1 == "5");        // true
console.log(s1 === "5");       // false
Typeof s1=number . Type of “5” string
console.log(s1 !== 5);         // false
console.log(s1 != “5");        // false
console.log(s1 !== "5");       // true
```

```
console.log(s1 > 5);           // false
console.log(s1 > "4");         // true

console.log(s1 >= 5);          // true
console.log(s1 > "6");         // false
```

NOTE:
=== and !== check values and data types

NOTE:
JS converts string values to number value when we compare them

LOGICAL OPERATORS

Operator	Meaning
&&	And operator returns true if all TRUE
||	Or operator returns true if one or more is TRUE
!	Not operator returns the opposite of value. If value is true returns FALSE, if value is false returns TRUE

```
let s2 = true;
let s3 = true;
console.log(s2 && true);          // true
console.log(s2 && s3);            // true
console.log(s2 && s3 && false);   // false

s3 = false;
console.log(s2 || s3 || false);   // true

s3 = null;
console.log(s2 && s3);            // null
console.log(s2 || s3);            // true

console.log(!s2);                 // false
console.log(!s3);                 // true

s3 = null;
console.log(!s3);                 // true
```

COMPARATORS/CONDITIONALS: IF ELSE STATEMENT

```
if(light===green){
    goStraight();
    }else{
    stop();
    }
```

=== is equal to
!== is not equal to
> is greater than
< is less than
>= is greater or equal to
<= is less than or equal to

=== is equal to
== is equal to

Difference:
=== checks value and data type (1!==“1”)
== checked only the value (1 == “1”)

```
var a = 1;
var b = "1";
```

```
if(a===b){
    console.log(true);
}else{
    console.log(false);
}
```

```
false
```

```
if(a==b){
    console.log(true);
}else{
    console.log(false);
}
```

```
true
```

COMPARATORS/CONDITIONALS: IF ELSE STATEMENT

IMPORTANT NOTE:
0, **FALSE**, **null**, **undefined**, **""** , **NaN** considered as **FALSE.**

All other values are **TRUE**

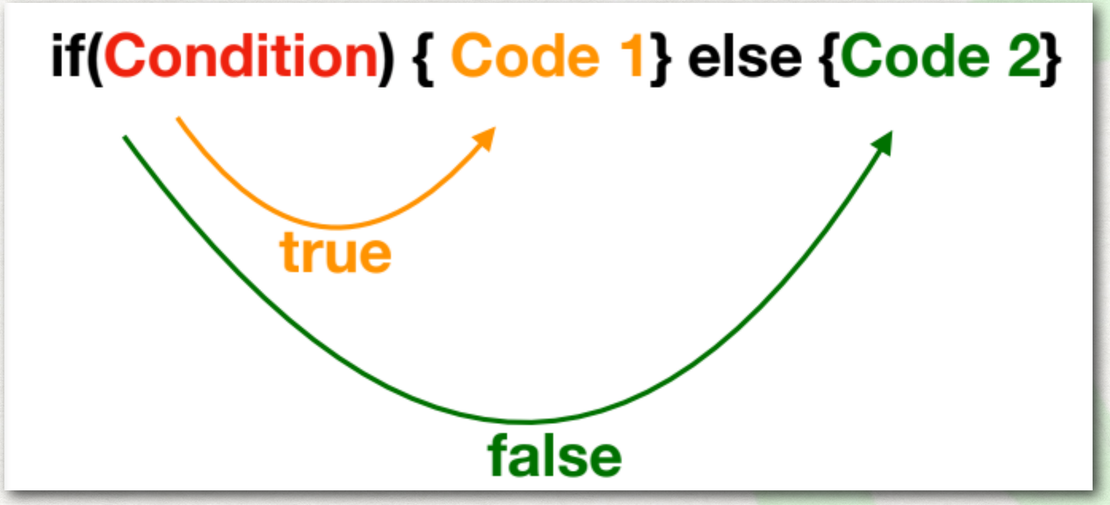
```
if(0){
  console.log('True');
}else{
   console.log('False');//"False"
}


if(null){
   console.log('True');
}else{
  console.log('False');//"False"
}
```

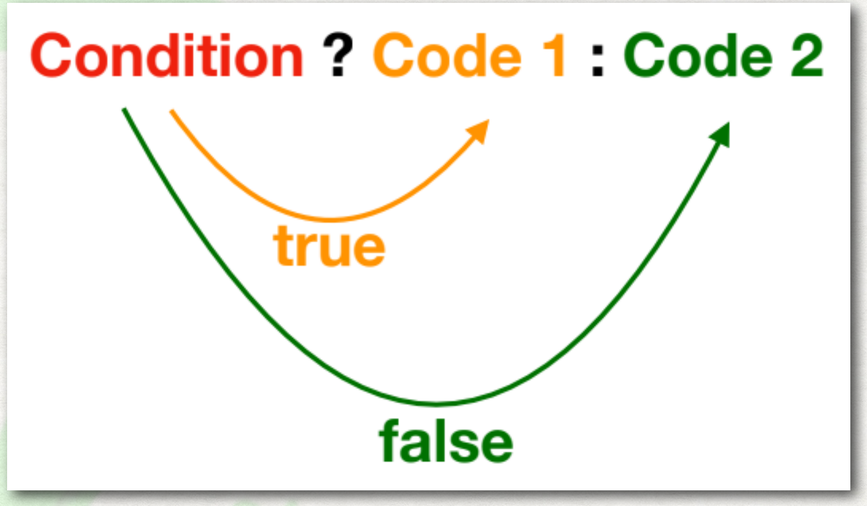
```
if(4){
  console.log('True');//"True"
}else{
  console.log('False')

}
if('apple'){
  console.log('True');//"True"
}else{
  console.log('False')
}
```

SWITCH STATEMENT

```
var color='blue';
switch(color){
    case 'red':
        console.log('color is red');
        break; //===>>Exit switch statement
    case 'blue':
        console.log('color is blue');
        break;
    default: //===> If there is no match then this line runs
        console.log('color is not red or blue');
        break;
}
```

TERNARY OPERATOR






```
var x = 10;
var z=12;
//Ternary
var y = (x > 5) ? (2*x) : ('x is not greater than or equal 5');
console.log(y);//20
//Nested Ternary
var result = (x > 5) ? ((z > 5) ? ('print true') : ('print false'))
                     : ('x is not greater than or equal 5');
console.log(result);//print true
```

True- 1st block

True

False

False - 2nd block

If x>5 get in the first block, otherwise get in the second block.

If z>5 then b"print true" otherwise "print false"

RANDOM NUMBER GENERATOR IN JS

```
var num = Math.random();

0-0.9999999999999999
```

16 Decimal Places

Randomization and control flow is important.

To Scale numbers we use Math.random() and then scale the number.

For exampleWe can get any number between 0(inclusive) -1(exclusive)

To get a random number between 1(inclusive)-10(exclusive), do followings:

```
var num = Math.random();
0.34235452352353745

num=num*10;              => 3.4235452352353745
num=Math.floor(num);  = 3
```

```
//Creating random number
var num=Math.random();
//Logging the number
console.log(num);//nums 0-1
```

```
//Creating random number
var num=Math.random();
//Get nums from 0-5
num*=6;
num=Math.floor(num);
//Logging the number
console.log(num);//nums 0-1
```

```
//Creating random number
var num=Math.random();
//Get nums from 1-6
num*=6;
num=Math.floor(num)+1;//add 1 to scale up
//Logging the number
console.log(num);
```

LOVE CALCULATOR

Get 2 names from user and store variables:**name1, name2.**

Create a variable **loveScore** and store the random number.

loveScore should be a random whole number between 1-100(inclusive) percent. Calculate the chance of a successful relationship. The closer to 100% the stronger

Task 1: Print '**name1** and **name2** 's love score is **loveScore'**

//Try using if and ternary

Task 2: Print 'You love each other' if loveScore is greater or equal to 80 %.

Print 'Not a strong relationship' otherwise

Task 3: Print 'You love each other' if loveScore is greater or equal to 80 %.

Print 'Modorate relationship' if loveScore is between is in between 50% and 80%

Print 'Not a strong relationship' otherwise

https://www.lovecalculator.com/

LEAP YEAR PRACTICE

Write a program to determine if the given year is a leap year.

Write a function : isLeap, and a parameter:year

-> divisible 4

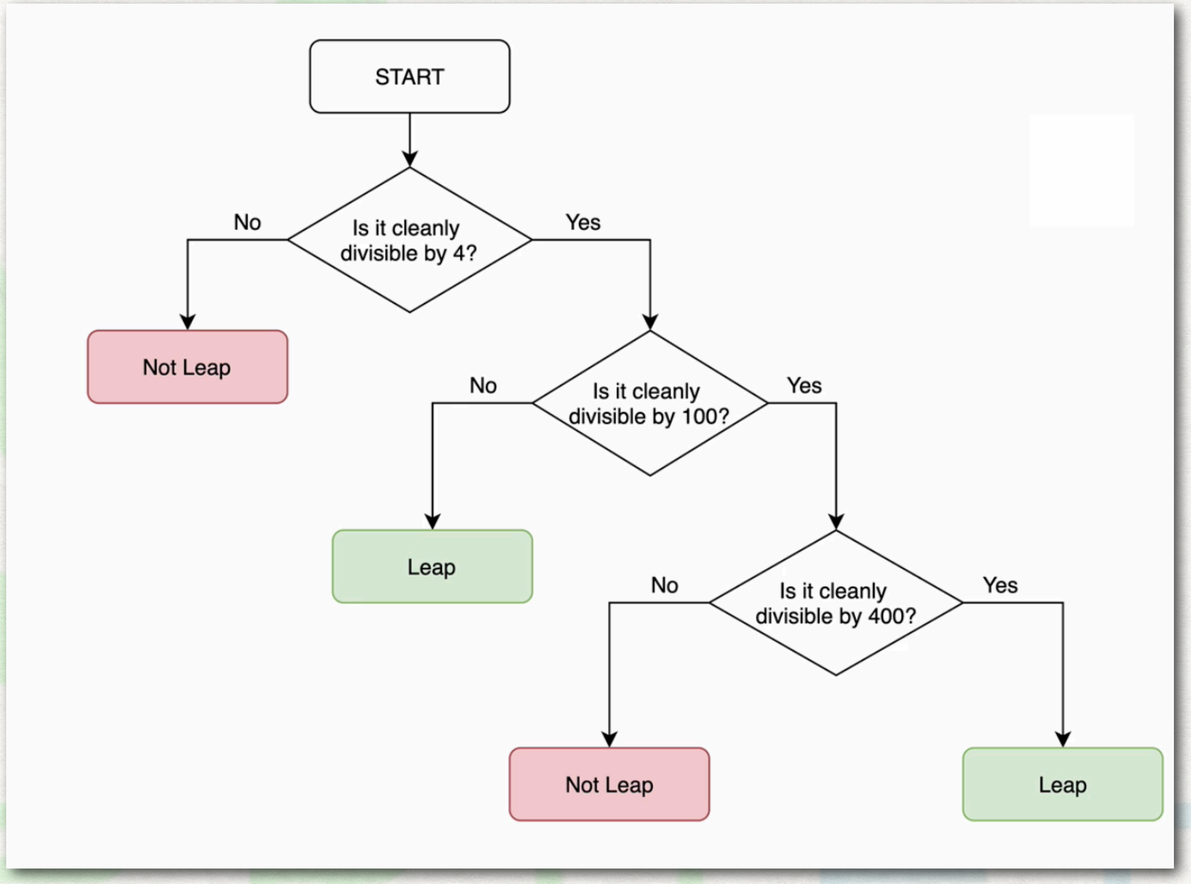
-> but not divisible by 100

-> unless divisible by 400

Here is the flow chart

Here is the flow chart

https://app.diagrams.net/



FUNCTIONS1 : CREATING AND CALLING

A function is a set of statement that performs a task or calculates a value

```
function greet(){//DECLARING THE FUNCTION

   //BODY OF THE FUNCTION

console.log('Hello World');

 }

greet();//calling the function
```

FUNCTIONS 2: PARAMETERS AND ARGUMENTS

Functions can have input. That inputs can change the behavior of the function

We use the input like the variable, we use it to calculate or do something inside

```
function greetPeople(name){//name is a parameter. Name IS NOT ACCESSABLE OUTSIDE OF THIS FUNCTION
   console.log('Hello '+name);
}
greet(“Sam"); —>> Sam is an ARGUMENT. It is not PARAMETER
```

```
function personInfo(firstName,lastName,dateOfBirth){
    console.log('Name: '+firstName+'\nLast Name: '+lastName+'\nDOB: '+dateOfBirth );
}
personInfo(‘John’,’Walker’,1990);
```

FUNCTIONS 3: OUTPUTS AND RETURN VALUES

```
//Create a function:totalCost
//Add two param:costPerItem,itemPrice
//Calculate and return price
function totalCost(costPerItem,itemPrice){
   return costPerItem * itemPrice;//Remainder
}
// Print the values of totalCost function for 4,5
// I can hold the value of the fuction in a variable cause of return keyword
// and use it in another function or in another calculation
console.log(totalCost(4,5));//20
var cost = totalCost(4,5);
console.log('Total Cost : '+cost); //20
//Creating an other function
//I can use previous function values
function remainingMoney(initialAmount){
    console.log(initialAmount-cost);//I can use the variable
    console.log(initialAmount-totalCost(6,8))//I can use the function within a function
}
remainingMoney(80);//32
console.log('REMAINING : '+remainingMoney(80));//undefined

//Creating an other function
//This function have return type
//I can use log to print
function remainingMoney1(initialAmount){
   var dif= initialAmount-cost;//I can use the variable
   return dif;
}
//Printing the function
console.log('REMAINING1 : '+remainingMoney1(80));//60
```

```
//Create a function:totalCost
//Add two param:costPerItem,itemPrice
//Calculate and return price
function totalCost(costPerItem,itemPrice){
    return costPerItem * itemPrice;//Remainder
}

// Print the values of totalCost function for 4,5
// I can hold the value of the fuction in a variable cause of return keyword
// and use it in another function or in another calculation
console.log(totalCost(4,5));//20
var cost = totalCost(4,5);
console.log('Total Cost : '+cost); //20
```

```
//Creating an other function
//I can use previous function values
function remainingMoney(initialAmount){
     console.log(initialAmount-cost);//I can use the variable
     console.log(initialAmount-totalCost(6,8))//I can use the function within a function
}

remainingMoney(80);//32
console.log('REMAINING : '+remainingMoney(80));//undefined
```

```
//Creating an other function
//This function have return type
//I can use log to print
function remainingMoney1(initialAmount){
     var dif= initialAmount-cost;//I can use the variable
     return dif;
}
//Printing the function
console.log('REMAINING1 : '+remainingMoney1(80));//60
```

PRACTICE QUESTION

Write function:bmiIndex

That takes two inputs: height,weight.

And returns BMI Index to the closest whole number.

BMI=weight/(height*height)

_____

Create a function: customerInfo

That takes two parameters: name,cardNum

Returns the info as the following: customerfirstletter capital and visible, rest is not visible (U**).replaceAll()

Customer cart number first 12 digit is not visible and last 4 visible only(**** **** **** 0000)

```
function bmiCalculator(weight, height){
//      var bmi =weight/(height*height);
    var bmi =weight/(Math.pow(height,2));
    return bmi;

}

var bmi=bmiCalculator(75,1.5);
console.log(bmi);
console.log(Math.round(bmi));//round nearest whole
console.log(Math.floor(bmi));//round down
console.log(bmi.toFixed(2));//2 digit after decimal point
```

```
function info(name,cardNum){
    var firstLetter=name.slice(0,1).toUpperCase();
    var restOfName=name.slice(1).replaceAll(/[a-z]/gi,"*");
    var specialChars="\n**** **** **** ";
    var last4Digit=cardNum.slice(12,16);
    return firstLetter+restOfName+specialChars+last4Digit;

}
console.log(info("asdgasasg","3452346234576345"));
```

ARRAYS

```
[]  => used to create array
var prices=[6,10,14,0,7];
```

So far we have seen we can store single data as string, number, or boolean

How can we store collection of related items into the same container or the same variable:ARRAY

Arrays are the collection of the items that are related

We can use array when we store similar items in a single variable

```
var prices=[6,10,14,0,7];
var myPrice=prices[1]; ==> 10
prices.lenght;  ===>5
prices.includes(10);===> true
```

ARRAYS

Crate an car inventory: carInventory

Add car brands in the inventory : toyota, honda, bmw, tesla, jaguar, toros

Prompt user : Which car you are looking for?

Respond true if car exist, respond false otherwise.

```
var carInventory=["toyota", "honda", "bmw", "tesla", "jaguar", "toros"];
carInventory.includes(carName);//This returns true or false based on the carName
```

```
var carInventory=["toyota", "honda", "bmw", "tesla", "jaguar", "toros"];
var carName=prompt("Which car do you like?").toLowerCase();
// carInventory.includes(carName);//This returns true or false based on the carName
var isExist=carInventory.includes(carName);
if(isExist){
    alert("We have in our inventory.");
}else{
    alert("Sorry!Check again later.");
}
```

FIZZ BUZZ PROBLEM

```
var arr =[];
arr.push(1);//adding item in the array
arr.push(5);//adding another item in the array
push(); is used to add item AT THE END OF THE ARRAY
.............................
arr.pop(); //REMOVE THE LAST ITEM FROM THE ARRAY
```

This problem is based on a game in England

Multiple of 3: “Fizz”. |||||| Multiple of 5: “Buzz”. |||||||| Multiple of 3&&5(multiple of 15): “FizzBuzz”

CALL THE FUNCTION 15 times

[1,2,”Fizz”,4,”Buzz”,””Fizz”,7,8,”Fizz”,”Buzz”,11,”Fizz”,13,14,”FizzBuzz”]

Write a program that prints the numbers from 1 to 100. But for multiples of three print “Fizz” instead of the number and for the multiples of five print “Buzz”. For numbers which are multiples of both three and five print “FizzBuzz”.

SOLUTION

```
var arr=[];
var count = 1;
function fizzBuzz(){
if(count%3==0&&count%5===0){
    arr.push("FizzBuzz");
}else if(count%3===0){
    arr.push("Fizz")
}else if(count%5===0){
    arr.push("Buzz")
}else{
   arr.push(count);
}
count++;//Increasing the number
console.log(arr);
}
fizzBuzz();//1
fizzBuzz();//1,2
fizzBuzz();//1,2,Fizz
fizzBuzz();//1,2,Fizz,4
fizzBuzz();//1,2,Fizz,4,Buzz
fizzBuzz();//1
fizzBuzz();//1,2
fizzBuzz();//1,2,Fizz
fizzBuzz();//1,2,Fizz,4
fizzBuzz();//1,2,Fizz,4,Buzz
fizzBuzz();//1
fizzBuzz();//1,2
fizzBuzz();//1,2,Fizz
fizzBuzz();//1,2,Fizz,4
fizzBuzz();//1,2,Fizz,4,Buzz
```

```
var arr=[];
var count = 1;

//everythime we call the function,
//we want to increase add the next number in the array
function fizzBuzz(){
arr.push(count);
console.log(arr);
}

fizzBuzz();//1
fizzBuzz();//1,1,1
fizzBuzz();//1,1,1
```

```
var arr=[];
var count = 1;

//everythime we call the function,
//we want to increase add the next numbe
function fizzBuzz(){
arr.push(count);
count++;//Increasing the number
console.log(arr);
}

fizzBuzz();//1
fizzBuzz();//1,2
fizzBuzz();//1,2,3
```

```
var arr=[];
var count = 1;

//everythime we call the function,
//we want to increase add the next number in the array
function fizzBuzz(){
//Instead of 3 put Fizz:
//Check if count divisible by 3
if(count%3===0){
    arr.push("Fizz")
}else{
   arr.push(count);
}
//instead of 5 put Buzz
//Instead of 15 put FizzBuzz
count++;//Increasing the number
console.log(arr);
}

fizzBuzz();//1
fizzBuzz();//1,2
fizzBuzz();//1,2,Fizz
```

```
var arr=[];
var count = 1;

//everythime we call the functio
//we want to increase add the nex
function fizzBuzz(){
//Instead of 3 put Fizz:
//Check if count divisible by 3
if(count%3===0){
    arr.push("Fizz")
}else if(count%5===0){
//instead of 5 put Buzz
//Instead of 15 put FizzBuzz
    arr.push("Buzz")
}else{
   arr.push(count);
}
//instead of 5 put Buzz
//Instead of 15 put FizzBuzz
count++;//Increasing the number
console.log(arr);
}

fizzBuzz();//1
fizzBuzz();//1,2
fizzBuzz();//1,2,Fizz
fizzBuzz();//1,2,Fizz,4
fizzBuzz();//1,2,Fizz,4,Buzz
```

```
var arr=[];
var count = 1;
function fizzBuzz(){
//instead of 5 put Buzz
//Instead of 15 put FizzBuzz
if(count%3==0&&count%5===0){
    output.push("FizzBuzz");
}else if(count%3===0){
//Instead of 3 put Fizz:
//Check if count divisible by 3
    arr.push("Fizz")
}else if(count%5===0){
//instead of 5 put Buzz
//Instead of 15 put FizzBuzz
    arr.push("Buzz")
}else{
   arr.push(count);
}

count++;//Increasing the number
console.log(arr);
}
```

```
fizzBuzz();//
fizzBuzz();//,2
fizzBuzz();//,2,Fizz
fizzBuzz();//,2,Fizz,4
fizzBuzz();//,2,Fizz,4,Buzz
fizzBuzz();//
fizzBuzz();//
fizzBuzz();//
fizzBuzz();//
fizzBuzz();//
fizzBuzz();//
fizzBuzz();//
fizzBuzz();//
fizzBuzz();//
fizzBuzz();//
```

```
(15) [1, 2, "Fizz", 4, "Buzz", "Fizz", 7, 8, "Fizz", "Buzz", 11, "Fizz", 13, 14, "FizzBuzz"]
```

WHO PAYS THE BILL?

Write a function whoPays(){}

This functions selects a random person from names array and returns that person

Make sure your code for works different array sizes.

Eg:

var names =[“Sam”,”Tom”,Cindy”,”Cole”,”Jim”];

Use Math.random() to get a random number

Use that number as index of array to get random name

Tom pays the bill

Cindy pays the bill

WHO PAYS THE BILL? SOLUTION

Write a function whoPays(names){}

This functions selects a random person from names array and returns that person

Make sure your code for works different array sizes.

Eg:

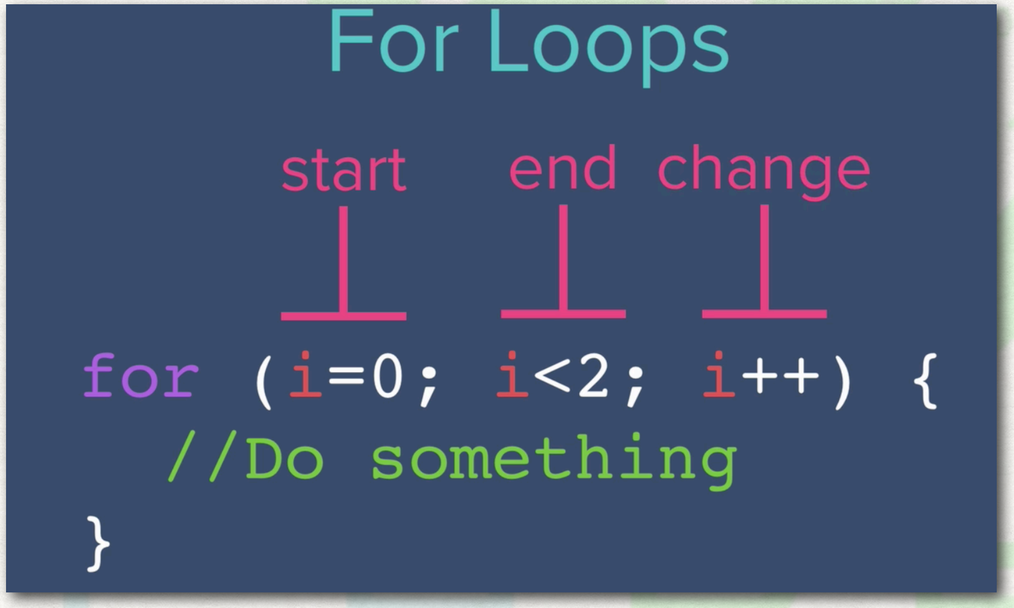
var names =[“Sam”,”Tom”,Cindy”,”Cole”,”Jim”];

Tom pays the bill

```
var names=["Sam","Tom","Cindy","Cole","Jim"];

function whoPays(){
    var numOfPeople=names.length;
    var randomnumber=Math.floor(Math.random()*numOfPeople);
    var randomName=names[randomnumber];
    return randomName+ " pays the bill"
}

console.log(whoPays());
```

FOR LOOP

```
for(var i=1;i<4;i++) {  ——>>>Start, Condition, Change is in here
console.log(i);  —>>>This code runs 3 times(while i=1,2,3)
}
```



FIZZ BUZZ WITH FOR LOOP

```
var arr=[];
var count = 1;
function fizzBuzz(){
if(count%3==0&&count%5===0){
    arr.push("FizzBuzz");
}else if(count%3===0){
    arr.push("Fizz")
}else if(count%5===0){
    arr.push("Buzz")
}else{
   arr.push(count);
}
count++;//Increasing the number
console.log(arr);
}
fizzBuzz();//1
fizzBuzz();//1,2
fizzBuzz();//1,2,Fizz
fizzBuzz();//1,2,Fizz,4
fizzBuzz();//1,2,Fizz,4,Buzz
fizzBuzz();//1
fizzBuzz();//1,2
fizzBuzz();//1,2,Fizz
fizzBuzz();//1,2,Fizz,4
fizzBuzz();//1,2,Fizz,4,Buzz
fizzBuzz();//1
fizzBuzz();//1,2
fizzBuzz();//1,2,Fizz
fizzBuzz();//1,2,Fizz,4
fizzBuzz();//1,2,Fizz,4,Buzz
```

```
var arr=[];

function fizzBuzz(){
    //LOOP STARTS
    for(var count = 1;count<=15;count++){
if(count%3==0&&count%5===0){
    arr.push("FizzBuzz");
}else if(count%3===0){
    arr.push("Fizz")
}else if(count%5===0){
    arr.push("Buzz")
}else{
   arr.push(count);
}
//LOG HERE TO LOG IN EACH STEP
console.log(arr);
}
//LOG HERE WIF YOU WANT TO LOG HERE
}

fizzBuzz();//Instead of writing the function 10 time, use a loop
```

WHILE LOOP

```
var i=1;
while(i<4) {  ———>>>Run the code in the body while I is less than 4
console.log(i);  —>>>This code runs 3 times(while i=1,2,3)
i++;   —->>>MAKE SURE TO ADD PROPER INCREMENT/DECREMENT TO AVOID INFINITE LOOP
}
```

A programmer's wife tells him: "while you're at the store, get some milk".

He never comes back.

MODIFY FIZZ BUZZ PROBLEM USING WHILE LOOP

```
var arr=[];
var count = 1;
function fizzBuzz(){
if(count%3==0&&count%5===0){
    arr.push("FizzBuzz");
}else if(count%3===0){
    arr.push("Fizz")
}else if(count%5===0){
    arr.push("Buzz")
}else{
   arr.push(count);
}
count++;//Increasing the number
console.log(arr);
}
fizzBuzz();//1
fizzBuzz();//1,2
fizzBuzz();//1,2,Fizz
fizzBuzz();//1,2,Fizz,4
fizzBuzz();//1,2,Fizz,4,Buzz
fizzBuzz();//1
fizzBuzz();//1,2
fizzBuzz();//1,2,Fizz
fizzBuzz();//1,2,Fizz,4
fizzBuzz();//1,2,Fizz,4,Buzz
fizzBuzz();//1
fizzBuzz();//1,2
fizzBuzz();//1,2,Fizz
fizzBuzz();//1,2,Fizz,4
fizzBuzz();//1,2,Fizz,4,Buzz
```

```
var arr=[];
var count = 1;

function fizzBuzz(){
    //LOOP STARTS
    while(count<=15){
if(count%3==0&&count%5===0){
    arr.push("FizzBuzz");
}else if(count%3===0){
    arr.push("Fizz")
}else if(count%5===0){
    arr.push("Buzz")
}else{
   arr.push(count);
}
count++;//Increasing the number
//LOG HERE TO LOG IN EACH STEP
console.log(arr);
}
//LOG HERE WIF YOU WANT TO LOG HERE
}

fizzBuzz();//Instead of writing the function 10
```

```
var arr=[];
var count = 1;

function fizzBuzz(){
    //LOOP STARTS
    while(count<=15){
if(count%3==0&&count%5===0){
    arr.push("FizzBuzz");
}else if(count%3===0){
    arr.push("Fizz")
}else if(count%5===0){
    arr.push("Buzz")
}else{
   arr.push(count);
}
count++;//Increasing the number
//LOG HERE TO LOG IN EACH STEP
console.log(arr);
}
//LOG HERE WIF YOU WANT TO LOG HERE
}

fizzBuzz();//Instead of writing the function 10 time, use a loop
```

DO WHILE LOOP

```
var i=1
do{  ———>>>Get in the body first
console.log(i);  —>>>This code runs 3 times(while i=1,2,3)
i++;   —->>>MAKE SURE TO ADD PROPER INCREMENT/DECREMENT TO AVOID INFINITE LOOP
}while(i<4);——>>Condition is at the end.
```

WHICH ONE TO USE

```
while (something is true) {
    //Do something
}

for (i=0; i<2; i++) {
  //Do something
}
```

=>STATE. RUN CODE WHILE PROGRAM IN CERTAIN STATE

=>ITERATE. RUN CODE MANY TIMES UNTIL CONDITION MET.

FIBONACCI GENERATOR

0,1,1,2,3,5,8,13,21,...

Write a code to the term number and print the array of fibonacci numbers

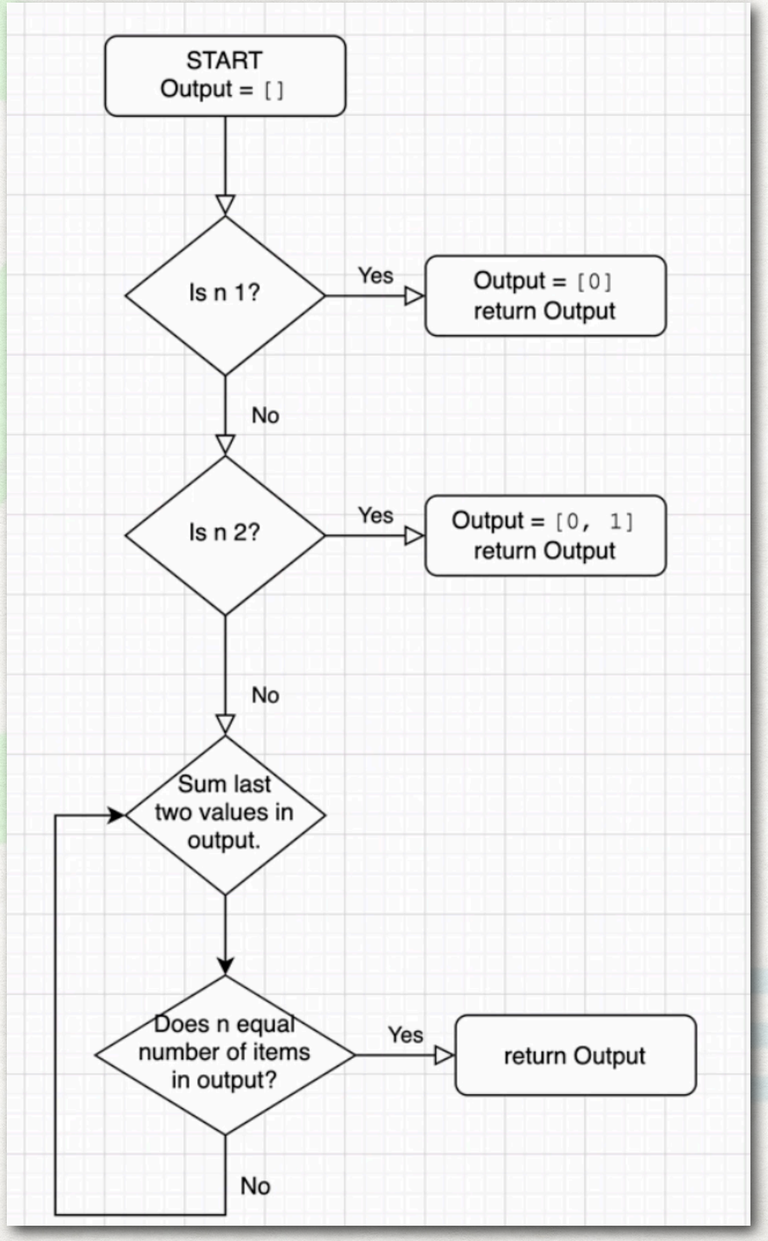
Eg:

fibNumbers(3); => [0,1,1]

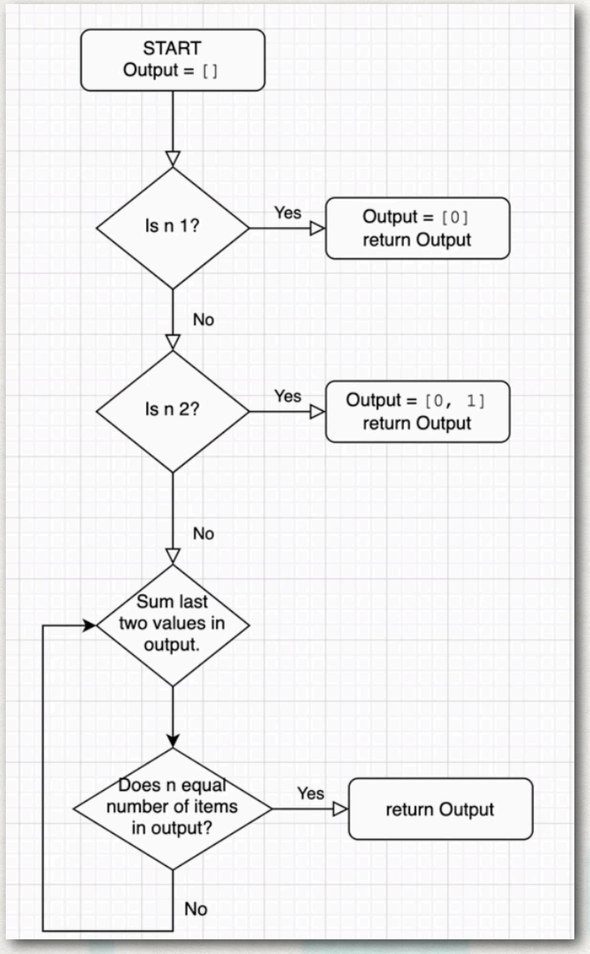
fibNumbers(6); => [0,1,1,2,3,5]

FLOWCHART CAN BE CREATED AT

https://app.diagrams.net/



FIBONACCI GENERATOR SOLUTION




```
function fibonacciGenerator (n) {

    var output = [];
    if (n === 1) {
        output = [0];
    }
    else if (n === 2) {
        output = [0, 1];
    }

    return output;
}
```

```
function fibonacciGenerator (n) {

    var output = [];
    if (n === 1) {
        output = [0];
    }
    else if (n === 2) {
        output = [0, 1];
    }
    else {
        output = [0, 1];
        output.push(output[0] + output[1]); //1
    }

    return output;
}
```

Day_3.js × index.html

```
1  function fibNumbers(n){
2      var output=[];
3      if(n===1){
4          output=[0];
5      }else if(n===2){
6          output=[0,1];
7      }else{
8          output=[0,1];
9          for(var i=2;i<n;i++){
10         output.push(output[output.length-2]+output[output.length-1]);
11         }
12     }
13     return output;
14 }
15
16 console.log(fibNumbers(6));
```

```
output=fibNumbers(6);
console.log(output);
```

```
function fibNumbers(n){
    var output=[];
    if(n===1){
        output=[0];
    }else if(n===2){
        output=[0,1];
    }else{
        output=[0,1];
        for(var i=2;i<n;i++){

output.push(output[output.length-2]+output[output.length-1]);
        }
    }
    return output;
}

console.log(fibNumbers(6));
```

HIGHER ORDER FUNCTIONS

Higher order functions are functions that can take other functions as input.

function function1(function2){ } ==> function2 is a parameter of function1

Create a calculator function that takes 2 number and a function, and calculates based on the operator

Eg: calculator(3,4,add); ==> 7

calculator(3,4,subtract); ==> -1

```
function add(num1,num2){
    return num1+num2;
}
add(3,4);
```

```
function subtract(num1,num2){
    return num1-num2;
}
subtract(3,4);
function multiply(num1,num2){
    return num1*num2;
}
multiply(3,4);
function divide(num1,num2){
    return num1/num2;
}
divide(3,4);
```

DEBUGGER:

Debugs the code:

In snippet, write debugger;

On the next line write the function to debug

Then click on step into to check the steps

We can use console to debug:

debugger; shift+enter go to next line. Write the function to debug. And step into to check steps

```
//Creating calculator function
function calculator(num1,num2,operator){
    return operator(num1,num2);
}

calculator(3,4,multiply);//12
```

```
calculator(3,4,multiply);//12
debugger;
calculator(4,5,subtract);//Function I want to debug
//1. calling calculator and assiging operator function=subtruct
//2. returns 4-5 that is in the subtract function
```

UNDERSTANDING OF JAVASCRIPT OBJECT

```
var person = {
fName : 'John',
lName: 'Cash',
age:19
}
```

KEYS: fName, lName, age

VALUES:John, Cash, 19

```
var user={
name:"Sam",
yearOfBirth: 2000,
address:{
        street:'50 street',
        city:'Dallas',
        state:'Texas'
 },
calculateSalary: function(){
//body of the function
}
}
```

KEYS: name, yearOfBirth,address, calculateSalary

VALUES: Sam, 2000,…

If a key has a function as value, it's a method

UNDERSTANDING OF JAVASCRIPT OBJECT

```
let fname='Ahmet';
let age=20;

let person = {
    //in this curly braces, we can pass more than 1 key-value pairs
    fName:'Ahmet',
    age:20
};//{} is called object literal. We pass js objects using this symbol

console.log(person);// we see key values pairs and properties ofth  person object
console.log(person.fName);//Accessing the name inside person object using dot operation.
// To change hte name of the person there are 2 ways
//1. changing with dot notaiton
person.fName='John';
console.log(person.fName);
//2. bracket notation
//reassing fName in the person object with bracket notation. use single or double quote but single is common
person['fName']='Sam';
console.log(person.fName);//prints Sam
//Dot or [] notation is better? Answer is . notation is shorter and better. But bracket notation has its own uses.
//For example we definet another variable to assign fName property and access fName property throgh that variable
let selection='fName';//assigning the property to a selection variable
console.log("SELECTION "+person[selection]);//Sam
person[selection]='James';//accessing the fName property dynamically and assiging James value
console.log(person[selection]);//James
console.log(person.fName);//James
//The last way is not common. dot notation is the most common one.
```

DEEPER UNDERSTANDING OF JAVASCRIPT OBJECT

We can use js object to hold the properties.

Instead of assigning a single value, we use {} and key-value pairs to assign value for different properties

Properties	person1	person2	person3
Name	“Tim”	“Jim”	“Sam”
Age	20	23	19
hasWorkPermit	TRUE	FALSE	TRUE
Languages	[“English”,” Spanish”]	[“English”,” German”]	[“English”]

```
var person1={
name:”Tim”,
age: 20,
hasWorkPermit: true,
languages:[“English”,” Spanish”]
};
```

```
console.log(person1.name); ===> Tim

console.log(person2.hasWorkPermit);===>false

console.log(person1.languages[1]); ==> Spanish

console.log(person1); ===>
name: "Tim", age: 20, hasWorkPermit: true, languages: Array(2)}
```

Create an object to store person2 and print the person’s all information

CONSTRUCTOR

Constructor is a special function that creates an instance of a class which is knows as object.

We can create constructors when we need to create multiple objects of the same kind.

The purpose is to create an object and set the values

Constructors name must be capitalized.

Constructors take inputs. Match the inputs with the property names inside the constructor body using this keyword

In Javascript, constructor gets called when you declare an object using new keyword

We first need to define a constructs Then use that constructor with “new” keyword

```
//DEFINING CONSRUCTOR

function Car(brand, model,year){

// this is a reference to the object in the code

this.brand=brand;

this.model=model;

this.year=year;

this.setMileague=function(mile){

this.mile=mile;

return mile;}}
```

```
//USING THE CONSTRUCTOR

var car1=new Car('Tesla','Model3',2020)

var car2=new Car('Honda','Accort',2004)

console.log(car1);
```

New keyword does 3 things:

1. Create an empty object

2. Make ‘this’ point to that object

3. And returns the object from construct

UNDERSTANDING OF JAVASCRIPT OBJECT

Constructor Practice:

Create a constructor :Person

And create person1 object

Then log person1 on the console

Do same for person2, and person3 using same constructor as homework

Properties	person1	person2	person3
Name	“Tim”	“Jim”	“Sam”
Age	20	23	19
hasWorkPermit	TRUE	FALSE	TRUE
Languages	[“English”,”Spanish”]	[“English”,”German”]	[“English”]

UNDERSTANDING OF JAVASCRIPT OBJECT

```
var person1={
name:“Tim”,
age: 20,
hasWorkPermit: true,
languages:[“English”,”Spanish”],
exercise: function(){
alert(“Exercise is in progress...”);
walk();
run();
};
```

```
function Person(name, age,hasWorkPermit,languages){
this.name=name;
this.age=age;
this.hasWorkPermit=hasWorkPermit;
this.languages=languages;
this.exercise=function(){
    alert("Exercise is in progress...")
}
};

var person1=new Person('Ahmet',18,true,'English');
person1.exercise();//calling the method
```

DEEPER UNDERSTANDING OF JAVASCRIPT OBJECT

Constructor: Solution

```
function Person(name, age,hasWorkPermit,languages){
this.name=name;
this.age=age;
this.hasWorkPermit=hasWorkPermit;
this.languages=languages;
};
var person1=new Person("Tim",20,true,["English","Spanish"]);
console.log(person1);
var person2=new Person("Jim",23,false,["English","German"]);
console.log(person2);
```

Properties	person1	person2	person3
Name	“Tim”	“Jim”	“Sam”
Age	20	23	19
hasWorkPermit	TRUE	FALSE	TRUE
Languages	[“English”,” Spanish”]	[“English”,” German”]	[“English”]

```
function Person(name, age,hasWorkPermit,languages){
this.name=name;
this.age=age;
this.hasWorkPermit=hasWorkPermit;
this.languages=languages;
};
```

Creating the person1 object using the constructor. We can use this constructor anytime we initialize specific object

```
var person1=new Person(“Tim”,20,true,[“English”,”Spanish”]);
var person2=new Person(“Jim”,23,false,[“English”,”German”]);
```

DEEPER UNDERSTANDING OF JAVASCRIPT OBJECT

Constructor's can have methods.That method can do multiple actions

Below we have log and alert function in exercise

```
function Person(name, age,hasWorkPermit,languages){
this.name=name;
this.age=age;
this.hasWorkPermit=hasWorkPermit;
this.languages=languages;
this.exercise=function(){
    console.log("Started Exercizing");
    alert("Exercise is in progress...");
}
};
var person1=new Person("Tim",20,true,["English","Spanish"]);
console.log(person1);//Person {name: "Tim", age: 20, hasWorkPermit: true, languages: Array(2)}
var person2=new Person("Jim",23,false,["English","German"]);
console.log(person2);//Person {name: "Jim", age: 23, hasWorkPermit: false, languages: Array(2)}
console.log(person2.name);//Jim
console.log(person2.languages);//["English","German"]
person2.exercise();//LOGS THE MESSAGE AND POPS UP THE ALERT
```

PRACTICE

1.Create a function which returns the number of `true` values there are in an array.
Eg:

```
countTrue([true, false, false, true, false]) → 2

countTrue([false, false, false, false]) → 0
```

2.A repdigit is a positive number composed out of the same digit. Create a function that takes an integer and returns whether it's a repdigit or not.

```
isRepdigit(66) → true

isRepdigit(6666) → true

isRepdigit(0) → true

isRepdigit(-11) → false
```

3.Create a function that *counts* the integer's **number of digits.**
Eg:

```
count(318) → 3

count(-92563) → 5
```

4.Create a function that takes an array of numbers and returns the **second largest** number.

```
secondLargest([10, 40, 30, 20, 50]) → 40

secondLargest([25, 143, 89, 13, 105]) → 105

secondLargest([54, 23, 11, 17, 10]) → 23
```

JAVASCRIPT ES6 ARROW FUNCTIONS

```
// Traditional Function
function (a){
  return a + 100;
}

// Arrow Function Break Down

// 1. Remove the word "function" and place arrow between the argument and open.
(a) => {
  return a + 100;
}

// 2. Remove the body brackets and word "return" -- the return is implied.
(a) => a + 100;

// 3. Remove the argument parentheses
a => a + 100;
```

```
// Traditional Function
function (a, b){
  return a + b + 100;
}

// Arrow Function
(a, b) => a + b + 100;
```

```
// Traditional Function (no arguments)
let a = 4;
let b = 2;
function (){
  return a + b + 100;
}

// Arrow Function (no arguments)
let a = 4;
let b = 2;
() => a + b + 100;
```

```
// Traditional Function
function (a, b){
  let chuck = 42;
  return a + b + chuck;
}

// Arrow Function
(a, b) => {
  let chuck = 42;
  return a + b + chuck;
}
```

```
// Traditional Function
function bob (a){
  return a + 100;
}

// Arrow Function
let bob = a => a + 100;
```

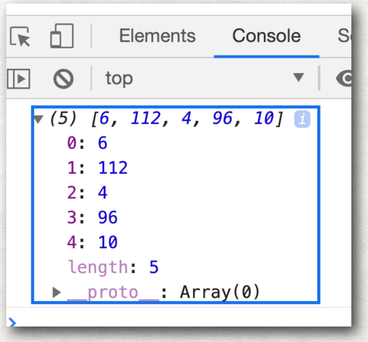
//Map –Create a new array by doing something with each item in an array.

Let's use map to pass another function inside this map.

1.For example, create function to double the numbers and loop thought double:

```
//create a function
function double(x){
    return x*2;
}
//pass a function in map to loop through double
function
const newNumber=numbers.map(double);
console.log(newNumber);
```

```
//SHORTENING MAP: COMBINE ABOVE CODE:
const newNumber1=numbers.map(
    function double(x){
        return x*2;
    }
);
console.log(newNumber1);
```



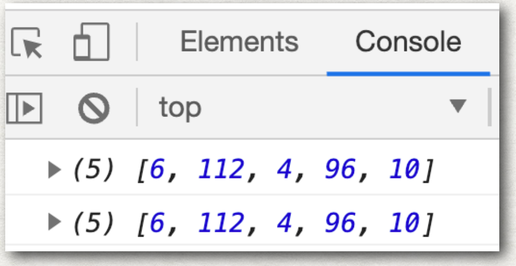
2. We can do the same thing with for each loop:

```
//with for each loop:
var newNumbers=[];
function double1(x){
    newNumbers.push(x*2);
}
numbers.forEach(double1);
console.log(newNumbers);
```

We can shorten this for each loop by putting the function inside the foreach:

```
const newNumbers1=[];
numbers.forEach(function (x){
    newNumbers1.push(x*2);
});
console.log(newNumbers1);
```

Here with foreach, we have to create a new empty array and then every time we do something with each item inside the array we push the new array. We can do this with map easily.



index.js

```
var numbers = [3, 56, 2, 48, 5];

//Map –Creates a new array by doing something with each item in an array.
//Map let us use a function inside a function and loop it
// create a function
function double(x){
    return x*2;
}
const newNumber=numbers.map(double);
console.log(newNumber);
//SHORTENING MAP: COMBINE ABOVE CODE:
const newNumber1=numbers.map(
    function double(x){
        return x*2;
    }
);
console.log(newNumber1);

//with for each loop:
const newNumbers=[];
function double1(x){
    newNumbers.push(x*2);
}
numbers.forEach(double1);
console.log(newNumbers);

// SHORTENNING FOR EACH: We can put double1 function in the for each:
const newNumbers1=[];
numbers.forEach(function (x){
    newNumbers1.push(x*2);
});
console.log(newNumbers1);
```

//Filter – Create a new array by keeping the items that return true.

Let's use map to filter the array based on a specific condition

1. For example, create function to double the numbers:

```
//This filter function will take each num inside numbers array
and return the only the onces with specific condition
//and put them inside a new array called newNumbers
//LET'S CREATE A FILTER TO GET NUMBERS LESS THAN 10

const newNumbersArray=numbers.filter(
    function (num){
    return num>10;
});
console.log(newNumbersArray);
```

```
▶ (2) [56, 48]
```

1. We can do the same thing with for each loop:

```
//with for each loop:
var newNumbersArray1=[];
numbers.forEach(function(num){
    if(num>10){
        newNumbersArray1.push(num);
    }
});
console.log(newNumbersArray1);
```

```
▶ (2) [56, 48]
```

```
//Filter – Create a new array by keeping the items that return true.
//This filter function will take each num inside numbers array and
//return the only the onces with specific condition
//and put them inside a new array called newNumbers
//Let;s return only numbers greater than 10
const newNumbersArray=numbers.filter(
    function (num){
    return num>10;
});
console.log(newNumbersArray);

//with for each loop:
var newNumbersArray1=[];
numbers.forEach(function(num){
    if(num>10){
        newNumbersArray1.push(num);
    }
});
console.log(newNumbersArray1);
```

```
//Reduce – Accumulate a value by doing something to each item in an array.
```

RESUCE is used to accumulate values

For example, create function to add the numbers with for each first:

```
//Let's Add all of the numbers  With forEach:
var newNumber2=0;
numbers.forEach(
    function(eachNum){
        newNumber2+=eachNum;
    }
);
console.log(newNumber2);
```

```
114
```

We can do the same thing with reduce function:

```
var newNumber3=numbers.reduce(
function(accumulator,eachNumber){
console.log("accumutator : "+accumulator);
console.log("eachNumber: "+eachNumber);
return accumulator+eachNumber;
}
);
console.log(newNumber3);
```

```
accumutator : 3
eachNumber: 56
accumutator : 59
eachNumber: 2
accumutator : 61
eachNumber: 48
accumutator : 109
eachNumber: 5
114
```

Those 3 functions are the most useful javascript function

Now we will talk find a find index which are relatively new

```
//Reduce – Accumulate a value by doing something to each item in an array
//Let's Add all of the numbers :
var newNumber2=0;
numbers.forEach(
    function(eachNum){
        newNumber2+=eachNum;
    }
);
console.log(newNumber2);

//Do Same thing using reduce
var newNumber3=numbers.reduce(
function(accumulator,eachNumber){
console.log("accumutator : "+accumulator);
console.log("eachNumber: "+eachNumber);
return accumulator+eachNumber;
}
);
console.log(newNumber3);
```

//FIND - FIND THE FIRST ITEM THAT MATCHES FROM AN ARRAY.
//FINDINDEX - FIND THE INDEX OF THE FIRST ITEM THAT MATCHES.

Find function will find the first item that matches in the array.

This will not loop through entire array unlike map. It will stop as soon as it finds the first matching item

findIndex does same think but it will return the first matching item's index.

For example, create function to add the numbers with for each:

```
//Find function will find the first item that matches in the
array.
//Let's return the first number that is greater that 10
const newNumber4=numbers.find(
    function(eachNum){
        return eachNum>10;
    }
);
console.log(newNumber4);
```

56

We can do the same thing with reduce function:

```
//Find function will find the INDEX first item that matches in
the array.
//Let's return the first number's INDEX that is greater that 10
//index start at 0
const newNumber5=numbers.findIndex(
    function(eachNum){
        return eachNum>10;
    }
);
console.log(newNumber5);
```

1

```
//Find function will find the first item that matches in the array.
//Let's return the first number that is greater that 10
const newNumber4=numbers.find(
    function(eachNum){
        return eachNum>10;
    }
);
console.log(newNumber4);


//Find function will find the INDEX first item that matches in the array.
//Let's return the first number's INDEX that is greater that 10
//index start at 0
const newNumber5=numbers.findIndex(
    function(eachNum){
        return eachNum>10;
    }
);
console.log(newNumber5);
```

JAVASCRIPT ES6 ARROW FUNCTIONS
KEEP GOINF ON THE PROJECT MAP-FILTER-REDUCE

```
// ================ARROW FUNCTIONS=========================
var numbers=[3,56,2,48,5];
//1.Create function seperatelly and use inside the map
function square (x){
    return x*x;
}
const newNumbers=numbers.map(square);
console.log(newNumbers);

//OR PUT HTE FUNCTION IN THE MAP DIRECTLY:
const newNumbers1=numbers.map(function square (x){
    return x*x;
});
console.log(newNumbers1);

//WE CAN DELETE THE NAME OF THE INNER FUNCTION (ANAYNOMUS FUNCTION)
const newNumbers2=numbers.map(function (x){
    return x*x;
});
console.log(newNumbers2);


// ARROW FUNCTIONS TAKES THIS CONCEPT ONE STEP AHEAD
// Delete function keyword and use FAT ARROR SYMBOL:
const newNumbers3=numbers.map( (x) => {
    return x*x;
});
console.log(newNumbers3);
// WE CAN MAKE THIS SHORTER WHEN THERE IS ONLY ONE INPUT::
//  teh inner paranthesis  around x
//EVEN WE CAN DELETE THE {} and return keyword and inner ;
// This is too short to understand and may be hard to understand
const newNumbers4=numbers.map( x => x*x );
console.log(newNumbers4);
```

JAVASCRIPT ES6 ARROW FUNCTIONS
KEEP GOINF ON THE PROJECT MAP-FILTER-REDUCE TASK HOMEWORD

```
// ARROW FUNCTION TASK: CONVERTU EACH OF THE FUNCTION to ARROW FUNCTION
//YOUR CHALLANGE IS TO CONVERT EACH FUNCTION AND SIMPLIFY
var numbers = [3, 56, 2, 48, 5];

//Map -Create a new array by doing something with each item in an array.
const newNumbers = numbers.map(function (x) {
  return x * 2;
});

////Filter - Create a new array by keeping the items that return true.
const newNumbers = numbers.filter(function(num) {
  return num < 10;
});

Reduce - Accumulate a value by doing something to each item in an array.
var newNumber = numbers.reduce(function (accumulator, currentNumber) {
    return accumulator + currentNumber;
})

//Find - find the first item that matches from an array.
const newNumber = numbers.find(function (num) {
  return num > 10;
})

//FindIndex - find the index of the first item that matches.
const newNumber = numbers.findIndex(function (num) {
  return num > 10;
})
```

```
SOLUTION:
//YOUR CHALLANGE IS TO CONVERT EACH FUNCTION AND SIMPLIFY
var numbers = [3, 56, 2, 48, 5];

//Map -Create a new array by doing something with each item in an array.
const newNumbers11 = numbers.map(x =>x * 2);
console.log(newNumbers11);
////Filter - Create a new array by keeping the items that return true.
const newNumbers22 = numbers.filter(num => num < 10);
console.log(newNumbers22);
// Reduce - Accumulate a value by doing something to each item in an array.
//BELOW HAS TWO INPUT PARAMETER AND ONE SINGLE RETURN. WE CAN'T REMOVE INPUT PARANTHESIS, BUT SHORTEN THE RETURN
var newNumbers33 = numbers.reduce( (accumulator, currentNumber) => accumulator + currentNumber);
console.log(newNumbers33);
//Find - find the first item that matches from an array.
const newNumbers44 = numbers.find(num => num > 10);
console.log(newNumbers44);
//FindIndex - find the index of the first item that matches.
const newNumbers55 = numbers.findIndex(num => num > 10);
console.log(newNumbers55);
```

JAVASCRIPT ES6 SPREAD OPERATOR

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Spread_syntax

USE THE SAME FOLDER OR Import the starting folder. Install and start

What are the ways to reduce the code we wrote in setContact? Usually it will use by using spread operator.SO let's learn how spread operator works

1. Inside index.js create two arrays:

```
const citrus = ["Lime", "Lemon", "Orange"];
const fruits = ["Apple", "Banana", "Coconut"];
2. How can add all items in citrus array in the fruits array AT THE END
3. One was to push each citrus item in the fruits array one bu by using a
   loop
4. Or I can simply use spread operator to add one array into another:
const fruits = ["Apple", "Banana", "Coconut", ...citrus];
console.log(fruits);
5. We can add spread operator in any order. Let's use it in the 2nd
place:
const fruits = ["Apple", ...citrus, "Banana", "Coconut", ];
console.log(fruits)
```

index.js

```
1  import React from "react";
2  import ReactDOM from "react-dom";
3  import App from "./components/App";
4
5  ReactDOM.render(<App />, document.getElementById("root"));
6
7  const citrus = ["Lime", "Lemon", "Orange"];
8  const fruits = ["Apple", ...citrus, "Banana", "Coconut", ];
9  console.log(fruits)
```

```
▶(6) ["Apple", "Lime", "Lemo
n", "Orange", "Banana", "Cocon
ut"]
```