// alert is a JavaScript function

alert('Learning JavaScript');

*let* varaible = 'current value';

// updating the variable value

varaible = 'updated value';

console.log(varaible);

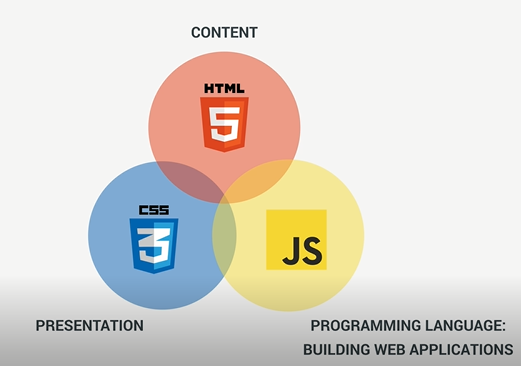
JavaScript is High Level, Object Oriented(Js is based on objects for storing data inform of properties), Multi paradigm(Use different styles of programming) programming Language.

Core Technologies of Web

1)HTML

2)CSS

3)JavaScript



JavaScript and Web Browser are two different things

JavaScript can also run outside web browser.

JavaScript can also be used on web server.

ES stands for ECMA Script

Modern JavaScrip

ES6-ES11

**Console:**

Console is just an environment that was build to execute small pieces of code and show results immediately ,does not show results of operation in code by default, unless we use console.log function.

**Values and Variables:**

Value is the smallest unit of information, that we have in JavaScript.

// Values and Variables

// Value is a piece of data, most fundamental of information that we have in programming

// A is a value

console.log("A")

// 23 is a value

console.log(23)

// 21 and 2 are the values

console.log(21+2)

**variables store the values.**

**In this way we can use values over and over again.**

// Declarig  a variable

// Assigning value to the nameofvariable

// Following code creates a real varaible in computer memory

// Value will be stored in nameofvariable

// Name varaible in camelcasing form

// Variables names cannot start with a number

// Variable name can only contain number , letters,under-score or dollar sign

// Do not use reserved key words in naming variables like new, function,name

//Do not start variable name with upper case letter like Person

//Variable name must be descriptive

*let* nameofvariable = 'Value';

// Variables that are all in upper case  are reserved for constants and constants

// will never change like number PI

*let* PI = 3.1415;

*let* county = 'Pakistan';

*let* continent = 'South Asia';

*let* population = 1200000000000000;

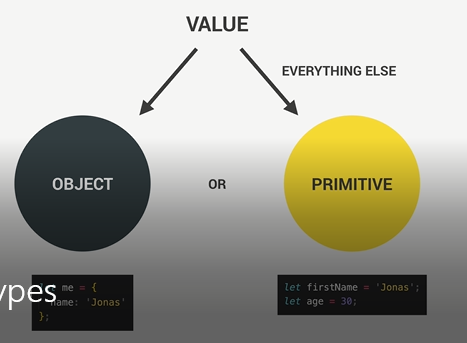
console.log(county, continent, population);

**Value:**

**In JavaScript every value is either an**

**1)Object**

**2)Primitive Value**



**There are seven primitive data types:**

**1)Number**

// Number data type

// Number are floating point numbers , hence they always have decimals

// Even if we don't define or see decimnals in numbers

// 23 is equivalent to 23.0

*let* age = 23;



**String**

// String data type

// String is a sequence of characters

*let* myName = 'Muhammad Asad Khalil Rao';



**Boolean:**

// Boolean data type

// Boolean data type can only take  either true logical value  or false logical value

// Boolean is always either true or false

*let* trueLogic = true;

**Undefined:**

// Undefined data type

// Undefined is the values taken by the variable that is not yet defined(empty value)

// Variable that is not yet defined is the variable that is declared but not assigned

// any value

*let* variable\_name;

**Null:**

Null also means empty value.

**Symbol:**

Value that is unique and cannot be changed.

**Big int:**

Larger integers than the number type can hold.

JavaScript has feature called **dynamic typing**; this means that **when we create a new variable,** we **do not have to manually define the data type of the value contained inside the variable.**

JavaScript automatically determines what is the data type of the value.

**Always** value has the type.

**Variable has no type**.

**Variable only store value (value have type)**

// Assingning new value with a different data type to the same varaible

*let* age = 23;

age = true;

console.log(age);

let nameofvariable=value(always value has the type)

**Type of Operator:**

*let* age = 23;

console.log(typeof age);

number

// Assingning new value with a different data type to the same varaible

age = true;

console.log(age);

// typeof is an operator that shows the type of value

console.log(typeof age);

boolean

*let* myName = 'Muhammad Asad Khalil Rao';

console.log(typeof myName);

string

**Dynamic Typing:**

Changing the type of the value that is hold by a variable.

*let* age = 23;

console.log(typeof age);



// Assigning new value with a different data type to the same variable

age = true;

console.log(typeof age);



**Undefined:**

When ever we declare an empty variable, the value of the variable is undefined, and type of the variable is also undefined.

Empty variable automatically holds the undefined value.

// Undefined type value of the variable

// type of the value of the new\_variable is undefined

*let* new\_variable;

console.log(typeof new\_variable);



Reassigning the variable

// Undefined type value of the variable

// type of the value of the new\_variable is undefined

*let* new\_variable;

console.log(typeof new\_variable);

// Reassigning new value to undefined variable

new\_variable = 'I have a new value';

console.log(typeof new\_variable);



**Error existing in the typeof operator:**

For null, both the value and type of the value are null.

But JavaScript’s typeof operator gives us object in case of null, in reality type of null value is null. Hence this is regarded as bug or error in JavaScript.

// ERROR in the type of operator

*let* null\_value = null;

console.log(typeof null\_value);



**Overall code**

// Number data type

// Number are floating point numbers , hence they always have decimals

// Even if we don't define or see decimnals in numbers

// 23 is equivalent to 23.0

*let* age = 23;

console.log(typeof age);

// String data type

// String is a sequence of characters

*let* myName = 'Muhammad Asad Khalil Rao';

// Boolean data type

// Boolean data type can only take  either true logical value  or false logical value

// Boolean is always either true or false

*let* trueLogic = true;

// Undefined data type

// Undefined is the values taken by the variable that is not yet defined

// Variable that is not yet defined is the variable that is declared but not assigned

// any value

*let* variable\_name;

console.log(variable\_name);

// Assingning new value with a different data type to the same varaible

age = true;

console.log(age);

// true is a boolean value

// typeof is an operator that shows the type of value

console.log(typeof age);

console.log(typeof myName);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DYNAMIC TYPING\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Dynamic typing means we can easily change the type of a value  that is hold by a

// variable

// Undefined type value of the variable

// type of the value of the new\_variable is undefined

*let* new\_variable;

console.log(typeof new\_variable);

// Reassigning new value to undefined variable

new\_variable = 'I have a new value';

console.log(typeof new\_variable);

// ERROR in the type of operator

*let* null\_value = null;

console.log(typeof null\_value);

**Ways of declaring variable:**

// There are three different ways of declaring variable

**1)let**

**2)var**

**3)const**

// We use the const key word to declare variable that are not suppose to change

// at any point in the future

*const* birthYear = 2001;

birthYear = 2002;



**We cannot declare empty const variable.**

// We cannot declare empty const varaible

*const* empty\_varible;

Declaring a variable and assigning it a value without var,let and const key words does not create a variable but it creates a property on global object.

// Declaring a variable and assigning it a value without var,let and const key words does not create a variable but it creates a property on global object.

global\_property = 'Global Property';

console.log(global\_property);

**Operators in JavaScript**

Transform values, combine multiple values.

Types of operators:

**Assignment Operator:**

// = is called the assignment operator

*let* x = 10 + 2;

// Following code is equivalent to x=x+10;

x += 10;

console.log('The value of x is ' + x);

// Following code means x=x\*(-1)

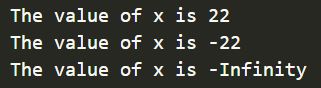
x \*= -1;

console.log('The value of x is ' + x);

// Following code means x=x/0

x /= 0;

console.log('The value of x is ' + x);



//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Assignment Operators\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

// = is called the assignment operator

*let* x = 10 + 2;

// Following code is equivalent to x=x+10;

x += 10;

console.log('The value of x is ' + x);

// Following code means x=x\*(-1)

x \*= -1;

console.log('The value of x is ' + x);

// Following code means x=x/0

x /= 0;

console.log('The value of x is ' + x);

*let* y = -13;

// Following code means  y=y+1;

y++;

console.log('The value of y is ' + y);

// Following code means  y=y-1

y--;

console.log('The value of y is ' + y);

**Comparison Operators: These operators give us the Boolean values**

console.log(3 > -1);

true

console.log(3 < -1);

false

*let* birth\_year = 2001;

*let* current\_year = 2023;

// - is the mathematical operator

*let* age = current\_year - birth\_year;

console.log(age >= 22);

true

**Order in which operators are executed:**

Comparison operator has precedence of 9.

Graphical user interface, text, application

Description automatically generated

The subtraction operator has precedence of 11.

Since Subtraction operator has higher precedence hence will be executed first.

Table

Description automatically generated with medium confidence

**Subtraction Operator is executed from the left to right:**

// 23-2 is first then 21-2 hence result is 19

console.log(23 - 2 - 2);

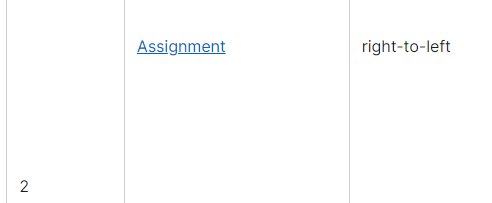
**Defining Two variables at same time:**

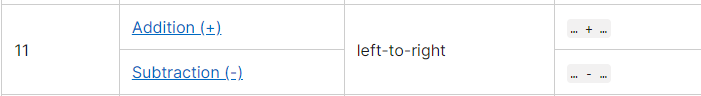
// Defining two variables at the same time

*let* x, y;

// Result of the subtraction will assigned to y then to x

x = y = 25 - 10 - 5;





Subtraction from left to right will first occur then assignment operation from right to left will occur.

()operator has highest precedence.

**String Concatenation**

We use the + sign to concatenate the strings.

*const* first\_name = 'Asad';

*const* job = 'Web developer';

*const* birth\_year = 2001;

*const* current\_year = 2023;

// We use the + to concatenate the string

console.log(

  'My name is ' +first\_name +' I work as a ' +job +' I was born in' +

    // JavaScript will simply convert birth\_year number into string to concatenate

    birth\_year +

'  I am ' +

       // JavaScript will simply convert current\_year - birth\_year  number into string to concatenate

    (current\_year - birth\_year) +' years old'

);

**Template Literals**

We write a string in normal way and insert variables directly into the string.

Use `` back ticks to write template literals.

*const* first\_name = 'Asad';

*const* job = 'Web developer';

*const* birth\_year = 2001;

*const* current\_year = 2023;

console.log(

  `My name is ${first\_name}. I work as a ${job}. I was born in ${birth\_year}. I am ${

    current\_year - birth\_year

  } years old`

);

${ nameofvariable } to extract the value of the variable.

**\n\**

To get to net line or make string come to new line we use **\n\**.

console.log(

  'My name is Asad \n\

I am a web developer\n\

Nice to meet you.'

);

**Using Template Literals(No need of \n\)**

If we need a multiple line string, would be better to use template literals.

console.log(

  `My name is Asad

I am a web developer

Nice to meet you.`

);

**If Else**

*const* candidate\_age = *Number*(prompt('Enter age of candidate'));

*const* minimum\_age = 18;

if (console.log(candidate\_age == minimum\_age || candidate\_age > minimum\_age)) {

  console.log('Eligible');

} else {

  console.log(

    `Candidate below 18 is not eligible and will be eligible after ${

      minimum\_age - candidate\_age

    }`

  );

}

The expression in side the () of if evaluated and turns out to be true or false, if the expression inside () of if turns out to be true then code inside {} of if will be executed.

The code in {} of the else block will be executed when ever the condition is false

Century variable declared inside the code block {} of if will only be accessible inside the code block {} of if

*const* birth\_Year = 1998;

if (birth\_Year <= 2000) {

  // century varaible declared inside the code block {} of if will

  // only be accessible inside the code block {}of if

*let* century = 20;

} else {

  //  century varaible declared inside the code block {} of else will

  // only be accessible inside the code block {}of if

*let* century = 21;

**}**

**Remedy**

So, we need to defined century variable outside the if else and assign it value inside the code block {} of if and code block {} of else.

*const* birth\_Year = 1998;

*let* century;

if (birth\_Year <= 2000) {

  century = 20;

} else {

  century = 21;

}

**Coding Challenge**

// Coding Challenge

*let* marks\_mass = *Number*(prompt("Enter Mark's weight"));

*let* marks\_height = parseFloat(prompt("Enter Mark's height"));

*let* john\_mass = *Number*(prompt("Enter John's weight"));

*let* john\_height = parseFloat(prompt("Enter John's height"));

*let* marks\_BMI = marks\_mass / Math.pow(marks\_height, 2);

*let* john\_BMI = john\_mass / Math.pow(john\_height, 2);

if (marks\_BMI > john\_BMI) {

  console.log(

    `Mark's BMI is (${marks\_BMI})  higher than John's BMI (${john\_BMI})`

  );

} else {

  console.log(

    `John's BMI is (${john\_BMI})  higher than Mark's BMI (${marks\_BMI})`

  );

}

**Type Conversion**

**Type conversion:**

Manually(explicitly) converting one type to another

**Example:**

Converting a String to a Number manually.

Converting a Number into Boolean manually.

**Code:**

Input field data from the input field usually come inform of strings.

*let* input\_fiedl\_data = '1999';

console.log(typeof (input\_fiedl\_data + 10));



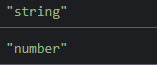
**Converting string to a number:**

Use Number function.

*Number*()

*let* input\_fiedl\_data = '1999';

console.log(typeof *Number*(input\_fiedl\_data));



**NaN is Not A Number**

JavaScript gives us a NaN value whenever an operation that involves numbers fails to produce a new number.

**Example:**

Number is a function converts string data type to number data type.

*Number*()

console.log(*Number*('string'));



NaN is actually an invalid number, it’s still a number but somehow an invalid.

**Converting Number or Boolean to a String**

We use the

*String*()

function to convert Number to string.

console.log(typeof(*String*(123)))



console.log(typeof *String*(true));



**Type Coercion:**

JavaScript automatically converts the types behind the scenes (completely hidden from us) for us.

Type Coercion happens when operator is dealing with two values that have different types.

JavaScript will then behind the scenes convert one of the values to match the other value so in the end operation can be executed.

**Example:**

console.log(typeof ('I am ' + 23 + 'years old' + true));



23 is a number yet it is converted into String.

True is a Boolean yet it is converted into Boolean.

+ operator triggers a coercion to string (**numbers** are **converted to strings and concatenated**).

Whenever there is an operation between a string and number, the number will be treated a string.

Not all the operators like + do coercion.

// JavaScript convert 23 string to number

// JavaScript convert 10 string to number

//23-10-3=10

console.log("23"-"10"-3)

- operator triggers a coercion to number (strings are converted to numbers **and concatenated**).

**\* operator converts the string to numbers:**

console.log('23' \* '2');



**/operator converts string to number:**

console.log('23' / '2');



**Logical Operator converts string to number**

**Logical operator includes >, < or == and =>**

console.log('32' < '95');



console.log('32' <= '32');



**Falsy Values:**

Values which are not exactly false but will become false when we try to convert them into Boolean.

**In JavaScript there are only five falsy values:**

1. **0**
2. **“”Empty string**
3. **undefined**
4. **null**
5. **NaN (Not A Number)**

All of the above mentioned five values are not false initially but will be converted into false when we attempt to convert the above-mentioned values into Boolean.

**Truthy:**

Value that will be converted into true are called Truthy values,

**Example:**

1. Number other than 0
2. String that is not empty

The above mentioned will be converted to true.

**Converting Number to Boolean:**

// Converting 0 Number to boolean

console.log(*Boolean*(0));



// Converting Non-Zero Number to boolean

console.log(*Boolean*(-13.4));



**Converting String to Boolean**

// Converting Empty string to Boolean

console.log(*Boolean*(''));



// Converting Non-Empty string to Boolean

console.log(*Boolean*('Asad'));



**Converting Empty Object to Boolean**

//Converting Empty Object to Boolean

console.log(*Boolean*({}));



**Converting undefined to Boolean**

//Converting undefined Object to Boolean

console.log(*Boolean*(undefined));



**Converting NaN to Boolean**

//Converting NaN Object to Boolean

console.log(*Boolean*(NaN));



**money is a number(could be 0 , negative, positive)**

In the () brackets of if JavaScript will convert number to Boolean according to truthy and falsy value concept.

*const* money = 1;

if (money) {

  console.log('Dont spend it all');

} else {

  console.log('Get a Job');

}



*const* money = -120.4;

if (money) {

  console.log('Dont spend it all');

} else {

  console.log('Get a Job');

}



**But incase of 0 it will be false**

*const* money = 0;

// In the () brackets of if JavaScript will convert number to boolean

if (money) {

  console.log('Dont spend it all');

} else {

  console.log('Get a Job');

}



*let* height;

console.log(typeof height);

if (height) {

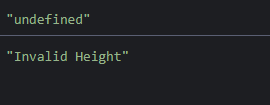
  console.log('Valid Height');

} else {

  console.log('Invalid Height');

}

**height is not defined hence it will be considered false inside the () of if**



**=== is called the Strict Equality Operator:**

To know whether something is exactly equal to something, we use ==== operator

===operator will return either true(when both values are the same ) or false value (Boolean value)

If something is exactly equal to something then === returns true

*let* age = 18;

if (age === 18) console.log('You just turned 18');





For the equality operator to return true value, both sides need to be exactly the same.

**===** operator does not perform type coercion.

**Loose Equality operator:**

**==** is called the loose equality operator.

Loose Equality operator does type coercion.

console.log('18' == 18);



== loose equality operator converts string “18” to number

console.log(18 == "18");

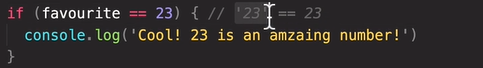


== loose equality operator converts number 18 to string

**Assignment operator is the single equal =**

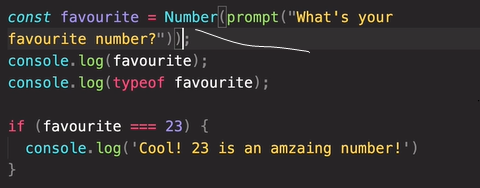
Prompt is a function that gives us the prompt window and we can input something in the prompt window.

Prompt takes the value in as a string.

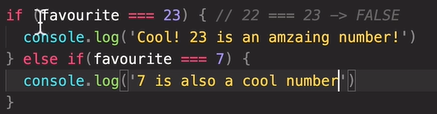


“23” string from the prompt input will be converted into number as loose equality operator does type coercion.

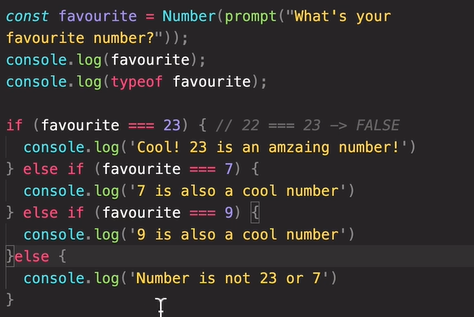
If === strict equality operator is used, then we would have to do type coercion manually.



**Adding more condition to if block using elseif**



**We can add more than one else if block**



If none of the conditions are true, then in the end else block executed.

**Not Equal Operator**

**!==**

One equal is the Loose Version

Two Equal is the Strict Version



The above if will be evaluated only on number other than 23.

**Boolean Logic**

Boolean Logic is a branch of computer science which uses true and false values to solve complex logical problems.

It uses several logical operators to combine true or false values.

Most Basic Logical Operators

**1)AND**

All variables need to be true for the result to be true.

AND operator returns true only if all(more than two) conditions are true.

**Example:**



Result would be true only if all(A,B,C) of them are true.

**2)OR**

OR operator will be true if just one of the variables if true.

OR operator returns true only if all(more than two) conditions are true.

**3)NOT:**

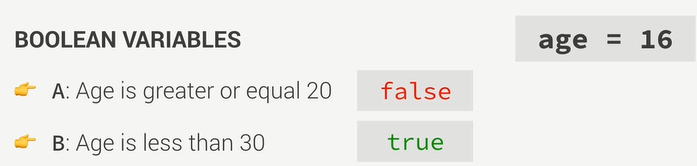
Not operator only acts on one Boolean value.

It inverts the Boolean value.

**Example:**

If A is true then NOT A will become false.

If A is false then NOT A will become true.



**NOT**

A white background with black text

Description automatically generated

**AND**

A close-up of a person's face

Description automatically generated

**OR**

A white surface with a black border

Description automatically generated

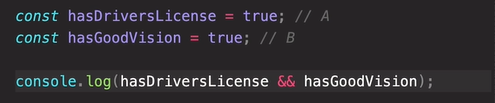
**NOT and AND**

NOT operator has precedence over the AND and OR operator



**AND operator in JavaScript**

**&&**





**OR operator in JavaScript**







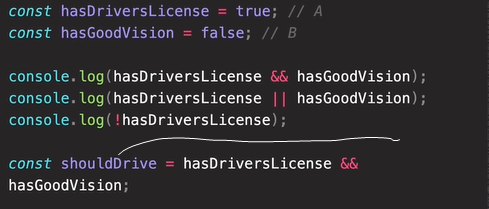
**NOT Operator in JavaScript**



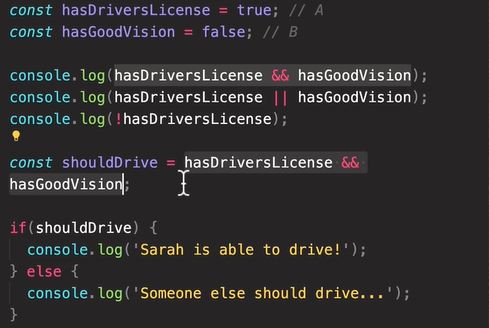


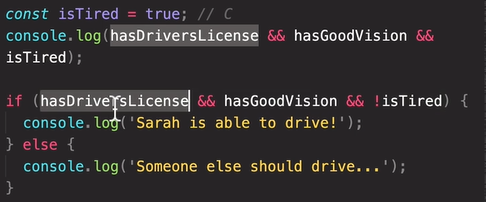
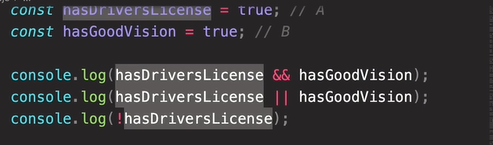


**Boolean Variable**



shouldDrive is a Boolean Variable, could be true or false, could be used in if else.





Code Challenge

*const* dolphin\_score\_one = *Number*(prompt('Enter First Match Score of Dolphin'));

*const* dolphin\_score\_two = *Number*(prompt('Enter Second Match Score of Dolphin'));

*const* dolphin\_score\_three = *Number*(

  prompt('Enter Third Match Score of Dolphin')

);

*const* sum\_dolphin = dolphin\_score\_one + dolphin\_score\_two + dolphin\_score\_three;

*const* number\_of\_mactches = 3;

*const* average\_dolphin = sum\_dolphin / number\_of\_mactches;

*const* koalas\_score\_one = *Number*(prompt('Enter First Match Score of Koalas'));

*const* koalas\_score\_two = *Number*(prompt('Enter Second Match Score of Koalas'));

*const* koalas\_score\_three = *Number*(prompt('Enter Third Match Score of Koalas'));

*const* sum\_koalas = koalas\_score\_one + koalas\_score\_two + koalas\_score\_three;

*const* average\_koalas = sum\_koalas / number\_of\_mactches;

*const* minimum\_score = 100;

if (average\_dolphin > average\_koalas && average\_dolphin >= minimum\_score) {

  console.log('Dolphin  Wins');

} else if (

  average\_dolphin < average\_koalas &&

  average\_koalas >= minimum\_score

) {

  console.log('Koalas  Wins');

} else if (

  average\_dolphin === average\_koalas &&

  average\_dolphin >= minimum\_score &&

  average\_koalas >= minimum\_score

) {

  console.log('It is a Draw');

} else {

  console.log('No one wins the trophy');}

**Switch Statements**

Compare one value with multiple different options.

switch () {

 case :

  break;

 default:

  break;

}

Inside the () of switch we put variable which is to be compared with multiple options.

Inside case we include one of the multiple options.

**Comparison** of a variable to the **multiple** options(cases)is **made** in **strict equality way**.

*let* week\_day = prompt('Enter one of the days of week');

// Mapping one activity to each day

// we are switching is the day

// We are comparing week\_day to multiple options

switch (week\_day) {

  // After the case and the colon,all the lines are executed

  case 'Monday': //week\_day===Monday , if this is true then code after : is executed

 console.log('Today is Monday');

  console.log('Feed the cat');

    break;

  default:

    break;

}

Everything after : of the case is executed.

To **execute** the **same code for different cases,** we use following code:

(Skip the break part)

  case 'Sunday': week\_day===Sunday

  case 'Saturday': week\_day===Saturday

    console.log('Today is Sunday');

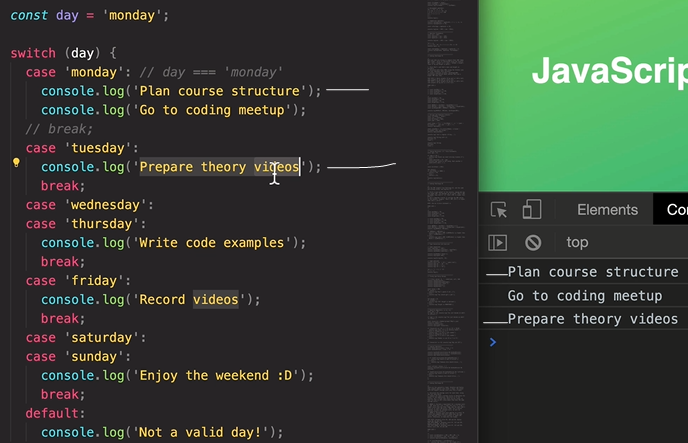
The code after semi-colon after Saturday will be executed for both Sunday and Saturday.

  //Default case is executed if all the other cases fail.

  default:

    console.log('Invalid Day');

Without the break the code continues executing continuously unless and until it encounter break.



**Capitalizing the first letter of String input**

*let* week\_day =

  week\_day\_input.charAt(0).toUpperCase() + week\_day\_input.slice(1);

**charAt(0) get the first letter of the input string.**

**toUpperCase method does capitalization of first letter**

**slice(1) get rest of the string starting from second character.**

**+is used for concatenation.**

**Code**

*let* week\_day\_input = prompt('Enter one of the days of week');

*let* week\_day = week\_day\_input.charAt(0).toUpperCase() + week\_day\_input.slice(1);

// Mapping one activity to each day

// we are switching is the day

// We are comparing week\_day to multiple options

switch (week\_day) {

  // After the case and the colon,all the lines are executed

  case 'Monday': //week\_day===Monday , if this is true then code after : is executed

    console.log('Today is Monday');

    console.log('Today is first day of week');

    break;

  case 'Tuesday':

    console.log('Today is Tuesday');

    console.log('Today is second day of week');

    break;

  case 'Wednesday':

    console.log('Today is Wednesday');

    console.log('Today is third day of week');

    break;

  case 'Thursday':

    console.log('Today is Thursday');

    console.log('Today is fourth day of week');

    break;

  case 'Friday':

    console.log('Today is Friday');

    console.log('Today is fifth day of week');

    break;

  case 'Saturday':

    console.log('Today is Saturday');

    console.log('Today is sixth day of week');

    break;

  case 'Sunday':

    console.log('Today is Sunday');

    console.log('Today is seventh day of week');

    break;

  //Default case is executed if all the other cases fail.

  default:

    console.log('Invalid Day');}

// Using if else statement

if (week\_day === 'Monday') {

  console.log('Today is Monday (using if else)');

} else if (week\_day == 'Tuesday') {

  console.log('Today is Tuesday (using if else)');

} else if (week\_day === 'Wednesday' || week\_day === 'Thursday') {

  console.log('Today is Wednesday/Thursday (using if else)');

} else if (week\_day === 'Friday') {

  console.log('Today is Friday (using if else)');

} else if (week\_day === 'Saturday' || week\_day === 'Sunday') {

  console.log('Today is Saturday/Sunday (using if else)');

} else {

  console.log('Invalid Day(using if else)');}

|  |  |
| --- | --- |
| **Statements** | **Expressions** |
| Bigger piece of code  Does not produce a value on itself | Piece of code that produces value.  3+5 |
|  |  |

**Conditionary Operator**

**condition ? code to be executed in case of true : code to be executed in case of false condition**

**Example**

*let* age = 23;

// After the ? we write the code that is to be executed when

// condition is evaluated as true

// After ? only one line of code can be writte

age >= 18 //Condition

  ? //if part

console.log('Condition is True')

  : //else part

console.log('Condition is False');

Think of ? to be if

Think of : to be else

**Conditional Operator has three parts:**

**1)Condition**

age >= 18

**2)After ? comes the if part code**

console.log('Condition is True')

**3)After : comes the else part code**

console.log('Condition is False')

**Over-all:**

*let* age = 23;

age >= 18

  ? console.log('Condition is True')

  : console.log('Condition is False');

**Ternary Operator is an expression hence value is returned/produced:**



**We can have conditions inside template literals using ternary operator (as expression can be inserted inside template literals):**



A black background with yellow text

Description automatically generated

**Strict Mode in JavaSript:**

Strict mode is a special mode to write secure JavaScript code.

Write the following statement as first beginning of the script.

'use strict';

**Functions**

**Declaring Function:**

*function* Nameoffunction() {

  // body of the function

  // Start of Curly braces indicate the body of the function

  console.log('Hi');

}

**Invoking the function/Calling the function/Running the function:**

// Calling the function

Nameoffunction();

Since declaration of function has no parameters hence no arguments are required to be passed while calling the function.

Not all function need to return something and not all functions need to accept all parameters.

**Passing Data/Parameters into Functions:**

Parameters are like variables that are specific to only the function in which they are being passed.

Parameters represent the input data of the function in which they are being passed into.

Parameters will get defined on calling the function.

// Passing parameters/data to the name\_of\_function function

*function* name\_of\_function(*paramerter\_one*, *paramerter\_two*) {

  console.log(`Your first parameter is ${*paramerter\_one*} and your second parameter is

  ${*paramerter\_two*}`);

}

**Function Returning Data:**

The return value from the function can be used anywhere in code.

// Passing parameters/data to the name\_of\_function function

*function* name\_of\_function(*paramerter\_one*, *paramerter\_two*) {

*const* variable = *paramerter\_one* + *paramerter\_two*;

  // Returing the data from the function

  return variable;

}

**Calling the function:**

Arguments are the actual values of the function parameters.

*const* first\_argument = *Number*(prompt('Enter your first argument'));

*const* second\_argument = *Number*(prompt('Enter your second argument'));

// Calling the function

// Passing arguments to the function

// name\_of\_function(first\_argument, second\_argument);

// The data returned by the function must be printed

// first parameter and second\_parameter are the inputs of the  name\_of\_function

console.log(name\_of\_function(first\_argument, second\_argument));

Calling a function is necessary for the code in the function to be executed.





Console.log is a built-in function which is called like our own function, we call in using () and inside () we pass in the argument.

Result of calling console.log function is the message being printed on the console.

**Number:**

*Number*()

Number is another built in function in which we pass in the argument (usually string) , Number function is executed and converts string into number

*const* string = '123';

*const* number = *Number*(string);

The value returned by the built in Number function can be saved in any variable.

**Function Expression/Anonymous Function**

Function keyword without name

Function without a name

Define the parameter inside ()

Define function body.

*function* (*parameter\_one*,*parameter\_two*)

// Function body

{

}

Store all the function inside a variable

*const* function\_storing\_variable=*function* (*parameter\_one*,*parameter\_two*)

// Function body

{

}

The variable that is storing the function expression will then be the function.

Expression produces a value which will be stored inside the function\_storing variable.

// age\_calculate\_anynymous is a variable storing the value returned by  the anynynous function

*const* age\_calculate\_anynymous = *function* (

*birthYear\_parameter*,

*currentYear\_parameter*

) {

  // function body with {}

  if (birthYear === 0 || birthYear < 0 || birthYear > currentYear) {

    return 'Invalid Birth Year';

  } else {

*const* age = currentYear - birthYear;

    // To take the value of age out of the age\_calculate function

    // we return the value using return key word

    return age;

  }

};

**Calling an Anonymous Function**

*const* birth\_year\_anonymous\_function\_argument = *Number*(

  prompt('Enter Birth Year Argument')

);

*const* current\_year\_anonymous\_function\_argument = *Number*(

  prompt('Enter Current Year')

);

// Calling a anynymous function

console.log(

  age\_calculate\_anynymous(

    birth\_year\_anonymous\_function\_argument,

    current\_year\_anonymous\_function\_argument

  )

);

**Overall:**

// age\_calculate\_anynymous is a variable storing the value returned by  the anynynous function

*const* age\_calculate\_anynymous = *function* (

*birthYear\_parameter*,

*currentYear\_parameter*

) {

  // function body with {}

  if (birthYear === 0 || birthYear < 0 || birthYear > currentYear) {

    return 'Invalid Birth Year';

  } else {

*const* age = currentYear - birthYear;

    // To take the value of age out of the age\_calculate function

    // we return the value using return key word

    return age;

  }

};

*const* birth\_year\_anonymous\_function\_argument = *Number*(

  prompt('Enter Birth Year Argument')

);

*const* current\_year\_anonymous\_function\_argument = *Number*(

  prompt('Enter Current Year')

);

// Calling a anynymous function

console.log(

  age\_calculate\_anynymous(

    birth\_year\_anonymous\_function\_argument,

    current\_year\_anonymous\_function\_argument

  )

);

The return value of function expressions can be stored in variables,

|  |  |
| --- | --- |
| **Function Declaration** | **Function Expression** |
| Call function declaration before they are declared/defined. | Call function expression after they are declared/defined. |

**Arrow Functions:**

Specia form of function expression.

Shorter and faster to write.

Arrow function can be stored in the variable.

// Arrow function

*const* variable\_storing\_arrow\_function=(*parameter\_one*,*parameter\_two*)*=>*expression

All the value returned by **(Single line arrow function)**

(birth\_year,current\_year) *=>*   birth\_year-current\_year;

is stored inside a variable age\_calculate.

*const* age\_calculate=(birth\_year,current\_year) *=>*   birth\_year-current\_year;

The value of the expression after => will automatically be returned without us specifying the return keyword.

**Calling arrow function:**

// Calling arrow function

// and

// Saving the returned value by arrow function in variable

*const* result\_age = age\_calculate(current\_year\_argument, birth\_year\_argument);

console.log(result\_age);

**Overall:**

*const* age\_calculate = (*birth\_year\_parameter*, *current\_year\_parameter*) *=>*

*birth\_year\_parameter* - *current\_year\_parameter*;

*const* current\_year\_argument = *Number*(prompt('Enter current year'));

*const* birth\_year\_argument = *Number*(prompt('Enter current birth'));

// Calling arrow function

// and

// Saving the returned value by arrow function in variable

*const* result\_age = age\_calculate(current\_year\_argument, birth\_year\_argument);

console.log(result\_age);

Another Example (Multiple line arrow function):

*const* result\_years\_left = (

*parameter\_birth\_year*,

*parameter\_current\_year*,

*retirement\_age\_parameter*

) *=>* {

*const* age = *parameter\_current\_year* - *parameter\_birth\_year*;

  // In case of more than one line arrow function we have use return key word

  return *retirement\_age\_parameter* - age;

};

*const* birth\_Year\_argument = *Number*(prompt('Enter birth year'));

*const* current\_Year\_argument = *Number*(prompt('Enter current year'));

*const* retirement\_age\_argument = *Number*(prompt('Enter retirement age'));

// Capturing the return value from the arrow function and console logging it

console.log(

  result\_years\_left(

    birth\_Year\_argument,

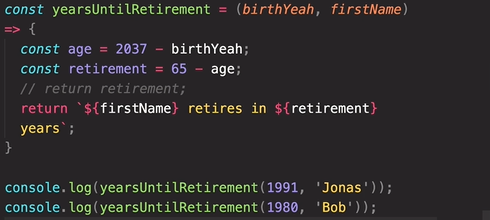
    current\_Year\_argument,

    retirement\_age\_argument

  )

);

**Another Example:**





**Calling One Function inside another function:**

**Function one:**

*function* function\_one(*function\_one\_paramaeter\_one*) {

  return *function\_one\_paramaeter\_one*;

}

**Function two:**

*function* function\_two(

*function\_two\_paramaeter\_one*,

*function\_two\_paramaeter\_two*

      ) {

        console.log(*function\_two\_paramaeter\_one*, *function\_two\_paramaeter\_two*);

        // The return value of the function\_one will be in below statement

       // hence we are console logging it console.log(function\_one(*function\_two\_paramaeter\_two*));

     // Calling function\_one inside function\_two and

    //sending function\_two\_paramaeter\_one as an argument

        console.log(function\_one(*function\_two\_paramaeter\_one*));

        return function\_one(*function\_two\_paramaeter\_one*);

      }

**Inputs:**

*const* function\_two\_argument\_one = prompt(

        'Enter first argumment of function two'

      );

*const* function\_two\_argument\_two = prompt(

        'Enter second argumment of function two'

      );

**Calling Function Two(No Console logging function two):**

// calling function\_two will cause the console.log statement of functrion\_two

      //  be executed

      function\_two(function\_two\_argument\_one, function\_two\_argument\_two);

**Code inside the Function two is executed:**

**Let us assume**

function\_two\_argument\_one

**be 1**

**and**

function\_two\_argument\_two

**be 2**

 console.log(*function\_two\_paramaeter\_one*, *function\_two\_paramaeter\_two*);



// The return value of the function\_one will be in below statementhence we are console logging it

 console.log(function\_one(*function\_two\_paramaeter\_two*));



// Calling function\_one inside function\_two and

        //sending function\_two\_paramaeter\_one as an argument

console.log(function\_one(*function\_two\_paramaeter\_one*));



 // The returned value will be printed using console.log and calling function two

return function\_one(*function\_two\_paramaeter\_one*);

**Console logging function two:**

      // Console logging function\_two

      // Following code will cause console.log statement inside the function\_two function to be executed

      console.log(

        function\_two(function\_two\_argument\_one, function\_two\_argument\_two)

      );

console.log(*function\_two\_paramaeter\_one*, *function\_two\_paramaeter\_two*);



        console.log(function\_one(*function\_two\_paramaeter\_two*));



        // Calling function\_one inside function\_two and        //sending function\_two\_paramaeter\_one as an argument

 console.log(function\_one(*function\_two\_paramaeter\_one*));



**Return value of function two is printed:**

return function\_one(*function\_two\_paramaeter\_one*);

 console.log(

        function\_two(function\_two\_argument\_one, function\_two\_argument\_two)

      );



**Alternate Example:**

*const* fruit\_one = *Number*(prompt('Enter the amount of first fruit'));

*const* fruit\_two = *Number*(prompt('Enter the amount of second fruit'));

*const* fruit\_pieces = *Number*(prompt('Enter number of pieces to be cut'));

*function* foodprocessor(*p\_one*, *p\_two*) {

        console.log(

          'First Fruit Pieces' + cut\_fruit\_pieces(fruit\_one, fruit\_pieces)

        );

        console.log(

          'Second Fruit Pieces' + cut\_fruit\_pieces(fruit\_two, fruit\_pieces)

        );

      }

      // Calling foodprocessor function with two arguments

      // fruit\_one and fruit\_two will replace the p\_one parameter and p\_two parameter

      // with fruit\_one and fruit\_two

      foodprocessor(fruit\_one, fruit\_two);

*function* cut\_fruit\_pieces(*fruit\_one*, *pieces*) {

        return *fruit\_one* \* *pieces*;

      }

**Flow of data between these functions:**

Calling the food processor function with the arguments fruit\_one and fruit\_two.The arguments will replace the parameters p\_one and p\_two in the function foodprocessor with the arguments fruit\_one and fruit\_two

 // Calling foodprocessor function with two arguments

      // fruit\_one and fruit\_two will replace the p\_one parameter and p\_two parameter

      // with fruit\_one and fruit\_two

      foodprocessor(fruit\_one, fruit\_two);

*function* foodprocessor(*p\_one*, *p\_two*) {

        console.log(

          'First Fruit Pieces' + cut\_fruit\_pieces(fruit\_one, fruit\_pieces)

        );

        console.log(

          'Second Fruit Pieces' + cut\_fruit\_pieces(fruit\_two, fruit\_pieces)

        );

      }

p\_one and p\_two parameters of the foodprocessor will be used to call the cut\_fruit \_pieces function.

cut\_fruit\_pieces(fruit\_one, fruit\_pieces)

fruit\_one and fruit\_pieces is the argument for the cut\_fruit\_pieces function.

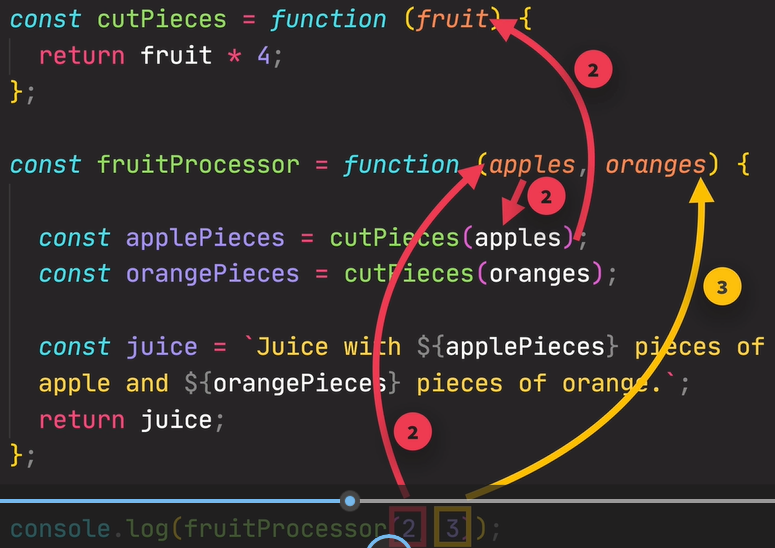
The fruit\_one and fruit\_pieces arguments will replace the fruit\_one and pieces parameters.

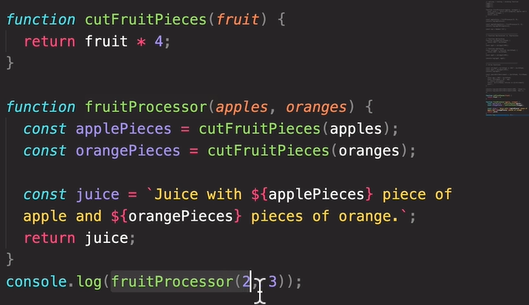
*function* cut\_fruit\_pieces(*fruit\_one*, *pieces*) {

        return *fruit\_one* \* *pieces*;

      }

**Arrow representation of data flow**





Result of console.log(fruitProcessor(2,3)) will cause the string juice which is being returned to be printed.

**Reviewing Functions**

**Function Expressions**

*const* age\_calculation = *function* (*birthYear\_parameter*, *currentYear\_parameter*) {

*const* age = *currentYear\_parameter* - *birthYear\_parameter*;

  // To take the value of age out of the age\_calculate function

  // we return the value using return key word

  return age;

};

// age\_calculate\_anynymous is a variable storing the value returned by  the anynynous function

*const* age\_calculate\_anynymous = *function* (

*birthYear\_parameter*,

*currentYear\_parameter*

) {

  // function body with {}

  if (

*birthYear\_parameter* === 0 ||

*birthYear\_parameter* < 0 ||

*birthYear\_parameter* > *currentYear\_parameter*

  ) {

    return 'Invalid Birth Year';

  } else {

    // Returning and Calling a anynymous function

    return age\_calculation(

      birth\_year\_anonymous\_function\_argument,

      current\_year\_anonymous\_function\_argument

    );

  }

};

*const* birth\_year\_anonymous\_function\_argument = *Number*(

  prompt('Enter Birth Year Argument')

);

*const* current\_year\_anonymous\_function\_argument = *Number*(

  prompt('Enter Current Year')

);

// Calling a anynymous function

console.log(

  age\_calculate\_anynymous(

    birth\_year\_anonymous\_function\_argument,

    current\_year\_anonymous\_function\_argument

  )

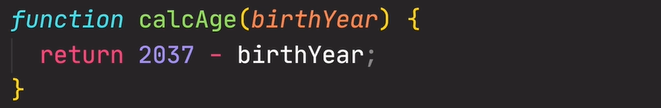
);

Once the value is returned by the function, the function execution is done, any code written after the return statement will get ignored and will not get executed. The return statement immediately exits the function.

**Three Type of Function**

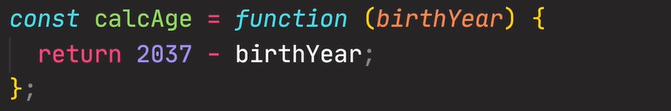
**Function Declaration:**

Can be used before they are declared in the code.



**Function Expressions:**

Function values stored in a variable.

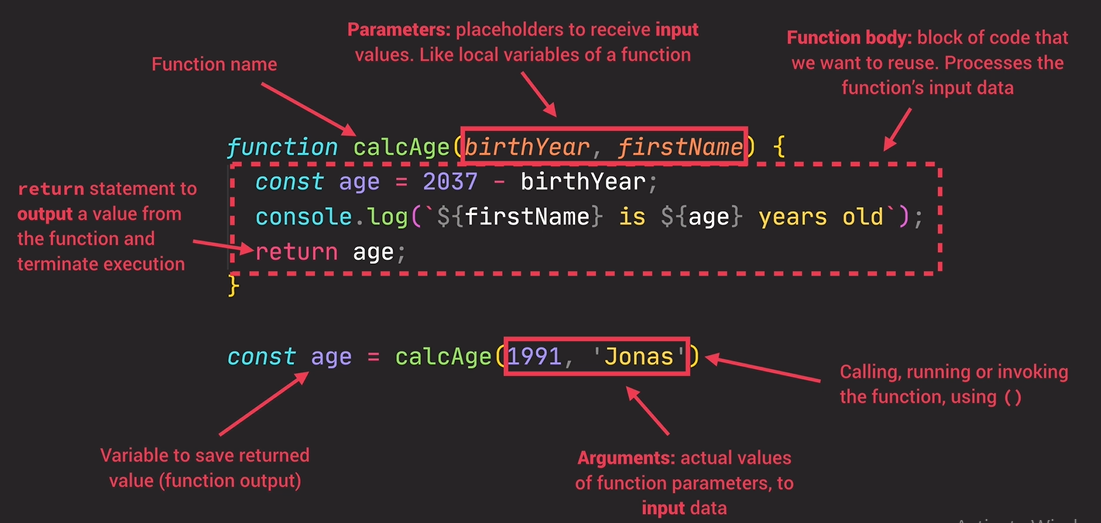


**Arrow Functions:**

Special Function expressions



We call the function using () and inside () we put arguments.



**Arrays**

Big Container in which we can throw variables and then later reference them.

**Literal syntax [] of creating arrays**

We use [] to create an array.

*const* name\_of\_arrays = [];

In array we can put different values separated by comma.

*const* name\_of\_arrays = ['Wilson', 'Michael', 'House', 'James', 'Gregory'];

**Alternate Way to create arrays:**

Creating array using Array function Array()

*const* name\_of\_arrays\_two=new *Array*()

which is called using ().The use of new keyword is essential.

**Inside () we specify the values.**

**Length property of array gives number of elements in array:**

.length is the property of array

name\_of\_arrays\_two.length

**To get the index of last element of array:**

console.log(name\_of\_arrays[name\_of\_arrays.length - 1]);

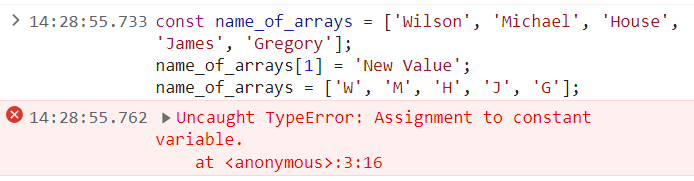
**Replacing elements in Array**

*const* name\_of\_arrays = ['Wilson', 'Michael', 'House', 'James', 'Gregory'];

name\_of\_arrays[1] = 'New Value'**;**

Since array is not a primitive value , hence we can change it although name\_of\_arrays is declared with const.

But we cannot assign a new array to existing name\_arrays which is declared constant.



In square brackets of arrays, in each position Java scripts expects an expression.

*const* current\_year = *Number*(prompt('Enter current year'));

*const* birth\_year = *Number*(prompt('Enter Birth year'));

*const* name\_of\_array\_three = ['Asad', 'Khalil', current\_year - birth\_year];

current\_year - birth\_year

Will produce a value that will then be stored in the array.

**Putting array inside an array:**

*const* current\_year = *Number*(prompt('Enter current year'));

*const* birth\_year = *Number*(prompt('Enter Birth year'));

*const* name\_of\_array\_three\_sub = ['A', 'B', 'C'];

*const* name\_of\_array\_three = [

  'Asad',

  'Khalil',

  current\_year - birth\_year,

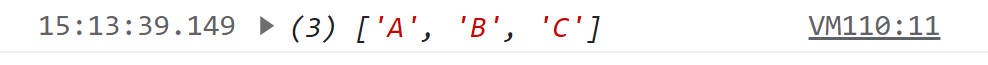
  true,

  name\_of\_array\_three\_sub,

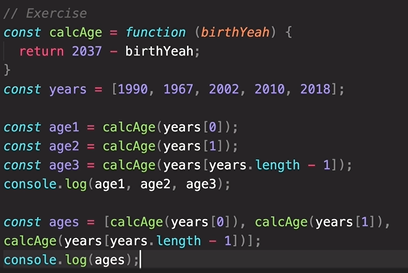
];

console.log(name\_of\_array\_three[name\_of\_array\_three.length - 1]);

name\_of\_array\_three\_sub is an other array inside the array name\_of\_array\_three.



We can also place a function call inside arrays:





**My attempt:**

*const* number\_of\_previous\_years = *Number*(prompt('Enter number of past years'));

*const* current\_entered = *Number*(prompt('Enter current year'));

*const* calcAge = *function* (*years*) {

  console.log(*years*);

  for (*let* index = 0; index < *years*.length; index++) {

    console.log(

      'The age of user in  ' +

*years*[index] +

        '  is ' +

        (current\_entered - *years*[index])

    );

  }

};

*const* years = [];

for (*let* index = 0; index < number\_of\_previous\_years; index++) {

*const* years\_entered = *Number*(prompt('Enter past years'));

  years[index] = years\_entered;

}

console.log(years);

console.log(calcAge(years));

**Built in Function to Apply on Arrays**

These are called methods (array operations).

**Push Method:**

Adds element to the end of array.

// Push method

*const* name\_of\_array = ['A', 'B', 'C', 'D'];

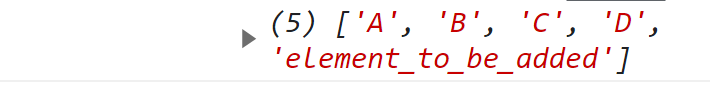
// Push method

name\_of\_array.push('element\_to\_be\_added');

console.log(name\_of\_array);

Push is a method which technically is a function. We call that function attached to the name\_of\_array array itself using dot.

And element\_to\_be\_added is added inside () as an argument to the push method.



The value returned by the push function is the length of the new array.

**Capturing the value returned by the push method:**

// Push method

*const* name\_of\_array = ['A', 'B', 'C', 'D'];

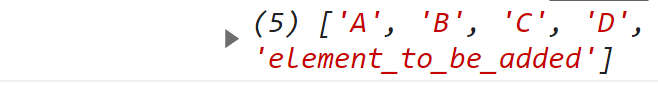
// Capturing the value returned by push method

*const* value\_returned\_by\_push\_method = name\_of\_array.push('element\_to\_be\_added');

console.log(value\_returned\_by\_push\_method);

console.log(name\_of\_array);





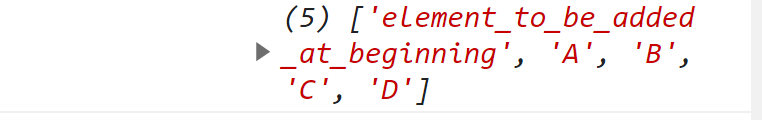
**Unshift Method:**

Add elements in the beginner of the array.

*const* name\_of\_array = ['A', 'B', 'C', 'D'];

name\_of\_array.unshift('element\_to\_be\_added\_at\_beginning');

console.log(name\_of\_array);



Unshift method also returns the length of the new array.

**Pop Method:**

Removes the last element of the array.

*const* name\_of\_array = ['A', 'B', 'C', 'D'];

name\_of\_array.pop();

console.log(name\_of\_array);



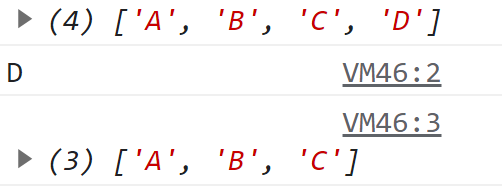
Pop method returns the removed element.

*const* name\_of\_array = ['A', 'B', 'C', 'D'];

console.log(name\_of\_array);

console.log(name\_of\_array.pop());

console.log(name\_of\_array);



**Shift Method:**

**To remove the first element from the array use unshift method**

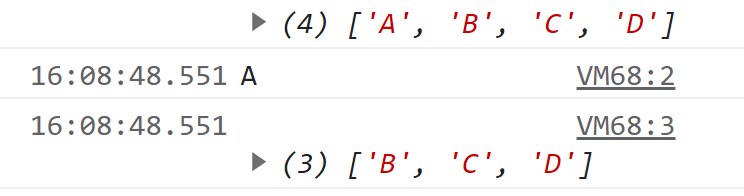
*const* name\_of\_array = ['A', 'B', 'C', 'D'];

console.log(name\_of\_array);

console.log(name\_of\_array.shift());

console.log(name\_of\_array)

**Shift method returns the first element to be removed.Shift method removes the first element of the array.**



**Index of method:**

Index of method tells us the position of element.

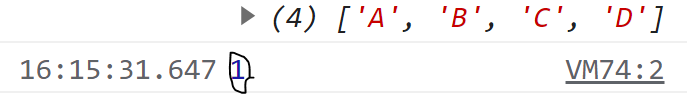
Inside the () we enter the element whose position is to be found.

IndexOf method returns the index of element which is mentioned in the name\_of\_array.

*const* name\_of\_array = ['A', 'B', 'C', 'D'];

console.log(name\_of\_array);

console.log(name\_of\_array.indexOf('B'));



If we try to find the index of an element which is not in the name\_of\_array, we will get **-1.**

**Includes method:**

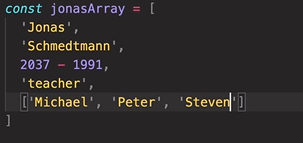
**Returns true if the element is in the array and false if the element is not in the array.**

**Uses strict equality for the check.**

**Does not do type coercion**

**Used to write conditionals in if and else if conditionals.**

**We can have array with in an array.**



**Objects:**

Creating objects using curly braces is called object literals syntax.

We write objects to group together different variables such as properties of object.

*const* name\_of\_object={}

In objects we define key value pairs.

Key is basically the variable name.

Value can be of any type we want.

*const* name\_of\_object = {

  key\_one: 'value',

  key\_two: 2,

  key\_three: true,

  key\_four: false,

object:value\_of\_object

};

**Example:**

An object of five keys/properties value pairs.

*const* student = {

  name: 'A',

  gender: 'male',

  roll\_number: 12345567,

  present: true,

  grade: 'Alpha',

  age: 2023-2001,

  friends: ['B', 'C', 'D', 'E', 'F', 'G'],

};

Each of the keys is also called the property.

**Difference in Object and Arrays**

|  |  |
| --- | --- |
| **Objects** | **Arrays** |
| Order of key/properties values does not matter when we want to retrieve them.  Use objects for unstructured data | Order in which we specify the elements matters because that is how we access the elements.  We can access array element using their order number.  Use arrays for unstructured data. |

**Getting Data from Object:**

Getting property from the objects is by using dot notations.

*const* student = {

  name: 'A',

  gender: 'male',

  roll\_number: 12345567,

  present: true,

  grade: 'Alpha',

  age: 20,

  friends: ['B', 'C', 'D', 'E', 'F', 'G'],

};

console.log(student.friends[5]);

nameofobject.nameofproperty/key

**Dot:**

In above code dot is an operator which will go into name of object and retrieve the value of the property (property specified after dot).

In dot notation we use real final property name to access the value of the object.

**Using Brackets Notations**

nameofobject["nameofproperty"]

**inside name of property must be specified in string format.**

*const* student = {

  name: 'A',

  gender: 'male',

  roll\_number: 12345567,

  present: true,

  grade: 'Alpha',

  age: 20,

  friends: ['B', 'C', 'D', 'E', 'F', 'G'],

};

console.log(student["roll\_number"]);



In bracket notation we can put any expression we like in brackets.

We can compute the property names that we want to access.



Both dot and bracket notation have high priority and both are executed from left to right.

*const* student = {

  firstName: 'John',

  lastName: 'Crane',

  name: 'A',

  gender: 'male',

  roll\_number: 12345567,

  present: true,

  grade: 'Alpha',

  age: 20,

  friends: ['B', 'C', 'D', 'E', 'F', 'G'],

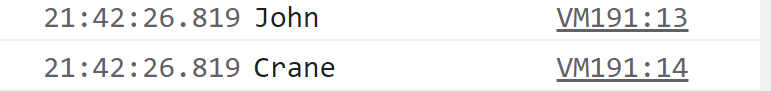
};

*const* property\_name = 'Name';

console.log(student['first' + property\_name]);

console.log(student['last' + property\_name]);

JavaScript will see the + operator and it will create the string firstName and then it will go to the student object and retrieve the value of the property.



In the square brackets we can put any expression we like.

When we need to compute the property name, we use bracket notation.

Prompt is built in function that we can use in JavaScript.

When we try to access a property on an object that does not exist, we get undefined.

*const* student = {

  firstName: 'John',

  lastName: 'Crane',

  name: 'A',

  gender: 'male',

  roll\_number: 12345567,

  present: true,

  grade: 'Alpha',

  age: 20,

  friends: ['B', 'C', 'D', 'E', 'F', 'G'],

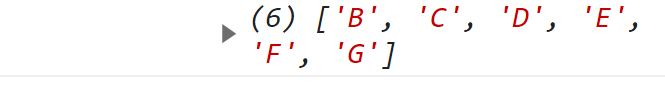
};

*const* interested\_In = prompt(

  'What do you want to know about me?Choose firstName,lastName,age,job and friends?'

);

console.log(student[interested\_In]);



**Undefined is a falsee value:**

It can be converted to false value and can be use in if else condition

*const* student = {

  firstName: 'John',

  lastName: 'Crane',

  name: 'A',

  gender: 'male',

  roll\_number: 12345567,

  present: true,

  grade: 'Alpha',

  age: 20,

  friends: ['B', 'C', 'D', 'E', 'F', 'G'],

};

*const* interested\_In = prompt(

  'What do you want to know about me?Choose firstName,lastName,age,job and friends?'

);

if (student[interested\_In]) {

  console.log(student[interested\_In]);

} else {

  console.log('Accessing Invalid Property');

}



Whenever the use tries to access the property that does not exist, else block is executed.

**Adding new properties to an Object:**

**Using Dot Noation**

name\_of\_object.new\_propery = 'some\_value';

**Example:**

*const* student = {

  firstName: 'John',

  lastName: 'Crane',

  name: 'A',

  gender: 'male',

  roll\_number: 12345567,

  present: true,

  grade: 'Alpha',

  age: 20,

  friends: ['B', 'C', 'D', 'E', 'F', 'G'],

};

student.location = 'Portugal';

console.log(student.location);



**Using bracket notation to add property to an Object:**

name\_of\_object['new\_property'] = 'new\_value';

*const* student = {

  firstName: 'John',

  lastName: 'Crane',

  name: 'A',

  gender: 'male',

  roll\_number: 12345567,

  present: true,

  grade: 'Alpha',

  age: 20,

  friends: ['B', 'C', 'D', 'E', 'F', 'G'],

};

student['email'] = '@student.com';

console.log(student["email"]);



**Challenge:**

*const* student = {

  firstName: 'John',

  lastName: 'Crane',

  name: 'A',

  gender: 'male',

  roll\_number: 12345567,

  present: true,

  grade: 'Alpha',

  age: 20,

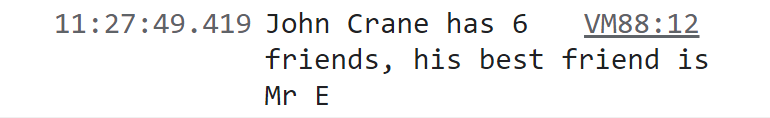
  friends: ['B', 'C', 'D', 'E', 'F', 'G'],

};

console.log(

  `${student.firstName} ${student.lastName} has ${student.friends.length} friends, his best friend is Mr ${student.friends[3]}`

);



**Object Methods:**

Key/property: value is the function.

Adding a function inside object as key value pair

name of the function as key/property

Writing function expression as value of the key/property

*const* student = {

  firstName: 'John',

  lastName: 'Crane',

  name: 'A',

  gender: 'male',

  roll\_number: 12345567,

  present: true,

  grade: 'Alpha',

  friends: ['B', 'C', 'D', 'E', 'F', 'G'],

  has\_driver\_liscense: true,

  // Adding a function inside object as key value pair

  // name of the function as key/property

  // Writing function expression as value of the key/property

  age\_calculation: *function* (*birth\_year*, *current\_year*) {

    if ((*current\_year* > 0) & (*current\_year* > *birth\_year*)) {

      return *current\_year* - *birth\_year*;

    } else {

      return 'Invalid Birth Year';

    }

  },

};

*const* birth\_year = *Number*(prompt('Enter Birth Year'));

*const* current\_year = *Number*(prompt('Enter Current Year'));

console.log(student.age\_calculation(birth\_year, current\_year));

Any function that is attached to an object is called method.

*const* student = {

  firstName: 'John',

  lastName: 'Crane',

  name: 'A',

  gender: 'male',

  roll\_number: 12345567,

  present: true,

  grade: 'Alpha',

  friends: ['B', 'C', 'D', 'E', 'F', 'G'],

  has\_driver\_liscense: true,

  // Adding a function inside object as key value pair

  // name of the function as key/property

  // Writing function expression as value of the key/property

  age\_calculation: *function* (*birth\_year*, *current\_year*) {

    if ((*current\_year* > 0) & (*current\_year* > *birth\_year*)) {

      return *current\_year* - *birth\_year*;

    } else {

      return 'Invalid Birth Year';

    }

  },

};

*const* birth\_year = *Number*(prompt('Enter Birth Year'));

*const* current\_year = *Number*(prompt('Enter Current Year'));

console.log(student.age\_calculation(birth\_year, current\_year));

Using function declaration instead of function expression may cause error.

If functions are considered as simply being values, then methods can be considered as property. Method happens to be a property that happens to hold function value.

**Calling Method Using Bracket Notation**

// Accesiung the method using bracket notation

console.log(student['age\_calculation'](birth\_year, current\_year));

**nameofObject[‘nameofmethod’](argument\_one,argument\_two))**

We access the method using [],we call the method using () and pass in the argument.

*const* student = {

  firstName: 'John',

  lastName: 'Crane',

  name: 'A',

  birth\_year: 2001,

  current\_year: 2023,

  gender: 'male',

  roll\_number: 12345567,

  present: true,

  grade: 'Alpha',

  friends: ['B', 'C', 'D', 'E', 'F', 'G'],

  has\_driver\_liscense: true,

  // Adding a function inside object as key value pair

  // name of the function as key/property

  // Writing function expression as value of the key/property

  age\_calculation: *function* (*birth\_year*, *current\_year*) {

    if ((*current\_year* > 0) & (*current\_year* > *birth\_year*)) {

      return *current\_year* - *birth\_year*;

    } else {

      return 'Invalid Birth Year';

    }

  },

};

*const* birth\_year = *Number*(prompt('Enter Birth Year'));

student['year\_of\_birth'] = birth\_year;

*const* current\_year = *Number*(prompt('Enter Current Year'));

student['present\_year'] = current\_year;

// Using the property of the student object as an argument for the age\_calculation  method

console.log(

  student['age\_calculation'](student['year\_of\_birth'], student['present\_year'])

);

**Special Variable :This**

In every method JavaScript gives us access to a special variable called **this**.

This key word allows us to access the value of the property directly from the object in every method.

**Reading the property directly from the object**

This keyword is basically equal to the object on which the method is called.

This key word is equal to the object called the method.

console.log(student.age\_calculation(birth\_year, current\_year));

student object is calling the age\_calculation method.

So inside age\_calculation, this keyword will point to object student.

*const* student = {

  firstName: 'John',

  lastName: 'Crane',

  name: 'A',

  birth\_year: 2001,

  current\_year: 2023,

  gender: 'male',

  roll\_number: 12345567,

  present: true,

  grade: 'Alpha',

  friends: ['B', 'C', 'D', 'E', 'F', 'G'],

  has\_driver\_liscense: true,

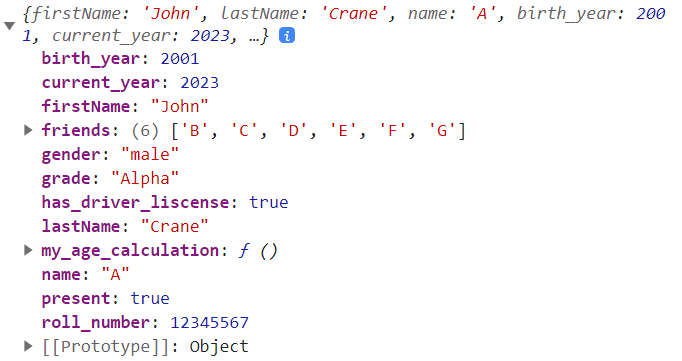
my\_age\_calculation: *function* () {

   console.log(this)

  },

};

console.log(student.my\_age\_calculation());



Since student object is calling the **my\_age\_calculation** **method** so **inside** **my\_age\_calcualtion** **method** **this** keyword **will point towards** the **student object**

**Using This Key Word:**

*const* student = {

  firstName: 'John',

  lastName: 'Crane',

  name: 'A',

  birth\_year: 2001,

  current\_year: 2023,

  gender: 'male',

  roll\_number: 12345567,

  present: true,

  grade: 'Alpha',

  friends: ['B', 'C', 'D', 'E', 'F', 'G'],

  has\_driver\_liscense: true,

my\_age\_calculation: *function* () {

    console.log(this);

    if ((this.current\_year > 0) & (this.current\_year > this.birth\_year)) {

      return this.current\_year - this.birth\_year;

    } else {

      return 'Invalid Birth Year';

    }

  },

};

*const* birth\_year = *Number*(prompt('Enter Birth Year'));

student['year\_of\_birth'] = birth\_year;

*const* current\_year = *Number*(prompt('Enter Current Year'));

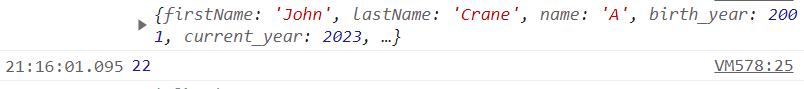
student['present\_year'] = current\_year;

// Accesing method of the object using dot

// Using () to call the age\_calculation method

//Since student object is calling my\_age\_calculation method hence this key word in my\_age\_calculation will point to student object

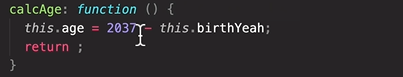
console.log(student.my\_age\_calculation());



**Creating new property on current object inside a method using this key word/using this key word to store new property**

**Syntax**

this.new\_property= expression



The value of the expression is stored inside the newly created property using this key word.

*const* student = {

  firstName: 'John',

  lastName: 'Crane',

  name: 'A',

  birth\_year: 2001,

  current\_year: 2023,

  gender: 'male',

  roll\_number: 12345567,

  present: true,

  grade: 'Alpha',

  friends: ['B', 'C', 'D', 'E', 'F', 'G'],

  has\_driver\_liscense: true,

 age\_calculation\_using\_new\_property\_this\_key\_word: *function* () {

    if ((this.current\_year > 0) & (this.current\_year > this.birth\_year)) {

      // new\_age\_property is a newly created property using this key word

      //this newly created property is attached to student object as

      //student object is calling the age\_calculation\_using\_new\_property\_this\_key\_word, hence this key word

      //points to the student object

      return this.new\_age\_property = this.current\_year - this.birth\_year;

    } else {

      return 'Invalid Birth Year';

    }

  },

};

*const* birth\_year = *Number*(prompt('Enter Birth Year'));

student['year\_of\_birth'] = birth\_year;

*const* current\_year = *Number*(prompt('Enter Current Year'));

student['present\_year'] = current\_year;

// Calling age\_calculation\_using\_new\_property\_this\_key\_word only once

student.age\_calculation\_using\_new\_property\_this\_key\_word();

// Retriving the value of the property new\_age\_property

console.log(student.new\_age\_property);

// Retriving the value of the property new\_age\_property

console.log(student.new\_age\_property);

**Challenge:**

**Create a method called summary and fetch all the data in that method**

*const* student = {

  firstName: 'John',

  lastName: 'Crane',

  name: 'A',

  birth\_year: 2001,

  current\_year: 2023,

  gender: 'male',

  roll\_number: 12345567,

  present: true,

  grade: 'Alpha',

  friends: ['B', 'C', 'D', 'E', 'F', 'G'],

  has\_driver\_liscense: true,my\_age\_calculation: *function* () {

    console.log(this);

    if ((this.current\_year > 0) & (this.current\_year > this.birth\_year)) {

      return this.current\_year - this.birth\_year;

    } else {

      return 'Invalid Birth Year';

    }

  },

  get\_summary: *function* () {

    return (this.overall\_summary = `${this.firstName}  ${this.lastName} was born in ${this.birth\_year}  he is ${this.my\_age\_calculation()} years old in ${current\_year} , he lives in. ${this.location}`);

  },

};

*const* birth\_year = *Number*(prompt('Enter Birth Year'));

student['year\_of\_birth'] = birth\_year;

*const* current\_year = *Number*(prompt('Enter Current Year'));

student['present\_year'] = current\_year;

*const* student\_location = prompt('Enter Location ');

student['location'] = student\_location; // Calling get\_summary only once

student.get\_summary();

// Retriving the value of the property overall\_summary

console.log(student.overall\_summary);

**Arrays:**

Arrays are special kind of objects, they have functions, in other words they have built in methods like push, pop, shift, etc, methods we can use to manipulate arrays.

**Challenge**

// CHALLENGE

*const* mark\_miller = {

  first\_name: 'Mark',

  last\_name: 'Miller',

  mass: '',

  height: '',

  caclBMI: *function* () {

    return (this.bmi\_mark\_miller = this.mass / (this.height \* this.height));

  },

};

*const* john\_smith = {

  first\_name: 'John',

  last\_name: 'Smith',

  mass: '',

  height: '',

  caclBMI: *function* () {

    console.log(this.mass, this.height);

    return (this.bmi\_john\_smith = this.mass / (this.height \* this.height));

  },

};

*const* mark\_mass = *Number*(prompt("Enter Marks's Mass"));

*const* mark\_height = *Number*(prompt("Enter Marks's Height"));

*const* john\_mass = *Number*(prompt("Enter John's Mass"));

*const* john\_height = *Number*(prompt("Enter John's Height"));

// GIVING VALUES TO THE PROPERTIES

mark\_miller['mass'] = mark\_mass;

mark\_miller['height'] = mark\_height;

john\_smith['mass'] = john\_mass;

john\_smith['height'] = john\_height;

// john\_smith.caclBMI method call is essential as without this method call

// bmi\_john\_smith would not exist, we need to explicitely call it

john\_smith.caclBMI();

// mark\_miller.caclBMI(); method call is essential as without this method call mark\_miller.bmi\_mark\_miller would not exist,we need to explicitely call it

mark\_miller.caclBMI();

if (john\_smith.bmi\_john\_smith > mark\_miller.bmi\_mark\_miller) {

  console.log(

    `${john\_smith.first\_name} ${john\_smith.last\_name}'s   BMI (${john\_smith.bmi\_john\_smith}) is higher than ${mark\_miller.first\_name} ${mark\_miller.last\_name} 's (${mark\_miller.bmi\_mark\_miller})`

  );

} else {

  console.log(

    `${mark\_miller.first\_name} ${mark\_miller.last\_name}'s BMI (${john\_smith.bmi\_john\_smith}) is higher than ${john\_smith.first\_name} ${john\_smith.last\_name} 's (${john\_smith.bmi\_john\_smith})`

  );

}

**Loops:**

**Loop statement has three parts.**

**1)Initial value of counter**

*let* index = 0

2)**Logical Condition** (Evaluated before each iteration of the loop, before each time code in the loop is executed)

If the condition is true, then the next loop iteration will run.

As soon as the logical condition is false, then the loop stops. Loop will keep running as long as the condition is true.

index < 10

**Update the Counter:**

Update the counter after each iteration.

index++

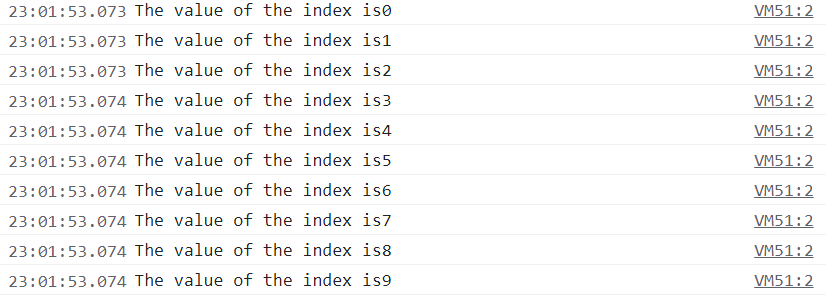
Code inside for loop will be repeated till the condition is false.

Example:

for (*let* index = 0; index < 10; index++) {

  console.log(`The value of the index is ${index}`);

}



for (*let* index = 11; index < 10; index++) {

  console.log(`The value of the index is ${index}`);

}

console.log('Out of the for loop');



For loop verifies before each repetition if the condition in logical part still holds true.

Since in above code the for loop verifies the logical condition before executing the code inside the for loop and since the condition is false hence code inside for loop is not executed.

**Applications of For Loops:**

*const* name\_array = [

  'Asad',

  'Khalil',

  2023 - 2001,

  'teacher',

  ['Michael', 'Peter', 'Steven'],

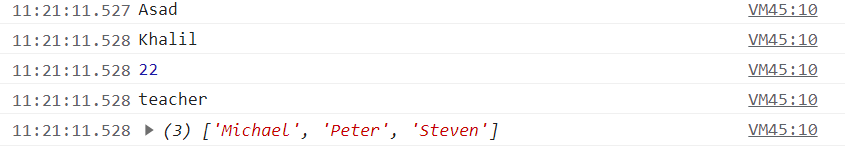
];

for (*let* index = 0; index < name\_array.length; index++) {

*const* element = name\_array[index];

  console.log(element);

}



name\_array.length is dynamically calculated value of array.

*const* name\_array = [

  'Asad',

  'Khalil',

  2023 - 2001,

  'teacher',

  ['Michael', 'Peter', 'Steven'],

];

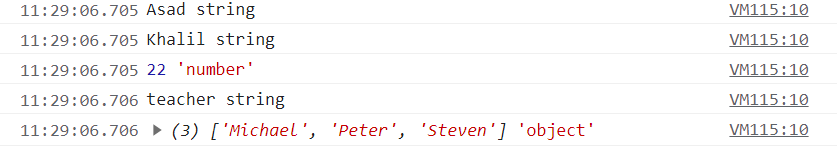
for (*let* index = 0; index < name\_array.length; index++) {

*const* element = name\_array[index];

// Reading elements from name\_array

  console.log(element,typeof(element));

}



**Adding Elements in Array**

*const* name\_array = [

  'Asad',

  'Khalil',

  2023 - 2001,

  'teacher',

  ['Michael', 'Peter', 'Steven'],

];

// Adding Elements in array

*const* empty\_array = [];

for (*let* index = 0; index < name\_array.length; index++) {

  // Adding the typeof elements of name\_array in empty\_array

  empty\_array.push(typeof name\_array[index]);

  console.log(empty\_array[index]);

} **Adding Ages according to Birth Years**

*const* birth\_years = [1991, 2007, 1969, 2020];

*const* empty\_new\_array = [];

for (index = 0; index < birth\_years.length; index++) {

*const* current\_year = 2023;

  empty\_new\_array.push(current\_year - birth\_years[index]);

}

console.log(empty\_new\_array);



**Continue and Break Statement**

Continue is to **exit the current iteration** of the loop and **continue to the next one**.

*const* my\_name\_array = [

  'Asad',

  'Khalil',

  2023 - 2001,

  'teacher',

  ['Michael', 'Peter', 'Steven'],

];

for (*let* index = 0; index < my\_name\_array.length; index++) {

  if (typeof my\_name\_array[index] !== 'string') continue;

  // After continue all the elements of type other than string will be skipped cause typeof my\_name\_array[index] !== 'string' will result in true only when typeof my\_name\_array[index] is not a string

  // and continue statement is executed and hence all the elements are skipped that are not string leaving behind the string elements

  console.log(my\_name\_array[index]);

}

A screenshot of a computer

Description automatically generated

**If the type of current element is not a string, then continue which results in exit of current iteration of loop and next one starts.**

**Break** is used to **completely terminate** **whole loop, not just the current iteration**.

*const* my\_name\_array = [

  'Asad',

  'Khalil',

  2023 - 2001,

  'teacher',

  ['Michael', 'Peter', 'Steven'],

];for (*let* index = 0; index < my\_name\_array.length; index++) {console.log("Without using break"+my\_name\_array[index]);if (typeof my\_name\_array[index] === 'number') break;

  console.log("Using break"+my\_name\_array[index]);

}

After the first number is found nothing else is printed, not even the line after break is executed and loop is terminated.

**Looping Backwards in arrays**

*const* my\_array = [

  'Asad',

  2023 - 2001,

  'student',

  ['Michael', 'Peter', 'Steven'],

];

console.log(my\_array.length);

for (*let* index = my\_array.length - 1; index >= 0; index--) {

  console.log(my\_array[index]);

}

**Loop inside a Loop**

Three different exercises, repeat each exercise five times.

Total 15 repetition

for (*let* index = 1; index < 4; index++) {

  console.log(`\_\_\_\_\_\_\_\_\_Starting Exercise Number ${index}\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_`);

  for (*let* index\_done = 1; index\_done < 6; index\_done++) {

    console.log(

      `Exercise ${index}, Number of exercise repition: ${index\_done}`

    );

  }

}

**While Loop**

while (condition) {}

While loop will keep running as long as the logical condition is true.

Initial value of the counter is defined outside the while loop.

In the end of the while loop change the value of the counter.

// Setting initial value of the counter

*let* rep = 1;

while (rep <= 10) {

  console.log(`Lifting weight repitition ${rep}`);

  // Change the value of the counter

  rep++;

}

Random Number:

Math.random();

Math.random will create random number from 0 to 1, those numbers will be decimal numbers.

Math.random()\*6;

Math.trunc will give us the number between 0 and 5

Math.trunc(Math.random()\*6);

Will give us the number from 1 to 6

Math.trunc(Math.random()\*6)+1;