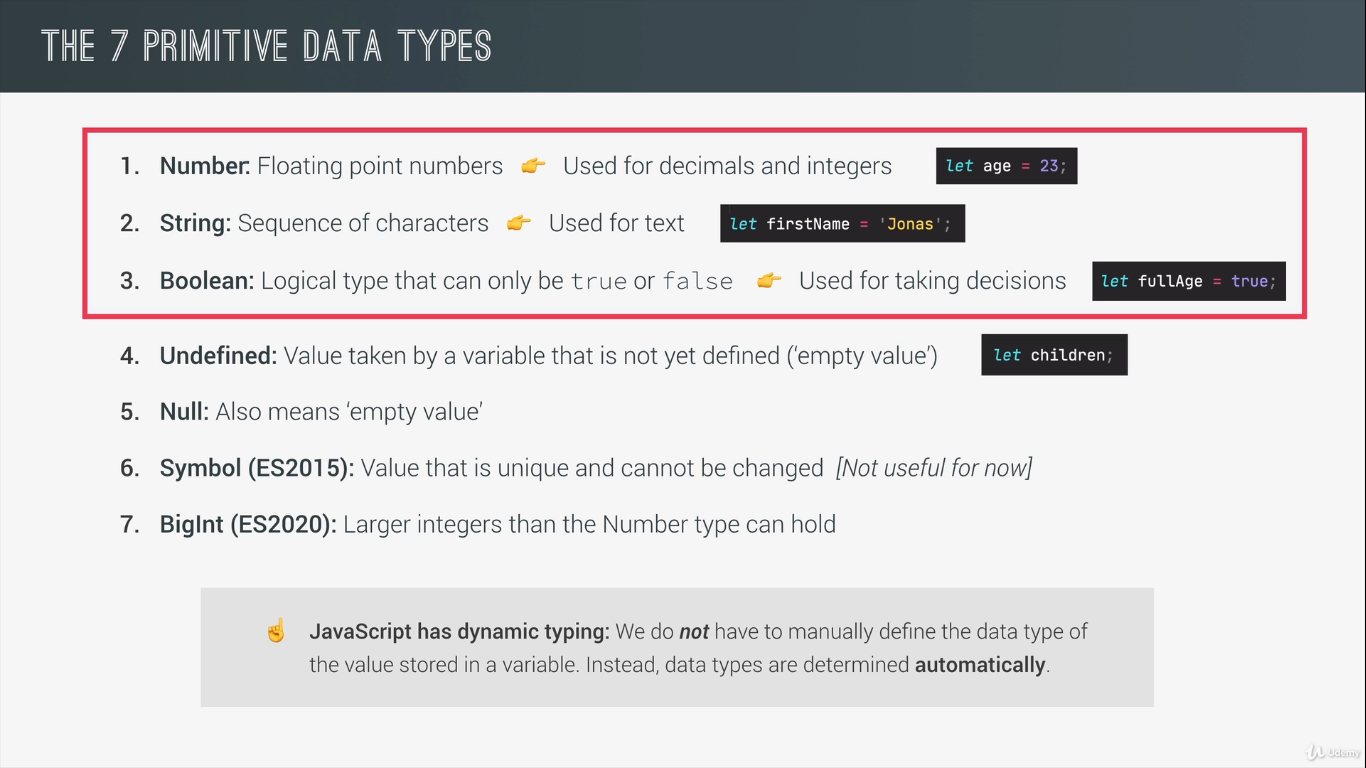
**2 May 2021**

**Data Type**

****

* Value has type , not Variable!
* typeof(…..)

**LET,CONST and VAR**

**Let:**

The **let** statement declares a block-scoped local variable, optionally initializing it to a value.

**Let x = 1;**

**if (x === 1) {**

**let x = 2;**

**console.log(x);**

**// expected output: 2**

**}**

**console.log(x);**

**// expected output: 1**

**Const:**

Constants are block-scoped, much like variables declared using the [let](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let) keyword. The value of a constant can't be changed through reassignment, and it can't be redeclared.

const number = 42;

try {

number = 99;

} catch (err) {

console.log(err);

// expected output: TypeError: invalid assignment to const `number'

// Note - error messages will vary depending on browser

}

console.log(number);

// expected output: 42

**Var:**

The **var statement** declares a function-scoped or globally-scoped variable, optionally initializing it to a value.

var x = 1;

if (x === 1) {

var x = 2;

console.log(x);

// expected output: 2

}

console.log(x);

// expected output: 2

**DAY 2**

**BASIC OPERATOR :**

//  Math operator

const now = 2037;

const ageShivam = now - 1997;

const ageUtkarsh = now - 1990;

console.log(ageShivam,ageUtkarsh);

console.log(ageUtkarsh\*2,ageUtkarsh/2,2\*\*3);

// here  2\*\*3 means  2\*2\*2

// Now we concatination of two string

let firstName = "Utkarsh";

let lastName="gandu";

console.log( firstName+" "+lastName);

        // Assigment operator

let x = 15;

x +=10;  // x+=10 is x = x+10;

x \*=4;

x++;

x--;

console.log(x);

// comparsion operators

console.log(ageUtkarsh>ageShivam);    //   <,>,>=,<=

console.log(ageShivam >= 18);

console.log(now-1990 > now -1997);

**OPERATOR PRECEDENCE:**

**Operator precedence** determines how operators are parsed concerning each other. Operators with higher precedence become the operands of operators with lower precedence.

let z , y;

z = y = 25-10-5;

console.log(z , y);

let averageAge = (ageUtkarsh + ageShivam)/2;

console.log(averageAge);

console.log(ageShivam, ageUtkarsh, averageAge);

**Day 3:**

**Strings , Templet and Literals:**

//          Strings ,Templete and literals

const firstName = "Utkarshgandu";

const year = 1990;

const job = "developer";

const utkarsh =  "I,m " + firstName +' , a' + (year-2021) + ' year old  ' + job+'!';

console.log(utkarsh);

const utkarshNew =`I,m ${firstName} , a ${year-2021} year old  ${job} !`;

console.log(utkarshNew);

// backtick use

console.log('Just a regular string......');

console.log ('string with \n\

multiple \n\

line');

console.log(`string with

multiple

line`);

**If else:->**

//  If else example

const shiavmAge = 13;

//const isoldEnough = shiavmAge >=18;

if(shiavmAge>=18){

    console.log("Shivam can start driving licence !");

}else{

    const yearLeft = 18 - shiavmAge;

    console.log(`Shivam is to young for driving licence. Wait for  ${yearLeft} years .`)

}

// control structure

// if {

// }else{

// }

const birthyear = 2001;

let centure ;

if(birthyear<=2000){

  centure=20;

}else{

    centure=21;

}

console.log(centure);

**Type conversion and coersion:-**

//    Type conversion and coercion

  //  conversion

const inputYear = '1991';

console.log(*Number*(inputYear), inputYear);

console.log(*Number*(inputYear)+18);

console.log(inputYear + 18);

// + convert into string

console.log(inputYear-10);

console.log(*Number*('shivam raj'));

console.log(NaN);

console.log(*String*(23) , 23);

//coercion

console.log('I am' + 23 + 'year old!');

console.log('23'- '10' -'3');

console.log('23' + '10' + '3'); // + always behav different

console.log('23'>'10');

let n = '1' + 1;

n -= 1;

console.log(n);

**Truthy and Falsy value:->**

// Truthy and Falsy  values

//  5 Falsy values :-> 0, null, undefined,' ', NaN

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

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

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

console.log(*Boolean*({})); // empty object

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

const money = 0;

if(money){

    console.log("Don't Spend to much money !");

}else{

    console.log("You should get a Job!");

}

Money 0 means falsy value that’s why else part excuted .

Money 100 means truthy value that’s why if part excuted.

**Equality operator == vs === :->**

The simplest way of saying that, == will not check types and === will check whether both sides are of same type. So, == is tolerant. But under the hood it converts to its convenient type to have both in same type and then do the comparison.

=== compares the types and values. Hence, if both sides are not same type, answer is always false. For example, if you are comparing two strings, they must have identical character sets. For other primitives (number, boolean) must share the same value.

**Rule for implicit coercion:** Comparison by using == does implicit type conversion under the hood. And rules for implicit coercion are as follows-

* If both operands are same type use ===
* undefined == null
* If one operands is string another is number, convert string to number
* If one is boolean and another is non-boolean, convert boolean to number and then perform comparison
* While comparing a string or number to an object, try to convert the object to a primitive type and then try to compare

Be careful while comparing objects, identifiers must reference the same objects or same array.

// == vs ===

const age = '18';

 if(age === 18) console.log(" You  just become an adult (srtict) ");

 if (age == 18 ) console.log(" You just become an adult (loose)");

 const fav=*Number*(prompt("What's your favourite Number :"));

 console.log( fav);

 if (fav ===23){

     console.log(" Cool ! 23 is an  amazing number !")

 }else if (fav===7){

    console.log(" Cool ! 7 is an  amazing number !")

 }

 else if( fav ===9 ){

    console.log(" Cool ! 9 is an  amazing number !")

 }

 else{

     console.log("The number is not 23 or 7 or 9")

 }

 if(fav != 23) console.log("The number is not cool!")

**Boolean Logics :-> AND(&&) , OR (||) and NOT (!)**

const hasDriversLicense =  true; //A

const hasGoodVision= false ; //B

console.log(hasDriversLicense && hasDriversLicense);

console.log(hasDriversLicense || hasGoodVision);

console.log(!hasDriversLicense);

const Utkarsh =  hasDriversLicense && hasGoodVision && !isTired;

if(Utkarsh){

    console.log("Utkarsh able to driver the car....");

}else

{

    console.log("Utkarsh not able to drive the car , someone else have to drive the car...")

}

const isTired = false;

console.log( hasDriversLicense || hasGoodVision || !isTired);

**Day 4**

**Challenge 3 :->**

// Challenge  3

/\*

 1. Calculate the average socre for each team ,using the test data below

  2. Compare the team's average scores to determine the winner of the competition, and print it to the console. Don't forget that can be a draw , so test  for that as well (draw means they have the same score average score  ).

  3. Bonus 1: Include a requirement for a minimum score of 100, with this rule , a team only wins if it has a higher score than the other team , and the same time a score of at least 100 point . HINT:  use a logical operator to test for minimum score , as well as multiple  else-if blocks.

  4. Bonus 2 : minimum  socre also applies to a draw! So  a draw  only happens when both teams have the same score and both have a score greater or equal 100 points , otherwise , no team wins the trophy .

  Test Data :  Dolphins score 96 , 108  , 89  and Koalas score : 88 , 91 , 110;

   Test Data :  Dolphins score 97, 112 , 101  and Koalas score : 189 , 95  , 123;  Test Data :  Dolphins score 97 , 112  , 101  and Koalas score :189 , 95 , 186;

\*/

const avgteamDolphins =  (189+ 95 + 186 )/3;

const avgteamKoalas = (189+ 95 + 186)/3;

 console.log (avgteamDolphins, avgteamKoalas);

if( avgteamDolphins>avgteamKoalas && avgteamDolphins >= 100 ){

    console.log(`Team Dolphins Score : ${avgteamDolphins}  ,  Team Koalas Score: ${avgteamKoalas}`);

    console.log( " Team Dolphin is Winner !");

}else if( avgteamKoalas >avgteamDolphins && avgteamKoalas>= 100 ) {

    console.log(`Team Dolphins Score : ${avgteamDolphins}  ,  Team Koalas Score: ${avgteamKoalas}`);

    console.log(" Team Koalas is winner !");

}else if ( avgteamDolphins === avgteamKoalas && avgteamDolphins>=100 && avgteamKoalas >= 100 ){

    console.log(`Team Dolphins Score : ${avgteamDolphins}  ,  Team Koalas Score: ${avgteamKoalas}`);

    console.log(" Match is Draw !");

}else {

    console.log (" No one win the trophy , Shivam is winner now");

}

**Day 5:->**

**Switch statement :->**

// switch statement

const day = 'sunday';

switch(day){

    case 'monday':

        console.log('Plane for course structure');

        console.log('Go to coding meetup');

        break;

        case 'tuesday':

            console.log('Prepaer for the Theory Videos');

            break;

            case 'wednesday':

                case 'thursday':

                    console.log('Write code examples');

                    break;

                    case'saturday':

                    case'sunday':

                    console.log('Enjoy the Weekend!');

                    break;

                    default:

                        console.log('Not Valid day!');

}

//  Compare with if else

if (day ==='monday') {

    console.log('Plane for course structure');

    console.log('Go to coding meetup');

}else if (day ==='tuesday') {

    console.log('Prepaer for the Theory Videos');

}else if(day === 'wednesday' ||  day === 'thursday'){

    console.log('Write code examples');

}else if (day === 'saturday' || day === 'sunday')

{

    console.log('Enjoy the Weekend!');

}else{

    console.log('Not Valid day!');

}

**Statments and Expressions :->**

**Expressions**

Any unit of code that can be evaluated to a value is an expression. Since expressions produce values, they can appear anywhere in a program where JavaScript expects a value such as the arguments of a function invocation. As per the MDN documentation, JavaScript has the following expression categories.

**Arithmetic Expressions:**

Arithmetic expressions evaluate to a numeric value. Examples include the following

10; // Here 10 is an expression that is evaluated to the numeric value 10 by the JS interpreter10+13; // This is another expression that is evaluated to produce the numeric value 23

**String Expressions:**

String expressions are expressions that evaluate to a string. Examples include the following

'hello';  
'hello' + 'world'; // evaluates to the string 'hello world'

**Logical Expressions:**

Expressions that evaluate to the boolean value true or false are considered to be logical expressions. This set of expressions often involve the usage of logical operators && (AND), ||(OR) and !(NOT). Examples include

10 > 9; // evaluates to boolean value true  
10 < 20; // evaluates to boolean value false  
true; //evaluates to boolean value true  
a===20 && b===30; // evaluates to true or false based on the values of a and b

**Primary Expressions:**

Primary expressions refer to stand alone expressions such as literal values, certain keywords and variable values. Examples include the following

'hello world'; // A string literal  
23; // A numeric literal  
true; // Boolean value true  
sum; // Value of variable sum  
this; // A keyword that evaluates to the current object

**Left-hand-side Expressions:**

Also known as lvalues, left-hand-side expressions are those that can appear on the left side of an assignment expression. Examples of left-hand-side expressions include the following

// variables such as i and total  
i = 10;  
total = 0;// properties of objectsvar obj = {}; // an empty object with no properties  
obj.x = 10; // an assignment expression// elements of arrays  
array[0] = 20;  
array[1] = 'hello';// Invalid left-hand-side errors  
++(a+1); // SyntaxError. Attempting to increment or decrement an expression that is not an lvalue will lead to errors.

Now that we have covered the basics of expressions, let’s dive a bit deeper into expressions.

**Assignment Expressions:**

When expressions use the = operator to assign a value to a variable, it is called an assignment expression. Examples include

average = 55;var b = (a = 1); // here the assignment expression (a = 1) evaluates to a value that is assigned to the variable b. b = (a = 1) is another assignment expression. var is not part of the expression.

The = operator expects an lvalue as its left-side operand. The value of an assignment expression is the value of the right-side operand such as 55 in the above example. As a side effect, the = operator assigns the value on the right side to the value on the left side.

**Expressions with side effects:**

As we just saw with assignment expressions, expressions with side effects are those that result in a change or a side effect such as setting or modifying the value of a variable through the assignment operator =, function call, incrementing or decrementing the value of a variable.

sum = 20; // here sum is assigned the value of 20sum++; // increments the value of sum by 1function modify(){  
 a \*= 10;  
}var a = 10;  
modify(); // modifies the value of a to 100.

**Statements**

A statement is an instruction to perform a specific action. Such actions include creating a variable or a function, looping through an array of elements, evaluating code based on a specific condition etc. JavaScript programs are actually a sequence of statements.

Statements in JavaScript can be classified into the following categories

**Declaration Statements:**

Such type of statements create variables and functions by using the var and function statements respectively. Examples include

var sum;  
var average;// In the following example, var total is the statement and total = 0 is an assignment expressionvar total = 0;// A function declaration statement function greet(message) {  
 console.log(message);  
}

**Expression Statements:**

Wherever JavaScript expects a statement, you can also write an expression. Such statements are referred to as expression statements. But the reverse does not hold. You cannot use a statement in the place of an expression.

var a = var b; // leads to an error cause you cannot use a statement in the place of an expressionvar a = (b = 1); // since (b = 1) is an assignment expression and not a statement, this is a perfectly acceptable line of codeconsole.log(var a); // results in error as you can pass only expressions as a function argument

Stand alone primary expressions such as variable values can also pass off as statements depending on the context. Examples of expression statements includes the following

// In the following example, sum is an expression as it evaluates to the value held by sum but it can also pass off as a valid statement.sum;// An expression statement that evaluates an expression with side effectsb = 4+38;

**Conditional Statements:**

Conditional statements execute statements based on the value of an expression. Examples of conditional statements includes the if..else and switch statements.

// Syntax of an if statement. If the expression following the if statement evaluates to a truthy value, statement 1 is executed else statement 2 is executed.if (expression)   
 statement 1  
else   
 statement 2

**Loops and Jumps :**

Looping statements includes the following statements: while, do/while, for and for/in. Jump statements are used to make the JavaScript interpreter jump to a specific location within the program. Examples of jump statements includes break, continue, return and throw.

**Function Expressions vs Function Declarations:**

A function expression, particularly a named function expression, and a function declaration may look the same but their behavior is very different.

A function expression is part of a variable assignment expression and may or may not contain a name. Since this type of function appears after the assignment operator =, it is evaluated as an expression. Function expressions are typically used to assign a function to a variable. Function expressions are evaluated only when the interpreter reaches the line of code where function expressions are located.

// A function expression. We assign a function to the variable num and use it to call the function.var num = function message(x) {  
 return x + x;  
 }num(7); // returns 14// An anonymous function expression. Behaves exactly like a named function expression.var num = function (x) {  
 return x + x;  
 }num(7); // returns 14

Only function expressions can be immediately invoked. Such types of function expressions are referred to as Immediately Invoked Function Expression (IIFE).

// An Immediately Invoked Function Expression(function () {  
 console.log('Immediately Invoked Function Expression.');  
})();

On the other hand, **function declarations are statements** as they perform the action of creating a variable whose value is that of the function. Function declaration falls under the category of declaration statements. Also, function declarations are hoisted to the top of the code unlike function expressions. Function declarations must always be named and cannot be anonymous.

// Example of a function declaration. Function declarations always start with the function keyword.function greet(message) {  
 return "Hi " + message;  
}

**Ternary Operator :->**

The **conditional (ternary) operator** is the only JavaScript operator that takes three operands: a condition followed by a question mark (?), then an expression to execute if the condition is [truthy](https://developer.mozilla.org/en-US/docs/Glossary/Truthy) followed by a colon (:), and finally the expression to execute if the condition is [falsy](https://developer.mozilla.org/en-US/docs/Glossary/Falsy). This operator is frequently used as a shortcut for the [if](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/if...else) statement.

[**Syntax**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Conditional_Operator#syntax)

condition ? exprIfTrue : exprIfFalse

[**Parameters**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Conditional_Operator#parameters)

*condition*

An expression whose value is used as a condition.

*exprIfTrue*

An expression which is evaluated if the *condition* evaluates to a [truthy](https://developer.mozilla.org/en-US/docs/Glossary/Truthy) value (one which equals or can be converted to true).

*exprIfFalse*

An expression which is executed if the *condition* is [falsy](https://developer.mozilla.org/en-US/docs/Glossary/Falsy) (that is, has a value which can be converted to false).

// Ternary operator

const age = 23;

age >= 18 ? console.log('I love to drink Wine') : console.log('I love to drink Water!');

function getFee(*isMember*) {

    return (isMember ? '$2.00' : '$10.00');

  }

  console.log(getFee(true));

  // expected output: "$2.00"

  console.log(getFee(false));

  // expected output: "$10.00"

  console.log(getFee(null));

  // expected output: "$10.00"

**JAVASCRIPT PART 2**

**Day 6:->**

**Use Strict Mode:->**

'use strict';

console.log("Hello Javascript Part 2 !");

let hasDriversLicense = false ;

const passTest = true;

if (passTest) hasDriverLicense = true;

if (hasDriversLicense) console.log(" I can drive ");

 let interface = "Fun";

 let private = 20;

Here , hasDriverLincense is not declare but When We, run the code in console , we don’t got error because of bug . So, we Use ‘use strict ‘ to find the minnor bug in program or some reserved key which is used in javascript upcoming version.

Uncaught SyntaxError: interface is a reserved identifier

**Function :->**

Functions are one of the fundamental building blocks in JavaScript. A function in JavaScript is similar to a procedure—a set of statements that performs a task or calculates a value, but for a procedure to qualify as a function, it should take some input and return an output where there is some obvious relationship between the input and the output. To use a function, you must define it somewhere in the scope from which you wish to call it.

[**Function declarations**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Functions#function_declarations)

A **function definition** (also called a **function declaration**, or **function statement**) consists of the [function](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/function) keyword, followed by:

* The name of the function.
* A list of parameters to the function, enclosed in parentheses and separated by commas.
* The JavaScript statements that define the function, enclosed in curly brackets, {...}.

For example, the following code defines a simple function named square:

**function square(number) {**

**return number \* number;**

**}**

**Function expression**

The **function** keyword can be used to define a function inside an expression.

You can also define functions using the [Function](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/Function) constructor and a [function declaration](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/function).

[**Syntax**](https://developer.mozilla.org/en-US/docs/web/JavaScript/Reference/Operators/function#syntax)

The expression is not allowed at the start of a statement.

**function [name]([param1[, param2[, ..., paramN]]]) {**

**statements}**

// Function declartion & expression

// function declartion

function birthYear(*yourYear*){

    const age = 2021 - birthYear ;

    return age;

}

// same process with different method

function calAge(*birthYear*){

    return 2021 - *birthYear*;

}

const yourAge = calAge(1997);

console.log(`Your age is :-> ${yourAge} and you are a looser!`)

// function expression

const calcAge2 = function(*birthYear*){  // this is also called an

    return 2021-*birthYear*;

}

// If we call above function expression , it gives error

// cannot access "calcAge2 " before initilization.

// but we have not any error in function declartion.

// Note: process called hoisting this

**Day 7**:->

**Function calling other function**

// Function calling other function

const  cutPieces =  function (*fruit*)

{

    return *fruit* \* 4;

};

function fruitProcessor(*apple*,*orange*){

    const applePieces = cutPieces(*apple*);

    const orangePieces = cutPieces(*orange*);

    const juice = `Juice with ${applePieces} pieces of apples and ${orangePieces} pieces of oranges`;

    return juice;

};

console.log(fruitProcessor(5,9));

**Example 2:->**

const calcAge = function(*bornYear*) {

  return 2021 - *bornYear*;

}

const yearunitRetirement = function(*firstName* , *bornYear*){

    const age = calcAge(*bornYear*);

    const retirement = 65 - age ;

    if (retirement >0) {

        return retirement;

        console.log(`${*firstName*} retires in ${retirement} years`);

    }else{

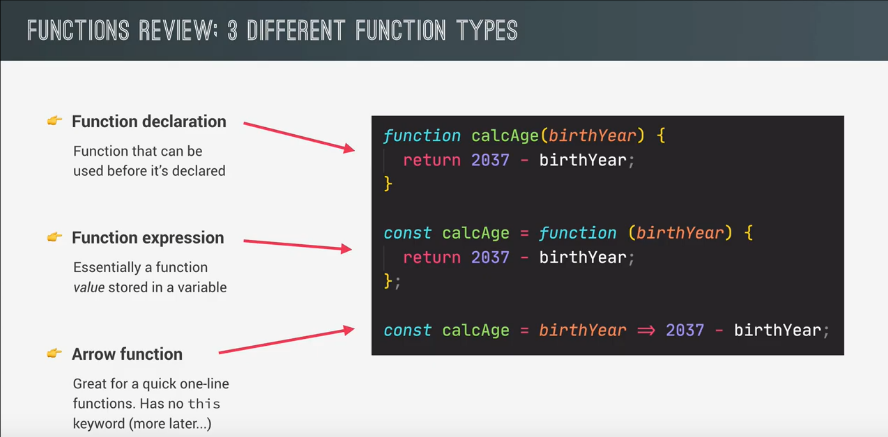
    return -1;

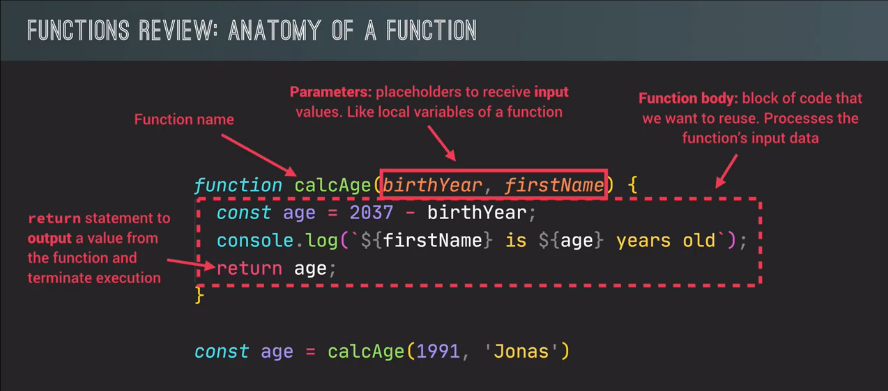
    console.log(`${*firstName*} has already retires  `);

    }

}

console.log(yearunitRetirement('Utkarsh',1990));

****

****

// Challenge no 1

/\*

Back to the two gymnastics teams , the Dolphins and the Koalas! There is a new gymnastics discipline , which  works differently.

Each team  completes 3 times , and then the average of the 3 scores is calculated (so one average score per team).

\*/

const calcAverage = (*a*,*b*,*c*) =>( *a*+ *b*+*c*)/3;

console.log (calcAverage(10,4,2));

let scoreDolphins = calcAverage( 4,4,4);

let scoreKoalas = calcAverage();

console.log(scoreDolphins,scoreKoalas);

const checkWinner = function (*avgDolphins*,*avgKoalas*) {

  if (*avgDolphins* >= 2\**avgKoalas*){

      console.log(`Dolphins win \*\*\*\*\*  ${*avgDolphins*}  vs ${*avgKoalas*}`);

  }else if (*avgKoalas* >= 2\**avgDolphins*) {

       console.log(`Koalas  win \*\*\*\* ${*avgKoalas*} vs ${*avgDolphins*}`);

  }else{

      console.log('Both team are looser');

  }

}

checkWinner(scoreDolphins,scoreKoalas);

checkWinner(576,111);

// test 2

// Dolphins score (85, 54,51)

// koalas score (23,34,27)

scoreKoalas = calcAverage(85, 54,51);

scoreDolphins =calcAverage (23,34,27);

console.log(scoreKoalas,scoreDolphins);

checkWinner(scoreDolphins,scoreKoalas);

**Array (data structure):->**

The JavaScript **Array** class is a global object that is used in the construction of arrays; which are high-level, list-like objects.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*      ARRAYS  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

 const studentOne = 'Shivam';

 const  studentSecond = 'Raina';

 const studentThird ='Rytham';

//In Array we can create in one place

const student = ['shivam ', 'raina ','rytham '];

console.log(student);

//Common Operation

const fruits =['apple' ,'banana'];

console.log(fruits.length);

//Access an Array item from end of Array

let first = fruits[0];

console.log(first);

let last = fruits[fruits.length-1];

console.log(last);

//Loop over an Array

fruits .forEach(function(*item*,*index*,*array*) {

    console.log(*item*,*index*);

});

const friends =['Shivam', 'suman', 'utkarsh'];

const  firstName = 'Raj';

const jonas = [ firstName ,'Maddy' , 2021 - 1997 , 'suraj' , friends];

console.log(jonas);

const  calAge=function(*year*){

    return 2021- *year*;

}

const years = [1991,1990,1994,1995,1997];

const age1 = calAge(years[0]);

const age2 = calAge(years[1]);

const age3 = calAge(years[2]);

const age4 = calAge(years[years.length -1]);

console.log (age1,age2,age3,age4);

**method & Operation:->**

//  \*\*\*\*\*\*\*\*\*\*opertaion \*\*\*\*\*\*\*\*\*\*\*

//push & pop

// push

const meet = ['sam' , 'maddy ', 'naddy'];

meet.push('nuke'); //last

meet.unshift('shivam raj'); //start

console.log(meet);

//pop

const frnd = ['sunny', 'gin' , 'sum', 'petter'];

const popped = frnd.pop();  // last

console.log(frnd);

console.log(popped);

frnd.shift();   //start

console.log(frnd.indexOf('sum'));

console.log(frnd.indexOf('gin'));

console.log(frnd.indexOf('bob'));

console.log(frnd);

 frnd.unshift(23);

console.log(frnd.includes('gin'));

console.log(frnd.includes('sum'));

console.log(frnd.includes('bob'));

console.log(frnd.includes(23));

if( frnd.includes('sum')){

    console.log('sum is present ');

}else{

    console.log('sum is absent');

}

**Challenge:->**

// challenge 2

// create a funtion to calculate the tip on the basis of condition if bill  is bettween or equal to 50>= , <=300 they tip 15%  else they tip 20%

// create array list of Bills with the help of  bills data create tips array

const calcTip = function(*bill*){

    return *bill* <=300 && *bill*>=50 ? *bill*\*0.15 : *bill*\*0.20;

}

const bills =[40,200,250];

const  tiplist = [ calcTip (bills[0]) , calcTip(bills[1]),calcTip(bills[0])   ];

const  total = [bills[0]+tiplist[0 ],bills[1]+tiplist[1 ], bills[2]+tiplist[2]];

console.log(tiplist , bills , total);

**Array vs object**

// Array vs object

// Array

const shivamArray = [

    'Shivam',

    'raj',

    2021-1997,

    ['suman' , 'raina' ,'utkarsh']

];

console.log(shivamArray);

const shivamObject = {

    firstName: 'shivam',

    lastName:'raj',

    age : 2021-1997,

    friends: ['suman' ,'raina' , 'utkarsh' ]

};

console.log(shivamObject);