* function expressions :

```
// Function declaration
function calcAge(birthYear) {
  return 2023 - birthYear;
}

const age = calcAge(2001);
console.log(age);

// Function expression
const calcAge2 = function (birthYear) {
  return 2023 - birthYear;
};

const age2 = calcAge2(2001);
console.log(age2);
```

→ functions are like values similar Number, Boolean, String and these can be stored in variables

* → function declarations are hoisted where as expressions are not.

→ Its a developer's preference.

* Arrow Functions :

```
// Function Declaration
function yearsUntilRetirement1(birthYear, firstName) {
  const age = 2023 - birthYear;
  const retirement = 65 - age;
  return `${firstName} retires in ${retirement} years`;
}
console.log(yearsUntilRetirement1(2001, "Anurag"));

// Function Expression
const yearsUntilRetirement2 = function (birthYear, firstName) {
  const age = 2023 - birthYear;
  const retirement = 65 - age;
  return `${firstName} retires in ${retirement} years`;
};
console.log(yearsUntilRetirement2(2001, "Anurag"));

// Arrow Functions
const yearsUntilRetirement3 = (birthYear, firstName) => {
  const age = 2023 - birthYear;
  const retirement = 65 - age;
  return `${firstName} retires in ${retirement} years`;
};
console.log(yearsUntilRetirement3(2001, "Anurag"));
```

→ not hoisted

→ do not have access to 'this' keyword.

↓

more about 'this' later

```
const checkEven2 = (num) => num % 2 == 0;

/*
const checkEven2 = (num) => {
  return (num % 2) == 0;
}
*/
```

★ Objects : (key, value pairs)

```
const myInfoArr = [
  "Anurag",
  "Student",
  2023 - 2001,
  8128039044,
  ["Rahul", "Shiva", "Ram"],
];
console.log(myInfoArr);
console.log(myInfoArr[0], myInfoArr[1]);

const myInfoObj = {
  firstName: "Anurag",
  profession: "Student",
  age: 2023 - 2001,
  phone: 8128039044,
  friends: ["Rahul", "Shiva", "Ram"],
};
console.log(myInfoObj);
console.log(myInfoObj.firstName, myInfoObj.age);
console.log(myInfoObj["firstName"], myInfoObj["age"]);
```

} create object, object literal syntax

→ In arrays accessing was through order (index) of elements, In objects element has a name, value (key, value pair)
name → key / property

→ {} (obj), [] (arr)

```
myInfoObj.location = "United States";
myInfoObj["instagram"] = "anurag_nampally";
console.log(myInfoObj);
```

→ you can add some more properties (keys).

Q: print output

"Anurag has ___ (no. of) friends, and ___ (1st friend) is his best friend"

```
console.log(
  `${myInfoObj.firstName} has ${myInfoObj.friends.length} friends, and ${myInfoObj.friends[0]} is his best friend`
);
```

★ Writing functions in Objects and "this" keyword :

```
const myInfoObj = {
  firstName: "Anurag",
  profession: "Student",
  birthYear: 2001,
  phone: 8128039044,
  friends: ["Rahul", "Shiva", "Ram"],
  calcAge: function () {
    this.age = 2023 - this.birthYear;
  },
};

// check whether a property/key exists
if (myInfoObj.age) {
  // undefined => false
  console.log(myInfoObj.age);
} else {
  console.log("Age doesnt exist hence I am calculating it");
  myInfoObj.calcAge();
}

console.log(myInfoObj);
console.log(myInfoObj.age);
```

→ functions are like values hence can be assigned to a property.

→ 'this' means object which is calling the function.

→ arrow functions will have access to the calling object, here this refers to a global level ⇒ a "window"

Example :

let's use objects to implement the calculations!
Remember: BMI = mass / (height * height) (mass in kg and height in meters).

Your tasks:
For each of them, create an object with properties for their full name, mass, and height (Mark Miller and John Smith).

Name these objects as mark and john, and their properties exactly as fullName, mass and height.

Create a calcBMI method on each object to calculate the BMI (the same method on both objects). Assign the BMI value to a property, and also return it from the method.

Log to the console who has the higher BMI, together with the full name and the respective BMI. Example: "John Smith's BMI (28.3) is higher than Mark Miller's (23.9)!".

```
const mark = {
  fullName: "Mark Miller",
  mass: 78,
  height: 1.69,
  calcBMI: function () {
    this.bmi = this.mass / this.height ** 2;
    return this.bmi;
  },
};

const john = {
  fullName: "John Smith",
  mass: 92,
  height: 1.95,
  calcBMI: function () {
    this.bmi = this.mass / this.height ** 2;
    return this.bmi;
  },
};

mark.calcBMI();
john.calcBMI();

// "John Smith's BMI (28.3) is higher than Mark Miller's (23.9)!"
if (mark.bmi > john.bmi) {
  console.log(
    `${mark.fullName}'s BMI (${mark.bmi.toFixed(2)}) is higher than ${
      john.fullName
    }'s (${john.bmi.toFixed(2)})!`
  );
} else {
  console.log(
    `${john.fullName}'s BMI (${john.bmi.toFixed(2)}) is higher than ${
      mark.fullName
    }'s (${mark.bmi.toFixed(2)})!`
  );
}
```

★ Destructuring Arrays :

1

```
const arr = [2, 3, 4];
const a = arr[0];
const b = arr[1];
const c = arr[2];
console.log(a, b, c);

const [x, y, z] = arr;
console.log(x, y, z);

// its easy to swap two variables
let primary = 12;
let secondary = 23;
console.log(primary, secondary);

[primary, secondary] = [secondary, primary];

console.log(primary, secondary);
```

3

```
// nested destructuring
const nested = [2, 4, [5, [6, 7, 8]]];
const [, , [a, [b, c, d]]] = nested;
console.log(a, b, c, d);

// default values
const [p = 1, q = 1, r = 1] = [8, 9];
console.log(p, q, r);
```

2

```
const library = {
  name: "Book Haven Library",
  location: "123 Main Street, Anytown, USA",
  categories: ["fiction", "non-fiction", "mystery", "drama"],
  fiction: ["FictionBook1", "FictionBook2", "FictionBook3"],
  mystery: ["mysteryBook1", "mysteryBook2", "mysteryBook3"],
  // order will buy two books from fiction
  order: function (fictionIdx1, fictionIdx2) {
    return [this.fiction[fictionIdx1], this.fiction[fictionIdx2]];
  },
};

// retrive 1st two fictional books
console.log(library.fiction[0], library.fiction[1]);
// using destructure
const [first, second] = library.fiction;
console.log(first, second);

// retrive 2nd and 4th categories
console.log(library.categories[1], library.categories[3]);
// using destructure
const [, two, , four] = library.categories;
console.log(two, four);

// how to return more than one value from a function
const [book1, book2] = library.order(1, 2);
console.log(book1, book2);
```

* Destructuring Objects :