1. Ten exercises – write simple function declarations

1.1 – Function that prints a greeting

Write a **function declaration** named `sayHello` that:

- Takes **no parameters**.
- Inside the function, prints the text:
    - `"Hello from the function"`
- Then **call** the function once.

1.2 – Function that prints numbers 1–3

Write a **function declaration** named `printOneToThree` that:

- Takes **no parameters**.
- Inside, uses a loop to print the numbers `1`, `2`, `3` (each on its own line).
- Call the function once.

1.3 – Function that prints an array length

Write a **function declaration** named `printLength` that:

- Creates an array of **4 names** inside the function.
- Prints the length of the array (should be `4`).
- Call the function once.

1.4 – Function that prints a fixed object

Write a **function declaration** named `printStudent` that:

- Creates an object with keys `name` and `age`, values `"Dana"` and `16`.
- Prints both `name` and `age` inside the function.
- Call the function once.

1.5 – Function that prints even numbers up to 10

Write a **function declaration** named `printEvensToTen` that:

- Uses a loop inside the function to print the even numbers: `2, 4, 6, 8, 10`.
- Call the function once.

1.6 – Function with return: sum of 2 numbers

Write a **function declaration** named `sumTwoNumbers` that:

- Has **no parameters**.
- Inside the function, creates two numbers: `a = 5`, `b = 7`.
- Calculates the sum and **returns** it.

Outside the function:

- Call `sumTwoNumbers`, store the returned value in a variable named `result`.
- Print `"The sum is: X"` (X is the variable).

1.7 – Function with return: total array length

Write a **function declaration** named `getNamesLength` that:

- Creates an array of names: `["Dana", "Noa", "Yossi", "Ali"]`.
- Returns the **length** of the array.

Outside the function:

- Call the function, store the result in `len`.
- Use an `if` to print `"Big class"` if `len` is at least 4, otherwise print `"Small class"`.

1.8 – Function with return: is passing grade

Write a **function declaration** named `getGrade` that:

- Inside the function, creates a variable `grade = 72`.
- Returns the value of `grade`.

Outside:

- Call `getGrade`, store in `studentGrade`.
- If `studentGrade` is at least 60, print `"Passed"`, otherwise `"Failed"`.

1.9 – Function with return: product of 3 fixed numbers

Write a **function declaration** named `getProduct` that:

- Inside, creates three numbers: `x = 2`, `y = 3`, `z = 4`.
- Returns the product of the three.

Outside:

- Call the function, store in `prod`.
- Print `"Product is: X"` where X is `prod`.

1.10 – Function with return: total price with fixed discount

Write a **function declaration** named `calculateFinalPrice` that:

- Inside, creates `price = 200` and `discount = 20`.
- Returns `price - discount`.

Outside:

- Call the function, store in `finalPrice`.
- Print `"Final price: X"` where X is `finalPrice`.

---

2. Ten exercises – functions with parameters + scope

Students must **write the code**, and then think about outer/inner scope and “what will happen”.

2.1 – Parameter used inside (no scope conflict)

Write a function declaration named `printScore(score)` that:

- Has one parameter `score`.
- Prints `"Score is: SCORE"`.

Outside:

- Create `let score = 80;`
- Call `printScore(score);`

Question for students:
Which value is printed, and where does it come from (parameter or outer variable)?

2.2 – Parameter vs outer variable

Write a program that:

- Creates `let grade = 50;` outside any function.
- Declares a function named `increaseGrade(grade)` that adds `10` to the **parameter** and prints it inside.

Call `increaseGrade(grade);` and **after that** also print the **outer** `grade`.

Question:
Does the outer `grade` change or stay `50`? What is printed inside vs outside?

2.3 – Modify outer variable inside function (no parameter)

Write a program that:

- Creates `let counter = 0;` outside any function.
- Declares a function named `incrementCounter()` that **adds 1 to the outer `counter`** and prints `counter`.
- Call `incrementCounter()` 3 times.
- Then print `counter` one more time outside the function.

Questions:
What values are printed on each call? What is the final value outside?

2.4 – Parameter shadows outer variable (different values)

Write a program that:

- Creates `let name = "Outer";` outside.

- Declares a function `printName(name)` that prints the parameter value.
- Call `printName("Inner");` and then also print the outer `name`.

Question:
What two values are printed and which is from where?

2.5 – Local variable only inside function

Write a program that:

- Declares a function `createSecret()` that creates `let secret = 1234;` **inside** the function and prints it.
- Call `createSecret();`
- Then try to print `secret` **outside** the function.

Question:
What happens when trying to print `secret` outside? Is it accessible or not?

2.6 – Function with 2 params

Write a function that:

- Has two parameters: `a` and `b`.
- Returns their sum.

Outside:

- Create `let a = 100;`
- Call the function with `(5, 7)` and print the returned value.

Question:
Does the outer `a` affect the function result?

2.7 – Object parameter, modify inside

Write a function that:

- Receives an object parameter `student`.
- Inside, changes `student.grade` to `100`.
- Prints the object.

Outside:

- Create a `student` object with `grade = 70`.
- Call the function with that object, then print `student.grade` outside.

Question:
Did the outer object change?

2.8 – Array parameter

Write a function that:

- Receives an array of numbers as a parameter.
- Prints the sum of all values, but does **not** change the array.

Outside:

- Create `numbers = [1,2,3,4];`
- Call the function and then print the array.

Question:
Is the array different after the function call?

2.9 – Parameter with default thinking (student decides)

Write a function that:

- Takes a single parameter `message`.
- If the function is called with a string, print it.
- If called with no argument at all, print `"No message"`.

Students must decide how to implement this and think about:
What happens when you call `printMessage("Hi")` vs `printMessage()`?

2.10 – Outer vs inner with `const`

Write a program that:

- Creates `const baseScore = 50;` outside.
- Declares a function `addBonus(baseScore)` that adds `20` to the **parameter** and prints the result.
- Calls `addBonus(baseScore)` and then prints `baseScore` outside.

Question:
Does the outer `baseScore` change? Why or why not?

---

3. Ten exercises – function expressions with arrow functions

All of these should be written as **arrow functions assigned to variables**.

3.1

Create an arrow function assigned to a variable `sayHi` that:

- Takes no parameters.
- Prints `"Hi from arrow function"`.
- Then call `sayHi()`.

3.2

Create an arrow function `doubleNumber` that:

- Takes one parameter `n`.
- Returns `n * 2`.

Call it with `5`, store the result, and print it.

3.3

Create an arrow function `sumThree` that:

- Takes three parameters: `a`, `b`, `c`.
- Returns their sum.

Call it with `1, 2, 3` and print the result.

3.4

Create an arrow function `getFirstElement` that:

- Takes an array parameter.
- Returns the first element of the array.

Call it with the array `["a","b","c"]` and print the returned value.

3.5

Create an arrow function `isAdult` that:

- Takes an `age` parameter.
- Returns `true` if age is at least 18, otherwise `false`.

Call it with `16` and with `20`, print both results.

3.6

Create an arrow function `square` that:

- Takes a number parameter and returns its square.
- Call it in a loop for numbers 1–3 and print the results.

3.7

Create an arrow function `getLength` that:

- Takes a string parameter and returns its length.

Call it with `"hello"` and print the result.

3.8

Create an arrow function `toUpperArray` that:

- Takes an array of strings.
- Returns a **new** array where each string is uppercase.

Call it with `["a","b","c"]` and print the returned array.

3.9

Create an arrow function `sumArray` that:

- Takes an array of numbers and returns the sum of all numbers.

Call it with `[10, 20, 30]` and print the result.

3.10

Create an arrow function `createStudent` that:

- Takes two parameters: `name` and `age`.
- Returns an object `{ name: name, age: age }`.

Call it with `"Dana", 16` and print the returned object.

---

4. Final exercise – general task broken into multiple functions

Use **everything**: variables, operators, if, loops, arrays, objects, functions.

4.1 – Student grades mini-system

Goal: Build a small program that manages student grades using **multiple functions**, not one big function.

The program must:

1. Have an array of **numbers** representing grades, for example: `70, 85, 90, 55, 100`.
2. Have an object representing general **settings**, for example:
    - `passGrade` (like 60)
    - `excellentGrade` (like 90)
3. Implement at least **four separate functions** (you can use function declarations or arrow functions):

    a) **`calculateAverage(gradesArray)`**
    - Receives the array of grades.
    - Calculates and **returns** the average grade (number).
4. **b)** **`countPassed(gradesArray, passGrade)`**
    - Receives the grades array and the `passGrade` from the settings object.
    - Returns how many grades are **greater than or equal to** `passGrade`.
5. **c)** **`getExcellentGrades(gradesArray, excellentGrade)`**
    - Receives the grades array and the `excellentGrade` from settings.
    - Returns a **new array** containing only the grades that are at least `excellentGrade`.
6. **d)** **`printReport(gradesArray, settings)`**

    - Receives the grades array and the settings object.
    - Inside, calls the other functions you wrote (`calculateAverage`, `countPassed`, `getExcellentGrades`).
    - Prints a full report, for example:
        - `"Average: X"`
        - `"Passed: Y students"`
        - `"Excellent grades: [ … ]"`
7. In the main code:
    - Create the `grades` array and `settings` object with precise values.
    - Call `printReport(grades, settings)` once.

You can optionally add:

- A function that adds a new grade to the array.
- A function that prints `"Great class"` if the average is above some number.