Background Story

Odessa is now knee-deep in her IT journey, facing new challenges and conquering them one by one. Her latest obstacle involves streamlining budget reports for her internship project. After spending countless hours staring at lines of repetitive code, she had an 'Aha!' moment. Why not condense these lengthy code blocks into neat little packages called functions?

As she delves into the world of functions, Odessa realizes the power they hold. Functions not only reduce errors but also make her code cleaner and more organized. It's like decluttering her room, but instead of clothes and shoes, she's tidying up her code. With functions, Odessa transitions from merely writing code to designing logic.

In her pursuit to become a full-stack developer, Odessa knows that mastering functions is a crucial step forward. le_04_function_overload copyBy understanding function declarations, parameters, and return values, she equips herself with the tools to create efficient and elegant solutions. Let's join Odessa as she explores the concept of function overload, turning mundane code into logical masterpieces.

Theory & Lecture Content

In JavaScript, functions are reusable blocks of code designed to perform a specific task. They help in reducing code duplication, improving code readability, and organizing logic efficiently. Let's break down the key aspects of functions:

1. Function Declarations

A function declaration consists of the `function` keyword, followed by the function name and a pair of parentheses `()`. This is then typically followed by a pair of curly braces `{}` where the function's logic resides.

```
// Function declaration
function greet() {
  return "Hello, World!";
}
```

2. Parameters

Parameters are placeholders in a function definition. They allow us to pass values into a function when it's called. Parameters are listed inside the parentheses of the function declaration.

```
// Function with parameters
function greet(name) {
  return `Hello, ${name}!`;
}
```

3. Return Values

Return values are the output of a function. When a function is called, it can perform tasks and then return a value using the `return` keyword.

```
// Function with return value
function addNumbers(a, b) {
  return a + b;
}
```

Resources:

- MDN Web Docs: Functions

Exercises

Exercise 1: Greet Function

Problem Statement:

Create a function called `greetUser` that takes a `name` parameter and returns a greeting message. If no name is provided, the function should return "Hello, Guest!"

Starter Code:

```
function greetUser(name) {
  // Write your code here
}
```

Full Solution Code:

```
function greetUser(name) {
  if (!name) {
    return "Hello, Guest!";
  } else {
    return `Hello, ${name}!`;
  }
}
```

Exercise 2: Calculate Age

Problem Statement:

Write a function `calculateAge` that takes a `birthYear` parameter and calculates the age based on the current year (2022). Return the calculated age.

Starter Code:

```
function calculateAge(birthYear) {
  // Write your code here
}
```

Full Solution Code:

```
function calculateAge(birthYear) {
  return 2022 - birthYear;
}
```

Exercise 3: Find Maximum Number

Problem Statement:

Define a function `findMax` that takes two parameters `a` and `b` and returns the greater of the two numbers.

Starter Code:

```
function findMax(a, b) {
  // Write your code here
}
```

Full Solution Code:

```
function findMax(a, b) {
  return a > b ? a : b;
}
```

Test Cases

Test Case 1:

```
test("Greet function with name", () => {
  expect(greetUser("Odessa")).toEqual("Hello, Odessa!");
});

test("Greet function without name", () => {
  expect(greetUser()).toEqual("Hello, Guest!");
});
```

Test Case 2:

```
test("Calculate Age function", () => {
  expect(calculateAge(1998)).toEqual(24);
});
```

Test Case 3:

```
test("Find Max function", () => {
  expect(findMax(10, 5)).toEqual(10);
  expect(findMax(7, 15)).toEqual(15);
});
```

Closing Story

Odessa's journey into the realm of functions has empowered her to think beyond just writing lines of code. By mastering function declarations, parameters, and return values, she's now equipped to architect logical solutions to complex problems. Functions have become her building blocks, paving the way for cleaner, more efficient code.

As Odessa continues her quest to become a top-tier developer and startup founder, she realizes that functions are not just about reducing errors but about designing scalable and elegant systems. Join Odessa in her next adventure as she dives into the world of function expressions and arrow functions, unlocking even more possibilities in her coding odyssey.