Functional Programming

Muhammad Zahir
Junejo

Lecture - Housekeeping

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly.
  - Please review Code of Conduct (in Student Undertaking Agreement) if unsure
- No question is daft or silly - **ask them!**
- Q&A session at the end of the lesson, should you wish to ask any follow-up questions.
- Should you have any questions after the lecture, please schedule a mentor session.
- For all non-academic questions, please submit a query: www.hyperiondev.com/support

Lecture Objectives

1. The concept of treating functions as objects and values in JavaScript.
2. Introduction to functional programming and its key concepts.
3. Exploring closures, callbacks, and higher-order functions.
4. Understanding how these concepts contribute to code modularity, reusability, and predictability.

What is Functional Programming?

- Functional Programming (FP) is a programming paradigm focused on treating computation as the evaluation of mathematical functions.
- Emphasizes immutability, statelessness, and composing functions to perform tasks.
- Key Concepts:
  - Functions (can be assigned to variables, passed as arguments, and returned from functions).
  - Avoidance of shared state and mutable data.
  - Focus on pure functions and immutability.
- Benefits:
  - Readability, reusability, and maintainability of code.
  - Easier to reason about and test.

Treating Functions as Values

- Assigning to Variables:
  - Functions can be assigned to variables.
- Passing as Arguments:
  - Functions can be passed as arguments to other functions.
- Returning from Functions:
  - Functions can be returned from other functions.

```
const greet = function(name) {
  return `Hello, ${name}!`;
};

const sayHello = greet;
const result = sayHello('Alice');
```

Closures

- A closure is a function that remembers the variables in its scope, even if the function is executed outside that scope.
- Use Cases:
  - Encapsulation: Maintain private data and expose selected methods.
  - Data Persistence: Retain data between function calls.
  - Callbacks: Retain context in asynchronous operations.

```
function createCounter() {
  let count = 0;
  return function() {
    count++;
    return count;
  };
}
const counter = createCounter();
```

Callbacks

- A callback is a function passed as an argument to another function to be executed later.
- Common Use Cases:
  - Handling asynchronous operations (e.g., API calls).
  - Event handling (e.g., button clicks).

```
function fetchData(url, callback) {
  // Fetch data from the server
  const data = ...;
  callback(data);
}
function displayData(data) {
  console.log(data);
}
fetchData('https://api.example.com/data', displayData);
```

Higher-Order Functions

- Higher-Order Functions (HOFs) are functions that take one or more functions as arguments or return a function.
- Benefits:
  - Promote code modularity and reusability.
  - Enable abstracting common patterns.
- Examples:
  - map(), filter(), reduce() for array manipulation.
  - Event listeners and handlers.

Benefits of Functional Programming

- Readability and Maintainability:
  - Modular code using pure functions is easier to understand.
  - Easier to debug and test.
- Concurrency and Parallelism:
  - Pure functions don't have side effects, making them suitable for parallel execution.
- Predictable Behavior:
  - Functions only depend on their arguments, leading to more predictable outcomes.
- Reusability and Composition:
  - Compose functions to build complex functionalities.

References

- https://developer.mozilla.org/en-US/docs/Glossary/Callback_function
- https://blog.bitsrc.io/understanding-higher-order-functions-in-javascript-75461803bad
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Closures

Questions and Answers

Thank You!