Part 1

Question 1:

1a(i):Imperative: Imperative programming entails explicitly coding every step to solve a problem, relying on understanding functions rather than pre-built models. According to Meni's first lecture its a language based on commands from register to another. It encompasses languages like C++, python, Java and Assembly emphasizing step-by-step instruction over model reliance.

1a(ii):Procedural: a type of imperative programming that structures programs around procedures, maintaining imperative features by defining the order in which statements are executed, and improving maintainability and quality through modularity and structured control flow.

1a(iii):Functional: The program is an expression or a series of expressions, not a sequence of commands .Running the program is a calculation of the expressions, finding their value and not the execution of commands.

1b: Procedural paradigm provides additional tools and principles that can lead to more organized, readable, and maintainable codebases. An easy way to explain is that "imperative" programming means that the computer gets a list of commands and executes them in order, when "procedural programming" (which is also imperative) allows splitting those instructions into procedures (or functions).

1c: Functional programming focuses on expressions; Procedural programming focuses on statements which means that the functional programming is a programming paradigm that treats computation as the evaluation of mathematical functions and avoids state and mutable data. It emphasizes the application of functions, in contrast with the procedural programming style that emphasizes changes in state.

Question 2:

According to the written code here is the step that the programmer did to get the getDiscountedProductAveragePrice:

First: He went through the inventory array filter out only the discounted products.

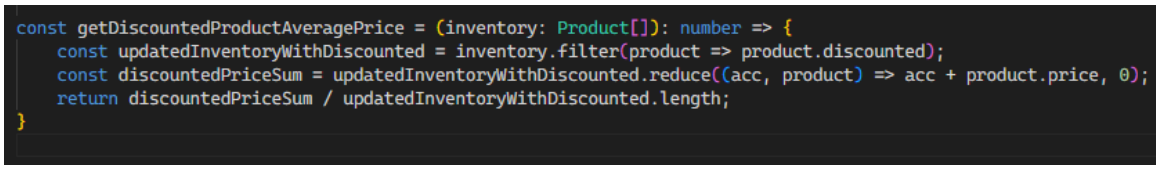
```
const updatedInventoryWithDiscounted = inventory.filter(product =>
product.discounted);
```

Second: He calculates the sum of the prices in the new inventory.

```
const discountedPriceSum = updatedInventoryWithDiscounted.reduce((acc, product) =>
acc + product.price, 0);
```

Third: return sum of the inventory after filter size.

```
return discountedPriceSum / updatedInventoryWithDiscounted.length;
```

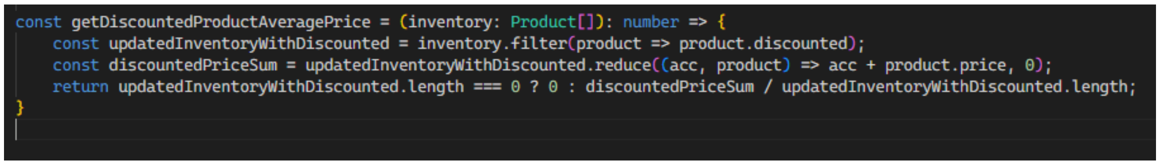


```
const getDiscountedProductAveragePrice = (inventory: Product[]): number => {
    const updatedInventoryWithDiscounted = inventory.filter(product => product.discounted);
    const discountedPriceSum = updatedInventoryWithDiscounted.reduce((acc, product) => acc + product.price, 0);
    return discountedPriceSum / updatedInventoryWithDiscounted.length;
}
```

(Note :handling with the if statement that check if the inventor length =0 could add here for example:)

return updatedInventoryWithDiscounted.length === 0 ? 0 : discountedPriceSum / updatedInventoryWithDiscounted.length;

so here is the final answer:



```
const getDiscountedProductAveragePrice = (inventory: Product[]): number => {
    const updatedInventoryWithDiscounted = inventory.filter(product => product.discounted);
    const discountedPriceSum = updatedInventoryWithDiscounted.reduce((acc, product) => acc + product.price, 0);
    return updatedInventoryWithDiscounted.length === 0 ? 0 : discountedPriceSum / updatedInventoryWithDiscounted.length;
}
```

Question 3:

(a) (x, y) => x.some(y)

According to the function structure its takes 2 arguments (the first is array and the second is function) the function check whether at least one element in the array passes the test implemented by the provided function so:

-x: T[]

-y: (value: T) => Boolean

So they type is: <T>(x: T[], y: (value: T) => boolean) => boolean

(b) x => x.reduce((acc, cur) => acc + cur, 0)

According to the function the array elements must be numbers cause we see “acc+cur” so:

-x: number[]

So the type is : (x: number[]) => number

(c) (x, y) => x ? y[0] : y[1]

According to the function structure its takes 2 arguments and return whether the first or the second element of an array based on x value(as we see y array should have at least two elements) so :
-x: boolean

-y: [T, T]

So the type is: <T>(x: boolean, y: [T, T]) => T