# TypeScript – Regular Exam – 27 April 2024

## Print and Sum

Write a program that:

* Receives a **start number** as a first parameter
* Receives an **end number** as a second parameter
* Prints all the numbers from the **given start** to the **given end (inclusively)**
* Prints the sum of the printed numbers

**Note:** All the numbers will be integers.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 5,  10 | 5 6 7 8 9 10  Sum: 45 |
| 0,  26 | 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26  Sum: 351 |
| 50,  60 | 50 51 52 53 54 55 56 57 58 59 60  Sum: 605 |

## City Taxes

Create a function **cityTaxes()**. It should receive a city's **name** (string), **population** (number) and **treasury** (number)as arguments. You will need to set as **properties** of an **object** and **return** the object. In addition to the input parameters, the object must have a property taxRate with an initial value of **10**, and three **methods** for managing the city:

* collectTaxes() **-** increase the **treasury** by population \* taxRate
* applyGrowth(percentage) **-** increase the population by the **given percentage**
* applyRecession(percentage) **-** Decrease the treasury by the **given percentage**

Round down the values after each calculation.

### Input

Your solution will receive three **valid** parameters. The methods that expect parameters will be tested with valid input.

### Output

Return an **object** as described above. The methods of the object modify the object and don’t return anything.

|  |  |
| --- | --- |
| **Code** | **Output** |
| const city =  cityTaxes('Tortuga',  7000,  15000);  console.log(city); | {  name: 'Tortuga',  population: 7000,  treasury: 15000,  taxRate: 10,  collectTaxes: [Function: collectTaxes],  applyGrowth: [Function: applyGrowth],  applyRecession: [Function: applyRecession]  } |
| **Testing with code** | |
| **Code** | **Output** |
| const city =  cityTaxes('Tortuga',  7000,  15000);  city.collectTaxes();  console.log(city.treasury);  city.applyGrowth(5);  console.log(city.population); | 85000  7350 |

# 3. Vending Machine

Your task is to create a repository that stores drinks by creating the classes described below.

### Drink

You need to create a class **Drink** with the following properties:

* **name – string**
* **price - number**
* **volume - number**

The class **constructor** should receive **name, price** and **volume**.

Override the **toString()** method in the following format:  
**"Name: {name}, Price: ${price}, Volume: {volume} ml"**

### VendingMachine

Create a class **VendingMachine** that has **drinks** (an array that stores the **Drink entities**). All entities inside the repository have the **same properties**. The **VendingMachine** class should have the following **properties**:

* **buttonCapacity - number**
* **drinks – Drink[]**

The class **constructor** should receive **buttonCapacity**, also it should initialize the **drinks** as an empty collection.

Implement the following features:

* **Method addDrink(drink: Drink)** – **adds** an **entity** to the **collection** of Drinks, **if** the **capacity** **allows it**.
* **Method removeDrink(name: string)** – **removes** a **drink by** **the** **given name,** if such **exists**, and **returns true** if it is removed, otherwise – **false**
* **Method getLongest()** – **returns** the **Drink** as **string** in the format **of the overriden toString() method** with the **biggest value** of **volume** property
* **Method getCheapest()**– **returns** the **Drink** as **string** in the format **of the overriden toString() method** with the **lowest value** of **price** property
* **Method buyDrink(name: string)** – **returns a string** in the format **of the overriden toString() method**
* **Method getCount** - **returns** the number of **drinks**, **available** in the vending machine
* **Method report()** – **returns** a **string** in the following **format**:
  + **"Drinks available:  
    {Drink1}  
    {Drink2}  
    (…)"**

### Constraints

* The **names** of the drinks will be **always unique**.
* You will always have drinks added before receiving methods, manipulating the drinks in the VendingMachine.

### Examples

This is an example of how the **VendingMachine** class is **intended to be used**.

|  |
| --- |
| **Sample code usage** |
| import { Drink } from "./Drink";  import { VendingMachine } from "./VendingMachine";  function main() {  //Initialize the repository (VendingMachine)  const vendingMachine = new VendingMachine(6);  //Initialize Entity (Drink)  const tea = new Drink("Tea", 1.5, 300);  const coffee = new Drink("Coffee", 2.0, 120);  const hotChocolate = new Drink("Hot Chocolate", 2.5, 200);  const latte = new Drink("Latte", 3.5, 220);  const cappuccino = new Drink("Cappuccino", 2.8, 180);  const mocha = new Drink("Mocha", 2.1, 150);  const herbalTea = new Drink("Herbal Tea", 1.75, 300);  //Get Count  console.log(vendingMachine.getCount);  //0  //Add Drinks  vendingMachine.addDrink(tea);  vendingMachine.addDrink(coffee);  vendingMachine.addDrink(hotChocolate);  vendingMachine.addDrink(latte);  vendingMachine.addDrink(cappuccino);  vendingMachine.addDrink(mocha);  //Try to add drinks when the capacity is full  vendingMachine.addDrink(herbalTea);  //Get Count  console.log(vendingMachine.getCount);  //6  //Remove Drink  console.log(vendingMachine.removeDrink("Herbal Tea"));  //false  console.log(vendingMachine.removeDrink("Tea"));  //true  //Get Longest Drink  console.log(vendingMachine.getLongest());  //Name: Latte, Price: $3.5, Volume: 220 ml  //Get Cheapest Drink  console.log(vendingMachine.getCheapest());  //Name: Coffee, Price: $2.0, Volume: 120 ml  //Buy a specific Drink  console.log(vendingMachine.buyDrink("Cappuccino"));  //Name: Cappuccino, Price: $2.8, Volume: 180 ml  //Drinks Report  console.log(vendingMachine.report());  //Drinks available:  //Name: Coffee, Price: $2.0, Volume: 120 ml  //Name: Hot Chocolate, Price: $2.5, Volume: 200 ml  //Name: Latte, Price: $3.5, Volume: 220 ml  //Name: Cappuccino, Price: $2.8, Volume: 180 ml  //Name: Mocha, Price: $2.1, Volume: 150 ml  }  main(); |