```
'use strict';

// Data needed for a later exercise
const flights =
  '_Delayed_Departure;fao93766109;txl2133758440;11:25+
    _Arrival;bru0943384722;fao93766109;11:45+_Delayed_
    Arrival;hel7439299980;fao93766109;12:05+_Departure
    ;fao93766109;lis2323639855;12:30';

// Data needed for first part of the section
const restaurant = {
  name: 'Classico Italiano',
  location: 'Via Angelo Tavanti 23, Firenze, Italy',
  categories: ['Italian', 'Pizzeria', 'Vegetarian',
    'Organic'],
  starterMenu: ['Focaccia', 'Bruschetta', 'Garlic
    Bread', 'Caprese Salad'],
  mainMenu: ['Pizza', 'Pasta', 'Risotto'],
  // Lecture 103: Array Destructuring
  order: function (starterIndex, mainIndex) {
    return [this.starterMenu[starterIndex],
      this.mainMenu[mainIndex]];
  },
  // Lecture 104: Object Destructuring
  // Notice that the time and mainIndex are not in
    the same order.
  // Because order doesn't matter when it comes to
    objects.
  // You can specify default arguments as well.
  orderDelivery: function ({ starterIndex, time,
    mainIndex, address, hi = 2 }) {
    // console.log(
    //   `Order received:
      ${this.starterMenu[starterIndex]} and
      ${this.mainMenu[mainIndex]} will be delivered
      to ${address} at ${time}.`
    // );
    //console.log(hi);
  },

  ////// Lecture 106: Rest Pattern and parameters
  orderPizza: function (mainIngredient,
```

```
      ...otherIngredients) {
      //console.log(mainIngredient, otherIngredients);
    },

    openingHours: {
      thu: {
        open: 12,
        close: 22,
      },
      fri: {
        open: 11,
        close: 23,
      },
      sat: {
        open: 0, // Open 24 hours
        close: 24,
      },
    },
  };

  //////// Lecture 103: Array Destructuring
  let arr = [2, 3, 4];
  const a1 = arr[0];
  const b1 = arr[1];
  const c1 = arr[2];
  // With destructuring, we can declare all these
    variables at the same time.
  const [a2, b2, c2] = arr; // this is a destructuring
    assignment.
  //console.log(a1, b1, c1, a2, b2, c2);

  // If you do this, don't forget to declare the arrays
    using const.
  // You are NOT destroying the original array in this
    process.

  // You need not unpack everything at a go
  let [first, second] = restaurant.categories;
  //console.log(first, second); // Italian, Pizzeria

  // What if you would like to skip the second element?
  [first, , second] = restaurant.categories;
```

```
68  //console.log(first, second); // Italian, Vegetarian
69
70  // let's say the owner wants to switch the main
      category and secondary category.
71  let [main, , secondary] = restaurant.categories;
72  //console.log(main, secondary);
73
74  // Without Destructuring
75  const temp = main;
76  main = secondary;
77  secondary = temp; // we create a temporary value of
      main using temp
78
79  // With Destructuring
80  [secondary, main] = [secondary, main];
81  //console.log(main, secondary);
82
83  // We can also have a function return an array,
84  // and then immediately destruct the array to give
      many variables.
85  let starter;
86  [starter, main] = restaurant.order(2, 0);
87  //console.log(starter, main);
88
89  // Destructuring a nested array
90  const nested = [2, 4, [5, 6]];
91  let [one, , [, four]] = nested;
92  //console.log(one, four);
93
94  ////// Lecture 104: Destructuring objects
95  // Note that the names here will have to be the same
      as the property name.
96  const { name, openingHours, categories } = restaurant;
97  //console.log(name, openingHours, categories);
98
99  // But we can specify new names
100 // PropertyName: newvariable name
101 const {
102   name: restaurantName,
103   openingHours: hours,
104   categories: tags,
105 } = restaurant;
```

```
//console.log(restaurantName, hours, tags);

// we can also create default values for the case that the property doesn't exist on the object.
// So the PropertyName:NewVariableName = Default Value (if it doesn't exist in the object.)
const { menu: mains = [], starterMenu: starters = [] } = restaurant;
//console.log(mains, starters);

// Mutating variables
let a = 111;
let b = 999;
const obj = { a: 23, b: 7, c: 14 };
// {a,b} = obj; // syntax error!
// Do this instead - wrap it into parenthesis
({ a, b } = obj);
//console.log(a, b);

// Nested Objects
let { fri } = openingHours;
//console.log(fri);

({
  fri: { open, close },
} = openingHours);
//console.log(open, close);

// Arguments to Function through object Destructuring
// Usually in JS, there are a lot of different function arguments and it is difficult to remember the order of parameters.

// Instead of defining the parameters manually, we can just pass an object into the function as an argument, and the function will then immediately destructure the object.
restaurant.orderDelivery({
  time: '22:30',
  address: 'Via del Sole, 21',
  mainIndex: 2,
  starterIndex: 2.
```

```
  });

  ////// Lecture 105: The Spread Operator (...)
  const array = [7, 8, 9];
  const badNewArr = [1, 2, array[0], array[1],
    array[2]];
  //console.log(badNewArr);
  // Spread Operator (ES6)
  const newArray = [1, 2, ...array];
  //console.log(newArray);
  //console.log(...newArray); // individual elements
    within the array.

  ////// Lecture 106: Rest Pattern and parameters
  // It also has three dots, and it does basically the
    same thing as the spread operator.
  // It's goal is to pack elements into the array.
  // Spread => dots on the right hand side of
    ASSIGNMENT operator
  arr = [1, 2, ...array];
  // If we are destructuring the array,
  // REST pattern ==> LEFT side of the assignment
    operator
  let others;
  [a, b, ...others] = [1, 2, 3, 4, 5];
  //console.log(a, b, others); // the REST pattern
    basically collects stuff that are not used in the
    destructuring assignment.
  // So in this case, others = [3,4,5];

  // Consider this - let's say we would like the pizza
    and the risotto from the menus in the restaurant.
  // From right to left, we can use the SPREAD operator
    to unzip all the items in the main and starter menu
    to get a single array, and then extract the pizza
    and the risotto from the mainMenu, and zip the rest
    of the menu items into the others array using the
    REST pattern.
  let pizza, risotto;
  [pizza, , risotto, ...others] = [
    ...restaurant.mainMenu,
    ...restaurant.starterMenu,
```

```
    ...restaurant.starterMenu,
  ];
  //console.log(pizza, risotto, others);

  /// Rest Pattern using parameters
  // These are called rest parameters.
  // The rest syntax is taking multiple numbers or values and then packs them all into 1 array.
  // This function can accept any number of parameters.
  const add = function (...numbers) {
    console.log(numbers);
    let sum = 0;
    for (let i = 0; i < numbers.length; i++) sum += numbers[i];
    console.log(sum);
  };
  //add(2, 3);
  //add(5, 3, 7, 2);
  //add(8, 2, 5, 3, 2, 1, 4);

  // Ordering pizza
  // We can call this function with any number of arguments.
  restaurant.orderPizza('mushrooms', 'onion', 'olives', 'spinach');
  restaurant.orderPizza('mushrooms'); // others. => an empty array.

  /////// Lecture 107: Short Circuiting (&& and ||)

  // The OR operator doesn't always have to be a boolean.
  // They can use any data type.
  // They can return any data type (IN JAVASCRIPT!)
  // they do something called short circuit evaluation.
  console.log(3 || 'Jonas'); // ==> 3
  // If they take in two values that are not booleans, then they will return a non-boolean.
  // Short Circuiting for the OR operator means that if the first value is a value representing a true, it will immediately return that first value.
  // If the first operand is true, then the other operand will not even be considered
```

```
    operand will not even be considered.
console.log('' || 'Jonas'); // ==> Jonas
console.log(true || 0); // ==>  true
console.log(undefined || null); // ==> null - if both values are false, there is NO short circuiting, so it returns null.
console.log(undefined || 0 || '' || 'Hello' || 23 || null); // ==> hello
// Because 'Hello' is true, it will shortcircuit the entire evaluation and returns 'Hello'

console.log(3 && 'Jonas'); // ==> Note that Jonas is returned.
console.log(undefined && 'Jonas'); // ==> undefined.

/////// Lecture 108: The Nullish Coalescing Operator (??) (ES 2020)
const guestCorrect = restaurant.numGuests ?? 10;
console.log(guestCorrect);
// The nullish coalescing operator works with the idea or with the concept of nullish values instead of falsy values.
// Nullish values are null and undefined (it does not include a zero or the empty string.) This means that the zero or empty string are truthy values as well.

/////// Lecture 109: Coding Challenge 1

```