# Exercises: Introduction to TypeScript

**Note: You should use types in your code as much as possible.**

## Even Sum

Write a simple function that **calculates** whether the **sum of 3 numbers is even**. Your function should receive **three** **integers (whole) numbers**, calculate their sum and return a **Boolean** value that is **true** if the sum is even or **false** otherwise.

### Examples

|  |  |
| --- | --- |
| **input** | **output** |
| 1,2,3 | true |

|  |  |
| --- | --- |
| **input** | **output** |
| 2,2,3 | false |

## Day of the Week

Write a function that receives a number and prints the equivalent **day of the week**, using the following table:

|  |  |
| --- | --- |
| **Number** | **Day** |
| 1 | Monday |
| 2 | Tuesday |
| 3 | Wednesday |
| 4 | Thursday |
| 5 | Friday |
| 6 | Saturday |
| 7 | Sunday |
| Any other number | error |

* The **input** comes as a single number argument.
* Avoid using conditional statements like **if…else** and **switch** to determine the **Day**, instead try using an **enum**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1 | 'Monday' |
| 5 | 'Friday' |
| -1 | 'error' |

## Format Person

Write a function that takes in a tuple with exactly two elements:

1. The first parameter is a **string** and represents the person’s **name.**
2. The second parameteris a **number** and represents the person’s **age**.

In case a correct tuple is passed, your function should return a string with the following format:

**'Hello, my name is <name> and my age is <age>'**.In case an invalid tuple is passed, TS should mark it as an error.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| ['Ivan', 20] | Hello, my name is Ivan and my age is 20 |
| ['Joro', 30] | Hello, my name is Joro and my age is 30 |
| ['Ivan', 20, 'Ivanov'] | **TS Error** |
| ['Joro', '25'] | **TS Error** |
| [] | **TS Error** |

## Convert Arrays

Write a function that takes in an array of strings and returns a tuple, where:

1. The first parameter is a **string** and represents the concatenated text from the entire array**.**
2. The second parameteris a **number** and represents the length of the concatenated text.

The **input** comes as an **array of string** elements.

The **output** is returned as a tuple of exactly 2 elements.

### Examples

|  |
| --- |
| **Input** |
| ['How', 'are', 'you?'] |
| **Output** |
| ['Howareyou?', 10] |

|  |
| --- |
| **Input** |
| ['Today', ' is', ' a ', 'nice', ' ', 'day for ', 'TypeScript'] |
| **Output** |
| ['Today is a nice day for TypeScript', 34] |

## Summarize Person

Write a function that summarizes information about a person and returns a prefixed tuple with the information. Your function should accept **7 parameters:**

1. **id: number - required**
2. **firstName: string – required**
3. **lastName: string – required**
4. **age: number – required**
5. **middleName: string – optional**
6. **hobbies: string[] – optional**
7. **workInfo: [string, number] - optional**

Summarize the information and return it in a tuple of exactly 5 elements, using the following logic:

1. The first element of the tuple should be of type number and hold the **id**
2. The second element of the tuple should be of type string and should hold the full name of the person:
   1. If the middleName was not provided or it is an empty string, the full name should have the following structure **'{firstName} {lastName}'**
   2. If the middle was provided and is not the empty string, the full name should have the following structure **'{firstName} {middleName} {lastName}'**
3. The third element of the tuple should be of type number and hold the **age**
4. The fourth element of the tuple should be of type string and hold the hobbies of the person:
   1. If hobbies was not provided or the person has 0 hobbies, show **'-'**
   2. If the person has 1 or more hobbies, show a string representing the hobies joined by **', '**
5. The fifth element of the tuple should be of type string and represent the person’s job and salary
   1. If the workInfo tuple was not provided, show **'-'**
   2. If the workInfo tuple was provided show the position and salary in the format **'{position} -> {salary}'**

Your function should always **return a tuple of 5 elements.**

### Examples

|  |
| --- |
| **Input** |
| 12, 'Eliot', 'Des', 20, 'Braylen', ['tennis', 'football', 'hiking'], ['Sales Consultant', 2500] |
| **Output** |
| [12, 'Eliot Braylen Des', 20, 'tennis, football, hiking', 'Sales Consultant -> 2500'] |
| **Input** |
| 20, 'Mary', 'Trent', 25, undefined, ['fitness', 'rowing'] |
| **Output** |
| [20, 'Mary Trent', 25, 'fitness, rowing', '-'] |
| **Input** |
| 21, 'Joseph', 'Angler', 28 |
| **Output** |
| [21, 'Joseph Angler', 28, '-', '-'] |
| **Input** |
| 21, 'Kristine', 'Neva', 23, '' |
| **Output** |
| [21, 'Kristine Neva', 23, '-', '-'] |

## Reversed Day of the Week

Write a function that prints a number between 1 and 7 when a **day of the week** is passed to it as a string and an **error message** if the string is **not recognized**.

* The **input** comes as a single-string argument.
* Avoid using conditional statements like **if...else** and **switch** to get the **number**, instead try using an **enum**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 'Monday' | 1 |
| 'Friday' | 5 |
| 'Invalid' | error |

## Unknown Response

Write a function that takes in exactly one parameter of **unknown** type, a response object that may or may not contain a value and returns a string based on its contents:

* The parameter provided will **always be a valid object**, that has **code** and **text** properties
* If the provided object has a **value** property of **type string**, return the **value**.
* Otherwise return **'-'**.
* **Hint**: If you know information that TypeScript cannot deduce, it may be acceptable to ignore type checks.

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| { code: 200, text: 'Ok', value: [1, 2, 3] } | - |
| { code: 301, text: 'Moved Permanently', value: 'New Url' } | New Url |
| { code: 201, text: 'Created', value: { name: 'Test', age: 20 } } | - |
| { code: 201, text: 'Created', value: 'Object Created' } | Object Created |
| { code: 404, text: 'Not found' } | - |
| { code: 500, text: 'Internal Server Error' } | - |

## Custom Type Guard

Write a **type predicate function** that receives exactly **one parameter** of type **unknown** and **narrows** it down to a **string[]**, but only if the array has at least 1 element.

Your type predicate function should return a single **Boolean** value:

* If the provided parameter was a valid string[] with at least 1 entry , it returns true.
* Otherwise returns **false**.

Your function should be usable as a **type guard**:

|  |
| --- |
| **Sample code** |
| let **arr**: unknown = …;  if(**isNonEmptyStringArray**(arr)) {      console.log(arr.length); // allowed  } |

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| {} | false |
| { test: 'one' } | false |
| [] | false |
| undefined | false |
| null | false |
| [12, 13] | false |
| ['test', 123] | false |
| ['a', 'b', 'c'] | true |

## Friday the 13th

Write a function that takes in a **single** **parameter** an **unknown[]**, and then:

* **Type narrows** each value of the array that is a **valid date** and **falls on a Friday the 13th**
* Prints each date that meets the above requirement in the following **format '<day>-<month>-<year>'** where **month** is the full name of the month.

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| [      {},      new Date(2025, 4, 13),      null,      new Date(2025, 5, 13),      '13-09-2023',      new Date(2025, 6, 13),  ] | 13-June-2025 |
| [      new Date(2024, 0, 13),      new Date(2024, 1, 13),      new Date(2024, 2, 13),      new Date(2024, 3, 13),      new Date(2024, 4, 13),      new Date(2024, 5, 13),      new Date(2024, 6, 13),      new Date(2024, 7, 13),      new Date(2024, 8, 13),      new Date(2024, 9, 13),      new Date(2024, 10, 13),      new Date(2024, 11, 13)  ]; | 13-September-2024  13-December-2024 |