Chapter 4 – Union Types

1. Introduction

A picture containing shape

Description automatically generated

* 1. any type is very unspecific.
  2. string. This type is very specific
  3. Union Types strikes balance between the two
     1. Allows to combine multiple specific types

1. Defining Unions
   1. Unions allow us to define multiple allowed *type members*
   2. Separating each type member with a vertical line character **|**

Example

|  |
| --- |
| let ID: string | number;    // number  ID = 1;    // or string  ID = '001';    console.log(`The ID is ${ID}.`); |

Exercise

|  |
| --- |
| function printNumsAndStrings(statement: string | number) {  console.log(`ℹ️ LOG:: ${statement}`);  }  printNumsAndStrings('hello!');  printNumsAndStrings(4); |

1. Type Narrowing
   1. Type with unions give flexibility, but there is more to consider
      1. We may want to perform different logic that does one thing for string and another for numbers 🡪 done using **Type Guard**

Example (Type Guard)

|  |
| --- |
| function formatValue(value: string | number) {  // Write your code here  }  formatValue('Hiya');  formatValue(42);  unction getMarginLeft(margin: string | number) {  // margin may be a string or number here    if (typeof margin === 'string') {  // margin must be a string here  return margin.toLowerCase();  }  } |

Exercise

|  |
| --- |
| function formatValue(value: string | number) {  // Write your code here  if (typeof value == "string") {  console.log(value.toLowerCase());  }  if (typeof value == "number") {  console.log(value.toFixed(2));  }  }  formatValue('Hiya');  formatValue(42); |

1. Inferred Union Return Types
   1. Typescript is able to infer types in many cases
   2. One of the useful case is return type of function
      1. If successful, return **Book** type
      2. If error, return **string** type

Exercises

|  |
| --- |
| function getBook() {  try {  return getBookFromServer();  } catch (error) {  return `Something went wrong: ${error}`;  }  } |