Chapter 3 – Complex Types

1. Introduction
   1. TypeScript makes it very easy to keep track of element types in arrays
   2. Manual type-checking is needlessly difficult and adds complications

|  |
| --- |
| let customersArray = ['Custy Stomer', 'C. Oostomar', 'C.U.S. Tomer', 3432434, 'Custo Mer', 'Custopher Ustomer', 3432435, 'Kasti Yastimeur'];  //Write Your Code here:  function checkCustomersArray() {  for (el of customersArray) {  if (typeof el != 'string') {  console.log(`Type error: ${el} should be a string!`);  }  }  }  function stringPush(val) {  if (typeof val != 'string') {  return;  }  customersArray.push(val);  } |

1. Array Type Annotations
   1. Type annotation for array types is fairly straightforward: [] after the element type
   2. Throw errors when elements of the wrong type are added

|  |
| --- |
| let names: string[] = ['Danny', 'Samantha'];  let names: string[] = ['Damien'];  names.push(666) // Type Error! |

* 1. Alternate - use the Array<T> syntax, where T stands for the type.

|  |
| --- |
| let names: Array<string> = ['Danny', 'Samantha']; |

Exercise

|  |
| --- |
| // Arrays:  let bestNumbers: number[] = [7,77,4];  let bestLunches: string[] = ['chicken soup', 'non-chicken soup'];  let bestBreakfasts: string[] = ['fasting', 'oatmeal', 'tamago kake gohan', 'any kind of soup'];  let bestBooleans: boolean[] = [true, false]; |

1. Multi-dimensional Arrays
2. Done using [][]

|  |
| --- |
| let arr: string[][] = [['str1', 'str2'], ['more', 'strings']]; |

Exercise

|  |
| --- |
| // Arrays:  let bestNumbers: number[] = [7,77,4];  let bestLunches: string[] = ['chicken soup', 'non-chicken soup'];  let bestBreakfasts: string[]= ['fasting', 'oatmeal', 'tamago kake gohan', 'any kind of soup'];  let bestBooleans: boolean[] = [true, false];  // Multidimensional Arrays:  let bestMealPlan: string[][] = [bestLunches, bestBreakfasts, ['baked potato', 'mashed potato']];  let bestBooleansTwice: boolean[][] = [bestBooleans, bestBooleans];  let numbersMulti: number[][][] = [ [[1],[2,3]], [[7],bestNumbers] ]; |

1. Tuples
   1. Tuple acts like arrays
      1. Has .length properties
      2. Can access element using [index]
      3. But can’t assign an array to a tuple variable

|  |
| --- |
| let tup: [string, string] = ['hi', 'bye'];  let arr: string[] = ['there','there'];  tup = ['there', 'there']; // No Errors.  tup = arr; // Type Error! An array cannot be assigned to a tuple. |

* 1. Tuple types specify both the lengths and the orders of compatible tuples, and will cause an error if either of these conditions are not met

|  |
| --- |
| let numbersTuple: [number, number, number] = [1,2,3,4]; // Type Error! numbersTuple should only have three elements.  let mixedTuple: [number, string, boolean] = ['hi', 3, true] // Type Error! The first elements should be a number, the second a string, and the third a boolean. |

Exericse

|  |
| --- |
| let favoriteCoordinates:[  number, number, string,  number, number, string, number];      favoriteCoordinates = [17, 45, 'N', 142, 30, 'E', -100];  favoriteCoordinates[6] = -6.825; |

1. Array Type Interface
   1. Typescript infers to type[] before tuple
   2. TypeScript can infer variable types from initial values and return statements

|  |
| --- |
| let examAnswers= [true, false, false];  examAnswers[3] = true; // No type error. |

* 1. We also get the same kind of type inference when we use the .concat() method
     1. **concatResult** as an array of numbers, not a tuple.

|  |
| --- |
| let tup: [number, number, number] = [1,2,3];  let concatResult = tup.concat([4,5,6]); // concatResult has the value [1,2,3,4,5,6]. |

Exercise

|  |
| --- |
| // Don't change this part:  let dogTup: [string, string, string, string] = ['dog', 'brown fur', 'curly tail', 'sad eyes'];  // Your code goes here:  let myArr = dogTup.concat("");  myArr[50] = "not a dog"; |

1. Rest Parameters
   1. Assigning types to [rest parameters](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/rest_parameters) is similar to assigning types to arrays
      1. Here the rest parameters are, ‘h’, ‘h’, ‘H’, ‘H’, ‘H’, ‘H’, ‘!’

|  |
| --- |
| function smush(firstString, ...otherStrings){  let output = firstString;  for(let i = 0; i < otherStrings.length; i++){  output = output.concat(otherStrings[i]);  }  return output;  }  smush('a','h','h','H','H','H','!','!'); // Returns: 'ahhHHH!!'. |

* 1. The above is not safe.
     1. Allows other types in **otherStrings**
  2. Is made safe using typescript

|  |
| --- |
| function smush(firstString, ...otherStrings: string[]){  /\*rest of function\*/  } |

Exercise

|  |
| --- |
| function addPower(p: number, ...numsToAdd: number[]): number {  let answer = 0;  for(let i = 0; i < numsToAdd.length; i++){  answer += numsToAdd[i] \*\* p;  }  return answer;  }  addPower('a string', 4, 5, 6); |

1. Spread Syntax
   1. Assigning types to [rest parameters](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/rest_parameters) is similar to assigning types to arrays
      1. Here the rest parameters are, ‘h’, ‘h’, ‘H’, ‘H’, ‘H’, ‘H’, ‘!’
   2. If not used, is awkward to read

|  |
| --- |
| gpsNavigate(40, 43.2, 'N', 73, 59.8, 'W', 25, 0, 'N', 71, 0, 'W') |

* 1. We can use tuple variables that represent the starting and ending coordinates to improve readability of code

|  |
| --- |
| let codecademyCoordinates: [number, number, string, number, number, string] = [40, 43.2, 'N', 73, 59.8, 'W'];  let bermudaTCoordinates: [number, number, string, number, number, string] = [25, 0 , 'N' , 71, 0, 'W'];  gpsNavigate(...codecademyCoordinates, ...bermudaTCoordinates);  // And by the way, this makes the return trip really convenient to compute too:  gpsNavigate(...bermudaTCoordinates, ...codecademyCoordinates);  // If there is a return trip . . . |

Exercise

|  |
| --- |
| function performDanceMove(moveName:string, moveReps:number, hasFlair:boolean):void{  console.log(`I do the ${moveName} ${moveReps} times !`);  if(hasFlair){  console.log('I do it with flair!');  }  }  let danceMoves: [string, number, boolean][] = [  ['chicken beak', 4, false],  ['wing flap', 4, false],  ['tail feather shake', 4, false],  ['clap', 4, false],  ['chicken beak', 4, true],  ['wing flap', 4, true],  ['tail feather shake', 4, true],  ['clap', 4, true],  ];  for (let danceMove of danceMoves) {  performDanceMove(...danceMove);  } |

Quiz

1. The code snippet prints the content of praises, a two-dimensional array of random strings, on the console. Fill in the code to initialize the variable, praises, with the correct value so that it only has five random messages.

Text

Description automatically generated

* 1. []

1. Which of these variable declarations is a tuple with correctly-assigned values?

Graphical user interface

Description automatically generated

* 1. B: const heading: [string, string, string, string] = ['Product', 'Price', 'Sale', 'Quantity']

1. Fill in the code to define a generic type for an array in TypeScript.

A picture containing text, screen, screenshot, hand

Description automatically generated

* 1. Array, <, >, [
  2. ]

1. Fill in the code with the correct return type for the function mySort().

Graphical user interface, text

Description automatically generated

* 1. Any[]

1. Which of the following is a valid function call for policy()?

Graphical user interface

Description automatically generated

* 1. A: Const adult: [string, number, Boolean] = [“Don”, 21, false];

1. Fill in the code to match the array types with the array values.

Graphical user interface, diagram, text

Description automatically generated with medium confidence

* 1. String[][], [‘Cereal’, ‘Toast’]
  2. String[][][], [lunch]

1. Which of these statements does NOT describe a tuple correctly?

Graphical user interface, website

Description automatically generated

* 1. D: An array can be assigned to a tuple when their elements are of the same type

1. Which of the following code snippets will generate a TypeScript type error?

Graphical user interface

Description automatically generated

* 1. A: let areacodes: number[] = [617, 413, 857, null];