step10_tuples

Tuples

Tuples in TypeScript are a special type of array that allow you to store a fixed number of elements with specific types. Unlike regular arrays, tuples can contain elements of different types and are defined with a specific structure.

Defining Tuples

You can define a tuple by specifying the types of its elements in square brackets.

Example

```
let user: [string, number] = ["Alice", 30];
```

Accessing Tuple Elements

You can access tuple elements using index notation, just like with arrays.

```
console.log(user[0]); // Output: "Alice"

console.log(user[1]); // Output: 30
```

Modifying Tuple Elements

You can modify elements in a tuple by assigning new values, as long as the types match.

```
user[1] = 31; // Update age

console.log(user); // Output: ["Alice", 31]
```

Using Tuples with Functions

Tuples can be used as function parameters and return types, allowing for structured data handling.

```
function getUserInfo(): [string, number] {
   return ["Bob", 25];}

let userInfo = getUserInfo();

console.log(userInfo); // Output: ["Bob", 25]
```

Destructuring Tuples

You can destructure tuples to extract values more conveniently.

```
let [name, age] = user;
console.log(name); // Output: "Alice"
console.log(age);  // Output: 31
```

Optional and Rest Elements in Tuples

Tuples can include optional elements or rest elements using `?` or `....`

Optional Elements:

```
let userWithOptional: [string, number?] = ["Charlie"];
userWithOptional[1] = 28; // Optional age
console.log(userWithOptional); // Output: ["Charlie", 28]
```

Rest Elements:

```
let tupleWithRest: [string, ...number[]] = ["Scores", 100, 90, 85];
console.log(tupleWithRest); // Output: ["Scores", 100, 90, 85]
```

example from repo.

```
//Tuple types have the advantage that you can accurately describe the type of an array of mixed types
var tuple: [number, string] = [1, "bob"];
var secondElement = tuple[1];  // secondElement now has type 'string'
// Typically an array contains zero to many objects of a
// single type. TypeScript has special analysis around
// arrays which contain multiple types, and where the order
// in which they are indexed is important.
// These are called tuples. Think of them as a way to
// connect some data, but with less syntax than keyed objects.
// You can create a tuple using JavaScript's array syntax:
```

```
const failingResponse = ["Not Found", 404];


// but you will need to declare its type as a tuple.
const passingResponse: [string, number] = ["{}", 200];
// If you hover over the two variable names you can see the
// difference between an array ( (string | number)[] ) and
// the tuple ( [string, number] ).
// As an array, the order is not important so an item at
// any index could be either a string or a number. In the
// tuple the order and length are guaranteed.
if (passingResponse[1] === 200) {
  const localInfo = JSON.parse(passingResponse[0]);
  console.log(localInfo);}
// This means TypeScript will provide the correct types at
// the right index, and even raise an error if you try to
// access an object at an un-declared index.
passingResponse[2];
// A tuple can feel like a good pattern for short bits of
// connected data or for fixtures.
type StaffAccount = [number, string, string, string?];
const staff: StaffAccount[] = [
  [0, "Adankwo", "adankwo.e@"],
  [1, "Kanokwan", "kanokwan.s@"],
  [2, "Aneurin", "aneurin.s@", "Supervisor"],];
// When you have a set of known types at the beginning of a
// tuple and then an unknown length, you can use the spread
// operator to indicate that it can have any length and the
// extra indexes will be of a particular type:
```

```
type PayStubs = [StaffAccount, ...number[]];

const payStubs: PayStubs[] = [
  [staff[0], 250],
  [staff[1], 250, 260],
  [staff[0], 300, 300, 300],
];

const monthOnePayments = payStubs[0][1] + payStubs[1][1] + payStubs[2][1];

const monthTwoPayments = payStubs[1][2] + payStubs[2][2];

const monthThreePayments = payStubs[2][2];

// You can use tuples to describe functions which take

// an undefined number of parameters with types:

declare function calculatePayForEmployee(id: number, ...args: [...number[]]): number;

calculatePayForEmployee(staff[0][0], payStubs[0][1]);

calculatePayForEmployee(staff[1][0], payStubs[1][1], payStubs[1][2]);

//

// https://www.typescriptlang.org/docs/handbook/release-notes/typescript-3-0.html#tuples-in-rest-parameters-and-spread-expressions

// https://auth0.com/blog/typescript-3-exploring-tuples-the-unknown-type/
```