TypeScript Cheatsheet

TypeScript is a superset of JavaScript that adds static typing and other features to help developers write more robust and maintainable code. It provides type annotations, interfaces, enumerations, generics, and other tools to help catch errors at compile-time rather than run-time. This cheatsheet provides an overview of some of the key features of TypeScript, along with code blocks for variables, functions, loops, conditionals, file manipulation, and resources for further learning.

Variables

```typescript
// Declaring a variable with a type annotation
let myString: string = 'Hello, World!';
let myNumber: number = 42;
let myBoolean: boolean = true;

// Declaring a variable without a type annotation
let myAny: any = 'This can be anything';

// Union types
let myUnion: string | number = 'This can be a string or a number';

// Type inference
let myInferredString = 'Hello, World!'; // Type is inferred as string
```

Functions

```typescript
// Declaring a function with a return type and parameter types
function addNumbers(a: number, b: number): number {
  return a + b;
}

// Declaring a function with optional parameters
function sayHello(name?: string): void {
  if (name) {
    console.log(`Hello, ${name}!`);
  } else {
    console.log('Hello!');
  }
}

// Declaring a function with default parameters
function sayGoodbye(name: string = 'World'): void {
  console.log(`Goodbye, ${name}!`);
}

// Declaring an arrow function
const multiplyNumbers = (a: number, b: number): number => {
  return a * b;
};
```

Loops

```
// for loop
for (let i: number = 0; i < 10; i++) {
  console.log(i);
}

// for...of loop
const myArray: number[] = [1, 2, 3, 4, 5];
for (const item of myArray) {
  console.log(item);
}

// forEach loop
myArray.forEach((item: number) => {
  console.log(item);
});

// while loop
let i: number = 0;
while (i < 10) {
  console.log(i);
  i++;
}

// do...while loop
let j: number = 0;
do {
  console.log(j);
  j++;
} while (j < 10);
```

Conditionals

```
// if statement
if (myNumber === 42) {
  console.log('The answer to the ultimate question of life, the universe, and
everything');
}

// if...else statement
if (myBoolean) {
  console.log('The boolean is true');
} else {
  console.log('The boolean is false');
}

// ternary operator
const result = myNumber > 0 ? 'Positive' : 'Negative or zero';

// switch statement
switch (myString) {
  case 'Hello':
    console.log('The string is "Hello"');
    break;
  case 'World':
    console.log('The string is "World"');
    break;
  default:
    console.log('The string is something else');
    break;
}
```

Interfaces

```
interface Person {
  name: string;
  age: number;
  address?: string; // optional property
  readonly id: number; // read-only property
}

const person: Person = {
  name: 'John',
  age: 30,
  id: 12345,
};

// Extending an interface
interface Employee extends Person {
  department: string;
}

const employee: Employee = {
  name: 'Jane',
  age: 25,
  id: 67890,
  department: 'Marketing',
};
```

Classes

```
class Animal {
  private name: string; // private property
  protected species: string; // protected property

  constructor(name: string, species: string) {
    this.name = name;
    this.species = species;
  }

  public getName(): string {
    return this.name;
  }

  public getSpecies(): string {
    return this.species;
  }
}

class Dog extends Animal {
  private breed: string;

  constructor(name: string, breed: string) {
    super(name, 'Dog');
    this.breed = breed;
  }

  public getBreed(): string {
    return this.breed;
  }
}

const myDog: Dog = new Dog('Fido', 'Labrador');
console.log(myDog.getName()); // Output: Fido
console.log(myDog.getSpecies()); // Output: Dog
console.log(myDog.getBreed()); // Output: Labrador
```

Generics

```
// Generic function
function identity<T>(arg: T): T {
  return arg;
}

const myString: string = identity<string>('Hello, World!');
const myNumber: number = identity<number>(42);

// Generic class
class Box<T> {
  private value: T;
  constructor(value: T) {
    this.value = value;
  }

  public getValue(): T {
    return this.value;
  }
}

const myStringBox: Box<string> = new Box<string>('Hello, World!');
const myNumberBox: Box<number> = new Box<number>(42);
```

Enumerations

```
enum Direction {
  Up = 'UP',
  Down = 'DOWN',
  Left = 'LEFT',
  Right = 'RIGHT',
}

const myDirection: Direction = Direction.Up;
console.log(myDirection); // Output: UP
```

File manipulation

```
import MyComponent from './MyComponent';

export default function App(): JSX.Element {
  return <MyComponent />;
}
```

Resources

- TypeScript documentation
- TypeScript playground
- React with TypeScript tutorial
- TypeScript Deep Dive
- TypeScript Weekly newsletter