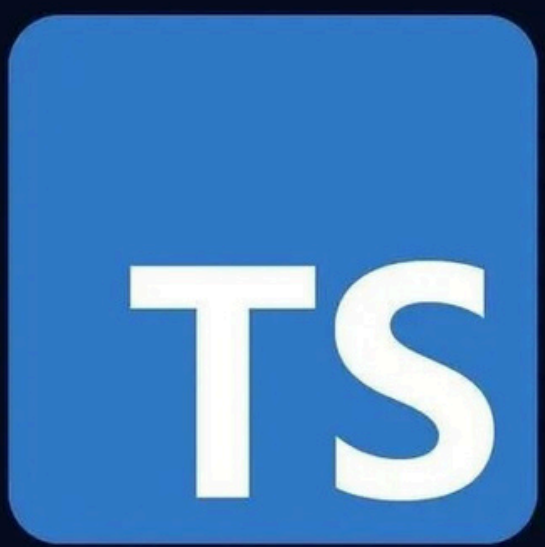
TS

TypeScript Cheatsheet



TS

Basic Types

index.ts

```ts
// numeric data type
let age: number = 42;
// string data type
let name: string = 'John';
// boolean data type
let isDone: boolean = false;

// function return type
function foo(): void {
  console.log('Hello, world!');
}

// anything can be a value
let x: any = 42;
// null data type
let nullValue: null = null;
// undefined data type
let undefinedValue: undefined = undefined;
```

TS

Function

index.ts

```ts
// 1. Function with typed parameters and return type
function add(a: number, b: number): number {
  return a + b;
}

// 2. Function with optional parameter
function greet(name?: string): void {
  console.log(`Hello, ${name ?? 'world'}!`);
}

// 3. Function with default parameter
function repeat(text: string, times: number = 3): string {
  return text.repeat(times);
}

// 4. Function with rest parameter
function sum( ...values: number[]): number {
  return values.reduce((total, value) => total + value, 0);
}

// 5. Function with overloaded signatures
function convert(value: string): number;
function convert(value: number): string;
function convert(value: string | number): string | number {
  if (typeof value === 'string') {
    return parseInt(value, 10);
  } else {
    return value.toString();
  }
}
```

TS

Interfaces

index.ts

```ts
// 1. Basic interface
interface Person {
  name: string;
  age: number;
}

// 2. Interface with optional property
interface User {
  id: number;
  email?: string;
}

// 3. Interface with readonly property
interface Point {
  readonly x: number;
  readonly y: number;
}

// 4. Interface with function property
interface Calculator {
  add(a: number, b: number): number;
}
```

TS

Interfaces (Cont)

index.ts

```ts
// 5. Interface extending another interface
interface Employee extends Person {
  department: string;
}

// 6. Interface extending multiple interfaces
interface Shape {
  draw(): void;
}
interface Rectangle extends Shape {
  width: number;
  height: number;
}

// 7. Interface with index signature
interface Dictionary<T> {
  [key: string]: T;
}

// 8. Interface with call signature
interface Greeter {
  (name: string): string;
}
```

TS

Generics

index.ts

```ts
// 1. Generic function
function identity<T>(arg: T): T {
  return arg;
}

// 2. Generic class
class Stack<T> {
  private items: T[] = [];

  push(item: T) {
    this.items.push(item);
  }

  pop(): T | undefined {
    return this.items.pop();
  }
}

// 3. Generic interface
interface KeyValuePair<K, V> {
  key: K;
  value: V;
}

// 4. Generic type alias
type Queue<T> = T[];

// 5. Generic constraint
function find<T extends { id: number }>(items: T[], id: number): T | undefined {
  return items.find(item => item.id === id);
}
```

TS

Type Assertion

index.ts

```ts
// 1. Angle bracket syntax
let name1: any = 'John';
let length1: number = (<string>name1).length;

// 2. as syntax
let name2: any = 'John';
let length2: number = (name2 as string).length;

// 3. Assertion with union type
let value: string | number = '42';
let length3: number = (<string>value).length;

// 4. Assertion with type intersection
type Person = { name: string };
type Employee = { department: string };
let john: Person & Employee = {
  name: 'John',
  department: 'IT'
};
let name4: string = (<Person>john).name;

// 5. Assertion with type narrowing
let user: { id: number; name: string } | null = { id: 42, name: 'John' };
if (user !== null) {
  let name5: string = user.name;
}
```

TS

Classes

index.ts

```ts
// declare a class with constructor and methods
class Person {
  constructor(public firstName: string, public lastName: string, public age: number) {}

  getFullName(): string {
    return `${this.firstName} ${this.lastName}`;
  }
}

// class inheritance
class Student extends Person {
  constructor(firstName: string, lastName: string, age: number, public studentId:
number) {
    super(firstName, lastName, age);
  }

  getStudentInfo(): string {
    return `${this.getFullName()}, Age: ${this.age}, Student ID: ${this.studentId}`;
  }
}

// access modifiers for class members
class Teacher extends Person {
  private salary: number;

  constructor(firstName: string, lastName: string, age: number, salary: number) {
    super(firstName, lastName, age);
    this.salary = salary;
  }

  getSalary(): number {
    return this.salary;
  }
}
```