Cheatsheets / **Learn TypeScript**

Type Narrowing

TypeScript Union Type Narrowing

Since a variable of a union type can assume one of several different types, you can help TypeScript infer the correct variable type using type narrowing. To narrow a variable to a specific type, implement a type guard. Use the `typeof` operator with the variable name and compare it with the type you expect for the variable.

```
const choices: [string, string] = ['NO', 'YES'];
const processAnswer = (answer: number | boolean) => {
  if (typeof answer === 'number') {
    console.log(choices[answer]);
  } else if (typeof answer === 'boolean') {
    if (answer) {
      console.log(choices[1]);
    } else {
      console.log(choices[0]);
    }
  }
}
processAnswer(true);    // Prints "YES"
processAnswer(0);       // Prints "NO"
```

TypeScript Type Guard

A TypeScript type guard is a conditional statement that evaluates the type of a variable. It can be implemented with the `typeof` operator followed by the variable name and compare it with the type you expect for the variable.

```
// A type guard implemented with the typeof operator
if (typeof age === 'number') {
  age.toFixed();
}
```

TypeScript Type Guard Accepted Types with `typeof`

The `typeof` operator may be used to implement a TypeScript type guard to evaluate the type of a variable including `number`, `string` and `boolean`.

TypeScript Type Guard with `in` operator

If a variable is a union type, TypeScript offers another form of type guard using the `in` operator to check if the variable has a particular property.

```
/*
In this example, 'swim' in pet uses the 'in' operator to check if the property .swim is present on pet. TypeScript recognizes this as a type guard and can successfully type narrow this function parameter.
*/
function move(pet: Fish | Bird) {
  if ('swim' in pet) {
    return pet.swim();
  }
  return pet.fly();
}
```

TypeScript Type Guard `if-else` Statement

If a variable is of a union type, TypeScript can narrow the type of a variable using a type guard. A type guard can be implemented as a conditional expression in an `if` statement. If an `else` statement accompanies the `if` statement, TypeScript will infer that the `else` block serves as the type guard for the remaining member type(s) of the union.

```
function roughAge(age: number | string) {
  if (typeof age === 'number') {
    // In this block, age is known to be a number
    console.log(Math.round(age));
  } else {
    // In this block, age is known to be a string
    console.log(age.split(".")[0]);
  }
}
roughAge('3.5');  // Prints "3"
roughAge(3.5);    // Prints 4
```

TypeScript Type Guard `if` Statement Function Return

If a variable is of a union type, TypeScript can narrow the type of a variable using a type guard. A type guard can be implemented as a conditional expression in an `if` statement. If the `if` block contains a `return` statement and is not followed by an `else` block, TypeScript will infer the rest of the code block outside the `if` statement block as a type guard for the remaining member type(s) of the union.

```
function formatAge(age: number |
string) {
  if (typeof age === 'number') {
    return age.toFixed(); // age must
be a number
  }
  return age; // age must not be a
number
}
console.log(formatAge(3.5));    //
Prints "4"
console.log(formatAge('3.5'));  //
Prints "3.5"
```