Cheatsheets / **Learn TypeScript**

Union Types

TypeScript Union Type

TypeScript allows a flexible type called any that can be assigned to a variable whose type is not specific. On the other hand, TypeScript allows you to combine specific types together as a union type.

```
let answer: any;    // any type
let typedAnswer: string | number; //
union type
```

TypeScript Union Type Syntax

TypeScript lets you create a union type that is a composite of selected types separated by a vertical bar, `|`.

```
let myBoolean: string | boolean;

myBoolean = 'TRUE';  // string type
myBoolean = false;   // boolean type
```

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 Function Return Union Type

TypeScript infers the return type of a function, hence, if a function returns more than one type of data, TypeScript will infer the return type to be a union of all the possible return types. If you wish to assign the function's return value to a variable, type the variable as a union of expected return types.

```
const popStack = (stack: string[]) => {
  if (stack.length) {
    return stack[stack.length-1]; // return type is any
  } else {
    return null;      // return type is null
  }
};
let toys: string[] = ['Doll', 'Ball', 'Marbles'];
let emptyBin: string[] = [];
let item: string | null = popStack(toys); // item has union type
console.log(item);  // Prints "Marbles"
item = popStack(emptyBin);
console.log(item);  // Prints null
```

TypeScript Union of Array Types

TypeScript allows you to declare a union of an array of different types. Remember to enclose the union in parentheses, `(...)`, and append square brackets, `[]` after the closing parenthesis.

```
const removeDashes = (id: string | number) => {
  if (typeof id === 'string') {
    id = id.split('-').join('');
    return parseInt(id);
  } else {
    return id;
  }
}
// This is a union of array types
let ids: (number | string)[] = ['93-235-66', '89-528-92'];
let newIds: (number | string)[] = [];
for (let i=0; i < ids.length; i++) {
  newIds[i] = removeDashes(ids[i]); // Convert string id to number id
}
console.log(newIds);  // Prints
[9323566, 8952892]
```

TypeScript Union Type Common Property Access

As a result of supporting a union of multiple types, TypeScript allows you to access properties that are common among the member types without any error.

```
let element: string | number[] =
'Codecademy';
// The .length property is common for
string and array
console.log(element.length);      //
Prints 10
// The .match method only works for a
string type
console.log(element.match('my')); //
Prints ["my"]

element = [3, 5, 1];
// The length property is common for
string and array
console.log(element.length);      //
Prints 3
// The .match method will not work for
an array type
console.log(element.match(5));  //
Error: Property 'match' does not exist
on type 'number[]'.
```

TypeScript Union of Literal Types

You can declare a union type consisting of literal types, such as string literals, number literals or boolean literals. These will create union types that are more specific and have distinct states.

```
// This is a union of string literal
types
type RPS = 'rock' | 'paper' |
'scissors' ;
const play = (choice: RPS): void => {
  console.log('You: ', choice);
  let result: string = '';
  switch (choice) {
    case 'rock':
      result = 'paper';
      break;
    case 'paper':
      result = 'scissors';
      break;
    case 'scissors':
      result = 'rock';
      break;
  }
  console.log('Me: ', result);
}
const number =
Math.floor(Math.random()*3);
let choices: [RPS, RPS, RPS] = ['rock',
'paper', 'scissors'];
play(choices[number]);
```