2. Tipus de dades

Typescript

Duració: 3h

Definició de variables

Com a javascript la definició de variables s'ha de fer servir let o const seguit del nom de la variable.

- **let**: definir una variable.
- **const**: definir una constant.

Després del nom definim el tipus de dada que serà la variable.

- number
- string
- boolean

També podem definir vectors amb tipus.

- number[]
- string[]

```
const pi : number = 3.14
let word : string = 'Hello'
let num : number = 5
let bool : boolean = true
let vector : number[] = [1,2,3]
let bigNum : bigint = 10000000n
```

Tipus de dades

- number: Enters i Decimals.
- bigint: Enters molt grans. No poden operar amb numerics.
- boolean: Valor binari que pot acceptar els valors true o false.
- string: Representa un text.
- null: valor que indica que no hi ha valor.
- undefined: Al definir una variable sense valor.
- void: No hi ha valor es fa servir per indicar que una funció no retona valor.
- any: Qualsevol valor anterior.
- unknown: Permet qualsevol valor però s'ha de validar el tipus abans de poder consultar o modificar el seu valor.

Exercicis (15 min)

1. Crea la variable str de tipus string i la variable num de tipus numeric imprimeix per consola el resultat.
2. Crea una variable bigint amb el valor d'un trillon: 1.000.000.000.000 i intenta sumar-la amb un numeric.
3. Crea un array de números i imprimeix per consola tots els valors més grans de 0 fes servir una variable booleana per emmagatzemar la condició.

Funcions

Typescript ens permet definir funcions indicant el tipus de dada que acepta cada paràmetre i el tipus de valor que retorna la funció:

```
function countChars(param : string) : number {
    return param.length
}
```

Enum

Definició d'un enum:

```
enum Color {
    Red,
    Green,
    Blue,
    Pink
};
```

Imprimir enum:

```
enum Color {Red, Green, Blue, Pink};
let c: Color = Color.Green;
console.log(c) // 1
let colorName = Color[c]
console.log(colorName) // "Green"
```

Enum

Ts compila els enums com objectes/diccionari. Podem veure aquest funcionament fent una impresió per consola de tipus de l'enum:

```
console.log(Color)
```

```
{
    0: 'Red',
    1: 'Green',
    2: 'Blue',
    3: 'Pink',
    Red: 0,
    Green: 1,
    Blue: 2,
    Pink: 3
}
```

Enum

Assignar valors a personalitzats a l'enum

```
enum Color {
    Red = '#f00',
    Green = 'green',
    Blue = 3,
    Pink
};
```

```
console.log(Color.Red) //'#f00'
console.log(Color.Green) //'green'
console.log(Color.Blue) //3
console.log(Color.Pink) //????
```

¿Quin valor retorna Color.Pink?

Exercicis (15 min)

1. Defineix un enum amb les mides de les samarretes: XS, S, M, L, XL. Fes una funció on li passem un preu i la mida i ens retorni el preu incrementat en intervals de 5% el preu original. Ex getPriceFromSize(Size.M, 10) // -> 11

2. Crea un enum amb els dies de les setmanes i crea una funció que imprimeix el número del dia de la setmana. Ex printWeekNomberDay(Week.Monday) // 1

3. Crea un enum amb els colors de l'arc de sant martí amb el valor en hexadecimal del color. Crea una funció que afegeixi a document DIVs amb els colors passats per paràmetres. Ex printColors(colors)

```
red: 'EB1416' orange: '#FFA500' yellow: 'FAAB36' green: '#79C314', blue: '#487DE7' indigo: '#4B369D'
violet: '#70369D'
```

Funcions Genèriques

1. Definició: Les funcions genèriques utilitzen tipus genèrics que s'especifiquen amb parèntesis angulars (<T>).
2. Flexibilitat de tipus: Permeten treballar amb diferents tipus de dades sense perdre la seguretat de tipus.
3. Múltiples tipus genèrics: Es poden fer servir diversos tipus genèrics en una mateixa funció.
4. Restriccions: Es poden aplicar restriccions als tipus genèrics fent servir la paraula clau extends.

```
function echo<T>(param: T): T {
   return param;
}

function echo<T1, T2>(p1: T1, p2: T2 ): [T1,T2]{
   return [p1, p2]
}

function print<T extends IPersona>(arg: T): string
{
   return arg.firstname + arg.lastname;
}
```

typeof

El operador `typeof` permet determinar el tipus de dades d'una variable o expressió. Retorna una cadena que indica el tipus del valor sense avaluar-lo.

```
console.log(typeof 42);         // "number"
console.log(typeof "Hola món");// "string"
console.log(typeof true);       // "boolean"
console.log(typeof {});         // "object"
console.log(typeof undefined); // "undefined"
```

Exercicis

1. Implementa una funció on retorna el nom del tipus de la dada passada per paràmetre.
2. Implementa la funció que accepta una array de qualsevol tipus i ordena'l de menor a mayor (fes servir l'algoritme d'ordenació de bombolla).

Tipus Alias

Els alias són una forma de crear tipus personalitzats i reutilitzables que permeten assignar un nom a qualsevol tipus de dada facilitant la seva reutilització. Millora la llegibilitat al tenir un nom més semàntic i un únic lloc on es defineix el tipus.

```
type UserId = number;

const id: UserId = 123;

type Car = {
 brand: string,
 model: string,
 year: number
};
```

Tipus Union

Els tipus d'unió (union types) permeten que una variable pugui tenir més d'un tipus. Això és útil quan un valor pot ser de diferents tipus en funció del context.

Només podrem accedir a les propietats que els dos tipus comparteixen.

```
type enumerable = string | Array<any>

function foo(p : enumerable){
    p.length
    p.includes('a')
    p.split(' ') // Error

}
```

Tipus Guard

Un type guard en TypeScript és una tècnica que permet obtenir informació sobre el tipus d'una variable, generalment dins d'un bloc condicional.

```
function processValue(value: string | number) {
    if (typeof value === 'string') {
        // TypeScript sap que 'value' és un
string
        console.log(value.toUpperCase());
    } else {
        // TypeScript sap que 'value' és un
number
        console.log(value.toFixed( 2));
    }
}
```

Type Guard amb tipus personalitzats

```
function isTriangle(forma: Triangle | Cercle) : forma is Triangle{
   return (forma as Triangle).alçada != undefined
}

function area(forma : Triangle | Cercle){

   if(isTriangle(forma)){
       forma.
   }

   if("base" in forma){
       forma.
   }
}
```

Cast

En TypeScript, hi ha dues maneres principals de fer un cast a una variable.

```
let someValue: unknown = "això és una cadena";
let strLength: number = (someValue as string).length;

let someValue: unknown = "això és una cadena";
let strLength: number = (<string>someValue).length;
```

Sobrecarrega de funcions

La sobrecarrega de funcions permet definir quines signatures de funcions són acceptades. TS no permet sobecàrrega directament, però si ens permet acceptar diferents tipus per un sol pàrametre fent servir les definicions definim quines signatures són valides.

```
// Signatures de sobrecàrrega
function combinar(a: string, b: string): string;
function combinar(a: number, b: number): number;

// Implementació
function combinar(a: string | number, b: string | number): string
| number {
 if (typeof a === "string" && typeof b === "string") {
   return a + b;
 } else if (typeof a === "number" && typeof b === "number") {
   return a + b;
 }
 throw new Error("Tipus de paràmetres no coincideixen");
}
```

```
let c1 = combinar('a', 'b')
let c2 = combinar(1, 2)
let c3 = combinar('a', 1)
let c4 = combinar(1, 'a')
```

Exercicis

1. Crea un type alias amb el nom de UserName de tipus string i crea una variable amb el alias creat i assigna-li un valor. Intenta fer un console.log de la variable creada.

2. Crea un type alias per a un objecte que representi un usuari amb les següents atributs: id (number) nom (string) email (string). Després, crea un array d'usuaris utilitzant aquest type alias i una funció que accepti un usuari i retorni un string amb el seu nom i correu.

3. Crea un type alias per a triangle amb els atributs base i alçada un altre type alias per a cercle amb l'atribut radi. Crea una funció que accepti un paràmetre de les dues formes i retorni l'àrea de la forma passada per paràmetre.