



PARSEANDO

Como vimos, los resultados de los valores obtenidos con el método `prompt()` no siempre coinciden con el tipo de dato que necesitamos. Es decir, si le pedimos al usuario que ingrese su edad y la guardamos en una variable para usar la misma y sumársela al año actual, veremos que no obtenemos el resultado pretendido. 😞

```
let edad = prompt("ingrese su edad");  
console.log(edad+2021);
```

Está claro que si a un texto le sumamos un número no obtendremos la sumatoria, sino que tendremos la concatenación de ambos como un texto.

`parseInt()`

Para no incurrir en errores como el anterior u otros tantos que pueden surgir de no comprobar el tipo de dato que estamos manipulando tenemos la función **`parseInt()`**. Esta función parsea una cadena de texto y devuelve un número.

```
parseInt("22");  
parseInt(prompt("Ingrese edad"));
```



De nuevo a lo de siempre, si no guardamos estos datos en ningún lado, difícilmente podamos hacer algo con ellos. Para eso, implementamos variables que almacenan el resultado de las funciones. Veamos su resultado.

```
let a = parseInt("22");
let b = parseInt(prompt("Ingrese edad"));
let c = parseInt("22"+"150");
let d = parseInt(22+150)
let e = parseInt(22+parseInt("150"));
let f = parseInt(22.55);

console.log(a);
console.log(b);
console.log(c);
console.log(d);
console.log(e);
console.log(f);
```

Al combinar y probar distintas posibilidades obtendremos distintos resultados, lo fundamental es entender el funcionamiento de cada método y función para aplicarlo según nuestras necesidades. Como vemos, en un caso puntual observamos que la función `parseInt()` solo nos devuelve la parte entera del número que ingresemos, por lo que si tenemos decimales los mismo quedarán truncados.

parseFloat()

Acá entra en juego esta otra función, que tiene el mismo objetivo que la anterior, pero en este caso si nos retorna los números decimales que existan.



```
console.log(parseFloat(22.34));  
console.log(parseFloat(22.3456284));
```

Si fuimos probando estas funciones y además por curiosidad, o error —ambos son sumamente útiles 😊—, intentamos parsear un texto, vimos que el resultado obtenido no es un número.

NaN

La propiedad **NaN** nos indica que el valor no es un número (**Not A Number**), por lo que esto nos produciría un error si queremos realizar alguna operación aritmética con este valor.


Pongamos este ejemplo de una situación que nos produciría un error. Supongamos que en el siguiente código, al ejecutarse, en el cuadro de diálogo del prompt el usuario, por error o a propósito —cosa que debemos tener en cuenta como programadores 😬—, ingresa un texto “su edad”.



```
let edad = parseInt(prompt("Ingrese su edad"));  
  
if(edad>18){  
  console.log("Es mayor de edad");  
}else{  
  console.log("Es menor de edad");  
}
```






 Claramente no estamos exentos de que el usuario sea un [troll](#), por eso, siempre tenemos que buscar maneras de validar los datos que el usuario puede manipular.

Desafío:

Te invitamos a resolver el siguiente desafío para seguir practicando. Para ello, podés abrir el VS Code y pegar el último bloque de código implementado.

- ¿Cuál es el resultado de este código?
- ¿Es correcto lo que arroja en base a lo que ingresó el usuario?
- ¿Dónde podría existir un problema?
- ¿Cómo podríamos solucionarlo y llegar a un mejor resultado utilizando los métodos que ya conocemos? → Tip

 Animate a refactorizar el código, pensar en los posibles errores y cómo salvarlos.

→ Tip: Tenemos la función **isNaN()**, la cual nos devuelve true si el valor dado como parámetro es NaN. Para conocer más podés ingresar [acá](#).

EXTRA

Math()

Como un apartado de esta sección, traemos para revisar información de un objeto que quizás nos sirva en algún momento de nuestro desarrollo. Estamos hablando de [Math](#), que consta de muchísimas propiedades y métodos que nos pueden ser de utilidad.