ANÁLISE

Como já vimos, os resultados dos valores obtidos com o método prompt() nem sempre correspondem com o tipo de dado de que necessitamos. Ou seja, se pedirmos ao usuário que introduza a sua idade e a armazene numa variável para utilizar e adicioná-la ao ano corrente, veremos que não obtemos o resultado pretendido. 😐

```
let idade = prompt("introduza sua idade");
console.log(idade+2021);
```

É evidente que se acrescentarmos um número a um texto, não obteremos a soma dos dois, mas teremos o encadeamento dos dois como um só texto.

parseInt()

Para evitar erros como o anterior ou outros erros que possam surgir da não verificação do tipo de dados que estamos manipulando, temos a função **parseInt()**. Esta função analisa uma sequência de texto e devolve em um número.

```
parseInt("22");
parseInt(prompt("Introduza idade"));
```

Mais uma vez, se não armazenarmos esses dados em algum lugar, dificilmente podemos fazer algo com eles. Para isso, implementamos variáveis que armazenam o resultado das funções. Vejamos o resultado.

```
let a = parseInt("22");
let b = parseInt(prompt("Introduza idade"));
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);
```

Combinando e testando diferentes possibilidades, obteremos resultados diferentes. O fundamental é entender o funcionamento de cada método e função para aplicá-lo conforme nossas necessidades. Como podemos ver, num caso de pontualidade, observamos que a função parseInt() só devolve a parte inteira do número que introduzimos, por isso, se tivermos decimais, serão truncados.

parseFloat()

Aqui entra em jogo esta outra função, que tem o mesmo objetivo que a anterior, mas neste caso devolve os números decimais que existem.

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

Se testamos estas funções e também por curiosidade, ou engano —ambos são extremamente úteis🙂—, tentamos analisar um texto, vimos que o resultado obtido não é um número.

NaN

A propriedade **NaN** nos indica que o valor não é um número **(Not A Number)**, portanto, isto causaria um erro se quiséssemos realizar qualquer operação aritmética com este valor.

Tomemos este exemplo de uma situação que conduziria a um erro. Suponhamos que no código seguinte, quando executado, na caixa de diálogo do prompt, o usuário, por engano ou de propósito — algo que precisamos ter em mente como programadores🧐 —, introduza um texto “sua idade”.

```
let idade = parseInt(prompt("Introduza sua idade"));

if(idade>18){
console.log("É maior de idade");
}else{
console.log("É menor de idade");
}
```

🚨 Claramente não estamos isentos do fato do utilizador ser um troll, portanto, temos

sempre de procurar formas de validar os dados que o utilizador pode manipular.

Desafio:

Te convidamos a resolver o seguinte desafio a fim de continuar a praticar. Para isso, abra o VS Code e cole o último bloco de código implementado.

- Qual é o resultado deste código?
- É correto o que retorna com base no que o utilizador introduziu?
- Onde poderia existir um problema?
- Como poderíamos resolvê-lo e alcançar um resultado melhor utilizando os métodos que já conhecemos? → Tip

🎈 Seja encorajado a refazer o código, pense em possíveis bugs e em como corrigi-los.

→ Dica: Temos a função **isNaN( )**, a qual nos retorna se o valor dado como parâmetro é NaN. Para saber mais, clique aqui.

EXTRA

Math()

Como uma separação desta seção, trazemos para rever informações sobre um objeto que nos possa ser útil em algum momento do nosso desenvolvimento. Estamos falando de Math, que tem muitas propriedades e métodos que podem ser úteis.