Eccezioni

Gestione degli errori

- Debugging
- Alcuni tipi di errori non possono essere evitati
    - Non è una buona idea esibire i messaggi di errore all’utente in un sistema software
      Bisogna gestire gli errori, dare feedback all’utente e poi continuare l’esecuzione dal punto giusto

Sistema di gestione delle eccezioni

Eccezione:

- Meccanismo di interruzione del programma in caso di errore
- Viene ‘lanciata’ quando l’errore compare
    - Automatiche
    - Possibilità di lanciare anche ‘manualmente’ a bisogno
- Può essere ‘catturata’ nel posto più adatto del programma

Le eccezioni in JavaScript possono essere gestite utilizzando il costrutto **try-catch**.
Il blocco `try` contiene il codice che potrebbe generare un'eccezione, mentre il blocco `catch` viene eseguito se un'eccezione viene sollevata nel blocco `try`.

Il blocco `catch` riceve l'eccezione come parametro, che può quindi essere utilizzato per determinare la causa dell'errore e adottare le opportune azioni correttive.

```
try {
```

```
  //comandi

}  catch (e){

  //comandi gestione errore
}
```

Cosa succede?

- Il codice del blocco **try** viene eseguito, fino al primo comando che genera un errore, incluso nelle chiamate annidate di funzioni. I comandi oltre l'errore non vengono eseguiti e viene restituito il controllo alla funzione corrente.
- Se compare un errore, si esegue il blocco **catch**. La variabile **e** contiene informazioni sull'errore.
- Se non c'è nessun errore, il blocco **catch** viene saltato e si continua l'esecuzione del programma.

> 📌 Tante volte può essere utile poter lanciare un'eccezione noi stessi. Per poter segnalare un problema specifico al nostro programma

È possibile sollevare un'eccezione utilizzando il costrutto `throw`

```
throw new Error("Si è verificato un errore durante l'elaborazione dei dati.");
```

In questo esempio, il costrutto `throw` viene utilizzato per creare un nuovo oggetto `Error` con il messaggio di errore specificato come parametro.
L'eccezione viene quindi sollevata e il controllo viene passato al blocco `catch` più vicino che gestisce l'eccezione.

È possibile utilizzare qualsiasi tipo di oggetto per sollevare un'eccezione, ma l'oggetto deve avere una proprietà `message` che contiene il messaggio di errore.

📌 `new Error()` utile ma non permette di riconoscere errori di natura diversa e nasconde gli altri errori. Infatti è utile creare i nostri di oggetti che implementano Error

```
// Gerarchia eccezioni/errori
class CalcError extends Error {}
class OperandError extends CalcError {}
class OperatorError extends CalcError {}
```

```
// Funzione calc
function calc(a) {
  [op,...x] = a
  for (let i of x) {
    if (!Number(i))
      throw new OperandError("Expecting numbers only")
  }
  switch (op) {
    case '+':
      return x.reduce((y,z)=>(y+z),0)
    case '-':
      [x1,...x] = x
      return x.reduce((y,z)=>(y-z),x1)
    case '*':
      return x.reduce((y,z)=>(y*z),1)
    case '/':
      [x1,...x] = x
      return xx.reduce((y,z)=>(y/z),x1)
    default:
      throw new OperatorError("Unknown operator(currently supporting only +,-,*,/)")
  }
}
```

```
// Gestione eccezioni
try {
  a = calc(['+',1,2,3])
  console.log(a)
} catch (e) {
  if (e instanceof OperandError) {
    console.log("OperandError: " + e.message)
  }
```

```
  else if (e instanceof OperatorError) {
    console.log("OperatorError: " + e.message)
  }
  else {
    console.log("Not working, that's it!")
  }
} finally {
  console.log("Cannot leave without saying bye!")
}
```

Il blocco `finally` viene utilizzato per eseguire del codice che deve essere eseguito sempre, indipendentemente dal risultato dell'esecuzione di un blocco di codice `try` o `catch`

I blocchi **finally** e **catch** possono essere omessi, ma almeno uno ci deve essere

I comandi **try-catch-finally** possono essere annidati