ΠΕΡΙΕΧΟΜΕΝΑ:

1. Συντακτικό Εξαιρέσεων
    1. try…catch
    2. finally
2. Αντικείμενα - Εξαιρέσεις:
    1. Μέθοδοι Αντικειμένων
    2. Built-in Αντικείμενα
3. throw και Εξαιρέσεις Χρήστη

- Όταν προκαλείται ένα σφάλμα κατά την εκτέλεση του προγράμματος π.χ.: (prj: error)

```
// x is undeclared
console.log(x);
```

- η JS σταματά την εκτέλεση του προγράμματος, δημιουργεί ένα **αντικείμενο λάθους (error)** (εδώ το ReferenceError) και το **"ρίχνει" (throw)** σε εμάς.
- Αυτά τα αντικείμενα λέγονται εξαιρέσεις (exceptions): Ο ρόλος τους είναι να προσφέρουν μία ευκαιρία ώστε να κάνουμε αποδοτικό χειρισμό του λάθους.
  - **Αν δεν χειριστούμε (προγραμματιστικά) την εξαίρεση**, το πρόγραμμα τερματίζει και μας επιστρέφεται η εξαίρεση που προέκυψε, καθώς και η σειρά εκτέλεσης των μεθόδων **(traceback)** που προκλήθηκε η εξαίρεση.

```
▶ Uncaught ReferenceError: x is not defined
    at error.html?_ijt=f79g…ELOAD_ON_SAVE:14:17
```

  - **Αν χειριστούμε (catch, handle)** την εξαίρεση, τότε η εκτέλεση του προγράμματος **συνεχίζει κανονικά.**
  - Ο χειρισμός της εξαίρεσης γίνεται με ένα block try-catch:
    - **try**: Σε αυτό το block βάζουμε τον κώδικα στον οποίο ενδέχεται να προκύψει η εξαίρεση
    - **catch(error)**: Εδώ βάζουμε τον κώδικα που θα τρέξει αν προκύψει εξαίρεση.

Παράδειγμα 1: try_catch

```
try {
  console.log(x);
}
catch(error) {
   console.log("Error: " + error);
}
```

Άσκηση 1:

Κατασκευάστε ένα πρόγραμμα που περιέχει δύο λάθη:

- Το πρώτο λάθος να είναι μία κλήση της eval() με κώδικα που να περιέχει ένα απλό συντακτικό λάθος.
- Το δεύτερο λάθος να είναι η κατασκευή (με τη συνάρτηση - κατασκευαστή), ενός πίνακα με εξωφρενικά μεγάλο μέγεθος.

Περιτυλίξτε κάθε λάθος σε ένα μπλοκ try και για το κάθε ένα από αυτά γράψτε ένα μπλοκ catch το οποίο απλά να τυπώνει το αντικείμενο - εξαίρεση.

Παρατηρήστε τον τύπο του αντικειμένου-εξαίρεσης.

- Το ολοκληρωμένο συντακτικό είναι:

```
try {
        …
} catch (error) {
        …
} finally {
        …
} …
```

  - Έπειτα από κάθε λέξη-κλειδί ακολουθεί μπλοκ κώδικα.
  - **try:** Εδώ μπαίνει ο "επίφοβος" κώδικας.
  - **catch:** Το όνομα error είναι της αρεσκείας μας, θα αρχικοποιηθεί με το αντικείμενο - εξαίρεση που θα προκύψει.
    - Συνηθίζεται να χρησιμοποιείται ο τελεστής instanceof για να εξειδικεύσουμε τον τύπο της εξαίρεσης
  - **finally (προαιρετικό):** Κώδικας που τρέχει σε κάθε περίπτωση: είτε μετά το χειρισμό της εξαίρεσης, είτε αν ο κώδικας του μπλοκ της try τρέξει χωρίς προβλήματα.

Παρατήρηση:

- Συνήθως στο μπλοκ finally κλείνουμε πόρους που έχουμε δεσμεύσει, όπως π.χ. μία απομακρυσμένη σύνδεση με ένα API κ.λπ.

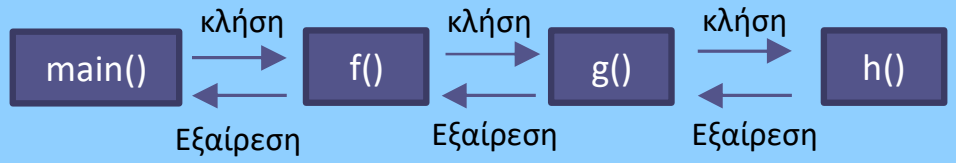
Παράδειγμα 2: finally

```
try {
   let array = new Array(1123123123123123);
   eval("let x=4.3.2");
}
catch (error) {
   if (error instanceof SyntaxError) {
      console.log("An instance error occurred!: " +  error)
   }
   else if (error instanceof RangeError) {
      console.log("Error with ranges: " + error)
   }
   else {
      console.log("Error: " + error);
   }
}
finally {
   console.log("I will handle something important");
}
console.log("Continuing execution...");
```

Σημείωση για τη χρήση με return:

- Ακόμη κι αν στις try/catch υπάρχει εντολή return, το block finally θα εκτελεστεί (και αυτό θα είναι το τελικό αποτέλεσμα αντί για το return: βλ. και return.html

- Έστω ότι έχουμε μια διαδοχή κλήσης μεθόδων όπως στο παράδειγμα 3:



- Όταν προκύπτει ένα λάθος (π.χ. εδώ στην h())
  - Η h() είτε θα χειριστεί το λάθος, είτε (αυτόματα) ο έλεγχος επιστρέφει στη g()
  - Η g() θα κάνει το ίδιο: είτε θα χειριστεί το λάθος, είτε (αυτόματα) ο έλεγχος επιστρέφεται στην f()
- Αν καμία μέθοδος δεν χειριστεί το λάθος τότε το πρόγραμμα θα τερματίσει και θα εμφανιστεί η σειρά κλήσης των μεθόδων

Σημείωση:

- Δείτε ότι με τον τρόπο αυτό, μπορούμε να χτίσουμε ένα "τείχος άμυνας λαθών" στο πρόγραμμά μας.
- Π.χ. η f() θα μπορούσε να είναι ένα ολοκληρωμένο πρόγραμμα που να καλεί διάφορες συναρτήσεις.
  - ωστόσο θα μπορούσαμε σε αυτήν να κάνουμε τον έλεγχο όλων των σημαντικών λαθών.
  - και οποτεδήποτε συνέβαινε κάποιο σημαντικό λάθος, η f() να αναλάμβανε επίσης να επαναφέρει το πρόγραμμα σε κατάσταση καλής λειτουργίας.

Παράδειγμα 3: CallStack

```
function h() {
   let array = new Array(142345234523412);
}

function g() {
   h();
}

function f() {
   try {
      g();
   }
   catch (error) {
      console.log(error);
   }
}

f();
```

- Κοινά properties για όλα τα αντικείμενα εξαιρέσεων:

μέθοδος	επεξήγηση
message	Μήνυμα της εξαίρεσης
name	Όνομα εξαίρεσης
cause	Αιτία εξαίρεσης
filename *	Όνομα αρχείου
lineNumber *	Αριθμός γραμμής
columnNumber *	Αριθμός στήλης

  - (Τα properties με * δεν υποστηρίζονται από όλους τους browsers - τα περισσότερα είναι μόνο στον Firefox)
- Η μοναδική μέθοδος του πρωτοτύπου είναι η **toString()**

Παράδειγμα 4: error_object

```
function printException(error) {
   console.log(error);
   console.log("message: " + error.message);
   console.log("name: " + error.name);
   console.log("name: " + error.filename);
   console.log("line: " + error.lineNumber);
   console.log("column: " + error.columnNumber);
   console.log("stack: " + error.stack);
}
```

Άσκηση 2:

Μελετήστε το αντικείμενο - εξαίρεση που προκύπτει όταν προσπαθούμε να κατασκευάσουμε πίνακα, ο όποίος έχει εκκεντρικά μεγάλο μέγεθος, με τη συνάρτηση του προηγούμενου παραδείγματος.

Άσκηση 3:

Τυπώστε το property “name” της τιμής null και μελετήστε το αντικείμενο - εξαίρεση που προκύπτει, με τη συνάρτηση του προηγούμενου παραδείγματος.

Built-in Αντικείμενα - Εξαιρέσεις:

- Τα επόμενα αντικείμενα - εξαιρέσεις δημιουργούνται εσωτερικά και "πετιούνται" σε περιπτώσεις λαθών

Αντικείμενο	Επεξήγηση
EvalError	Λάθη που προκαλούνται από την eval()
RangeError	Λάθος που προκαλείται όταν μια αριθμητική μεταβλητή ή παράμετρος είναι εκτός των επιθυμητών ορίων
ReferenceError	Λάθος που προκαλείται όταν προσπαθούμε να έχουμε πρόσβαση σε μία αναφορά που δεν είναι έγκυρη
SyntaxError	Συντακτικό Λάθος
URIError	Κακοκσχηματισμένο URI (μέθοδοι encodeURI() και decodeURI())
AggregateError	Σωρευτικό Λάθος (πολλά λάθη μαζεμένα σε ένα)

- Εκτός των παραπάνω υπάρχει και το γενικό λάθος (οποιαδήποτε περίπτωση εκτός των παραπάνω)

Αντικείμενο	Επεξήγηση
Error	Λάθος (που δεν μπαίνει σε μία από τις παραπάνω κατηγορίες)

Σημείωση:

- Τα λάθη - εξαιρέσεις που είδαμε, προκαλούνται σε συγκεκριμένες περιπτώσεις. Π.χ. το RangeError προκαλείται σε συγκεκριμένες περιπτώσεις:
  - Στον constructor Array()
  - Σε μεθόδους του Number.prototype (toExponential(), toFixed, toPrecision())
  - (και όχι σε κάθε περίπτωση που ίσως θα περιμέναμε)
- Ενώ μπορούμε να τα χρησιμοποιήσουμε για να προκαλέσουμε και τις δικές μας εξαιρέσεις (βλ. επόμενη διαφάνεια)

Παράδειγμα 5: uri

```
try {
   let encoded =
         encodeURI("https://www.psounis.com?name=Ψούνης");
   console.log(encoded);
   encoded =
         encodeURI("https\u{D800}www.psounis.com?name=Ψούνης");
                              // lone surrogate character
   console.log(encoded);
}
catch (error) {
   printException(error);
}
```

- **Εντολή throw**: Ρίχνει μια εξαίρεση (συγκεκριμένα οποιοδήποτε από τα built-ins της προηγούμενης διαφάνειας)

Παράδειγμα 6: check_valid_input

```
function processInteger(input) {
   if (!Number.isInteger(input))
      throw new TypeError("Number must be an integer");
   else if (input<0)
      throw new RangeError("Number must be >= 0");
   else if (input>50)
      throw new RangeError("Number must be <= 50");

   console.log("I am doing something with input: " + input);
}

let input = ["10", 1.5, 60, 33];

for (let value of input) {
   try {
      processInteger(value);
   }
   catch (error) {
      console.log(error);
   }
}
```

- **Κατασκευάζουμε μία δική μας κλάση - εξαίρεση**
  - Κληρονομώντας την κλάση Error και ορίζοντας τον κατασκευαστή της.

Παράδειγμα 7: my_exception

```
class ValidationError extends Error {
   constructor(message) {
      super(message);
      this.name = "ValidationError";
   }
}

function processInteger(input) {
   ...
}

let input = ["10", 1.5, 60, 33];

for (let value of input) {
   try {
      processInteger(value);
   }
   catch (error) {
      console.log(error);
   }
}
```

ΜΑΘΗΜΑ 12: Εξαιρέσεις

Ανασκόπηση - Βιβλιογραφία

JavaScript psounis

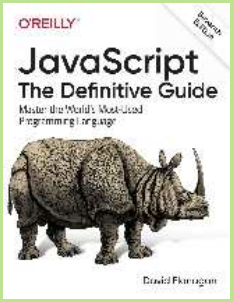
Ανασκόπηση:

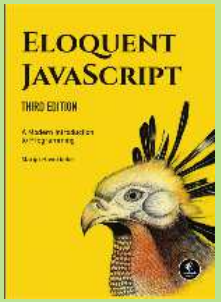
- 1: Εισαγωγή
- 2-3: Μεταβλητές - Σταθερές
- 4-5: Επιλογή - Επανάληψη
- 6-7: Πίνακες - Συναρτήσεις (Εισαγωγή)
- 8: Αντικείμενα (Ορισμός, Πρωτότυπα, Κληρονομικότητα)
- 9: Τύποι Δεδομένων (Ξανά)
- 10: Συναρτήσεις (scope, closures, this)
- 11: Modules
- 12: Εξαιρέσεις

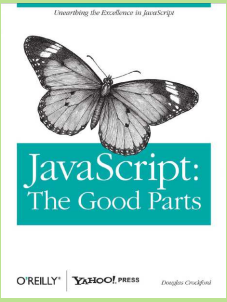
Επόμενα Βηματα:

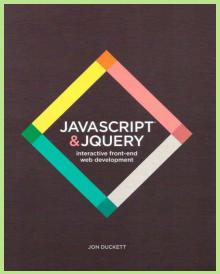
- Προχωρημένα Θέματα (JS Advanced): Iterators, Generators, Asynchronous Programming, built-in APIs
- Browser JS: events, listeners, selectors, html+css alterations, κ.α.
- jQuery
- Εργασία με APIs (π.χ. Google Maps)

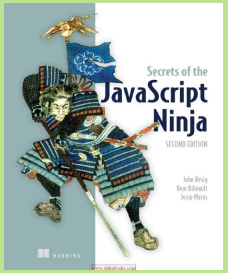
Βιβλιογραφία Σειράς:

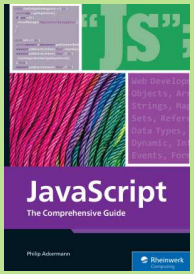












Web:

- https://developer.mozilla.org
- *https://www.stackoverflow.com ;-)*
- *Από το μάθημα 10.2 και μετά: OpenAI ChatGPT ;->*