SCI6373 Programmation documentaire

Cours 3

Été 2025

Plan (1/2)

- 2e script
  - Exemples 022, 024, 030
- Exercices après C2 - retour
- Se soucier de l'*expérience utilisatrice !*
- Valeurs plutôt vraies et plutôt fausses
- Quelques ajouts à notre boîte à outils
  - auto-réassignations, conversions explicites
  - .trim(), .replace()

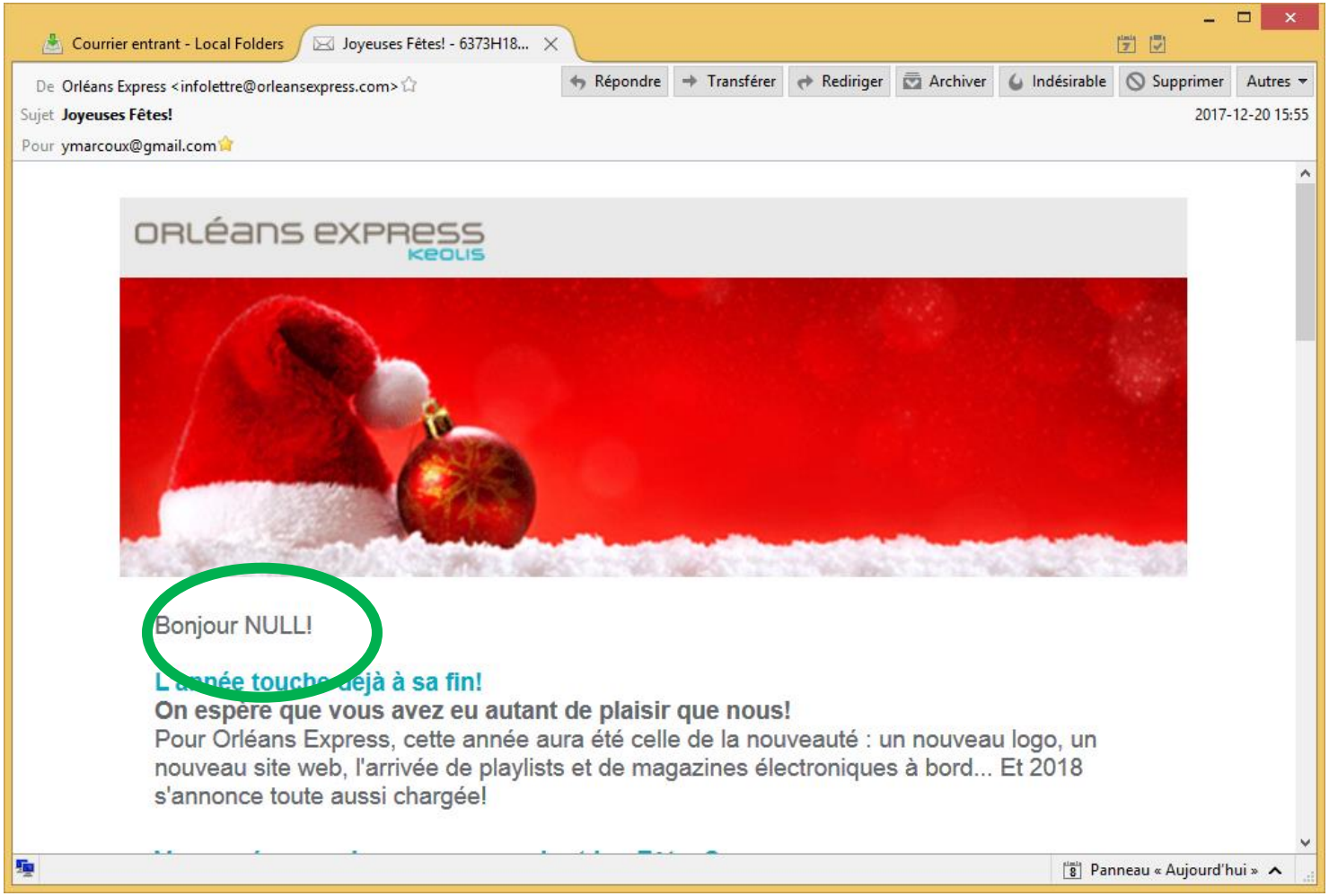
Plan (2/2)

- Structures de contrôle `if` et `if…else`
  - Et bonnes pratiques associées
- Aides à la programmation
- Fichiers JS séparés
- Fonction `console.log()`
- TP 1 à remettre vendredi soirée

Expérience utilisatrice (1/2)

- Peut être affectée par des petits détails de fonctionnement, par exemple :
  - Mauvais résultat s'il y a des blancs superflus dans un intrant
    - Peut provenir d'un copier-coller, ou d'un félin sur le clavier !
  - Message inadéquat (voire insultant) si erreur dans un intrant ou mauvais clic

`null` strikes again !



Expérience utilisatrice (2/2)

- Un script doit (le plus possible) être *tolérant* pour les erreurs / intrants inattendus
- Pas comme :
  - Exercices après C2 - 4h (saisons)
  - Exercices après C2 - 4f et 4g (prénom)
  - Exemples 022, 024, 030

Une 1e structure de contrôle

- Une structure de contrôle est une façon d'encadrer des blocs d'énoncés pour modifier l'ordre d'exécution séquentiel selon lequel les énoncés s'exécutent par défaut
- Cette structure de blocs encadrés constitue elle-même un *énoncé* (complexe)

Structure "if"

```
if (condition) {
   énoncés-si-condition-vraie
};
```

McPeak&Wilton p. 58

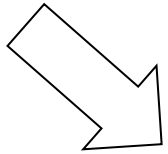
condition
Comme pour le choix conditionnel **? :**
condition est une expression
qui retourne `true` ou `false`
typiquement, **une comparaison**

- ( ) autour de *condition* obligatoires
- Entre les { } : un ou plusieurs énoncés
  - Exécutés seulement si *condition* est vraie
  - Si *condition* fausse : rien n'est exécuté
- 1 énoncé ou + entre { } s'appelle un ***bloc***

Pas de message si null

```
nom = prompt("Ton nom ?");
alert("Bonjour " + nom + " !");
```

Exemple : 030



```
nom = prompt("Ton nom ?");
if (nom != null) {
  alert("Bonjour " + nom + " !");
};
```

Exemple : 040

Indentation des blocs

- Il est ***extrêmement*** recommandé de décaler vers la droite les blocs imbriqués dans une structure de contrôle (ici **if)**
- S'appelle "indentation"
- Facilite ***énormément*** la lecture du code
- Peut se faire "en bloc" dans Notepad++ avec Tab et Maj+Tab
  - Démo

Structure de contrôle "if…else"

```
if (condition) {
        bloc-si-vrai
} else {
        bloc-si-faux
};
```

McPeak&Wilton p. 58

Parallèle avec choix conditionnel :

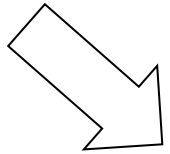
```
(condition) ? valeur-si-vrai : valeur-si-faux
```

Exemple :

```
(n > 10) ? "bon" : "mauvais"
```


Message fixe si null

```
nom = prompt("Ton nom ?");
if (nom != null) {
  alert("Bonjour " + nom + " !");
};
```



Exemple : 040

```
nom = prompt("Ton nom ?");
if (nom != null) {
  alert("Bonjour " + nom + " !");
} else {
  alert("Désolé, je ne peux vous saluer…");
};
```

Exemple : 050

Allègement possible (facultatif)

- Si un bloc d'énoncés ne contient qu'un seul énoncé, on peut éliminer les { }

```
nom = prompt("Ton nom ?");
if (nom != null)
  alert("Bonjour " + nom + " !");
else
  alert("Désolé, je ne peux vous saluer…");
```

Exemple : 060

Petit défi en classe

- Traiter le cas où l'utilisatrice ne saisit rien de la même façon que si elle clique Annuler
- Façon 1 :
  - Généraliser expression booléenne
- Façon 2 :
  - "Normaliser" les données
    - Transformer 2 cas différents (null et "") en un seul cas
    - C'est le PC DNC (diminuer le nombre de cas)

`?…:` versus `if…else`

- `?…:` est pour choisir une *valeur* parmi plusieurs possibles
- `if…else` est pour choisir une *action* (ou *série d'actions*) parmi plusieurs possibles

Bonne pratique avec **`if`** / **`if`**…**`else`** (1/2)

- Toujours en mettre *le minimum* dans un **`if`**, ou dans les 2 branches d'un **`if`**…**`else`**
- Tout traitement *commun* devrait être *en dehors* des parties conditionnelles
  - Évite de dupliquer des instructions
  - Moindre risque d'erreur
  - Plus facile à modifier
  - Code moins long

Bonne pratique avec **`if`** / **`if`**…**`else`** (2/2)

- S'applique aussi à l'opérateur (`?…:`)
  - Parfois, utiliser (`?…:`) au lieu d'un **`if`**…**`else`** permet d'en mettre plus en commun
- Exemples 090, 092 et 094
- S'applique aussi aux autres structures de contrôle, par exemple **`switch`**…**`case`** (à venir) qui est comme un **`if`** à "n" branches

"if" imbriqués

- Les "if" peuvent être imbriqués, i.e. :
  - Un des blocs (ou les deux) peuvent eux-mêmes contenir un ou des énoncés "if"
- Un *pattern* fréquent :
  - Le bloc "else" est lui-même un énoncé "if"
  - Vérifie "en cascade" des conditions mutuellement exclusives et choisit un bloc en conséquence
  - Exemple 100 (saison)

Aides à la programmation (1/2)

- Autocomplétion
- Coloration syntaxique
- Gestion des indentations
- Insertion / retrait de commentaires
- Pairage de délimiteurs
- Pliage / dépliage de blocs
- Indentation automatique (JSFormat)

Fichiers `.js` externe (1/2)

- Au lieu de mettre le JS *dans* un élément `<script>`, on peut utiliser l'attribut `src` :

```
<script src="monFich.js"></script>
```

- L'élément `script` *doit* être vide
- Mais les deux balises (début et fin) *doivent* être présentes telles quelles

Fichiers `.js` externe (2/2)

- Comme pour les scripts imbriqués :
  - Ces "appels" de scripts externes peuvent être n'importe où dans le document HTML
  - Les scripts appelés sont lus et exécutés *immédiatement* dès que le navigateur rencontre l'élément `script`
- Exemple 110

Avantages d’un fichier .js externe

- Un même fichier `.js` peut être « appelé » d’une multitude de pages HTML différentes
  - Comme pour les feuilles de style externes
- Les outils d'édition offrent plus d'aide à la programmation…

Aides à la programmation (2/2)

- Marche mieux Illustration avec le fichier JS de l'exemple 110 dans Notepad++

Valeurs JS quelconques interprétées comme valeurs booléennes (`true` ou `false`)

truthy et *falsy*

(plutôt vraies et plutôt fausses)

Valeurs quelconques traitées comme booléennes (1/2)

- Là où une valeur true/false est attendue :
  - (*condition*) d'un `if` ou `?…:` (ou `while`)
  - argument d'un opérateur booléen
- On peut mettre *n'importe quelle valeur* :
  - Sont traitées comme un `true` (plutôt vraies) :
    - Toute chaîne de caractères non vide
    - Un nombre différent de zéro
    - Tout pointeur à un objet
  - Sont traitées comme un `false` (plutôt fausses) :
    - La chaîne vide, le nombre 0, `NaN`, `null` et `undefined`

Valeurs quelconques traitées comme booléennes (2/2)

- Très utile pour alléger les expressions utilisées comme conditions
  - Ex.: (nom) au lieu de (nom != "")
- Permet de traiter `null` et "" d'un seul coup
  - Exemple 115 (comparer avec 060)

Auto-réassignations (1/3)

- Quelques opérateurs servent à effectuer des auto-réassignations en une seule opération (*McPeak&Wilton p. 29*)
- Syntaxe : `+=` , `-=` , `*=` , etc.
- Sémantique :

```
n += x ⇔ n = n+x
m -= 3 ⇔ m = m-3
m *= 3 ⇔ m = m*3
```

etc.

Auto-réassignations (2/3)

- N.B.: `+=` fonctionne tant avec des variables numériques que caractères
- Ex.:
```
n += 31;
msg += ' et à bientôt!';
```
- Très utile pour composer graduellement un message

Exemple 150

Auto-réassignations (3/3)

- L'utilisation de ces opérateurs est facultative, car on a toujours le choix d'écrire l'auto-réassignation au long
- C'est juste une façon un peu plus concise d'exprimer les choses
- Peuvent s'utiliser avec des LHS "complexes"

Incrémentation et décrémentations (1/5)

- Il s'agit de formes très compactes d'*auto-réassignation* pour variables *numériques*
- Exemples :

```
i++
x--
++nClients
--tailleFile
```


Incrémentations et décrémentations (2/5)

- Deux types :
  - Incrémentation: `++`
  - Décrémentation : `--`
- Deux formes :
  - L'opérateur *avant* la variable numérique

    Ex.: `++i`
  - L'opérateur *après* la variable numérique

    Ex.: `i++`

Incrémentations et décrémentations (3/5)

- Incrémentations :

  ++n

  - Retourne (et réassigne) la nouvelle valeur
  - En fait, c'est une abréviation de "+= 1"

  n++

  - Réassigne la nouvelle valeur, mais ***retourne l'ancienne***
  - En fait, c'est une abréviation de "(… += 1) - 1"

Incrémentations et décrémentations (4/5)

- Décrémentations :

  --n

  - Retourne (et réassigne) la nouvelle valeur
  - En fait, c'est une abréviation de "**-=** 1"

  n--

  - Réassigne la nouvelle valeur, mais ***retourne l'ancienne***
  - En fait, c'est une abréviation de "(… **-=** 1) + 1"

Incrémentations et décrémentations (5/5)

- Très utilisées dans les boucles ("while", "for", etc.)
- Ces opérateurs ont très haute priorité
  - Rarement besoin de les parenthéser
- Peuvent s'utiliser avec des LHS "complexes"

Conversions explicites (1/4)

- La plupart du temps, les conversions implicites (automatiques) de nombre à caractère et vice-versa font exactement ce qu'on veut
  - Ex.:
```
"3.56" * Math.PI
"La réponse est " + (1.15*x)
```


Conversions explicites (2/4)

- Mais il arrive qu'on veuille simplement convertir une valeur d'un type à un autre sans autre transformation
  - C'est ce qu'on appelle une conversion *explicite*
- Il y a de nombreuses façons de le faire; on montre ici les *plus simples*

Conversions explicites (3/4)

- De nombre à chaîne, concaténer avec la chaîne vide :

  nombre `+ ""` (ou `""` `+` *nombre*)

  – *nombre* représente n'importe quelle expression donnant une valeur numérique

- Exemples

```
123.45 + "" ↔ "123.45"
"" + (10+7) ↔ "17"
```

N.B.: On peut aussi faire `String(`*nombre*`)`
PYM : Oubliez ça !

Conversions explicites (4/4)

- De chaîne à nombre, utiliser le + unaire !

  +*chaîne*

  – *chaîne* représente n'importe quelle expression donnant une valeur caractère

- Exemples :

  `+"789"` ↔ `789`

  `+"0765"` ↔ `765`

  `+"78z9"` ↔ `NaN` (valeur simple NaN)

N.B.: On peut aussi faire `Number(`*chaîne*`)`
PYM : Oubliez ça !

N.B.: Pour vérifier si une expression donne NaN, il *faut* utiliser la fonction `isNaN(`*expression*`)`, qui retourne true ou false

Encore quelques opérations sur valeurs caractère

- *chaine*.trimRight()
  - Enlève les blancs à droite
- *chaine*.trimLeft()
  - Enlève les blancs à gauche
- *chaine*.trim()
  - Enlève les blancs des deux côtés (la plus utile)

' abc '.trim() → "abc"

'a c '.trimRight() → "a c"

(suite)

- *chaine*.replace('rech', 'remplac')
  - Remplace la *première occurrence* de 'rech' dans la chaîne sujet (*chaine*) par 'remplac'

    'abcb'.replace('b', 'BON') → 'aBONcb'
- Exemple : présenter une valeur numérique avec la virgule décimale :

  (7/13 + "").replace('.', ',') → "0,5384615384615384"

Oui, c'est possible de changer *toutes* les occurrences d'un coup avec `.replaceAll()`

Fonction `console.log()`

Expérience utilisatrice ↑

- Améliorer le plus possible avec trim() :
  - Exercices après C2 - 4h (saisons)
  - Exercices après C2 - 4f et 4g (prénom)
  - Exemples 030, 060
- À faire individuellement

À cet après-midi !

Exercice court #2

=

1d dans la série après C3

