# Labo 6B : expressions régulières

**Objectifs**

- Construire des expressions régulières
- Utiliser des expressions régulières dans des contextes et des buts divers (éditeurs de texte, scripts, transformation de texte style BBCode)
- Manipuler des expressions régulières en Javascript

## Exercice 1 : La syntaxe des expressions régulières

Les expressions régulières représentent des motifs (patterns) qui correspondent à une ou à plusieurs chaînes de caractères. Elles sont construites à partir d'une série de symboles dont certains ont une signification particulière.

L'objectif de cet exercice est de découvrir la syntaxe des expressions régulières à travers des exercices où il vous sera demandé de construire des expressions régulières qui permettent de capturer certaines chaînes de caractères données (c'est-à-dire dont le motif correspond à ces chaînes de caractères). Dans certains exercices, on donnera également une liste de chaînes de caractères qui devront **ne pas** correspondre à l'expression régulière à construire.

Les exercices sont groupés en plusieurs séries qui illustrent chacune certains types d'expressions régulières.

### Étape 1 : mise en place, séries d'exercices

Récupérez l'archive `regexp.zip` et extrayez les fichiers qu'il contient dans un répertoire. Ouvrez ensuite la page HTML sous Firefox.

L'interface vous permet de voyager entre les différentes séries et entre les exercices d'une même série. À chaque fois, l'objectif est d'entrer une expression régulière qui répond aux exigences demandées. Ces exigences sont les suivantes : l'expression régulière devra capturer tous les exemples cités dans la colonne « Doit capturer » sans toutefois correspondre aux exemples de la colonne « Doit ignorer ».

Les chaînes capturées par l'expression que vous encodez sont automatiquement signalées par un surlignage vert. Lorsque tous les exemples de la colonne de gauche sont intégralement surlignés et qu'aucun de ceux de la colonne de droite n'est intégralement surligné, l'exercice est réussi.

Pour vous familiariser avec l'interface, suivez les instructions de l'exercice 1.

### Étape 2 : caractères et classes prédéfinies

Le tableau ci-dessous indique comment créer un motif correspondant à n'importe quel caractère. Pour créer un pattern correspondant à une séquence de caractères, il suffit de concaténer les codes en question.

Par exemple, le pattern `a.C` correspond à toutes les chaînes de 3 caractères commençant par a et finissant par C.

| Syntaxe | Signification |
|---|---|
| **A 3 c ! =** | Le caractère en question<br>Valable pour tous les caractères sauf \^$.*+?()[]{}\| |
| **\\ \$ \(** | Le caractère qui suit \<br>Permet de capturer un caractère spécial : \^$.*+?()[]{}\| |
| **\t** | Tabulation |
| **\n** | Retour à la ligne |

| | |
|---|---|
| **.** | N'importe quel caractère sauf le retour à la ligne |
| **\d** | Un chiffre (0, 1, 2, 3, 4, 5, 6, 7, 8 ou 9) |
| **\D** | Un caractère autre qu'un chiffre |
| **\w** | Un caractère alphanumérique (A, …, Z, a, …, Z, 0, …, 9, _) |
| **\W** | Un caractère autre qu'alphanumérique |
| **\s** | Un « blanc » (espace, tabulation, retour à la ligne) |
| **\S** | Un caractère autre qu'un « blanc » |

Chacun des caractères spéciaux cités ci-dessus possède une signification propre, qui est abordée dans cet atelier. En voici un aperçu :

- `\` est utilisé comme indicateur de début d'un code spécial ;
- `^` et `$` sont utilisés pour imposer que le motif se trouve en début/fin de ligne ;
- `.` est utilisé pour représenter n'importe quel caractère (ou presque) ;
- `*`, `+`, `?`, `{` et `}` sont utilisés pour indiquer un motif répété ou optionnel ;
- `(` et `)` sont utilisés pour marquer des groupes ; et
- `[` et `]` sont utilisés pour définir des classes de caractères.

Notez que la classe `\w` correspondant aux caractères alphanumériques ASCII ; les lettres accentuées et autres lettres spéciales ne sont pas forcément reprises (cela peut varier d'une implémentation des expressions régulières à l'autre).

## Étape 3 : classes de caractères

Les symboles tels que `\d` ou encore `\S` permettent de capturer des catégories (ou classes) de caractères. À côté de ces classes prédéfinies, les expressions régulières permettent également de construire des classes définies « manuellement », en citant les options possibles entre crochets.

| Syntaxe | Signification |
|---|---|
| **[abcd]** | Un caractère parmi ceux qui sont cités |
| **[.!?]** | Entre crochets, on peut citer n'importe quel caractère (même un caractère spécial) sauf les trois suivants : ]-\ |
| **[[\]]** | Si on veut citer un de ces trois caractères, il faut l'échapper via \ ; l'expression régulière à gauche capture un crochet (ouvrant ou fermant) |
| **[a-z]** | Entre les crochets, on peut aussi utiliser des intervalles ; ici, n'importe quel caractère entre a et z |
| **[À-ÿ]** | N'importe quel caractère accenté |
| **[abc5-9]** | Un caractère parmi a, b, c, 5, 6, 7, 8, 9 |
| **[\wÀ-ÿ]** | Un caractère alphanumérique (au sens de la langue française)<br>Note : donné à titre d'information, inutile de l'utiliser dans les exercices |
| **[^abcd]** | Si on place ^ juste après le crochet ouvrant, on capture n'importe quel caractère *sauf* ceux qui sont cités ; ici, n'importe quel caractère sauf a, b, c et d |
| **[^0-9]** | N'importe quel caractère sauf un chiffre (= [^\d] = \D) |
| **[^^]** | N'importe quel caractère sauf ^ |

## Étape 4 : répétitions

Toutes les syntaxes présentées dans les étapes précédentes correspondent à un seul caractère, mais on peut les concaténer pour former des patterns pour des chaînes plus complexes. On peut aussi les faire suivre de décorateurs spécifiques pour indiquer des répétitions ou des éléments optionnels.

| Syntaxe | Signification |
|---|---|
| **a?** | Soit un « a » soit rien du tout (élément optionnel) |
| **10?1** | Correspond aux chaînes 11 et 101 (le 0 est optionnel) |
| **\d*** | Une séquence de 0, 1 ou plusieurs chiffres |
| **\s+** | Une séquence de 1 ou plusieurs blancs |
| **[abc]+** | Une séquence de lettres parmi a, b et c avec au moins une lettre |
| **\w{3}** | Une séquence de 3 caractères alphanumériques |
| **\w{3,}** | Une séquence d'au moins 3 caractères alphanumériques |
| **\w{3,6}** | Une séquence de 3, 4, 5 ou 6 caractères alphanumériques |

## Étape 5 : groupements et disjonctions

Les décorateurs du tableau ci-dessus ne s'utilisent pas seulement sur des patterns correspondant à 1 caractère. Pour les faire porter sur une séquence de caractères, il faut généralement utiliser des parenthèses.

Par exemple, l'expression régulière `A\d{3}` va capturer toutes les chaînes qui commencent par « A » suivi de trois chiffres. Si on ajoute des parenthèses, on obtient l'expression régulière `(A\d){3}` qui, elle, correspond à toutes les chaînes de la forme « A1A2A3 » (où 1, 2 et 3 peuvent être remplacés par n'importe quels chiffres).

Par défaut, chaque fois qu'on utilise des parenthèses dans une expression régulière, la chaîne qui correspond au contenu de ces parenthèses est mémorisée. En utilisant des notations telles que `\1` (pour le premier groupe de parenthèses), `\2` (pour le second groupe de parenthèses), etc., on peut construire des expressions régulières testant qu'une sous-chaîne apparaît plusieurs fois.

Les parenthèses permettent également de regrouper des alternatives, via la disjonction.

Le tableau suivant présente des exemples pour toutes ces nouvelles options.

| Syntaxe | Signification |
|---|---|
| **(ab)+** | Une chaîne répétant « ab » 1 ou plusieurs fois |
| **(\d)-\1** | Les chaînes « 0-0 », « 1-1 », « 2-2 », « 3-3 », …, « 9-9 » |
| **(\d)(\d)-\1\2** | Les chaînes telles que « 00-00 », « 01-01 », « 47-47 », …<br>= (\d\d)-\1 |
| **b(oule\|ill)** | Les chaînes « boule » et « bill »<br>\| indique une disjonction |
| **(?:\d)(\d)-\1** | Les chaînes telles que « 01-1 », « 42-2 », « 33-3 », …<br>(?: ) permet de grouper des symboles sans toutefois mémoriser le contenu ; \1 fait donc référence au (\d) |

Le fait de mémoriser des sous-expressions (en les entourant de parenthèses) n'est pas seulement utile pour tester si la sous-expression en question se répète ; c'est aussi un élément-clef qui prendra toute son importance dans les exercices de type « chercher/remplacer ».

Les références du type \1 sont aussi appelées des « backreferences ».

**Note - numérotation des groupements**

Comment savoir à quel numéro est associé un groupement ? La règle est simple : les numéros sont attribués aux parenthèses ouvrantes en lisant l'expression régulière de la gauche vers la droite. (Les groupes anonymes correspondant à la syntaxe `(?: … )` ne sont pas pris en compte.)

Par exemple, dans `(A(\d)(!))`, `\1` correspondra à la chaîne de 3 caractères, `\2` au chiffre et `\3` au point d'exclamation.

Si un groupement est répété, il ne reçoit qu'un seul numéro et sa référence correspond à la dernière correspondance de ce groupe. Ainsi, quand on fait correspondre `(\d-)*` au texte « 1-2-4- », la seule référence valable est `\1` (même si on utilise plusieurs fois `\d-`, cela ne crée pas de références `\2`, `\3` ou autre). Et, dans ce cas-ci, la référence `\1` correspondra à « 4- », à savoir la dernière correspondance. Par exemple, l'expression régulière `(\d)+\1` correspondra à toute séquence de chiffres dont le dernier est répété.

## Étape 6 : les assertions

Les expressions régulières construites jusqu'ici décrivent un motif qui correspond à certaines chaînes de caractères. Il est parfois nécessaire de restreindre la recherche d'un motif aux chaînes de caractères qui occupent des positions particulières.

Par exemple, pour rechercher tous les mots en -ou dans un texte, on pourrait considérer l'expression régulière `.*ou` ; cependant, celle-ci va capturer toutes les séquences de caractères qui se terminent par « ou ». Ainsi, dans « Un genou posé sur le goudron », l'expression régulière va capturer non seulement « genou » mais aussi « gou » de « goudron ». Pire encore, elle capturera aussi « nou », « enou » et « n genou ».

Les décorateurs décrits dans le tableau suivant permettent de traduire des conditions (on parle aussi « d'assertions ») relatives à la position de la chaîne recherchée.

| Syntaxe | Signification |
|---|---|
| **^M** | Un « M » situé au début du texte |
| **\.$** | Le point final situé à la fin du texte |
| **^.*$** | Le texte tout entier (vu que .* doit s'étendre du début du texte jusqu'à la fin) ; alors que .* seul pourrait capturer n'importe quelle partie du texte |
| **\bgu** | Toute chaîne « gu » située au début d'un mot |
| **al\b** | Toute chaîne « al » située à la fin d'un mot |
| **\bou\b** | Tout mot « ou » (mais pas les « ou » à l'intérieur d'un mot plus long) |

| | |
|---|---|
| **\Bou\B** | Toute chaîne « ou » située à l'intérieur d'un mot (ni seul, ni au début, ni à la fin d'un mot) |
| **#(?=\d)** | Tous les « # » situés juste avant un chiffre |
| **\.(?= )** | Tous les « . » situés juste avant une espace |
| **,(?! )** | Toutes les « , » qui ne sont pas suivies d'une espace |

Quelques remarques importantes :

- Les symboles `^` et `$` repèrent généralement le début et la fin du texte analysé ; ils correspondent donc à deux positions bien précises dans le texte. En mode « multiligne », leur signification est légèrement différente : ils correspondent à n'importe quel(le) début / fin de ligne. Voici ce que donnent les deux exemples du tableau en mode « multiligne ».

| Syntaxe | Signification |
|---|---|
| **^M** | Tout « M » situé en début de texte ou après un retour à la ligne |
| **\.$** | Tout « . » situé en fin de ligne (avant un retour à la ligne) ou à la fin du texte |
| **^.*$** | Toute ligne complète du texte |

- Le symbole `\b` est une abréviation pour « boundary » (bord, frontière) ; il représente n'importe quelle position correspondant à la frontière entre un mot et ce qui l'entoure, c'est-à-dire au début ou à la fin d'un mot.

  Plus précisément, le symbole `\b` correspond à toute position située entre deux caractères dont l'un est alphanumérique (lettre, chiffre ou _) et l'autre n'est pas alphanumérique (espace, retour à la ligne, autre symbole de ponctuation), ou bien au début ou à la fin du texte.

  Tout comme `\D`, `\W` et `\S` correspondent aux inverses de `\d`, `\w` et `\s`, le symbole `\B` correspond à l'inverse de `\b`. Il capture donc toutes les positions qui ne correspondent pas au début ni à la fin d'un mot.

- Les syntaxes `(?= … )` et `(?! … )` sont des « look-ahead » (= observer ce qui suit), le premier étant un look-ahead positif et le second, un look-ahead négatif. Le look-ahead positif capture n'importe quelle position qui est suivie par une chaîne de caractères qui correspond à l'expression régulière indiquée dans les parenthèses. À ne pas confondre avec `( … )` et `(?: … )` qui permettent respectivement de capturer un groupe qui sera mémorisé ou pas.

  Notez que la chaîne correspondant à un look-ahead n'est pas capturée. Ainsi, si on analyse le texte « En #3, les 7 mousquetaires » avec l'expression régulière `#(?=\d)`, la correspondance sera la chaîne « # ». Si on utilise l'expression régulière `#\d`, la correspondance sera la chaîne « #3 ».

  Utilisé en fin d'expression régulière, un look-ahead permet de ne cibler que les chaînes de caractères qui sont suivies par un élément donné (par exemple les « # », mais

seulement ceux suivis par un chiffre dans le cas de l'exemple `#(?=\d)` ).

Utilisé en début d'expression régulière, un look-ahead permet de combiner des conditions. Par exemple, l'expression régulière `(?=123)\d{5}` capture les chaînes composées de 5 chiffres et qui commencent à une position suivie par « 123 », c'est-à-dire les chaînes de 5 chiffres commençant par 123. Dans ce cas-ci, on aurait tout aussi bien pu utiliser `123\d{2}` mais ce n'est pas le cas pour `(?=.{0,5}X).{6}`.

Cette dernière expression régulière capture les séquences de 6 caractères qui commencent à une position à partir de laquelle on peut trouver une chaîne correspondant à `.{0,5}X` (c'est-à-dire une position telle qu'on trouve au moins un « X » dans les 6 caractères qui suivent). Notez que l'expression régulière `.{0,5}X` est équivalente à la disjonction `X|.X|..X|...X|....X|.....X`. Au final, l'expression régulière `(?=.{0,5}X).{6}` capturera toutes les chaînes de 6 caractères qui comportent au moins un X. Ce type d'expressions régulières permet entre autres de vérifier des conditions de sécurité sur les mots de passe (par exemple : au moins 8 caractères avec au moins un chiffre et au moins une lettre).

Un « look-ahead » négatif peut être utile pour cibler les chaînes de caractères qui ne correspondent pas à un motif donné. Par exemple, l'expression régulière `(?!34)\d\d` capturera tous les nombres à deux chiffres qui commencent à une position à partir de laquelle on ne peut pas lire 34, c'est-à-dire tous les nombres à deux chiffres sauf 34 !

## Exercice 2 : Expressions régulières sous Notepad++

Les expressions régulières sont utiles pour manipuler et transformer des textes contenant des données respectant un certain motif. La plupart des éditeurs de texte proposent des outils relatifs aux expressions régulières ; dans le cadre de cet exercice, on utilisera Notepad++.

### Étape 1 : utilisation de regexp sous Notepad++

Ouvrez le fichier intitulé `dates.txt` sous Notepad++. Ou, plutôt, ouvrez-en une copie afin de pouvoir repartir de la version originale plus tard. Jetez un coup d'œil à son contenu : des dates et, pour brouiller les pistes, quelques fractions.

Sous Notepad, dans le menu Search, cliquez sur « Find ». Dans la fenêtre qui s'ouvre, assurez-vous que l'option « Regular expression » (tout en bas) est bien sélectionnée. Cela vous permet d'indiquer votre motif de recherche sous la forme d'expressions régulières.

Entrez comme expression à rechercher `\d{4}`. Vous devriez être capable de déterminer le motif auquel cette expression correspond. Cliquez sur le bouton « Find All in Current Document » et observez le résultat de la recherche dans la nouvelle fenêtre.

### Étape 2 : utilisation de regexp pour des analyses statistiques

Notez que chacune des dates est écrite au format américain : le premier nombre est le mois, puis vient ensuite le jour et, en dernier lieu, l'année.

En utilisant une recherche par expression régulière (même procédé que dans l'étape précédente), répondez aux questions suivantes.

1. Combien y a-t-il de dates correspondant au mois de décembre (12) ?
   Réponse attendue : 8
   Question : avez-vous utilisé un décorateur de type « annotation » ?

2. Parmi toutes les dates, combien tombent pendant les grandes vacances (c'est-à-dire juillet ou août, mois 7 ou 8) ?
   *Réponse attendue : 16*

3. Combien y a-t-il de dates postérieures au 1er janvier 2000 ?
   *Réponse attendue : 9*

4. Parmi toutes les dates, combien ont un jour et un mois qui s'écrivent tous les deux avec un seul chiffre ?
   *Réponse attendue : 25*

5. Combien de fractions se trouvent dans le fichier ?
   *Réponse attendue : 8*

6. Parmi toutes les dates, combien ont un mois qui s'écrit en un seul chiffre et qui se retrouve parmi les 4 chiffres de l'année ?
   *Réponse attendue : 27*
   ^(\d)/.*/.*\1.*

## Étape 3 : regexp et remplacements

On utilise souvent les expressions régulières pour modifier le format de données répondant à un certain motif.

Dans un premier temps, on va appliquer cette technique pour s'assure que tous les numéros de jour sont bien écrits en deux chiffres (ainsi, le jour « 7 » sera écrit « 07 »). Les numéros de jour à un chiffre sont assez faciles à repérer : il s'agit d'un chiffre unique coincé entre deux caractères « / ».

Dans le menu Search, choisissez cette fois-ci l'option « Replace » et, une fois encore, assurez-vous que l'option « Regular expression » est bien sélectionnée.

Dans la partie « Find what », inscrivez l'expression régulière `/\d/`.

Dans la partie « Replace with », inscrivez `$&` puis cliquez sur « Replace all ».

Normalement, aucun changement ne devrait s'effectuer… c'est parce que, dans le cas d'un remplacement, la notation `$&` signifie « toute la chaîne de caractères capturée ». Bref, ici, vous venez de demander à Notepad++ de remplacer toutes les chaînes de caractères correspondant à `/\d/` par elles-mêmes…

L'objectif est ici d'ajouter un 0 devant le chiffre du jour. Pour ce faire, on a besoin de pouvoir demander le remplacement de la chaîne par « / » suivi de « 0 » suivi du chiffre du jour suivi de « / ».

Pour pouvoir faire référence au chiffre du jour, il faut le capturer dans un groupe. Remplacez donc l'expression régulière de recherche par `/(\d)/`. Désormais, si on veut faire référence à ce chiffre au sein de l'expression régulière, on peut utiliser la notation `/1`.

Pour faire référence au contenu d'un groupe au sein de l'expression de remplacement, on utilise une notation différente : `$1`. Indiquez donc comme expression de remplacement `/0$1/` puis cliquez sur « Replace all ».

Cette fois-ci, l'action désirée devrait être accomplie.

Note : si on veut utiliser le caractère $ dans la chaîne de remplacement, on le note `$$`.

## Étape 4 : applications du remplacement

En utilisant le remplacement, effectuez les modifications suivantes sur le fichier. Note : en cas d'erreur lors d'une opération de remplacement, vous pouvez utiliser Ctrl-Z pour revenir en arrière.

1. Assurez-vous que les numéros de mois soient tous écrits en 2 chiffres. Note : pour cibler les numéros de mois (et éviter les numérateurs des fractions), un « look-ahead » pourra être utile. (^.)(?=/.*/\d{4})

   0$1
2. Remplacez toutes les lignes qui ne comportent pas une date par le texte « Ceci n'est pas une date » suivi, entre parenthèses, du contenu de la ligne. Par exemple, la ligne « 7/8 » deviendra « Ceci n'est pas une date (7/8) ». Note : un look-ahead négatif pourrait être utile. \d{1,3}/\d{1,3}(?!/\d{4})$

   Ceci n'est pas une date ($&)
3. Finalement, transformez toutes les dates au format européen, à savoir jj/mm/AAAA.

   (\d{2})/(\d{2})/(\d{4})

   $2/$1/$3

## Exercice 3 : Manipulations textuelles

Dans cet exercice et les suivants, ce sera à vous de découvrir les opérations de recherche et/ou de remplacement à effectuer pour arriver aux objectifs décrits. Dans tous les cas, les expressions régulières peuvent vous faciliter grandement la tâche.

### Application 1 : renommage de fichiers

Créez un nouveau répertoire et placez-y le contenu de l'archive `movies.zip` (une bonne trentaine de fichiers avi plutôt vides). Disons qu'il s'agit d'une série de vidéos récupérés à gauche et à droite et correspondant à divers films inspirés de jeux vidéos.

Deux observations : dans les noms des fichiers, tous les espaces ont été remplacés par des points ; de plus, chaque nom de film est suivi de la date de parution entre parenthèses.

Votre objectif est de renommer tous ces fichiers pour placer des espaces au lieu des points et pour faire apparaître la date en tête de nom plutôt qu'en fin de nom (pour pouvoir trier les fichiers par année de parution par exemple).

**Récupérer la liste des fichiers.** Maintenez la touche Shift enfoncée et faites un clic droit dans le répertoire où vous avez placés les fichiers. Choisissez l'option « Open PowerShell Window here » pour ouvrir une fenêtre vous permettant d'entrer directement des commandes.

Entrez la commande `dir` (pour directory), ce qui vous donne la liste des fichiers du répertoire ainsi que diverses informations.

Pour obtenir uniquement les noms des fichiers, entrez `dir -Name`.

Et, finalement, pour envoyer ces informations dans un fichier texte plutôt que de les afficher à l'écran, utilisez `dir -Name > contenu.txt` (ou choisissez un autre nom pour le fichier recevant la liste des vidéos).

Vous pouvez maintenant fermer la fenêtre PowerShell (soit directement soit en entrant la commande `exit`).

**Préparer le travail.** L'instruction DOS/PowerShell permettant de renommer un fichier utilise la syntaxe suivante : `ren <ancientNom> <nouveauNom>`. L'idée est de partir du fichier `contenu.txt` et de le modifier jusqu'à obtenir une liste de commandes `ren` à exécuter pour obtenir le résultat final. Pour ce faire, ouvrez le fichier sous Notepad++.

Pour l'instant, chaque ligne contient le nom d'un fichier. Globalement, il va falloir remplacer chaque ligne par `ren nomFichier nouveauNom`, ce qu'on peut faire en plusieurs étapes. Dans une première étape, on peut se contenter d'avoir des lignes du style `ren nomFichier nomFichier` (des opérations qui ne font rien) puis, ensuite, modifier les seconds noms pour obtenir ce qu'on veut.

Comme il va falloir travailler sur ces seconds noms, autant se simplifier la tâche. Une manière de faire cela consiste à ajouter des caractères spéciaux qui permettront ensuite plus facilement de cibler ces « seconds noms ». On peut utiliser ici n'importe quel caractère qui n'apparaît pas dans les noms de films ; disons, par exemple, #.

En bref :

- Dans un premier temps, faites en sorte que chaque ligne devienne `ren nom #nom#`.
- Ensuite, ciblez les noms placés entre # et opérez-y les modifications nécessaires.

Pour rappel, les opérations nécessaires sont : (a) remplacer les points (mais pas tous) par des espaces… vous pouvez utiliser un look-ahead pour cibler les points à remplacer ; (b) déplacer l'année de parution en début de titre. (en cas de problème, voir l'indice en fin de section)

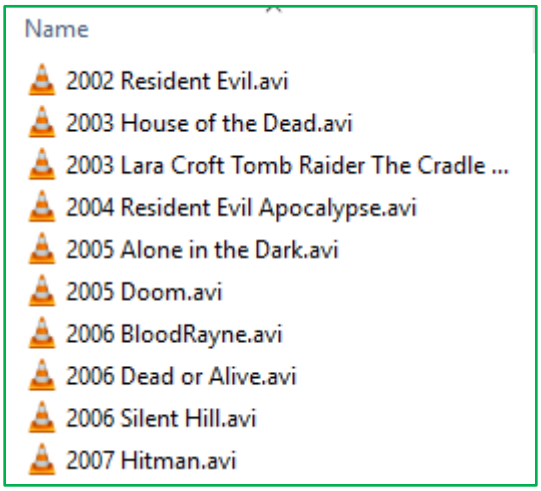
De plus, comme les nouveaux noms contiennent des espaces, il faut les encadrer par des guillemets. Au final, votre fichier devrait contenir des lignes similaires à ceci :

```
ren Alone.in.the.Dark(2005).avi "2005 Alone in the Dark.avi"
ren Assassin's.Creed(2016).avi "2016 Assassin's Creed.avi"
ren BloodRayne(2006).avi "2006 BloodRayne.avi"
ren Dead.or.Alive(2006).avi "2006 Dead or Alive.avi"
ren Doom(2005).avi "2005 Doom.avi"
```

**Lancer le travail.** Assurez-vous tout d'abord que le fichier de commandes est bien enregistré en encodage UTF-8 et qu'il se trouve bien dans le même répertoire que tous les fichiers vidéos.

Renommez-le ensuite en lui donnant l'extension `.bat` (pour Batch = ensemble de commandes).

Finalement, double-cliquez sur le fichier .bat pour l'exécuter. Si tout a été fait correctement, le contenu de votre répertoire devrait désormais ressembler à l'illustration ci-dessous.



## Application 2 : mise en forme d'un texte

Ouvrez le fichier `ip.html` sous Notepad++ et observez son contenu.

Il s'agit d'une page web reprenant une liste d'adresses IP. Les plus observateurs auront remarqué au premier coup d'œil que certaines des adresses sont incorrectes (une adresse IP devrait être constituée de 4 nombres entre 0 et 255 inclus séparés par des points ; hors, certaines des adresses comportent des nombres plus élevés que 255).

L'objectif ici est de rendre cette liste d'adresse IP plus lisible en la transformant en tableau similaire à celui présenté ci-contre.

| | | | |
|---|---|---|---|
| 37 | 60 | 71 | **268** |
| 214 | 71 | 166 | 215 |
| 201 | 220 | 253 | **260** |
| 6 | 2 | 32 | 255 |
| 128 | 208 | 153 | 164 |
| 176 | **262** | 30 | 238 |
| 124 | 189 | 221 | 47 |
| 26 | 164 | 97 | 102 |
| 215 | 216 | 131 | 84 |
| 149 | 152 | 82 | 145 |
| 158 | **286** | 193 | 61 |
| 61 | **261** | 117 | 128 |
| 145 | 151 | **266** | 189 |
| 119 | 157 | 37 | 81 |
| 96 | **281** | 221 | 241 |

Dans le tableau, les lignes correspondant à des adresses incorrectes seront écrites en rouge et, dans ces adresses, les nombres erronés (256 ou plus) seront affichés en gras. Pour ce faire, utilisez les styles définis dans le document HTML.

Une suggestion de plan d'action : repérez les adresses incorrectes (grâce à un look-ahead) et transformez les lignes en question en lignes du tableau (avec le style approprié) ; puis transformez les autres lignes (celles avec des adresses correctes) ; ensuite, occupez-vous des nombres à placer dans des cellules de tableau : construisez une expression régulière qui ciblera les valeurs incorrectes et « emballez-les » dans une balise `<td>` avec le style adéquat.

## Application 3 : balises BBCode sur forums

De nombreux gestionnaires de forums utilisent une syntaxe particulière pour permettre aux utilisateurs de poster des messages dont certaines parties sont en gras, en italique ou dans un style particulier. Dans la plupart des cas, il s'agit de la syntaxe appelée « BBCode » (ou Bulletin Board Code), qui utilise des crochets pour encadrer ses balises.

Le tableau suivant reprend quelques-unes des balises BBCode standards (la majorité des gestionnaires permettent également de définir de nouvelles balises BBCode en fonction des spécificités du forum en question).

| Syntaxe | Signification |
|---|---|
| **[b]…[/b]** | Texte en gras |
| **[i]…[/i]** | Texte en italique |
| **[url]…[/url]** | Transforme le contenu à l'intérieur des balises (qui est traité comme une URL) en lien menant vers cette URL.<br>Ex : [url]http://www.google.be[/url] |
| **[url=*url*]…[/url]** | Affiche le texte placé à l'intérieur des balises et le transforme en lien menant vers l'URL indiquée.<br>Ex : [url=http://www.google.be]Cliquez ici[/url] |

Dans le cadre de cet exercice, en plus des quatre balises BBCode décrites ci-dessus, on autorisera également l'utilisation du code suivant.

| Syntaxe | Signification |
|---|---|
| \"…\" | Affiche le texte placé entre les \" dans un style prédéfini correspondant à la classe « parole » (utilisez un span pour appliquer la classe à cette partie du texte). |

L'objectif de cet exercice est d'implémenter en Javascript un programme permettant de transformer le texte entré par l'utilisateur d'un forum en code HTML permettant d'afficher le message au bon format.

Le point de départ est la page HTML `forum.html`. Ouvrez ce fichier (ou une copie) sous Notepad++ et examinez-en le contenu.

La page en elle-même est divisée en deux parties. La partie supérieure (`#input`) permet à l'utilisateur d'entrer un message dans une textarea ; un bouton permet de vider la zone de texte et l'autre, d'envoyer le message entré. La seconde partie (`#forum`) est initialement vide (elle est située sous la séparation grise) ; elle est destinée à recevoir les messages postés.

Du côté Javascript, cela se fait via la fonction `ajoute()` qui lit le contenu de la textarea et s'occupe de créer un nouveau  de classe message dans la partie. Le contenu textuel de ce div est obtenu en appliquant la fonction `texteEnHTML` au texte entré. C'est donc dans le code de cette fonction `texteEnHTML` que vous devrez entrer les lignes Javascript responsables de la transformation du texte.

Quelques conseils :

- Vous utiliserez principalement la méthode « replace » définie dans `String.prototype.replace`. Consultez la documentation en ligne pour savoir comment cette méthode agit ; entre autres, une des questions que vous devriez vous poser est de savoir si cette méthode agit « en place » (en modifiant la chaîne de caractère) ou si elle produit une nouvelle chaîne de caractères.
- Commencez « petit ». Par exemple, concentrez-vous juste sur la balise [b] dans un premier temps, et effectuez quelques tests pour vérifier votre code.
- Tentez d'utiliser les deux manières d'écrire une expression régulière en Javascript : soit via un littéral entre « / » soit via la fonction constructrice `RegExp`. Dans le cas du littéral, pensez que les symboles « / » devront être échappés à l'intérieur des deux « / » qui encadrent l'expression (tout comme on doit échapper les guillemets dans une chaîne de caractères).
- Pensez aussi aux flags à utiliser !
- Une fois que vous aurez implémenté la balise [b], testez votre code avec le premier exemple donné ci-dessous. Si la portion de texte mise en gras est trop grande, consultez la note ci-dessous sur les quantificateurs greedy et lazy.
- Passez ensuite aux autres balises et effectuez d'autres tests (des tests personnels et les exemples donnés plus bas).

**Quatre exemples-tests et les résultats attendus** (vous pouvez copier/coller ces messages directement dans la textarea).

- C'est un premier test pour [b]mettre[/b] certaines parties du texte en [b]gras[/b].
- Bonjour tout le monde ! Moi, c'est [b]Jean-Kévin[/b] et j'adooooooore jouer à LOL. Je trouve ce jeu tout simplement génial, et d'ailleurs, je passe chaque jour sur le Reddit de Lol. C'est à l'adresse [url]https://www.reddit.com/r/leagueoflegends/[/url]. Vous pouvez m'y trouver sous le nom de [i]JeanKevinLovesLOL[/i].
- Salut. [b]Je[/b] trouve [b]que[/b] les [b]balises[/b] [url=https://en.wikipedia.org/wiki/BBCode]BBCode[/url] [b]sont[/b] vachement [b]fun[/b] alors [b]c'est[/b] possible [b]que[/b] j'en [b]abuse[/b] un [b]peu[/b].

- Une jeune femme s’avance. \"Bonjour, tout le monde !\" lance-t-elle sur un ton enjoué. Le nain assis à la taverne lève à peine les yeux de sa chope et émet un grognement. La femme se tourne vers lui. \"Que la langue naine est fascinante !\" s’exclame-t-elle. \"Permettez-moi de vous saluer à mon tour.\" Puis elle pousse un grognement si long et si guttural que même le nain ne peut s’empêcher de tourner la tête vers elle tout en levant un sourcil.

C'est un premier test pour **mettre** certaines parties du texte en **gras**.

Bonjour tout le monde ! Moi, c'est **Jean-Kévin** et j'adooooooore jouer à LOL. Je trouve ce jeu tout simplement génial, et d'ailleurs, je passe chaque jour sur le Reddit de Lol. C'est à l'adresse https://www.reddit.com/r/leagueoflegends/. Vous pouvez m'y trouver sous le nom de *JeanKevinLovesLOL*.

Salut. **Je** trouve **que** les **balises** BBCode **sont** vachement **fun** alors **c'est** possible **que** j'en **abuse** un **peu**.

Une jeune femme s’avance. *« Bonjour, tout le monde ! »* lance-t-elle sur un ton enjoué. Le nain assis à la taverne lève à peine les yeux de sa chope et émet un grognement. La femme se tourne vers lui. *« Que la langue naine est fascinante ! »* s’exclame-t-elle. *« Permettez-moi de vous saluer à mon tour. »* Puis elle pousse un grognement si long et si guttural que même le nain ne peut s’empêcher de tourner la tête vers elle tout en levant un sourcil.

**Note - quantificateurs « greedy » et « lazy »**

Les quantificateurs sont les décorateurs tels que `*`, `+` et `{…}` correspondant à des répétitions. Par défaut, ils sont en mode « greedy » (avare), ce qui signifie qu'ils vont capturer le plus grand nombre de caractères possible.

Par exemple, lors de l'analyse du texte « 12345a », les expressions régulières `\d*` et `\d+` vont automatiquement capturer les 5 chiffres.

Cela peut poser des problèmes quand on veut cibler des zones de texte délimités par des balises. Par exemple, dans le texte « a12abba345a », si on recherche des séquences de caractères entourées par des « a » via l'expression régulière `a.*a`, celle-ci capturera le texte tout entier, alors qu'on aurait peut-être préféré capturer « a12a » dans un premier temps, puis « a345a » ensuite.

Dans certains cas, on peut s'en sortir en imposant qu'il n'y ait aucune autre balise à l'intérieur du texte capturé. Dans le cas de l'exemple « a12abba345a », cela reviendrait à utiliser l'expression régulière `a[^a]*a`, qui impose que les caractères capturés soient « a », puis autre chose que a, puis un « a » final. Note : vous pouvez utiliser le code de l'Exercice 1 pour tester ces exemples en entrant les textes dans le cadre « test libre ».

Une autre manière de procéder consiste à imposer aux quantificateurs d'être « lazy », c'est-à-dire de capturer le moins de caractères possibles. Cela se fait en ajoutant un point d'interrogation après le quantificateur. Dans ce cas-ci, l'expression régulière devient `a.*?a`.

De manière générale, quand on se trouve dans un contexte de « balises » délimitant le texte, il est souvent préférable d'utiliser les versions lazy des quantificateurs.

| Syntaxe | Signification |
|---|---|
| **\d*?** | Une séquence de 0, 1 ou plusieurs chiffres (le moins possible) |
| **\s+?** | Une séquence de 1 ou plusieurs blancs (le moins possible) |
| **\w{3,}?** | Une séquence d'au moins 3 caractères alphanumériques (le moins possible) |
| **\w{3,6}?** | Une séquence de 3 à 6 caractères alphanumériques (le moins possible) |

## Application 3 (suite) : exercices supplémentaires

Voici quelques ajouts optionnels pour l'application de simulation de forum.

- Dans les applications web qui reçoivent des informations textuelles de la part des utilisateurs, il vaut mieux être prudent… Plutôt que de faire un long discours, entrez le message suivant dans la simulation de forum.

  ```
  Hello ! <span style="font-size: 250%">Je gâche le visuel du forum !</span> <div class="message">Hop!</div><script>alert("Message encombrant !");</script>
  ```

  Dans le cas de cette application, on ne peut guère faire pire que modifier le visuel, parce que les messages sont introduits via des « innerHTML ». Mais si les messages étaient stockés dans une base de données sur un serveur et envoyés au sein du code HTML initial de la page, les conséquences pourraient être pires. Entre autres, le code contenu dans les balises <script> s'exécuterait…

Il y a un moyen assez simple de faire en sorte que le texte entré par l'utilisateur ne puisse activer aucune balise HTML (et donc, aucun code interprétable), qui consiste à effectuer un remplacement (via une expression régulière). Voyez-vous lequel ?

- Vous pouvez également ajouter une fonctionnalité au forum : il repère toutes les adresses mail inclues dans le texte du message et les remplace par des liens permettant d'envoyer un message mail. Vous pouvez trouver des expressions régulières permettant de capturer les adresses mail en effectuant une recherche sur le web. Pour rappel, le code HTML pour permettre d'envoyer un mail est le suivant :

```
<a href="mailto:adresseMail">texte</a>
```

- La méthode `String.prototype.replace` peut également prendre une fonction comme second argument. Consultez sa description dans la documentation en ligne puis servez-vous en pour ajouter la fonctionnalité suivante au forum. Dans chaque message, on repérera les sommes d'argent indiquées sous la forme nombre + symbole « $ » et on les remplacera par la valeur équivalente en Euros (au taux de conversion actuel). Dans un premier temps, vous pouvez vous limiter aux sommes d'argent exprimées par un nombre entier de dollars.

---

### Indice pour l'application 1

Les points à remplacer sont ceux qui vérifient les deux conditions suivantes : (1) il y a encore au moins 1 point par après (car on ne doit pas remplacer le dernier point, qui précède l'extension) et (2) il n'y a qu'un symbole # par après. Ces deux conditions peuvent se traduire par deux look-ahead, le premier positif et le second négatif (pas 2 « # »).