INTRODUCTION À L'ALGORITHMIQUE

INSTRUCTIONS ET VARIABLES

2014-07

netcat_ / Glenn ROLLAND

INSTRUCTIONS

FORME DES INSTRUCTIONS

- Dans l'ordre chronologique
- Séparées par des ";"
- Les espaces blancs ne comptent pas (espaces, tabulations, lignes)

```
instruction;
instruction;
instruction;
```

COMMENTAIRES

```
// commentaire sur une seule ligne

/* commentaire qui démarre sur une ligne
   et qui finit seulement lorsqu'on
   rencontre la balise de fin */
```

MOTS RÉSERVÉS

```
if else for do while switch case break typeof new
delete null undefined ...
```

- Ces mots ont **un sens précis** pour le langage JavaScript
- Il **ne doivent pas être utilisés comme variables**

ENTRÉES - SORTIES

ENTRÉES - DEMANDER UNE VALEUR

- prompt(*unMessage*);

```
prompt("Bonjour monde");
```

SORTIES - AFFICHER UNE VALEUR

- alert();
- console.log();

```
alert("Bonjour monde");
```

VARIABLES

DÉCLARATION DE VARIABLES

- *Étiquettes* sur des zones de mémoire
- Attention à donner des noms significatifs !

```
var nomDeMaVariable;
var variable1, variable2;
```

ASSIGNATION DE VARIABLE

- Remplissage d'une case mémoire
- Attache à cette case une *étiquette* qui porte son nom

```
maVariable = 1;
jolieVariable = "toto";
```

UTILISATION DE VARIABLES

- Retrouve la case mémoire qui possède l'*étiquette* correspondant au nom de la variable

```
alert(maVariable);
maVariable = maVariable + 1;
monAutreVariable = uneAutreVariable;
```

EXERCICES

- **declare-store-display** : Déclarer trois variables, y stocker les valeurs 1, 2 et 3. Afficher le contenu de la deuxième variable.
- **prompt-store-display** : Demande à l'utilisateur son nom, puis lui affiche un message de bienvenue

TYPES DE VARIABLES

Type	Description
number	entiers ou flottants (contient la valeur **NaN**)
string	chaines de caractères
boolean	valeur booléenne (VRAI ou FAUX)
undefined	valeur non définie
function	fonctions
null	valeur nulle
object	structures de données complexes

TESTE DU TYPE AVEC `typeof`

```
typeof 2 // 'number'
typeof 3.3 // 'number'
typeof "toto" // 'string'
typeof true // 'boolean'
typeof undefined // 'undefined'
typeof function() { } // 'function'
typeof null // 'object'
typeof { } // 'object'
```

TYPAGE DES VARIABLES

- **fort ou faible** : selon capacité à
  - respecter les types et limiter les conversions
  - et assurer au plus tôt la détection des erreurs
- **statique ou dynamique** : vérification du type
  - lors de la compilation ?
  - ou bien à l'exécution ?
- **explicite, implicite** : est-ce que les types sont
  - indiqués à chaque fois par le développeur ?
  - devinés par le compilateur ou l'interpréteur ?

QUELQUES EXEMPLES DE TYPAGES

Langage	Sûreté	Dynamisme	Expressivité
Javascript	Faible	Dynamique	Implicite
C	Faible	Statique	Explicite
Java	Fort	Dynamique	Explicite
Ruby	Fort	Dynamique	Implicite
OCaml	Fort	Statique	Implicite

OPÉRATEURS ARITHMÉTIQUES

OPÉRATEURS ARITHMÉTIQUES

- Opérent sur des valeurs numériques (des littéraux ou des variables)
- Renvoient une valeur numérique.

ARITHMÉTIQUE DE BASE

- l'addition : +
- la soustraction : -
- la multiplication : *
- la division : /
- le modulo : %

ARITHMÉTIQUE... SUITE

- Incrément : ++
  - Opérateur unaire.
  - En préfixe (ex : ++x), renvoie la valeur *après* avoir ajouté un
  - En suffixe (ex : x++), renvoie la valeur *avant* d'ajouter un.
- Décrément : --
  - Opérateur unaire.
  - En préfixe (ex : --x), renvoie la valeur *après* avoir retiré un
  - En suffixe (ex : x--), renvoie la valeur *avant* de retirer un.
- Négation unaire : -

EXERCICES

- **prompt-and-sum** :
  - Demande à l'utilisateur un premier chiffre, puis un second.
  - Calcule le produit de la multiplication des deux chiffres.
  - Affiche le resultat.
- *Variante : essayer avec une addition à la place de la multiplication*
- *Attention : problème de parseInt :-)*

OPÉRATEURS BINAIRES

AND (ET) BINAIRE

Renvoie un 1 à chaque position binaire pour laquelle les bits des deux opérandes sont à 1.

```
a & b
```

OR (OU) BINAIRE

```
a | b
```

Renvoie un 1 à chaque position binaire pour laquelle au moins un des bits des deux opérandes est à 1.

XOR (OU EXCLUSIF) BINAIRE

Renvoie un 1 à chaque position binaire pour laquelle au plus un des bits des deux opérandes est à 1.

```
a ^ b
```

NOT (NON) BINAIRE

Inverse les bits de l'opérande.

```
~ a
```

DÉCALAGE BINAIRE À GAUCHE

Décale la représentation binaire de b bits sur la gauche et complète avec des zéros à droite.

```
a << b
```

DÉCALAGE BINAIRE À DROITE

Décale la représentation binaire de b bits sur la droite en ignorant les bits perdus.

```
a >> b
```

DÉCALAGE BINAIRE À DROITE EN COMPLÉTANT AVEC DES ZÉROS

Décale la représentation binaire de b bits sur la droite en ignorant les bits perdus et ajoute des zéros sur la gauche.

```
a >>> b
```

OPÉRATEURS DE COMPARAISON

ÉGALITÉ : ==

Renvoie `true` si les opérandes sont égaux.

```
3 == var1
"3" == var1

3 == '3'
```

INÉGALITÉ : !=

Renvoie `true` si les opérandes sont différents.

```
var1 != 4
var2 != "3"
```

ÉGALITÉ STRICTE : ===

Renvoie `true` si les opérandes sont égaux et de même type.

```
3 === var1
```

INÉGALITÉ STRICTE : !==

Renvoie `true` si les opérandes ne sont pas égaux et/ou ne sont pas de même type.

```
var1 !== "3"
3 !== '3'
```

SUPÉRIORITÉ STRICTE : >

Renvoie `true` si l'opérande gauche est supérieur (strictement) à l'opérande droit.

```
3 === var1
```

SUPÉRIORITÉ OU ÉGALITÉ : >=

Renvoie `true` si l'opérande gauche est supérieur ou égal à l'opérande droit.

```
longueur >= largeur
taille >= 3
```

INFÉRIORITÉ STRICTE : <

Renvoie `true` si l'opérande gauche est inférieur (strictement) à l'opérande droit.

```
a < b
"2" < "12"
```

INFÉRIORITÉ OU ÉGALITÉ : <=

Renvoie `true` si l'opérande gauche est inférieur ou égal à l'opérande droit.

```
var1 <= var2
var2 <= 5
```

EXERCICE

Écrire un programme qui :

- Demander le nom de l'utilisateur
- Stocke le résultat dans la variable `prenom`
- Teste si le prénom est "Marc"
- Affiche le résultat (`true` ou `false` ?)

OPÉRATEURS D'AFFECTATION

OPÉRATION ARITHMÉTIQUE ET AFFECTATION

Opérateur	Equivalence
`x += y`	`x = x + y`
`x -= y`	`x = x - y`
`x *= y`	`x = x * y`
`x /= y`	`x = x / y`
`x %= y`	`x = x % y`

OPÉRATION BINAIRE ET AFFECTATION

Opérateur	Equivalence
x <<= y	x = x << y
x >>= y	x = x >> y
x >>>= y	x = x >>> y
x &= y	x = x & y
x ^= y	x = x ^ y
`x	=y`

`x = x y`

CONVERSION DE TYPE

CONVERSIONS IMPLICITES

Dans certains contextes, Javascript effectue des conversions automatiques

```
1 == '1'
1 == true
```

CONVERSIONS IMPLICITES - CONTEXTES

- Opérateurs non-strict d'égalité : ==, < et >:
- Contexte boolen : `while(...)`, `if(...)`, etc:
  - `false` ⇒ `false`
  - `true` ⇒ `true`
  - `undefined` ⇒ `false`
  - `null` ⇒ `false`
  - `0` ⇒ `false`
  - `NaN` ⇒ `false`
  - `""` ⇒ `false`

CONVERSION IMPLICITES - QUELQUES RÈGLES

- Conversion vers un string
  - Si l'objet a une méthode `toString()`, elle est exécutée
  - Si la méthode `valueOf()` existe, elle est exécutée
  - Sinon une exception est lancée
- Conversion vers une valeur numérique:
  - Si la méthode `valueOf()` existe, elle est exécutée
  - Si l'objet a une méthode `toString()`, elle est exécutée
  - Sinon une exception est lancée
- On peut convertir explicitement: `String(obj)`, `Number(obj)`

CONVERSIONS EXPLICITES

- **parseInt**(*chaine*, *base*) - convertit une chaine de caractères en entier
- **parseFloat**(*chaine*, *base*) - convertit une chaine de caractères en flottant
- **Number**(*string*) - convertit une chaine de caractères en nombre (entier ou flottant, selon)
- **String**(*nombre*) - convertit un nombre en chaine de caractères

QUESTIONS

Suite: Structures conditionnelles