TRALALERE

Notions générales de programmation

Avec exemples en langage JavaScript

Bibliothèques Sans Frontières
Libraries Without Borders
INVESTISSEMENTS D'AVENIR
Caisse des Dépôts
l'Europe s'engage en Ile-de-France avec le FEDER
UNION EUROPÉENNE

V1.0
Auteur : Tralalere

Sommaire

Le javascript

Nous illustrons ci-après les grandes notions de programmation à l'aide d'exemples en javascript. Le javascript est un langage web, ce qui signifie qu'un navigateur Internet peut l'interpréter.

Le langage javascript permet de définir des instructions qui seront exécutées, les unes après les autres, de haut en bas.

Variables

La variable est une brique élémentaire de la programmation. Il s'agit d'un contenant, un « bocal », dans lequel on peut stocker une valeur. La valeur contenue dans une variable peut changer au cours du temps.

Le score peut être une variable, par exemple. Il est représenté par un nombre : il est généralement à 0 en début de partie et peut augmenter (ou parfois diminuer) selon les événements au cours de la partie.

Une variable peut stocker plein d'éléments différents : des nombres, des phrases, un statut (« vrai » ou « faux »), d'autres variables… On appelle ces différents éléments des « types ». On distingue généralement :

- les valeurs numériques

```
var chiffre = 0;
var chiffre = 0,5;
var chiffre = 1/2;
```

- les caractères

```
var lettre = 'a';
```

- les chaînes de caractères

```
var mot = 'mot';
var phrase = 'ceci est une phrase';
```

- les booléens (vrai/faux)

```
var test = true;
var test = false;
```

Pour déclarer une variable on utilise le mot-clé *var*. Une fois la variable spécifiée, il n'est plus nécessaire de répéter ce mot-clé :

```
var monChiffre = 18;
monChiffre = 22;
monChiffre = monChiffre - 2;
```

Ici, on déclare la variable « monChiffre » et on l'initialise à 18.
On remplace ensuite sa valeur par 22.
Enfin, on soustrait 2 à la valeur présente dans cette variable et on stocke le résultat dans cette même variable.
La valeur finale la variable « monChiffre » est donc 20.

Opérations

En code, nous trouvons les opérateurs suivants :

- « + » : addition
- « - » : soustraction
- « * » : multiplication
- « / » : division
- « ^ » : exposant. Il permet d'élever un nombre à une puissance.
- « % » : « modulo ». Il permet d'obtenir le reste d'une division entière.

Exemples :
3 / 2 = 1,5
6^3 = 6*6*6 = 216
17%7 = 3, car 17 divisé par 7 donne 2 et il reste 3
15%5 = 0, car 15 divisé par 5 donne 3 et il reste 0

Pour modifier une variable par le calcul, il existe quelques raccourcis :

Exemples :

i = i + 1 peut également s'écrire i += 1 ou encore i++
i = i + 10 peut également s'écrire i += 10

Comparateurs et conditions

Pour comparer des valeurs, nous utilisons les signes mathématiques suivants :

- « == » : strictement égal
- « > » : supérieur
- « >= » : supérieur ou égal
- « < » : inférieur
- « <= » : inférieur ou égal
- « != » : différent

```
if (i == 1)
/* Compare « i » à la valeur 1*/
/* Si « i » vaut 1, cette comparaison est égale à vrai (true)*/
/* Si « i » ne vaut pas 1 alors cette comparaison est égale à faux (false)*/

if (i != 1)
/* Compare « i » à la valeur 1*/
/* Si « i » vaut 1, cette comparaison est égale à faux (false)*/
/* Si « i » ne vaut pas 1 alors cette comparaison est égale à vrai (true)*/

if (i == 'salut')
/* Compare  « i » à la chaîne de caractère « salut »*/

var i = 0;
var j = 'test';
if (i == j)
/* Compare la variable « i » (0) à la variable « j » ('test') */
```

On peut aussi écrire la condition suivante :

```
if (i === 0)
/* Cette écriture compare les valeurs et le type */
```

On compare le caractère 'A' à la valeur 65. En ASCII (cf. chapitre dédié), 'A' vaut 65 donc la comparaison 'A' == 65 est vraie.

```
'A' === 'A' /*Vrai*/
'A' === 'B' /*Faux*/
'A' === 65 /*Faux*/
'A' == 65 /*Vrai*/
```

Règles logiques

Si on veut comparer plusieurs données, on peut utiliser les signes && (et) et || (ou).

```
if (vie >= 1 && saut >= 1)
/* Si la vie du joueur est supérieure à 1 ET que le saut du joueur est supérieur à 1, la condition est vérifiée et renvoie vrai (true)*/

if (vie >= 1 || saut >= 1)
/* Si la vie du joueur est supérieure à 1 OU que le saut du joueur est supérieur à 1, la condition est vérifiée*/
```

Boucles

Une boucle est une instruction permettant de répéter du code plusieurs fois à la suite.

Dans le cadre du développement d'un jeu, le code peut être répété sur plusieurs cycles du jeu, qu'on appelle les « frames* ».
À la manière d'une vidéo qui est une succession rapide de photos donnant l'illusion du mouvement, un jeu dessine à l'écran une image composée de dizaines de frames (images) par seconde qui vont produire un effet de mouvement, d'animation.

> **frame : En informatique, une frame est une image. Dans un jeu vidéo, la vitesse d'affichage se compte en Fps (frames per second). Si un jeu est fluide son affichage est compris entre 30 et 60 Fps.*
> *Si l'affichage est de 1 Fps, le programme affiche chaque nouvelle image à chaque seconde. Une animation qui défile à une image par seconde est très lente et donne un effet saccadé.*
>
> *Donc si on demande à notre programme de faire une action, comme danser dans une boucle, il va relancer cette fonction avant même qu'on voit le personnage commencer à danser. Cela revient à lui demander de recommencer à danser à l'infini . C'est parce que son animation est en plusieurs images et qu'on lui demande environ 30 fois par seconde de jouer la première image de son animation.*

La boucle « while » (tant que)

Le code inséré dans la boucle *while* s'effectue tant que la condition en paramètre de cette boucle est vérifiée.

```
var i = 0;

while (i < 10)
{
    i = i + 1;
}
/* ici la boucle effectue 10 tours, elle ajoute à i une unité à chaque tour */

while (true)
{
    console.log('coucou');
}
/* ici le paramètre est vrai, la boucle est donc infinie. Tant qu'on n'arrête pas le programme, « coucou » sera écrit dans la console, indéfiniment */
```

La boucle « for » (énumération)

La boucle *for* définit, dans ses paramètres, un compteur à incrémenter pour déterminer le nombre de tours. On peut utiliser ce compteur dans le code de la boucle.

Modèle :

```
for (initialisation ; condition de fin ; incrémentation)
{
    instruction;
}
```

Exemple :

```
for (var i = 0 ; i < 10 ; i ++)
{
    console.log(i);
}
/* ici la boucle effectue 10 tours, elle ajoute à i une unité à chaque tour */
/* Remarque : cet exemple a le même comportement que la boucle while avec le compteur i*/
```

Événement

L'événement est une instruction qui permet de déclencher du code lorsqu'un certain message ou état, émis par le programme, est capté.
Lorsque le joueur clique sur la page ou tape sur le clavier par exemple, des messages sont propagés par le programme, correspondant à l'action que l'utilisateur vient d'effectuer. On peut donc capter ces messages pour y lier une action à exécuter.

Exemple :

```
window.addEventListener('keydown', function(event)
{
    if (event.key !== undefined)
    {
        console.log('key: ' + event.key);
    }
    if (event.keyCode !== undefined)
    {
        console.log('keyCode: ' + event.keyCode);
    }
}, true);
```

Ce code affiche dans la console :

- Le mot « key », suivi du nom correspondant à la touche de clavier enfoncée ;
- Le mot « keyCode », suivi du numéro correspondant à la touche de clavier enfoncée.

Fonctions

Une fonction permet de rassembler un ensemble d'instructions. Elle permet d'alléger le code et de le rendre plus lisible car plutôt qu'écrire l'ensemble des instructions autant de fois que nécessaire, on appelle la fonction correspondante.
Une fonction peut retourner une valeur.

Exemple :

Déclaration de la fonction :

```
fonction FaireDuVelo()
{
    pedaler();
    if (routeTourne == vrai)
    {
        TournerGuidon();
    }
}
```

Appel de la fonction :

```
FaireDuVelo();
```

Cette fonction peut, par exemple, s'appeler à chaque fois que le joueur monte sur son vélo dans le jeu.

Syntaxe

Un projet se code à plusieurs. La lisibilité dans l'écriture de la syntaxe est donc une notion importante pour tout programmeur voulant échanger, partager et travailler son code avec les autres.

Il est possible de proposer une organisation du code plus lisible que celle proposée par l'application. Par exemple, au lieu de l'affichage par défaut :

```
toujours (fonction (){
if (personnage.estAuSol ()){
    personnage.avance()
}
})
```

On peut avoir :

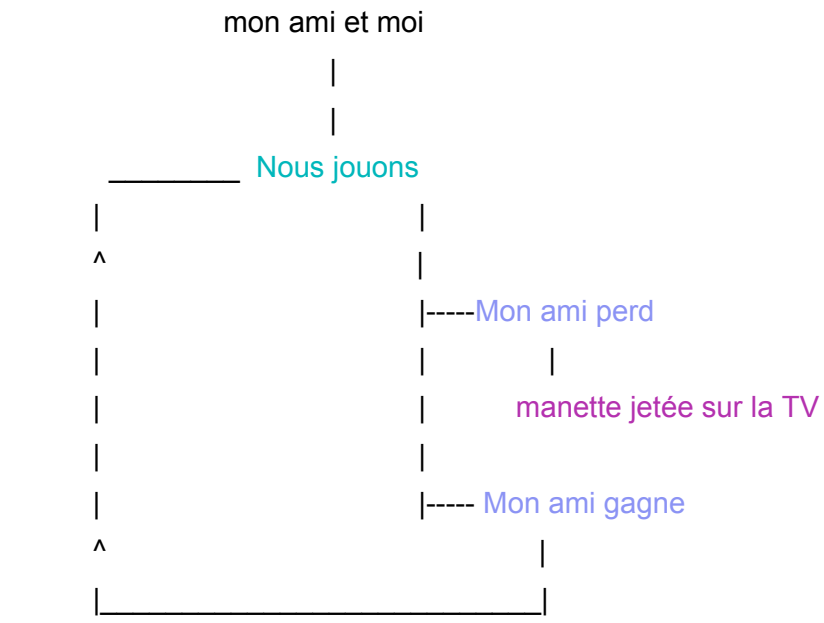
```
toujours (fonction ()
{
    if (personnage.estAuSol ())
    {
        personnage.avance()
```

```
    }
})
```

Langage Déconnecté

Exemple :

Mauvais Perdant



Réécriture en code avancé :

```
While (monAmiGagne)
{
        monAmiGagne = NousRejouonsUnePartie();
}
ManetteJetéeSurLaTV();
```

Ici, on a une boucle qui vérifie la condition Victoire/Défaite de mon ami.
monAmiGagne est une variable qui contient le résultat de la partie sous la forme vrai / faux.
NousRejouonsUnePartie() et ManetteJetéeSurLaTV() sont des fonctions.
Ce sont des comportements créés par l'utilisateur à l'aide d'une suite d'instructions. Ces instructions seront toutes exécutées dans le programme lors de l'appel de la fonction qui les regroupe.

On peut, par exemple, décortiquer la fonction NousRejouonsUnePartie()

```
fonction NousRejouonsUnePartie()
```

```
{
    var monScore = 0;
    var scoreDeMonAmi = 0;

    JouerNouvellePartie();
    monScore = ScoreDeLaPartie('moi');
    scoreDeMonAmi = ScoreDeLaPartie('monAmi');

    si (monScore > scoreDeMonAmi)
    {
        renvoie faux;
    }
    sinon
    {
        renvoie vrai;
    }
}
```

ASCII

L'ASCII est une norme informatique utilisée pour coder un caractère.
Selon le pays, les mêmes caractères peuvent s'écrire avec des symboles différents (exemple : les guillemets). L'ordinateur, qui ne comprend pas le langage humain, a besoin d'une norme pour faire la différence entre chaque caractère.
La norme ASCII définie une valeur pour chaque lettres, caractères et symboles.

C'est très important de créer un standard pour les alphabets. Avant l'ASCII, beaucoup de machines avaient leur propre façon d'encoder les caractères. L'affichage pouvait totalement varier d'une machine à l'autre, rendant compliqué l'échange de documents et d'informations.

Voici un tableau des différents symboles et de leurs valeurs en ASCII :

Code en base				Caractère	Signification
10	8	16	2		
33	041	21	0100001	!	Point d'exclamation
34	042	22	0100010	"	Guillemet
35	043	23	0100011	#	Croisillon9
36	044	24	0100100	$	Dollar
37	045	25	0100101	%	Pourcent
38	046	26	0100110	&	Esperluette
39	047	27	0100111	'	Apostrophe
40	050	28	0101000	(	Parenthèse ouvrante
41	051	29	0101001	)	Parenthèse fermante
42	052	2A	0101010	*	Astérisque
43	053	2B	0101011	+	Plus
44	054	2C	0101100	,	Virgule
45	055	2D	0101101	-	Trait d'union, moins
46	056	2E	0101110	.	Point
47	057	2F	0101111	/	Barre oblique
48	060	30	0110000	0	Chiffre zéro
49	061	31	0110001	1	Chiffre un
50	062	32	0110010	2	Chiffre deux
51	063	33	0110011	3	Chiffre trois
52	064	34	0110100	4	Chiffre quatre
53	065	35	0110101	5	Chiffre cinq
54	066	36	0110110	6	Chiffre six
55	067	37	0110111	7	Chiffre sept

56	070	38	0111000	8	Chiffre huit
57	071	39	0111001	9	Chiffre neuf
58	072	3A	0111010	:	Deux-points
59	073	3B	0111011	;	Point-virgule
60	074	3C	0111100	<	Inférieur
61	075	3D	0111101	=	Égal
62	076	3E	0111110	>	Supérieur
63	077	3F	0111111	?	Point d'interrogation
64	0100	40	1000000	@	Arobase
65	0101	41	1000001	A	Lettre latine capitale A
66	0102	42	1000010	B	Lettre latine capitale B
67	0103	43	1000011	C	Lettre latine capitale C
68	0104	44	1000100	D	Lettre latine capitale D
69	0105	45	1000101	E	Lettre latine capitale E
70	0106	46	1000110	F	Lettre latine capitale F
71	0107	47	1000111	G	Lettre latine capitale G
72	0110	48	1001000	H	Lettre latine capitale H
73	0111	49	1001001	I	Lettre latine capitale I
74	0112	4A	1001010	J	Lettre latine capitale J
75	0113	4B	1001011	K	Lettre latine capitale K
76	0114	4C	1001100	L	Lettre latine capitale L
77	0115	4D	1001101	M	Lettre latine capitale M
78	0116	4E	1001110	N	Lettre latine capitale N
79	0117	4F	1001111	O	Lettre latine capitale O
80	0120	50	1010000	P	Lettre latine capitale P
81	0121	51	1010001	Q	Lettre latine capitale Q

82	0122	52	1010010	R	Lettre latine capitale R
83	0123	53	1010011	S	Lettre latine capitale S
84	0124	54	1010100	T	Lettre latine capitale T
85	0125	55	1010101	U	Lettre latine capitale U
86	0126	56	1010110	V	Lettre latine capitale V
87	0127	57	1010111	W	Lettre latine capitale W
88	0130	58	1011000	X	Lettre latine capitale X
89	0131	59	1011001	Y	Lettre latine capitale Y
90	0132	5A	1011010	Z	Lettre latine capitale Z
91	0133	5B	1011011	[	Crochet ouvrant
92	0134	5C	1011100	\	Barre oblique inversée
93	0135	5D	1011101	]	Crochet fermant
94	0136	5E	1011110	^	Accent circonflexe (avec chasse)
95	0137	5F	1011111	_	Tiret bas
96	0140	60	1100000	`	Accent grave (avec chasse)
97	0141	61	1100001	a	Lettre latine minuscule a
98	0142	62	1100010	b	Lettre latine minuscule b
99	0143	63	1100011	c	Lettre latine minuscule c
100	0144	64	1100100	d	Lettre latine minuscule d
101	0145	65	1100101	e	Lettre latine minuscule e
102	0146	66	1100110	f	Lettre latine minuscule f
103	0147	67	1100111	g	Lettre latine minuscule g
104	0150	68	1101000	h	Lettre latine minuscule h
105	0151	69	1101001	i	Lettre latine minuscule i
106	0152	6A	1101010	j	Lettre latine minuscule j
107	0153	6B	1101011	k	Lettre latine minuscule k

108	0154	6C	1101100	l	Lettre latine minuscule l
109	0155	6D	1101101	m	Lettre latine minuscule m
110	0156	6E	1101110	n	Lettre latine minuscule n
111	0157	6F	1101111	o	Lettre latine minuscule o
112	0160	70	1110000	p	Lettre latine minuscule p
113	0161	71	1110001	q	Lettre latine minuscule q
114	0162	72	1110010	r	Lettre latine minuscule r
115	0163	73	1110011	s	Lettre latine minuscule s
116	0164	74	1110100	t	Lettre latine minuscule t
117	0165	75	1110101	u	Lettre latine minuscule u
118	0166	76	1110110	v	Lettre latine minuscule v
119	0167	77	1110111	w	Lettre latine minuscule w
120	0170	78	1111000	x	Lettre latine minuscule x
121	0171	79	1111001	y	Lettre latine minuscule y
122	0172	7A	1111010	z	Lettre latine minuscule z
123	0173	7B	1111011	{	Accolade ouvrante
124	0174	7C	1111100	|	Barre verticale
125	0175	7D	1111101	}	Accolade fermante