Collège Emile Zola - Toulouse

ACADÉMIE
DE TOULOUSE
*Liberté*
*Égalité*
*Fraternité*

**Collège Émile Zola**

17, rue Jean-Pierre Blanchard

31400 Toulouse

Tél. : 05 61 52 95 97

Fax : 05 61 52 11 08

Mèl : 0311338l@ac-toulouse.fr

Pr François LAVAUX

# Atelier 6ème : Informatique

Année scolaire 2023 - 2024

| Informatique | Principe de l'atelier | **6ème** |
|---|---|---|

L'atelier "Informatique" est un atelier de découverte de l'informatique, en particulier de la programmation.

Nous utiliserons l'explorateur de fichiers, un tableur, Scratch, Gimp et programmerons un peu en HTML. Le but est de faire **un premier pas dans la logique de la programmation informatique** et de découvrir l'algorithmique, quelques principes simples comme les entrées et les sorties, des boucles, des conditions élémentaires...
Le HTML permettra de découvrir un environnement avec compilateur.
Il y aura aussi quelques points historiques pour la culture générale.

Pédagogiquement, Scratch permettra de travailler la logique sans problèmes de compilation. Ils seront introduits avec le HTML mais, avec ce que nous ferons, il y aura alors moins de logique à réfléchir.

Les participants auront **plusieurs projets à mener pendant l'année**, qui prendront chacun plusieurs semaines. Le travail à la maison entre les séances n'est pas interdit mais ne concerne que ceux qui veulent aller plus loin que l'heure d'atelier ne le permet.

En terme de méthode, beaucoup d'interactions sont attendues dans le groupe. Il est parfaitement autorisé d'aider son voisin **mais totalement interdit de toucher son clavier ou sa souris**. Quand une question sera posée par un participant à l'enseignant, il y a de fortes chances qu'elle soit soumise au groupe pour analyse et réponse.

La participation à l'atelier est totalement libre et gratuite mais **la présence est obligatoire après inscription**.

La participation à l'atelier ne donnera pas lieu à des évaluations en classe, ni a des notes, mais il n'est pas exclu que les participants y prennent du plaisir ou y trouvent une certaine forme de satisfaction.

| Informatique | Programmation annuelle prévisionnelle | **6ème** |
|---|---|---|

Chaque semaine, la partie sur ordinateur occupe au moins la moitié de la séance.

| **Semaine** | **Contenu théorique** | **Partie informatique** |
|---|---|---|
| 1 | Présentation de l'atelier | Arborescence + interface du tableur |
| 2 | Octet, Bases, Langage hexadécimal | Tableur : convertisseur octet → hexadéc. |
| 3 | Instructions | Découverte de l'interface Scratch |
| 4 | Variables (+ renforcement instructions) | Scratch : programmes pour calculer |
| 5 | Un peu d'histoire de l'informatique | Tableur : comment le faire calculer ? |
| 6 | *Séance de révision et renforcement* | Révisions : Dossiers, Tableur, Scratch |
| 7 | Programmer une animation | Scratch : créer une petite animation |
| 8 | Programmer une animation (suite) | Animation Scratch (suite) |
| 9 | Comment améliorer son animation ? | Animation Scratch (suite) |
| 10 | *Séance de révision et renforcement* | Présentation commentée au groupe |
| 11 | Le bases de la retouche photo | Découverte de l'interface de Gimp |
| 12 | Créer un lutin à son image | Gimp et intégration dans Scratch |
| 13 | Introduction au HTML | Découverte de l'interface texte |
| 14 | Structure d'un fichier HTML | Programmation d'un site simple |
| 15 | Travailler le style | Programmation d'un site simple (suite) |
| 16 | Rendre le site dynamique | Programmation d'un site simple (suite) |
| 17 | Rendre le site dynamique (suite) | Programmation d'un site simple (suite) |
| 18 | *Séance de révision et renforcement* | Présentation commentée au groupe |
| 19 | Un peu d'histoire de d'Internet | Reprise en main de Scratch |
| 20 | Programmer un jeu : les bases | Scratch : créer un jeu |
| 21 | Les vies et le score | Scratch : créer un jeu (suite) |
| 22 | A propos des clones | Scratch : créer un jeu (suite) |
| 23 | Améliorer son jeu | Scratch : créer un jeu (suite) |
| 24 | *Séance de révision et renforcement* | Présentation commentée au groupe |

## 1. Le langage de l'ordinateur

Un ordinateur est un outil ayant des composants électroniques, réagissant au passage ou non du signal. Leur langage est donc **binaire** : **0** signifie *le courant ne passe pas* et **1** signifie *le courant passe.*

Remarque : Certains composants sont des **portes logiques** qui réagissent en fonction du signal reçu. *Exemples : la porte NON (NOT en anglais) renvoie le signal contraire du signal reçu, la porte ET (AND) a deux entrées et ne renvoie* 1 *que si elle reçoit* 1 *aux deux entrées, etc.*

Chaque fil propage donc un signal traité comme une information appelée **bit**.
Les fils sont regroupés en nappes de 8 fils et l'information ainsi transmise est un **octet**.
*Exemple : une nappe envoie* 00010011 *lorsque le courant passe dans ses fils de rang* 1 ; 2 *et* 5.

Ce nombre est traité en base 2 et il y a donc $2^8 = 256$ **valeurs possibles**.
Mais, cette base produit des nombres à l'écriture très longue. Elle a donc été traduite dans le système hexadécimal (base 16) avec les symboles 1 jusqu'à 9 puis $A, B, C, D, E$ et $F$.
*Exemple : « Onze » s'écrit* 1011 *en base* 2 *et* $B$ *en base* 16

Chaque octet est ensuite scindé en deux. Ainsi, chaque partie correspond à un nombre inférieur à 16, ce qui ne s'écrit qu'avec 1 **seul chiffre** en hexadécimal.
*Exemple : Une série de 4 nappes renvoie les octets* 10010010 00001010 11011101 01010010.
*(en fonction des fils qui reçoivent ou non du courant)*
*Les octets sont scindés en* 2 *et traduits en hexadécimal :*
$1001 \to 9 \quad 0010 \to 2 \quad 0000 \to 0 \quad 1010 \to A \quad 1101 \to D \quad 1101 \to D \quad 0101 \to 5 \quad 0010 \to 2$
*Le message transmis est donc* $920A\ DD52$.

## 2. Parler à l'ordinateur

Une information qui induit une action pour l'ordinateur s'appelle une **instruction**.
Un **programme informatique** est donc une **séquence d'instructions**.

On peut communiquer des instructions binaires (ou hexadécimales) à un ordinateur, cela s'appelle *programmer en langage machine*, mais c'est très difficile et demande beaucoup d'entraînement.
Il existe donc des **langages de programmation** où l'on saisit des instructions en langage naturel puis un **compilateur** se charge de les transformer en langage machine compréhensible par l'ordinateur.

Chaque langage de programmation (comme C++, PHP, Python, etc.) a sa propre **syntaxe** et sa propre logique. Une erreur dans l'une ou l'autre fait que le programme ne "tourne" pas correctement (ou pas du tout).
Pour l'apprentissage de la programmation on peut utiliser un logiciel de *programmation graphique* comme Scratch. On y emboîte des « blocs » ce qui évite les problèmes de syntaxe et permet de se concentrer sur la logique.

Une série de telles instructions pour obtenir une résultat est appelé **algorithme informatique**.
Le mot « algorithme » vient du nom du mathématicien perse Al-Khawarizmi, car il est le père de méthodes de calcul par répétition d'une action jusqu'à obtention du résultat.
L'algorithmique est l'étude de telles séries d'instructions, en dehors d'un langage de programmation.

Vocabulaire :

La personne qui crée le code est le **programmeur**.
La personne qui teste si le programme a des **bugs** est le **testeur**.
La personne qui utilise le programme est l'**utilisateur** .

La première version du programme qui fonctionne (mais minimale) est la version **alpha**.
La version du programme qui fonctionne mais reste à finaliser est la version **bêta**.
Les personnes qui "éprouvent" la version bêta sont appelés **bêta-testeurs**.
La version finale du programme est appelée **version 1.0**.
*Les mises à jours créeront les versions 1.1 ; 1.2 ; etc.*

### 3. Les instructions

On appelle **instruction d'entrée** une instruction qui demande à l'utilisateur de donner une information, par exemple au clavier ou à la souris.
*Exemple : le bloc "Demander... et attendre" en Scratch.*

On appelle **instruction de sortie** une instruction qui demande à l'ordinateur de faire quelque chose de visible pour l'utilisateur, par exemple afficher un objet à l'écran ou déclencher l'impression d'un document.
*Exemple : le bloc "Dire..." en Scratch.*

Il existe d'autres séries d'instructions qui ne font ni l'un, ni l'autre. Elles **calculent** quelque chose (sans afficher le résultat), **répètent** une action, **testent** si une condition est vérifiée, etc.
Ce sont celles qui sont le plus utilisées.
Remarque : Certains programmes n'ont aucune instruction d'entrée. C'est le cas des animations, où l'utilisateur regarde sans interagir.

Si on ne programme aucune instruction de sortie, l'ordinateur fera des actions mais l'utilisateur ne verra rien. C'est le cas pour les cookies informatiques par exemple, qui s'installent et récupèrent des informations "silencieusement". Par contre, il y a des sorties pour le programmeur qui récupère beaucoup d'informations sur l'utilisateur !
Attention : lorsqu'on programme un calcul, si on veut voir le résultat, il ne faut pas oublier d'utiliser une instruction de sortie !

### 4. Variable informatique

Le résultat d'uns instruction comme "*calcule 2+3*" est systématiquement perdue. L'ordinateur n'est pas pensé pour la garder *a priori* ou sans instruction qui lui demande explicitement de le faire.
Pour garder une "information" lors d'un programme, l'ordinateur a des "espaces de stockages" appelés **variables informatiques**.

Chaque variable a un **nom**, afin de pouvoir être appelée et utilisée, et un **type**, qui identifie ce qu'elle peut contenir (un mot, un nombre, etc.) Certains langages comme Scratch ou Python ont un typage dynamique et gèrent eux-même le type des variables lors de leur première utilisation.

Pour mettre une donnée dans une variable, on utilise un **instruction d'affectation**.
*Exemple : le bloc "Mettre... à..." en Scratch.*

L'histoire de l'informatique ne commence pas avec l'apparition des premiers ordinateurs. Avant eux, il y a eu des machines mécaniques, ainsi que purement théoriques et il serait difficile d'en faire la liste exhaustive.

On peut toutefois commencer par la **Pascaline**, première machine à calculer mécanique, inventée par le mathématicien français **Blaise Pascal** vers 1 640. Il en a l'idée en 1 642 alors qu'il n'a encore que 19 ans et mettra 3 ans (et 50 prototypes) pour la finaliser. L'idée de départ était d'aider son père, puis finalement d'en faire le commerce, mais ce fut un échec car sa conception la rendait trop coûteuse. La machine est composée de roues dentées à 10 positions et où le passage de 9 à 0 entraîne le mouvement de la roue à gauche. C'est naturel pour l'addition mais demande une astuce, passer deux fois par le complément à 9, pour la soustraction. Les multiplications et divisions sont faites à l'aide d'additions et soustractions successives.

En 1 834, le mathématicien anglais **Charles Babbage** imagine une « machine analytique », à savoir une machine à calculer programmable. Il passera sa vie à la construire sans parvenir à l'achever. En 1 842, le mathématicien italien Louis-Frédéric Ménabréa écrit un article et il est proposé à **Ada Lovelace**, fille du poète George Byron et épouse du comte de Lovelace, passionnée de mathématiques, d'en faire la traduction. Pour ce travail, elle écrira le premier **algorithme** de programmation de l'Histoire de l'humanité. Le premier informaticien était donc... une informaticienne !

Les premiers ordinateurs, au sens où on l'entend aujourd'hui, datent d'après la seconde guerre mondiale. Mais, ils ont des ancètres très proches dont l'**Havard Mark I**, premier grand calculateur américain, et le **Z3**, calculateur à relais électromécaniques allemand. A noter qu'ils étaient d'une taille colossale, occupant une pièce entière.
En parallèle, le mathématicien **Alan Turing** imagine un ordinateur théorique, ainsi que la théorie mathématique afférente, appelé aujourd'hui « machine de Turing ».

L'Harvard Mark I marque aussi le véritable début de l'aventure de la société IBM (International Business Machines) dont le principal but a été la commercialisation des ordinateurs. Il fallait donc en réduire la taille... et le prix. Par exemple, le « modèle 650 », lancé en 1 954 coûtait 500 000 dollards ! C'est d'ailleurs avec l'arrivée en France de ce dernier qu'à été inventé le mot **ordinateur**.

Après de nombreuses années de recherche et développement, IBM aboutira en 1 973 au premier Personnel Computer, le fameux PC.

*Augusta Ada King, comtesse de Lovelace (*1 815 − 1 852*)*

## 1. L'animation : un art de prestidigitation

En animation, il ne faut pas réfléchir à ce que l'on veut que l'objet fasse mais à ce qu'on va lui faire faire pour donner l'impression qu'il le fait.
*Exemple : en Scratch, pour donner l'impression qu'un lutin bouge ses pieds (ou ses ailes), on programme une répétition de "changement de costume", avec un délai entre chacun.*

Le principe clef est donc « **donner l'impression que...** » :
- Comment donner l'impression que l'objet bouge ?
- Comment donner l'impression que les personnages dialoguent ?
- Comment donner l'impression que beaucoup de temps s'écoule ?
- etc.

Le cinéma d'animation et la bande-dessinée ont posé des codes que l'on peut facilement utiliser.
*Exemple : on utilise une "bulle" (dont le vrai nom est **phylactère**) pour indiquer qu'un personnage parle dans un dessin.*
*Pour information, en France, le premier phylactère est apparu sur une affiche de* 1 792*. Mais des "rubans" ont une fonction similaire sur des tableaux et tapisseries du moyen âge.*

Remarque : Ces astuces ne fonctionnent que si l'utilisateur connaît ces codes. Par exemple, une animation qui utilise des codes du manga japonais sera incompréhensible pour une personne qui ne connaît pas du tout cette culture.

## 2. Déplacement d'un lutin en Scratch

En Scratch, les objets qui reçoivent un script sont appelés **lutins**.
Pour faire changer un lutin de place, il suffit de changer ses coordonnées. L'utilisation du bloc "Aller à ..." permet au programme de le faire (sans intervention humaine).

Attention : Lorsqu'on programme, on peut "attraper" les lutins avec la souris sur la scène et les déplacer. Mais, **ce n'est pas possible en "grand écran"**, en particulier pour l'utilisateur.
Il faut donc prévoir **tous** les déplacements dans le script du lutin, ainsi que **sa position de départ**.

Idée : quand on programme, lorsqu'on bouge le lutin sur la scène avec la souris, **le bloc "aller à..." se met automatiquement aux bonnes coordonnées**.

Changer le lutin de place et le voir se déplacer sont deux choses différentes :
- Enchaîner deux blocs "Aller à..." donne l'impression que le lutin se "téléporte".
- Enchaîner un bloc "Aller à..." puis "Glisser en... secondes à..." donne un meilleur rendu.

Idée importante : L'utilisation d'un bloc "Glisser à..." n'est pas toujours une bonne option car **c'est une action "insécable"** : le lutin fera cette action jusqu'au bout quoi qu'il arrive. Il est parfois préférable de programmer une **répétition de petites actions** pour obtenir le même rendu visuel mais garder la possibilité d'intervenir pendant le déplacement.

## 3. Améliorer le visuel du déplacement en Scratch

On peut programmer un lutin qui va et vient seul sur l'écran en mettant dans un boucle "Répéter..." les instructions "Avancer de... pas" et "Rebondir si le bord est atteint".

Pour éviter qu'un lutin ne se mette la tête en bas, il suffit de programmer la façon dont il se tourne/retourne.
Cela se fait dans son menu **direction** (sous la scène) ou avec la brique "Fixer le sens de rotation...", à mettre au début du script. Il y a 3 options : le lutin tourne sur lui-même **à 360°**, le lutin se retourne **gauche/droite** ou le lutin **ne tourne pas** et reste fixe.
Il est important d'adapter son choix selon que le lutin est vu de dessus ou de profil.

Ajouter une animation sur les membres du lutin améliore encore le rendu. Cela se fait en programmant un code **en parallèle** où l'on met dans un bloc "Répéter..." les instructions "Costume suivant" et "Attendre 0.1 secondes".
Attention : il faut que le lutin ait des costumes simulant un mouvement de déplacement (il bouge les pattes, bat des ailes, etc.). S'il a d'autres costumes, on peut soit choisir les bons en remplaçant "Costume suivant" par "Basculer sur le costume...", soit supprimer les costumes inutiles dans l'onglet "Costumes".

S'il n'a pas d'autre costume, on peut en fabriquer un en dupliquant le costume existant dans l'onglet "Costumes" puis en modifiant la copie grâce aux outils de dessin du Scratch. Cela demande un peu de pratique pour avoir un rendu acceptable.

Pour programmer un lutin qui ne se déplace pas de manière régulière, comme une feuille qui tombe, on peut réfléchir à comment utiliser "nombre aléatoire compris entre..." dans les instructions de déplacement. Cela permet de le faire aller tantôt un peu vers la gauche, tantôt un peu vers la droite, tout en le faisant aller vers le bas de façon régulière. Des instructions en parallèles s'avèrent parfois utiles.

### 4. Mouvement relatif en Scratch

Pour donner l'impression que le lutin bouge, on peut aussi le laisser fixe au milieu de la scène et faire bouger les autres éléments du décor.

L'exemple classique pour cela est la voiture sur la route :
- On découpe les roues du dessin de la voiture et on les transforme en deux lutins indépendants qui tournent sur eux-même.
- On place **au second plan** des objets comme des arbres qui traversent la scène de droite à gauche. Rendre leur apparition aléatoire améliore le rendu.

Pour des effets d'accélération, il suffit d'augmenter conjointement les vitesses de rotation des roues et de déplacement des éléments du décor. Pour cela, il est opportun de créer une **variable qui définit ces vitesses**. On met alors le nom de la variable dans la brique de déplacement. Par suite, cela permet de les modifier avec la brique "Ajouter... à la variable...".
Remarque : pour ralentir, il suffit d'ajouter une valeur négative.

### 5. Lutins ayant l'air de dialoguer en Scratch

Comme expliqué plus haut, il suffit d'utiliser des codes de la bande-dessinée, ici les phylactères.
Mais, le temps associé à ceux-ci et la gestion des délais entre chacun est importante.

Pour une "bulle" courte, la laisser 1 seconde convient. Pour une bulle longue, il faut, à la fois, laisser le temps à l'utilisateur de lire et donner l'impression que le lutin a mis du temps à la dire. Donc, 2 secondes est un temps adapté.
Pour plus de 2 secondes, il vaut mieux enchaîner plusieurs "bulles". Cela donne du dynamisme à

l'animation. Rien n'est pire que de garder fixe une longue bulle pendant plusieurs secondes.

Des personnages qui dialoguent normalement attendent 1 seconde avant de répondre. Sinon, cela donne l'impression qu'ils se sont coupé la parole.
Il faut aussi penser que les yeux de l'utilisateur cherchent les changements dans l'image et que son cerveau a un temps de réaction de 1 seconde.

Faire changer le lutin de costume pour illustrer des réactions de sa part améliore grandement le rendu (mais il faut qu'il ait de tels costumes). On peut aussi programmer de petits déplacements.

### 6. Lutin ayant le visage du programmeur

On peut sans trop de difficultés créer un lutin qui porte son propre visage. Pour cela il suffit de prendre quelques photos avec différentes expressions (bouche fermée, bouche ouverte,etc. ) puis, avec un logiciel de dessin adapté, découper les visages et les coller sur le corps d'un lutin existant.

Pour simplifier le travail de retouche, il est important de prendre une photo sur un fond uni d'une couleur bien différente de celle du visage. Cela permettra de supprimer la zone autour du visage par sélection de la couleur, en une seule fois.
L'outil gomme, associé à un zoom à 200% (ou plus) permet les dernières retouches si nécessaire.

Attention : il faut penser à enregistrer une image sans fond, au format ".png" par exemple sinon un rectangle blanc apparaîtra autour du lutin ainsi créé.

Des logiciels libres comme Photofiltre ou Gimp sont tout a fait adaptés pour cela.

## 1. C'est quoi le HTML ?

**HyperText Markup Langage** signifie « langage hypertexte avec des balises »

On parle d'hypertexte en présence de commandes qui permettent de passer d'une autre information à une autre grâce à une adresse informatique. Un exemple classique est un mot souligné en bleu sur lequel on peut cliquer pour changer de page Internet.

Pour comprendre la notion de **balise**, on peut commencer par parler de traitement de texte.
Aujourd'hui, lorsqu'on utilise un tel logiciel et à moins de changer les options d'affichage, on ne voit que le texte tel qu'il sera affiché ou imprimé.
Mais pour ce résultat, le traitement de texte utilise en fait beaucoup de commandes qui n'apparaissent pas (comme la tabulation, le retour-ligne, l'adresse des images que l'on veut voir avec leurs dimensions et leur position dans la page, etc.)

Un traitement de texte comme Latex demande de saisir tous ces éléments dans le fichier, ce qui les rend visible à l'utilisateur.

Il en va de même pour le HTML : le document doit être rédigé avec ses différentes commandes.
Elles sont sous forme de balise : <html> ; </html> ; <p> ; < /p> ; <img... / > ; etc.
La différence fondamentale c'est que le HTML sert à **l'affichage de pages sur Internet**.

Idée : Le premier fichier d'un site internet s'appelle toujours **index.html**

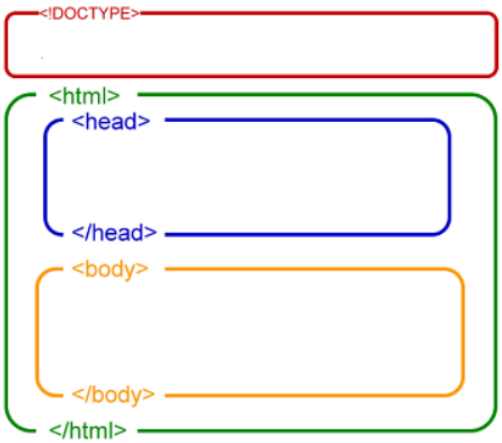
## 2. Structure d'un fichier HTML

Dans un fichier HTML, certaines balises vont par paires et d'autres sont dites « orphelines ».

Celles qui vont par paires utilisent le même mot-clef et définissent une "zone". La première est dite **ouvrante** et la seconde **fermante**.
*Exemple :* <p> ; < /p> *définit un paragraphe dans la page.*

Les balises orphelines servent à une action ponctuelle comme insérer une image, sauter un ligne, etc.
*Exemple :* <img src="test.jpg"/ > *insére l'image test.jpg (src signifie source) à l'endroit de cette balise.*
Remarque : Le HTML est à sa version 5 et il existe aussi le XHTML, une norme plus exigeante. Pour éviter les problèmes d'interprétation par le navigateur et rendre plus lisibles les codes, la norme est d'écrire les orphelines au format < ... / >

Idée : La structure générale d'un fichier HTML est toujours la même.
- une première balise indique que le fichier est en HTML : <!DOCTYPE html>
- une paire de balise définit le début et la fin du document : <html> < /html>
- Dedans, une première paire de balise définit la tête du document : <head> < /head>
- Une seconde paire de balise définit le corps du document : <body> < /body>




Dans la tête (head) du document, on va mettre des balises qui donnent des informations essentielles pour la page Internet mais qui n'apparaissent pas à l'écran : quel est l'alphabet ou le type de codage utilisé, s'il y a un style particulier à appliquer à la page, etc.
On dit que la tête contient des **métadonnées**.

Dans le corps (body), on va mettre tout ce qui doit être affiché à l'écran et le coder afin que cela corresponde à ce que l'on imagine.
On va donc créer différentes zones, qui s'enchaînent ou qui sont à côté les unes des autres. Elles contiendront du texte ou des images, seront centrées ou non, etc.

Il faut bien penser que les pages apparaitront différemment si elles sont affichées sur un écran de 24 pouces ou un smartphone ! Programmer la place des objets par rapport aux bords de l'écran et les uns par rapport aux autres est donc fondamental.
Un codage approximatif ou malvenu (comme mettre des espaces au lieu de tabuler par exemple) donne presque toujours des affichages horribles.

### 3. Les balises du head.

Deux choses sont essentielles.

La première est une balise qui définit l'**encodage de la page**. Par exemple, si le site est écrit en arabe, il ne faut pas qu'il s'affiche avec l'alphabet latin sinon ce sera incompréhensible.

Le plus répandu est l'UTF-8, qui a succédé à l'ASCII. C'est un ensemble de caractères, codés sur 8 bits, qui contient tout l'alphabet latin en majuscule et en minuscule, les lettres accentués et des caractères classiques comme $<$ , $=$ ( ) :.

La balise est : **<meta charset="utf-8">**

Les suivantes sont une paire, qui indique le **titre de la page**. Ce n'est pas une chose qui va s'écrire en haut de l'écran par exemple. C'est le nom que l'auteur donne à cette page.
C'est aussi ce qui apparaîtra comme nom si un moteur de recherche trouve la page.

Les balises sont : **<title>** ***le nom de la page*** **</title>**

Il peut y avoir beaucoup d'autres balises permettant de définir un style pour la page (couleur, fond, fonte d'écriture, etc.), des mots clefs pour un moteur de recherche, une description, le nom de l'auteur...

**4. Les balises de base du body.**

Les balises possibles sont tellement nombreuses qu'il faudrait un livre entier pour toutes les citer. Il s'agit ici d'en voir les toutes premières qui permettent d'écrire un peu de texte et d'y placer des images.

La paire **<p> ; < /p>** permet de délimiter un **paragraphe**. On peut, en particulier y écrire du texte.

Les paires avec un « h » signifiant « heading » (c'est à dire « titre ») ont 6 niveaux.
**<h1> < /h1>** est à utiliser pour les **titres les plus importants** (de niveau 1)
**<h2> < /h2>** est à utiliser pour les **sous-titres** (de niveau 2)
... et ainsi de suite, jusqu'au niveau 6, si besoin.
Remarque : dans le style par défaut, les titres de niveau 1 sont écrits plus gros que ceux de niveau 2 et ainsi de suite. Il y a aussi un enchaînement avec écrits en gras, puis en italique.

Pour **insérer une image**, on utilise la balise orpheline **<img ... / >**.
Cette balise a besoin d'au moins un **attribut** : la source(scr), c'est à dire l'adresse informatique où trouver l'image.

idée importante : Un attribut est une information ajoutée à une balise. Beaucoup sont optionnels. Ils s'écrivent tous de la même façon : motclef="*valeur*"

**5. Les attributs de <img ... / >**

C'est la première balise à ne pas utiliser dans sa forme "brute". Il y a plusieurs raisons à cela.

La première concerne l'**accessibilité des sites**. Une personne malvoyante ne peut pas voir une image (bien entendu). Elle utilise un logiciel qui lit ce qu'il y a sur la page. L'attribut **alt=** permet de définir un **texte alternatif à l'image**.
Ce texte apparaîtra aussi à la place de l'image si le navigateur n'arrive pas à la charger.

La deuxième concerne les **dimensions de l'image**. Sans cette information, le navigateur affichera l'image dans les dimensions qu'il trouvera dans ses proriétés. Si c'est un poster A3, ce sera la taille de l'image à l'écran !
Définir la **longueur (length)** et la **largeur (width)** permet d'adapter l'image à la page Internet. On peut mettre la taille en pixels(px) mais aussi en pourcentage(%). Il y a d'autres options pour adapter la taille de l'image à tous les écrans (24 pouces ou smartphone).

Le "lieu de stockage" de la source de l'image est important. Si l'attribut est scr="image.bmp", le navigateur cherchera l'image appelée "image" avec l'extension ".bmp" **dans le même répertoire que la page en cours**.
Si on a peu d'images, il est donc opportun de les ranger au même endroit que ses pages ".html". Mais un site a souvent beaucoup d'images et il est préférable de créer un sous-répertoire "/img" pour les ranger. Du coup, l'attribut s'écrira scr="img/image.bmp".

Remarque : C'est une adresse dite "relative", ce qui s'oppose à une adresse "absolue". Dans le second cas, on donne le chemin complet et exact de l'objet. Dans le premier, la navigateur cherche "en-dessous" du dossier en cours. Avec des pages Internet, on travaille beaucoup en adresses relatives car le navigateur ne va pas chercher les fichiers sur l'ordinateur du créateur du site !

Au final, la balise prendre une forme :
**<img scr="img/image.png alt="Image d'un truc" width="100px" lenght="200px"/ >**

### 6. Améliorer la présentation

Le texte, par défaut, va jusqu'au bout de la ligne (de l'écran) puis passe en dessous.

Idée : Il ne s'affiche donc pas pareil selon la taille de l'écran et le zoom appliqué à celui-ci. C'est une des différences fondamentales entre le traitement de texte et le navigateur.

On peut "forcer" le **retour à la ligne** avec la balise <**br**/ >.

La balise <img ... / > n'est pas la seule à accepter les attributs optionnels. C'est le cas pour toutes les balises de base du body par exemple.
On peut définir pour chaque paragraphe une fonte d'écriture, une couleur, une disposition du texte, etc. Pour ce faire il suffit d'ajouter un des attributs suivants dans la balise ouvrante de la paire.

style="color :..." → change la couleur du texte
style="font−size :...pt" → change la taille du texte
style="font−family :..." → change la fonte d'écriture
title="..." → affiche une bulle au passage de la souris
etc.

Idée importante : Cette méthode ne s'utilise que pour appliquer des changements **localement**. Si plusieurs paragraphe doivent avoir le même style, on n'écrit pas des attributs identiques dans les balises <p> de chacun. La méthode est alors globale et se ferait avec les **feuilles de style**(CSS).

Toutefois, **si c'est le même style pour toute la page, on peut le déclarer dans le head**.
Pour cela, on utilise la paire <**style**> <**style**/ > à l'intérieur de laquelle on va déclarer des attributs pour certaines balises.
Par exemple,

```
<head>
   <style> p {color : red} <style/ >
<head/ >
```

Ce code donne l'attribut *"le texte est écrit en rouge"* à toutes les balises <p>. Donc, dans toutes les zones déclarées comme paragraphe, le texte sera rouge.

### 7. Rendre le site dynamique

Nous n'avons pas encore exploité la partie hypertexte du HTML. Or c'est une de ses principales caractéristiques puisque c'est une partie de son nom !
Pour cela, nous allons voir deux des utilités de la paire <**a...**> <**a**/ >.
Remarque : "a" vient de « anchor » qui veut dire « ancre » en anglais.

La page Internet n'a pas de fin "vers le bas". Il y a, à droite, une barre de défilement, mais on peut aussi **envoyer le navigateur directement à un endroit de la page**.
Pour cela, il faut nommer un des "endroits" de la page. On utilise : <**a id="*nom*"**> <**a**/ >
On peut mettre des éléments entre les deux balises, mais généralement cela sert à nommer un endroit précis, comme un pixel, donc on le laisse vide.

A un autre endroit, on va de nouveau utiliser <a...> mais avec la formulation :

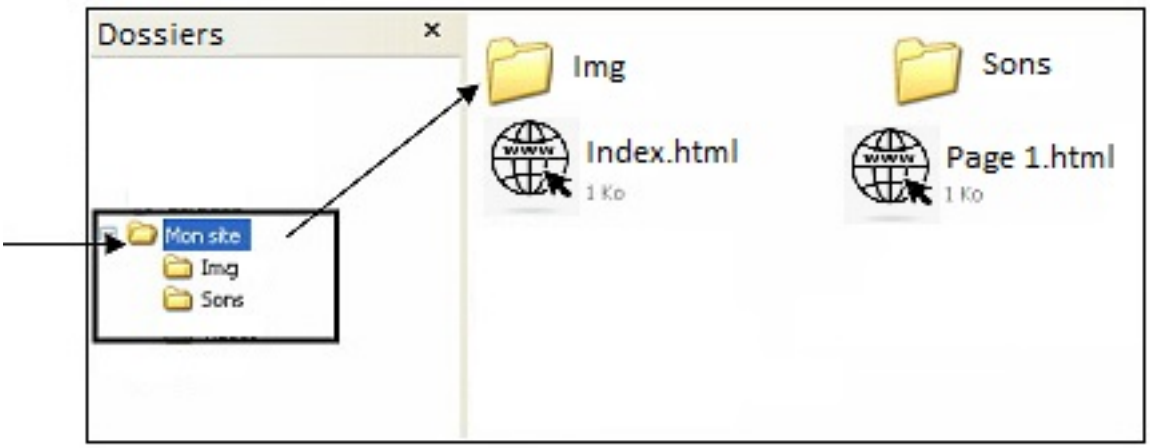
<**a href="#*nom*"**> **Cliquer ici pour aller à *nom*** <**a**/ >

Le texte saisi entre ces deux balises apparaîtra souligné en bleu et cliquer dessus amènera au niveau du <a id="*nom*">.
Remarque : On peut mettre une image ou autre chose à la place du texte, ou les deux.

Un lien peut aussi **renvoyer à une autre page Internet**. Pour cela, il suffit de changer la valeur de l'attribut href=.

*Exemple :* **<a href="#autrepage.html"> Cliquer ici pour aller à l'autre page <a/ >**
*renvoie vers la page autrepage.html*

Il faut de nouveau faire attention aux adresses. Ici, elle est relative et le navigateur va chercher un fichier qui s'appelle "autrepage" avec l'extension ".html" dans le répertoire en cours. Normalement, on met toutes les pages d'un site dans le même répertoire (sauf s'il est vraiment gigantesque).



L'histoire d'Internet commence officiellement à la fin des années 1950 mais des auteurs de science-fiction avait eu des idées du même genre bien avant, sans peut-être se douter que cela serait un jour possible.

En 1958, La société Bell crée le premier modem, appareil permettant de transmettre des données (en binaire) par l'utilisation d'une ligne téléphonique. C'est l'évènement premier car, avant cela, il était impossible pour des ordinateurs distants de "communiquer" entre-eux.

Un homme visionnaire va proposer une idée en janvier 1960 : J.C.R Licklider, un informaticien américain (qui aura bien d'autres idées comme la souris informatique par exemple). Il publie « Man-Computer Symbiosis », littéralement la symbiose Homme-Ordinateur, où il imagine « un réseau [d'ordinateurs] connectés les uns aux autres par télécommunication » dont les fonctions principales seraient d'être une bibliothèque géante, de permettre le stockage et la récolte d'informations.
Il parle même de « fonctions symbiotiques », ce qui n'est pas sans nous évoquer le lien avec nos smartphones.

A partir d'octobre 1962, il travaille pour le Département de la Défense des Etats-Unis à l'agence pour la recherche avancée sur le traitement de l'information. Après plusieurs années de développement, il lance l'ARPANET en 1968, premier réseau de transfert de "paquets de données" entre ordinateurs.

En 1971, 23 ordinateurs sont reliés via ARPANET et Ray Tomlison, un ingénieur américain, invente le concept de courriel. Il est d'ailleurs l'auteur du premier d'entre-eux.

Les choses vont alors prendre une dimension mondiale. On crée, en 1972, l'International Networking Group, qui a pour objectif de créer des protocoles identiques pour tous les pays du monde en matière de communication entre ordinateurs. L'équipe française, sous la direction du chercheur de l'IRIA Louis Pouzin, est très impliquée. La France a, à ce moment, créé son propre réseau qui porte le nom de Cyclades. Il est à la fois concurrent et partenaire d'ARPANET.
Cette collaboration donnera naissance au protocole TCP/IP en 1973, encore utilisé de nos jours.

Les premiers forums sont inventés pas des étudiant américains en 1979. Le nom "Internet" est adopté en 1983. Dans les dix années suivantes, le nombre d'ordinateurs connectés va passer d'une centaine à 100 000.

Le prochain tournant essentiel a lieu en 1990 où ARPANET disparaît et le réseau devient alors civil : c'est la naissance du World Wide Web (le fameux www. au début de toutes les adresses internet). Dés lors, le nombre d'ordinateurs connectés n'aura de cesse d'augmenter passant brutalement de 36 millions à 370 millions entre 1996 et l'an 2000. Les journaux parlent alors de « l'explosion de la bulle Internet ».
En 2023 on dénombre près de 5 milliards d'ordinateurs connectés.

Aujourd'hui, l'invention des smartphones a révolutionné notre rapport avec Internet, qui est vraiment devenu « symbiotique » . Il n'est plus besoin de s'installer devant un appareil fixe, relié à une ligne téléphonique, pour utiliser Internet. Cela change l'accès, mais aussi le rapport, au savoir... et bien d'autres choses encore !
Par exemple, avec le rapport au GPS, ou les achats de billets presque uniquement en ligne, il est déjà impossible pour certaines personnes de voyager sans leur téléphone.
Et que dire du rapport aux réseaus sociaux ? On peut légitimement se demander quelle sera la prochaine étape, voire s'en inquiéter. C'est pourquoi il semble primordial aujourd'hui d'avoir des connaissances solides en informatique pour ne pas se trouver dépassé... ou dépendant !

### 1. A propos de *répéter indéfiniment.*

La différence fondamentale entre une animation et un jeu est qu'il y a des **interactions** avec l'utilisateur. La première va permettre au joueur de déplacer son lutin.
Il va y avoir deux phases pour cela, dont une première "de recherche". On pourrait donc s'autoriser, dans un premier temps, à utiliser la "boite noire" : *Répéter indéfiniment.*

Mais, cette brique **n'a pas de sens en informatique**. Si un programme se répète à l'infini c'est qu'il y a un bug appelé "boucle infinie".

Cette démarche serait uniquement **à but pédagogique et pour une première fois**. En effet, elle permettrait d'isoler les problèmes et d'avoir tout de suite un premier résultat à tester.
Pourtant, il est fortement recommandé de l'éviter car... elle rajoute beaucoup de travail ! De plus, il faut apprendre à penser les choses dans l'ordre.

Ce n'est pas dans un second temps que le programmeur réfléchira à une condition d'arrêt du jeu (ce qui permettra d'utiliser la boucle : *Répéter jusqu'à...*). Il faut apprendre à **penser son jeu avant de le programmer**, en particulier, anticiper les conditions d'arrêt avant de programmer les déplacements (qui, en fait, en dépendent).

Le bloc *Répéter indéfiniment* peut servir pour programmer avec de tout jeunes élèves, à l'école primaire, qui ne sont pas en mesure de comprendre la notion de condition d'arrêt.

### 2. Les vies ou le score.

Ils forment, pour un jeu simple, la condition d'arrêt. Le jeu se termine quand on n'a plus de vie ou lorsqu'on a atteint un certain score. On peut être "gourmand" et prévoir les deux mais, pour un premier jeu, il vaut mieux choisir entre l'un ou l'autre.

Le choix ne change pas la programmation :
- Créer une **variable** qui s'appelle "Vies" ou "Score"
- La rendre visible à l'écran
- Ne pas oublier de la mettre au nombre voulu pour les vies (généralement 3) et à 0 pour le score **au début du programme**.
- Mettre chaque script dans une boucle « Répéter jusqu'à Vies = 0 »
  ou « Répéter jusqu'à Score = *Nombre voulu* »

Plus tard dans le programme, on mettra au moment opportun le bloc "Ajouter $-1$ à Vies" (ou $+1$ à score)

### 3. Déplacer le lutin du joueur.

Le principe de base est de permettre le déplacement au clavier. La code est simple :
**Si** la flèche du haut est pressée **Alors** s'orienter vers le haut et avancer de ... pas.
**Si** la flèche du bas est pressée **Alors** s'orienter vers le bas et avancer de ... pas.
etc.

Le nombre de pas définit la vitesse de déplacement. Il est opportun de **créer une variable** pour

celle-ci et d'utiliser son nom dans le code ci-dessus. Cela permet d'éviter de relire tout le texte pour tout corriger si on veut adapter la vitesse de déplacement. Cela ouvre aussi la **possibilité de modifier la vitesse en cours de jeu**, en agissant sur la variable (malus qui ralentit, bonus qui accélère, vitesse qui change selon l'avancée du jeu, etc.)

Pour que le lutin ne sorte pas de l'écran, il suffit de rajouter "Rebondir si le bord est atteint". Et, comme pour une animation, il faut penser aux options de rotation du lutin pour qu'il ne se retrouve pas la tête en bas.

### 4. Présence d'un autre lutin que celui du joueur.

Le plus simple comme premier jeu est un "jeu de poursuite". Un second lutin se déplace automatiquement et le joueur doit soit l'attraper (c'est lui le poursuivant) soit l'éviter. Dans le premier cas, le score augmente, dans le second cas il perd un vie.

Les déplacements de ce lutin sont à programmer comme pour une animation.

Il faut, par contre, prévoir les interactions entre les 2 lutins. Pour cela, le code est simple :

**Si** touche *nom de l'autre lutin* **Alors** ...

C'est ici qu'il faut augmenter le score ou baisser la vie.

Idée importante : le programme ne peut pas gérer que **le lutin 1 teste qu'il touche le lutin 2 pendant que le lutin 2 teste qu'il touche le lutin 1**.
La solution est qu'un seul des deux lutins fasse le test puis qu'il envoie **un message** à l'autre lutin si le test est positif.

Il est opportun que le poursuivant **fasse une pause après avoir touché** l'autre lutin, sinon l'ordinateur comptera plusieurs "touches" et le score va fortement augmenter ou (les vies diminuer).

Il est également primordial que les mouvements du lutin qui n'est pas le joueur ne soient ni trop aléatoires (le jeu devient injouable) ni toujours les mêmes (le jeu devient ennuyeux). Plusieurs stratégies de programmation sont possibles. La plus simple est de donner un angle aléatoire au début du jeu, qui fait que les rebonds seront différents à chaque partie.

### 5. A propos des clones.

Pour un jeu d'un niveau de programmation plus avancé, le nombre de poursuivants va évoluer au cours de la partie ou des objets vont par exemple tomber depuis le haut de la scène (plutôt que des poursuivants). Il peut y avoir également des bonus et des malus qui apparaissent.

Pour cela, il n'est pas question de multiplier le nombre de lutins, ayant tous le même script. On va créer un lutin qui **génère des clones de lui-même**.
Le principe est simple. Grâce au bloc "Créer un clone de moi-même", le lutin crée un clone là où il se trouve.
Le clone suivra le code qui se trouve sous le bloc "Quand je commence comme un clone". Ce code ne s'applique pas au lutin lui-même mais seulement à ses clones.

Il est parfois opportun de "cacher" le lutin mais de "monter" ses clones.