Un tutoriel dans le langage Toy...

Le problème: nous voulons changer la case de chaque lettre d'une phrase pour pouvoir rendre la phrase plus "criante".

Décortiquer le problème: nous voulons prendre chaque lettre une à une, et déterminer si elle est en minuscule. Si elle est en minuscule, nous la changeons vers sa version en majuscule.

Pour résoudre notre problème, nous utiliserons le langage Toy, puisque c'est le plusse meilleur langage de programmation de tous les temps (exagération à reconsidérer)...

Premièrement, nous savons que notre phrase devra se retrouver dans une variable. Nous pourrons alors lire chaque lettre une à une, tester pour connaître la case, et changer la lettre le cas échéant.

En Toy, on crée une variable ainsi:

text [phrase] "ceci est loi, respectez moi loi, loi, ici la loi!"

En lisant ce que contient notre variable phrase, nous comprenons mieux pourquoi c'est si important de résoudre notre problème. Car, tous et toutes le savent, une phrase criante, est une phrase qui se fait respecter.

Alors c'est ça, nous avons notre phrase, et nous voulons la changer. Comment faire ça dans le plusse meilleur langage au monde? Avec une boucle! En Toy, les boucles se nomment **loop**, parce que le Toy a le plusse meilleur vocabulaire des langages de programmation, parce que.

loop [

/]

Les choses bizarres qui viennent avec la boucle, c'est normal.

Maintenant, il faut lire un à un, chaque lettre de la phrase. Misère. Vous vous dites peut-être, est-ce qu'un ordinateur pourrait le faire à notre place? Oui, c'est ce que nous tentons d'accomplir. Mais pour ça, il faut être plus malin que l'ordinateur. Et c'est pourquoi, nous utilisons le Toy, le plusse meilleur langage.

Le Toy met à notre disposition, une multitude de fonctions. Alors, il y a la fonction **=**, pour copier une valeur dans une autre valeur (intuitif, non?). La fonction **count**, pour compter le nombre de lettre dans une phrase. La fonction **find**, pour trouver un caractère dans une phrase. La fonction **+**, pour ajouter quelque chose à quelque chose d'autre...Intuitif? Mais oui, c'est ça, le Toy!

Donc, nous avons plein de fonctions, dans le Toy. Mais ça nous prends une stratégie, si nous voulons faire émerger une piste de solutions dans notre problématique actuelle.

La stratégie: On lit chaque lettre de la phrase, et si la lettre est trop petite, on la met plus grosse. Paf!

La démarche de résolution de problème: Nous devons diviser le problème en sous-problèmes, et ce, jusqu'à ce que chaque sous problème soit suffisamment aisé à résoudre avec le plusse meilleur langage de tous.

Alors, la loop:

loop [

/]

et la variable qui se nomme phrase:

text [phrase] "ceci est loi, respectez moi loi, loi, ici la loi!"

Finalement, le problème est peut-être plus facile à résoudre de tête qu'avec un ordinateur. Mais, avec un ordinateur, c'est plusse meilleur! On est à l'an 2000, faut être dans le vent! Alors...

La soupe: **find** permet de trouver l'emplacement dans une phrase, d'une lettre précise. **count** permet de compter le nombre de lettre dans notre variable. Ah oui, j'oubliais. La fonction **=** est très intuitive. Voyez ceci:

=[phrase] 9, phrase, 12

Explication: On ne peut pas copier un nombre vers une phrase. Mais dans les paramètres, la phrase est circonscrite par deux nombres. Le premier nombre fait commencer le paramètre 'phrase' au nombrième lettre, et le second nombre, à partir du premier nombre, copie le nombre de lettres 12-9. Est laissé comme exercice à l'élève de trouver ce que cette fonction a copié au juste.

SÉRENDIPITÉ!

La fonction **count** compte, non? Alors, comptons! En fait, on va faire ça:

```
integer [nombre_de_lettres]
count [nombre_de_lettres] phrase

loop [nombre_de_lettres] [
     ???
/]
```

Bon, là, le problème est **???**. Étant le plusse meilleur langage de tous, le Toy, ne connaît pas l'incertitude. Mais nous oui, donc, un ordinateur est mieux, à condition qu'il soit bien programmé. Et avec le Toy, c'est du gâteau.

SÉRENDIPITÉ!

Nous allons lire une à une les lettres de la phrase! Ça va nous prendre deux variables **integer**, et une variable **text**...

```
integer [bonjour]
integer [bonsoir]
text [qui_suis_je]
```

Stratégie: **bonjour** commencera à la première position de la phrase. Donc,

=[bonjour] 0

Et **bonsoir** sera toujours une unité de plus que **bonjour**...

```
=[bonsoir]bonjour
+[bonsoir]1
```

Et on met ça dans la boucle:

```
loop [nombre_de_lettres] [
      =[bonsoir]bonjour
      +[bonsoir]1
/]
```

Et la boucle sera exécuté un nombre **nombre_de_lettres** de fois. Si vous avez l'esprit vif, vous remarquerez que **bonsoir** sera toujours égale à 1. Et vous n'avez pas tort.

SÉRENDIPITÉ!...Et INTUITION!...

On n'a qu'à copier bonsoir dans bonjour! Comme ceci:

```
loop [nombre_de_lettres] [
      =[bonsoir]bonjour
      +[bonsoir]1
      ???
      =[bonjour]bonsoir
/]
```

Encore les ??? Mais là c'est facile, on veut utiliser les variables bonjour et bonsoir pour copier la lettre de phrase vers la variable qui_sui_je. Comme ça:

```
=[qui_suis_je] bonjour, phrase, bonsoir
```

C'est simple. Maintenant faut chercher...

ATTENTION! Il ne faut pas créer de nouvelles variables dans une boucle. C'est interdit, et Toy sera confus si vous le faites.

Vous vous demandez pourquoi cet avertissement. C'est parce que nous devons créer de nouvelles variables...

Une variable pour chercher
Une variable pour trouver
et Une variable pour les contenir toutes (les lettres)

Simplement...

```
integer [chercher]
integer [trouver]
text [contenir]
```

Et on met ça avant la boucle.

Stratégie: Avec la variable qui_suis_je, on peut utiliser la fonction **find**. Imaginez ceci: On a l'alphabet au complet en lettres minuscules dans une variable, et on a l'alphabet en lettre majuscules dans une autre variable. Si on trouve l'emplacement d'une lettre dans un alphabet, l'emplacement sera la même que pour l'autre alphabet...vous voyez l'astuce?

Et souvenez-vous lorsqu'on a décortiqué le problème. Nous voulions tester chaque lettre pour savoir si elle était en minuscule, ou majuscule. Alors, si nous réussissons à trouver, dans l'alphabet minuscule, notre lettre, nous saurons que notre lettre est dans l'alphabet minuscule. Et parce que la fonction **find** retourne l'emplacement de la lettre dans notre alphabet, il suffira de copier la lettre de l'alphabet majuscule qui se trouve au même endroit. Malin, non?

Alors, ça nous prends deux autres variables.

```
text <petit> "abcdefghijklmnopqrstuvwxyz"
text <grand> "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
```

C'est tellement simple, que ça me fait dire que le Toy est le plusse meilleur langage de tous les autres!

Donc, on a notre boucle, on sait ce que nous voulons faire. Mais nous ne savons pas encore comment le faire.

Alors, ça prend un listing...

```
text [phrase] "ceci est loi, respectez moi loi, loi, ici la loi!"

text <petit> "abcdefghijklmnopqrstuvwxyz"
text <grand> "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

integer [nombre_de_lettres]
count [nombre_de_lettres] phrase

integer [bonjour]
integer [bonsoir]
text [qui_suis_je]

integer [chercher]
integer [trouver]
text [contenir]

loop [nombre_de_lettres] [
      =[bonsoir]bonjour
      +[bonsoir]1
      =[qui_suis_je] bonjour, phrase, bonsoir
      !!!
      =[bonjour]bonsoir
/]
```

C'est pas très impressionnant comme ça, mais on sait où on s'en va.
Les !!! sont là pour nous indiquer que tout n'est pas rose, au royaume de Toy. En fait, aucun travail vraiment utile sera accompli si la variable **qui_suis_je** n'est pas utilisée pour faire quelque chose ensuite.

Nous savons, pour une multitude de raisons, que la variable **qui_suis_je** va contenir une à une chaque lettre de notre phrase. Et si nous ajoutions une à une chaque lettre dans une autre phrase, mais que si la lettre à ajouter est minuscule, nous la changions pour une majuscule? BINGO!

Ça nous prend une nouvelle variable:

```
text <nouvelle_phrase>
```

Et ça nous prend de la logique, beaucoup de logique. Parce que c'est compliqué pour un ordinateur de penser, nous devons le faire à sa place...

Premièrement, transformer la lettre si elle est en minuscule. Mais comment savoir si elle est en minuscule? En cherchant son emplacement dans la variable **petit**, comme cela:

```
find[chercher] petit, qui_suis_je
```

La fonction **find** est un peu capricieuse, en ceci que la variable **qui_suis_je** doit contenir une seule lettre. Si ce n'est pas le cas, ou bien si **find** ne trouve pas, alors la variable **chercher** sera égale à -1. Si la variable **chercher** est égale à -1 après la recherche, alors nous saurons que **find** n'aura pas trouvé.

Donc, si nous comparons la variable **chercher** avec -1, et que la comparaison est positive, nous saurons que la lettre **qui_suis_je** n'a pas été trouvé dans les petites lettres. Ça peut vouloir dire que la lettre est déjà grosse, ou bien est un espace, par exemple. Alors, il suffirait de simplement l'ajouter à la variable ~~**contenir**~~ **nouvelle_phrase**, sans avoir besoin de la convertir d'abord.

```
=chercher|==-1|if[

/]
```

Ce code ressemble un peu à une boucle, mais avec l'exception que c'est un **if**. Cette 'boucle'-ci sera exécuté qu'une seule fois, et ce seulement si la variable chercher est égale à -1.

Explication... Toy étant le plusse meilleur langage de tous, il contient une variable caché, qui se nomme simplement **$**, qui est utilisée lors-qu'aucune variable est mise entre parenthèses carrées ou triangulaires avec une fonction. Par exemple, la boucle **loop** a la variable **nombre_de_lettres** entre parenthèses carrées. En plus, la fonction **if** ne comprends pas les parenthèses, alors pour **if**, c'est obligé d'utiliser la variable **$**. Alors on met la valeur de la variable **chercher** dans la variable **$**, on teste si la valeur de la variable **$** est égale à **-1** (avec la fonction intuitive **==**), et puis on fait le **if**, qui sera exécuté si la variable **$** égale alors à **1** (parce que **==** va remplacer -1 par un si **$** est égale à **-1**).

Aussi, lorsque **if** est exécuté, il remplace le contenu de **$** par le nombre 0. Mais si **if** n'est pas exécuté parce que **$** est 0, alors il remplace le 0 de $ par le nombre 1. Donc:

```
=chercher|==-1|if[
      ??A
/]|if[
      ??B
/]
```

Est la logique de décisions pour savoir si la lettre qui_suis_je doit être remplacé ou non par sa version plus grosse.

Nous savons que la variable nouvelle_phrase sera notre phrase transformée, Donc:

```
+[nouvelle_phrase]qui_suis_je
```

Doit être mis à la place du **??A**. Nous savons, bien évidemment, que ce qui remplacera le **??B** sera un processus décisionnel fort complexe. Comme celui-ci:

```
=[trouver]chercher
+[trouver]1
=[qui_suis_je] chercher, grand, trouver
+[nouvelle_phrase]qui_suis_je
```

En fait, c'est exactement le processus décisionnel fort complexe que nous devons exploiter si nous voulons progresser dans notre synergie opératoire!

Et là, qu'est-ce qu'on fait? Ben, c'est terminé, nous avons réussi!
RÉCAPITULATIF:

```
text [phrase] "ceci est loi, respectez moi loi, loi, ici la loi!"

text <petit> "abcdefghijklmnopqrstuvwxyz"
text <grand> "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

integer [nombre_de_lettres]
count [nombre_de_lettres] phrase

integer [bonjour]
integer [bonsoir]
text [qui_suis_je]

integer [chercher]
integer [trouver]
text [contenir]

loop [nombre_de_lettres] [
      =[bonsoir]bonjour
      +[bonsoir]1
      =[qui_suis_je] bonjour, phrase, bonsoir
      find[chercher] petit, qui_suis_je
      =chercher|==-1|if[
            +[nouvelle_phrase]qui_suis_je
      /]|if[
            =[trouver]chercher
            +[trouver]1
            =[qui_suis_je] chercher, grand, trouver
            +[nouvelle_phrase]qui_suis_je
      /]
      =[bonjour]bonsoir
/]
```

Et si vous faites imprimer la phrase à l'écran, comme ceci:

```
print nouvelle_phrase
```

Vous verrez ceci à l'écran:

CECI EST LOI, RESPECTEZ MOI LOI, LOI, ICI LA LOI!

Percutant comme message, non? Ça me fait penser aux macros...

Une **macro** en Toy, est comme une fonction, mais on lui fait faire ce que nous voulons. Syntaxe de base:

```
macro <bazou> <

/>
```

Et ensuite, pour utiliser notre macro, nous faisons ceci:

```
bazou
```

Et cette macro ne fait rien, parce qu'elle ne contient rien. Alors, l'astuce, c'est de mettre notre beau code que nous avons écrit, dans bazou, et ensuite, nous pourrions, osons rêver, utiliser bazou pour rendre percutant n'importe quel message!

Et là, sachez que le Toy est le plusse meilleur langage de tous. Quand vous créer une variable dans bazou, et que cette variable a des parenthèses triangulaires, alors c'est possible de mettre dans cette variable, un contenu de l'extérieur de la variable. Si la macro bazou est appelée avec une variable cible, alors c'est possible de copier vers cette variable avec une variable de bazou dont le nom de variable commence et se termine par des parenthèses normales:

```
macro <bazou> <
      text <entre>
      text <(sortie)>

      ???
/>
```

Notez qu'il ne faut pas mettre de lettres accentués comme nom de variable. Toy ne le permet pas.
Dans **???** on met notre code, qu'on change un peu...

```
text <phrase> "ceci est loi, respectez moi loi, loi, ici la loi!"

text <petit> "abcdefghijklmnopqrstuvwxyz"
text <grand> "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

macro <bazou> <
      text <entre>
      text <(sortie)>


      integer [nombre_de_lettres]
      count [nombre_de_lettres] entre

      integer [bonjour]
      integer [bonsoir]
      text [qui_suis_je]

      integer [chercher]
      integer [trouver]
      text [contenir]

      loop [nombre_de_lettres] [
            =[bonsoir]bonjour
            +[bonsoir]1
            =[qui_suis_je] bonjour, entre, bonsoir
            find[chercher] petit, qui_suis_je
            =chercher|==-1|if[
                  +<(sortie)>qui_suis_je
            /]|if[
                  =[trouver]chercher
                  +[trouver]1
                  =[qui_suis_je] chercher, grand, trouver
                  +<(sortie)>qui_suis_je
            /]
            =[bonjour]bonsoir
      /]
/>
```

Et on appelle bazou, comme ceci:

```
bazou[phrase]entre phrase
```

Et le travail est fait! Mais...

JULES CÉSAR!!!

Explications: Qui dit Jules César, qui conquêtes, et qui dit conquêtes, dit fierté. Qui dit fierté, est fier d'utiliser le plusse meilleur langage, et donc, ordinateur. Qui dit ordinateur, pense à codes secrets, donc, Jules César!...

Le code de César, c'est de remplacer chaque lettre d'une phrase par la lettre de treize positions plus loin. Je vous le promets, **bazou** sait faire ça. Mais **bazou** est superstitieux, alors il ne va pas vouloir du nombre 13...

Donc, à Rome on pense comme les Romains, et les chiffres romains sont tout à l'envers. Donc...

text <CESAR_CYPHER> "NOPQRSTUVWXYZABCDEFGHIJKLM"

On ajoute ensuite deux variables dans notre macro:

text <petit>
text <grand>

Et voilà! Nous sommes dorénavant des cryptopgraphes!

Bon, vous vous dites, que si bazou a une variable **petit** qui contient rien, et **grand** qui contient rien, alors **bazou** ne fera rien. Vrai! C'est pour cela qu'on va l'aider un peu...

bazou[phrase] entre phrase, petit petit, grand grand

Va transformer notre phrase pour la rendre criante de 'vérité'.

Et...

bazou[phrase] entre phrase, petit grand, grand CESAR_CYPHER

Va transformer notre phrase pour la rendre indéchiffrable!

Mais, si on n'a pas le droit de faire de la cryptographie, qu'est-ce qu'on fait alors?

On fait la même affaire:

bazou[phrase] entre phrase, petit grand, grand CESAR_CYPHER

Va transformer la phrase crypté en phrase lisible.

Et si par erreur vous faites:

bazou[phrase] entre phrase, petit CESAR_CYPHER, grand grand

Pas de problèmes, la même chose ce produira. C'est simple comme mathématiques!

Sachez seulement que **CESAR_CYPHER** a besoin que les lettres soient criantes. Si vous voulez crypter n'importe quelle phrase, vous aurez besoin d'un **petit_cesar**:

text <petit_cesar> "nopqrstuvwxyzabcdefghijklm"

Alors:

bazou[phrase] entre phrase, petit petit, grand petit_cesar
bazou[phrase] entre phrase, petit grand, grand CESAR_CYPHER

Est double-plus versatile! Il y a un message politique dans ce tutoriel en plus.

Telle est, la puissance de Toy!!!!!!

Fin du tutoriel.