Le TME solo a été plutôt réussi. Ceci est la correction de la version 2 (groupe de 15h00-15h45), mais les différences avec la version 1 (groupe de 14h00-14h45) sont mineures.

N'oubliez pas de lire attentivement l'énoncé : il fallait ajouter au moins un autre test après chaque définition de fonction, et deux tiers d'entre vous ont perdu 2,5 points en oubliant d'écrire ces tests.

Nous avons pu remarquer quelques lacunes sur les chaînes de caractères (voir Qu. 3.1), ainsi que sur le déboguage des fonctions utilisant des boucles.

Lorsqu'une fonction ne passe pas les tests, il faut essayer de comprendre pourquoi. Pour cela, vous pouvez :

— Evaluer votre fonction sur certaines valeurs, via la zone d'évaluation de MrPython.
— Utiliser la fonction `print` pour afficher les valeurs contenues par les variables.
— Simuler sur papier le comportement de la fonction que vous avez implémenté.

Par ailleurs, essayez d'utiliser des noms de variables plus clairs que a, b, c, ...

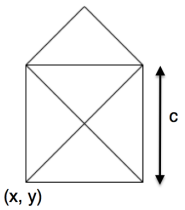
Le TME solo était sur 36 points, comme l'indique le barême dans cette correction. La note a ensuite été ramenée sur 20, et arrondie au demi-point supérieur.

La répartition des notes est la suivante :

3,5	4	4,5	5	5,5	5,5	7	8	8	9	9	9	10	10	11
11	11,5	12	12,5	13,5	14	14,5	14,5	17	17,5	18	18,5	18,5	20	

Exercice 1

1. (5 points) Écrire une fonction `enveloppeH` qui, étant donné trois réels x, y et c, retourne l'image d'une enveloppe carrée, ouverte sur le haut, de coin bas gauche (x, y) et de côté c, comme celle-ci :



Indication. On utilisera les fonctions :
`draw_triangle(x1, y1, x2, y2, x3, y3)` ou/et `draw_line(x1, y1, x2, y2)`,
`overlay(image1, image2,...)`,
`show_image(image)`.

Solution: Il ne fallait pas oublier de spécifier en hypothèse que $c > 0$. Attention : la fonction devait renvoyer une image, l'appel à `show_image` (ayant pour signature `Image -> NoneType`) n'était pas à faire dans la fonction, mais sur un appel de la fonction (c.f. question suivante). Il était possible de réaliser le dessin avec seulement trois appels à `draw_triangle`.

```
def enveloppeH(x, y, c):
    """ float * float * float -> Image
    Hyp: c > 0"""
    # triangle_gauche : Image
    triangle_gauche = draw_triangle(x, y, x+c, y, x, y+c)
    # triangle_droit : Image
    triangle_droit = draw_triangle(x+c, y, x+c, y+c, x, y)
    # triangle_bord : Image
    triangle_bord = draw_triangle(x, y+c, x+c/2, y+3*c/2, x+c, y+c)
    return overlay(triangle_gauche, triangle_droit, triangle_bord)
```

2. (2 points) Dans cet exercice, on ne demande pas de jeu de tests, mais simplement deux affichages :
   — l'image `enveloppeH(.1, -.4, .5)`
   — une deuxième image avec des valeurs que vous aurez choisies.

Solution:

```
show_image(enveloppeH(.1, -.4, .5))
show_image(enveloppeH(0, 0, .25))
```

Exercice 2

1. (7 points) Écrire une fonction `log_5` qui, étant donné un nombre $x \geq 1$, retourne le plus grand entier k tel que $5^k \leq x$. La fonction doit utiliser une boucle `while`.
   Par exemple :
   `log_5(125)` retourne 3
   `log_5(124)` retourne 2.

Solution: Il fallait donc faire une boucle while, qui parcourt les puissances de 5 jusqu'à dépasser x. En lançant les tests, on voit que la fonction ne doit pas retourner le compteur directement. En exécutant la fonction, on se rend vite compte que la valeur du compteur `k` est trop grande d'une unité, et qu'il fallait donc écrire `return k-1` .

```
def log_5(x):
    """ Number -> int
    Hyp: x >= 1"""
    # k : int
    k = 0
    # nb : Number
    nb = 1
    while nb <= x:
        k = k + 1
        nb = 5 * nb
    return k-1

assert log_5(125) == 3
assert log_5(124) == 2
assert log_5(1) == 0
```

Exercice 3

1. (6 points) Écrire une fonction `permutation2` qui, étant donné un mot s de longueur 3, retourne le mot obtenu en échangeant le deuxième caractère de s et son troisième caractère.
   Par exemple :
   `permutation2('col')` retourne `'clo'`.

Solution: Il y a eu trois sources d'erreurs pour cette question :
- Les indices des chaînes **commencent à 0** : pour la chaine `s = "bonjour"`, `s[1]` est le caractère 'o', tandis que `s[0]` est le caractère 'b'.
- Il n'y avait pas besoin d'une boucle pour implémenter la fonction.
- Il fallait préciser en hypothèse que la longueur de la chaîne était fixée à 3.

```
def permutation2(s):
    """ str -> str
    Hyp: len(s) == 3"""
    return s[0] + s[2] + s[1]

assert permutation2('col') == 'clo'
assert permutation2('abc') == 'acb'
```

2. (9 points) Un *facteur* d'un mot `s` est un découpage `s[i:j]`.
   Par exemple, `'namb'` est un facteur de `'topinambour'`.
   Écrire une fonction `secret2` qui, étant donné un mot s, retourne le mot obtenu en appliquant la fonction `permutation2` à tous les facteurs consécutifs de longueur 3 de s.
   Si la longueur de s n'est pas un multiple de 3, le dernier facteur (qui a 1 ou 2 caractères) est

inchangé.
Par exemple : `secret2('secret')` retourne `'scerte'`, `secret2('malabar')` retourne `'mlaaabr'`, `secret2('etudiant')` retourne `'eutdaint'`.

Solution: Cette question était la plus difficile du TME, en particulier pour la gestion des chaînes d'une longueur non multiple de 3. Une solution est de faire une boucle qui s'occupe d'appliquer `permutation2` au facteurs conséductifs de longueur 3 de s, puis d'ajouter le reste de la chaîne.

```
def secret2(s):
    """ str -> str"""
    # resultat : str
    resultat = ""
    # i : int
    for i in range(0, len(s)//3):
        resultat = resultat + permutation2(s[3*i:3*(i+1)])

    return resultat + s[3*(len(s)//3):len(s)]

assert secret2('secret') == 'scerte'
assert secret2('malabar') == 'mlaaabr'
assert secret2('etudiant') == 'eutdaint'
assert secret2('voyages') == 'vyoaegs'
```

Une autre solution est d'accumuler dans une variable auxiliaire le facteur, jusqu'à ce qu'il atteigne une longueur 3. Dans ce cas, le facteur est transformé avec la fonction `permutation2`, puis réinitialisé à la chaîne vide. Si la boucle est finie, on ajoute au résultat le dernier facteur, qui a une longueur plus petite que 3.

```
def secret2bis(s):
    """ str -> str"""
    # resultat : str
    resultat = ""
    # facteur : str
    facteur = ""
    # c : str
    for c in s:
        facteur = facteur + c
        if len(facteur) == 3:
            resultat = resultat + permutation2(facteur)
            facteur = ""

    return resultat + facteur
```

3. (7 points) Écrire une fonction `L_secret2` qui, étant donné une liste de mots L, retourne la liste obtenue en appliquant la fonction `secret2` à tous les mots de L. Par exemple :
`L_secret2(['je','hais','les','voyages','et','les','explorateurs'])`
retourne :
`['je', 'hias', 'lse', 'vyoaegs', 'et', 'lse', 'epxlroaetusr']`.

Solution: Cette question est un problème classique de transformation de listes utilisant le schéma `map`. Il suffit de parcourir les chaînes écrites dans la liste, et d'appeler `secret2` sur chacune d'entre elles.

```
def L_secret2(L):
    """ list[str] -> list[str]"""
    # res : list[str]
    res = []
    # s : str
    for s in L:
        res.append(secret2(s))
    return res

assert L_secret2(["je", "hais", "les", "voyages", "et", "les", "explorateurs"
   ]) == ["je", "hias", "lse", "vyoaegs", "et", "lse", "epxlroaetusr"]
assert L_secret2([]) == []
```