Exemples d'utilisation d'aides et de fonctions de correction dans un notebook jupyter

Les fonctions de correction sont dans un fichier python à part. Pour que le fichier ne soit pas visible dans l'application Capytale, le fichier doit être préfixé par le caractère underscore (tiret sous le 8). Ce fichier python doit donc être importé au début du notebook (voir capture d'écran ci-dessous).

```
1 from _test_fonctions import *
```

Dans la version 1 du notebook (version pour les élèves les plus autonomes), les élèves doivent compléter le code complet de la fonction. Le nom de la fonction doit être donné pour que la comparaison avec la fonction de correction puisse être effectuée (le nom de la fonction de correction est identique, mais suffixé par la chaine de caractère _correction). Dans cette capture d'écran ci-dessous, on peut voir la fonction à compléter, suivie des aides sur lesquelles les élèves peuvent cliquer à la demande.

```
1  def solution_glouton_monnaie(somme, pieces):
2      N = len(pieces)
3      rendu = [0]*N
4      k = N-1
5      # Complétez le code ici !
6
7      return rendu
8
9  sol = solution_glouton_monnaie(9,(1,2,5))
10 print("solution : {}, nombre de pièces : {}".format(sol,sum(sol)))
11 test_fonction(solution_glouton_monnaie, [(9,(1,2,5))])
```

Aide n°1

Aide n°2

Aide n°3

Aide n°4

Dans la version 2 du notebook (version pour les élèves qui ont besoin d'aide pour démarrer l'écriture des programmes), les programmes sont structurés avec des commentaires et sont des programmes à trous.

```
1  def solution_glouton_monnaie(somme, pieces):
2      # 1) Initialisation de la combinaison de pièces à rendre
3      N = len(pieces)
4      rendu = [0]*N
5      # 2) Boucle sur le système de pièces : on commence par la pièce de plus forte valeur
6      k = N-1
7      # ligne suivante à modifier !
8      while False :
9          # 3) Test sur la valeur de la pièce
10         if ... :
11             ...
12             ...
13         # 4) Passage à la pièce de valeur inférieure
14         else :
15             ...
16     return rendu
17
18 sol = solution_glouton_monnaie(9,(1,2,5))
19 print("solution : {}, nombre de pièces : {}".format(sol,sum(sol)))
20 test_fonction(solution_glouton_monnaie, [(9,(1,2,5))])
```

Selon le niveau d'autonomie des élèves, on peut donc soit donner la version 1, soit la version 2. On peut aussi laisser le choix aux élèves de choisir la version sur laquelle ils souhaitent travailler. Une fois la fonction écrite en totalité ou complétée par les élèves, le résultat retourné par la fonction est comparé avec celui de la fonction de correction pour les mêmes entrées. Un tableau récapitulatif affiche les deux résultats (soit en vert quand les résultats concordent, soit en rouge quand il y a une erreur). Dans la capture d'écran ci-dessous, la fonction écrite présente volontairement des erreurs, mais retourne cependant le bon résultat.

```python
def solution_glouton_monnaie(somme, pieces):
    # 1) Initialisation de la combinaison de pièces à rendre
    N = len(pieces)
    rendu = [0]*N
    # 2) Boucle sur le système de pièces : on commence par la pièce de plus forte valeur
    k = N-1
    while somme > 0  and k >= 0:
        # 3) Test sur la valeur de la pièce
        if pieces[k] <= somme :
            somme = somme - pieces[k]
            rendu[k] = rendu[k] + 1
        # 4) Passage à la pièce de valeur inférieure
        k = k - 1
    return rendu

sol = solution_glouton_monnaie(8,(1,2,5))
print("solution : {}, nombre de pièces : {}".format(sol,sum(sol)))
test_fonction(solution_glouton_monnaie, [(8,(1,2,5))])
```

```
solution : [1, 1, 1], nombre de pièces : 3
```

Paramètres	Résultat attendu	Résultat obtenu	Validation
(8, (1, 2, 5))	[1, 1, 1]	[1, 1, 1]	✅

Pour pouvoir tester la validité de la fonction, il est donc nécessaire de faire des tests supplémentaires avec des jeux de tests aléatoires.

```python
import random

jeu_test = [(random.randint(1,100), (1,2,5,10,20,50,100,200)) for i in range(5)]
test_fonction(solution_glouton_monnaie, jeu_test)
```

Paramètres	Résultat attendu	Résultat obtenu	Validation
(33, (1, 2, 5, 10, 20, 50, 100, 200))	[1, 1, 0, 1, 1, 0, 0, 0]	[1, 1, 0, 1, 1, 0, 0, 0]	✅
(77, (1, 2, 5, 10, 20, 50, 100, 200))	[0, 1, 1, 0, 1, 1, 0, 0]	[0, 1, 1, 0, 1, 1, 0, 0]	✅
(74, (1, 2, 5, 10, 20, 50, 100, 200))	[0, 2, 0, 0, 1, 1, 0, 0]	[1, 1, 0, 0, 1, 1, 0, 0]	❌
(12, (1, 2, 5, 10, 20, 50, 100, 200))	[0, 1, 0, 1, 0, 0, 0, 0]	[0, 1, 0, 1, 0, 0, 0, 0]	✅
(99, (1, 2, 5, 10, 20, 50, 100, 200))	[0, 2, 1, 0, 2, 1, 0, 0]	[1, 1, 1, 1, 1, 1, 0, 0]	❌

On peut donc voir sur cet exemple, que la fonction écrite par l'élève n'est pas valide et doit donc être corrigée. L'élève va donc devoir modifier sa fonction et relancer à nouveau les tests aléatoires pour valider sa fonction.

Enfin pour aider les élèves à l'écriture des programmes, les élèves peuvent cliquer sur les aides. Les élèves jouent le jeu en ne regardant celles-ci qu'en cas de besoin ou de blocage. Ces aides correspondent à des interventions qu'on aurait fait à l'oral, pour faire le point avec la classe, en cas de difficulté rencontrée par la classe. Sur la capture d'écran ci-dessous, l'élève a cliqué sur les aides n°1 et n°3 uniquement.

```python
def solution_glouton_monnaie(somme, pieces):
    N = len(pieces)
    rendu = [0]*N
    k = N-1
    # Complétez le code ici !

    return rendu

sol = solution_glouton_monnaie(9,(1,2,5))
print("solution : {}, nombre de pièces : {}".format(sol,sum(sol)))
test_fonction(solution_glouton_monnaie, [(9,(1,2,5))])
```

Aide n°1

Seules les pièces de valeur inférieure ou égale à la somme à rendre peuvent être rendues.

Aide n°2

Aide n°3

Si on rend la k-ieme pièce du système de pièces, alors on doit incrémenter `rendu[k]` de +1

Aide n°4

En conclusion, ces aides et fonctions de correction ont pour objectif de rendre les élèves plus autonomes. Elles permettent aussi de diminuer les interventions de l'enseignant lors de la séance. En effet les aides et les résultats des corrections étant accessibles aux élèves, ceux-ci n'ont pas à attendre la venue de l'enseignant pour commencer à corriger leurs programmes.