Méthologie de la Programmation
TP3 : erreurs en python
le programme du pendu

am@up8.edu

Septembre 2022

Dans ce TP :
- découverte, lecture et correction des messages d'erreur (mais pas que !) en python
- Écriture du code en python (documentation : `https://docs.python.org/3/`)
- Dépôt sur le moodle (cours Info-L1-MdP, clé d'inscription PROGRAMMATION) avant mardi prochain 13h d'une archive contenant uniquement des fichiers ".py" correspondant aux programmes des différents exercices.
  - `VOTRENOM_exercice_{x}.py`
  - `zip L1_MdP_TP3_VOTRENOM.zip exercice_1.py exercice_2.py exercice_3.py` permet de créer une archive contenant vos fichiers.
- **notation** :
  - le **nommage** des fichiers, des fonctions, et des variables sera pris en compte dans la notation.
  - votre programme doit comporter une fonction `main` (voir exemple) dans lequel vous effectuez des appels aux fonctions développées pour **tester et montrer leur bon fonctionnement**. Pensez à tester les cas limites.
  - rendre le TP avant la date spécifiée sur le moodle ne donne pas de bonus.
  - pas de rendu sur moodle $\Rightarrow 0$.

Pour commencer, téléchargez l'archive `https://alicemillour.github.io/assets/cours/MdlP/MdlP_2223_TP3.zip` et décompressez-la dans un dossier approprié.

Exercice 1 : les messages d'erreur se lisent de bas en haut !

Chaque fonction du fichier exercice1.py `erreur_x()` contient au moins une erreur. Dé-commentez les appels des fonctions et corrigez les erreurs au fur et à mesure.
Attention : Le comportement attendu de chaque fonction est **spécifié en commentaire**, une fonction qui s'exécute sans message d'erreur n'a pas nécessairement le comportement désiré.

Lorsque vous corrigez une erreur (par exemple une `TypeError`), notez en commentaire de votre correction le(s) message(s) associé(s) qui vous a(ont) permis de corriger le code, comme ceci :

```
1 def erreur_x():
2         """
```

```
3         Affiche : 3 petits chats
4         """
5         chaine = "petits chats"
6         num = 3
7         print(num + " " + chaine)
```

devient :

```
1 def erreur_x():
2         """
3         Affiche : 3 petits chats
4         """
5         chaine = "petits chats"
6         num = 3
7         print(str(num) + " " + chaine) # TypeError : unsupported
             operand type(s) for +: 'int' and 'str'
```

Exercice 2 : Un programme qui donnes les informations d'exécution d'un script

Vous avez vu en cours un programme qui affiche le nombre et la valeur de(s) paramètre(s) avec le(s)quel(s) il a été exécuté.

Complétez le fichier exercice2.py.

Les résultats d'exécution attendus sont les suivants (attention aux messages affichés qui ne sont pas les mêmes en fonction du nombre d'arguments) :

— $ VOTRE_NOM_exercice2.py

```
================================================
          Informations du programme
================================================

La ligne de commande contient 1 argument(s) :

- Le premier argument (indice 0) est le nom du script : affiche_parametres.py
```

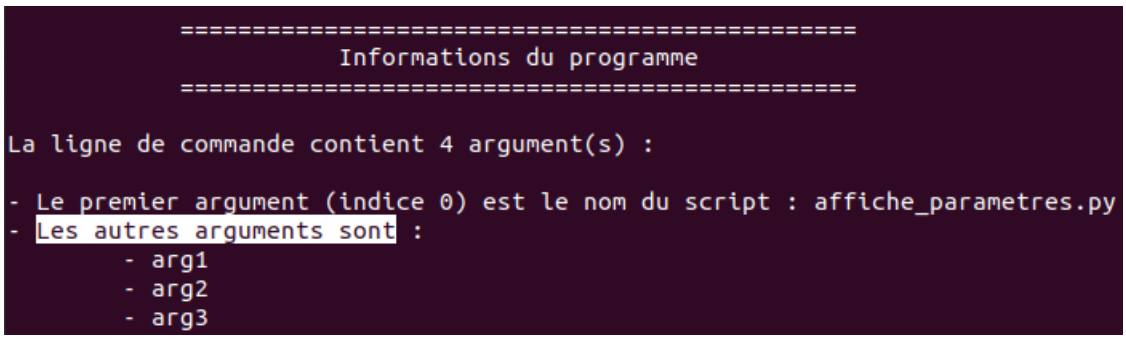
— $ VOTRE_NOM_exercice2.py arg1

```
================================================
          Informations du programme
================================================

La ligne de commande contient 2 argument(s) :

- Le premier argument (indice 0) est le nom du script : affiche_parametres.py
- L'autre argument est :
        - arg1
```

— $ VOTRE_NOM_exercice2.py arg1 arg2 arg3



Indices : vous devez utiliser la fonction `format()` sur une chaîne contenant n fois le motif « {} », n étant le nombre d'arguments passés au script python.
Vous devez passer en paramètre de la fonction `format` la liste des paramètres du script, en excluant la première valeur.
Vous pouvez choisir les éléments d'une liste comme sur l'exemple ci-dessous :

```
1 L = ["A","B","C","D"]
2 L[1] # "B"
3 L[-1] # "D"
4 L[2:3] # ["C","D"]
```

1 Exercice 3 : Jeu du Pendu

Dans cet exercice nous allons construire les premiers éléments d'un jeu du pendu interactif se jouant dans le terminal.
Le principe est le suivant :

1. le programme fait deviner à un joueur un mot saisi par un (autre) utilisateur.

Le fichier `exercice3.py` contient le squelette du script.

1.1 Fonction `get_mot()`

Implémentez la fonction `get_mot` qui demande à l'utilisateur de rentrer un mot à faire deviner.

1.2 Fonction `pendu_run()` : initialisation

Écrire une fonction `pendu_run()` qui prend en argument le mot retourné par la fonction `get_mot()`, crée et initialise les variables locales nécessaires (voir commentaire dans le fichier), et affiche à l'utilisateur les messages indiqués.

1.3 Fonction `pendu_run()` : jouer

Le but du jeu est donc de faire deviner le mot choisi au départ de la manière suivante :

- Le joueur à deux possibilités, soit entrer une lettre, soit tenter de deviner le mot complet.
- S'il se trompe de lettre il perd un essai.
- S'il tente d'entrer un mot complet qui n'est pas le mot attendu, il perd également un essai.

— S'il retente une lettre qu'il a déjà essayé, qu'elle soit bonne ou pas, il ne perd pas d'essai.

La partie s'arrête si le mot est deviné ou si le joueur n'a plus d'essais.

Implémentez ces fonctionnalités dans la fonction `pendu_run()`.