ACT MODULES EN PYTHON

Compétences travaillées :

APP	Mobiliser les concepts et les technologies adaptés au problème
ANA	Modéliser un problème en utilisant les objets conceptuels de l'informatique pertinents (fonctions et modules)
REA	Imaginer et concevoir une solution modulaire : décomposer en blocs, se ramener à des sous-problèmes simples et indépendants

1. Approche de la modularité avec les fonctions

La *programmation fonctionnelle* permet de simplifier l'écriture d'un programme.
Voyons ce que l'on peut faire pour améliorer encore l'aspect *modulaire* d'un programme.

1) Ecrire un programme energie.py qui demande à l'utilisateur la masse d'un objet **m**, sa vitesse **V** et son altitude **H** et qui calcule l'énergie mécanique associée.

L'énergie mécanique Em = Ec + Epp

avec l'énergie cinétique Ec = $\frac{1}{2}$ .m.V^2 et l'énergie potentielle Epp = m.g.H

Les trois énergies sont calculées à l'aide de 3 fonctions distinctes nommées Emeca, Ec et Epp.

2) Créer un nouveau programme fonctions.py dans lequel on écrira les 3 fonctions de calcul des énergies. Dans le programme energie.py, ajouter en première ligne l'instruction :

```
from fonctions import Emeca, Ec, Epp
```

Tester le programme et commenter l'intérêt de cette formulation.

Remarques :

- On parle du *module* fonctions.py et du programme energie.py.
- L'appel des fonctions du module se fait de façon nominative et explicite (« fonction par fonction »), mais on peut :

  → Importer l'ensemble : `from fonctions import *`

  → Importer et donner un alias : `from fonctions import Emeca as Em`
- Dans le cas où l'on a beaucoup de module, on les organisera en packages : concrètement, un dossier package qui contient les modules en question. L'instruction d'appel du module devient :

```
from packages.fonctions import Em, Ec, Epp
```

2. Application : création d'un QCM

Nous allons créer un module permettant la création et la correction de QCM en python.

Le QCM est stocké dans un tableau de tableaux du type :

```
QCMNSI = [["L'amie écureuil de Bob l'éponge s'appelle", "En dormant, Patrick bave plus que Gary", "Patrick est une étoile"],\
          ["Cindy", "*Vrai", "à 3 branches"],\
          ["Mindy", "Faux", "*à 5 branches"],\
          ["*Sandy", "match nul", "à 6 branches"]]
```

Il contient les questions sous forme de **chaînes de caractères** et la proposition correcte est précédée du caractère ***** .

On dispose d'un module qcm.py qui contient les fonctions utiles : elles sont nommées, les *docstrings* sont écrites, des ***assert*** également, mais elles restent à coder...

Il y a les 4 fonctions suivantes :

passageQCM(QCM)	*affiche_correction(QCM, i)*
affichageQCM(QCM, i)	*score_reponse(QCM, rep, i)*

Le résultat attendu de l'interface pour chaque question est comme ci-contre :

```
 L'amie écureuil de Bob l'éponge s'appelle
1  :  Cindy
2  :  Mindy
3  :  Sandy
Entrez votre N° de réponse (1, 2 ou 3) : 3

++++++++++++++++++++++++++++++++++++++++
La réponse correcte était :  3  avec ' Sandy '
Score actuel  1  sur  1
++++++++++++++++++++++++++++++++++++++++
```

Le fichier passage_qcm.py contient le programme principal.

Travail demandé :

1) **Coder** les fonctions du module.
2) **Tester** le programme.
3) **Décrire** précisément l'ensemble des fonctions : les entrées, les sorties et les traitements.