SE TESTER QUIZ

Vérifiez que vous avez bien compris les points clés des **fiches 1 à 4**.

1 Modularité

→ FICHE 1

1. Parmi les lignes qui suivent, trouver celle qui ne permet pas d'importer et d'utiliser la totalité du module `itertools` :

☐ **a.** `import itertools`
☐ **b.** `import itertools as itt`
☐ **c.** `from itertools import cycle`

2. On souhaite écrire une portion de code qui permette de savoir si une année est bissextile ou non. Qu'est-ce qui est le plus approprié ?

☐ **a.** Écrire un programme principal qui demande à l'utilisateur de taper une année et qui indique si elle est bissextile.
☐ **b.** Écrire une fonction qui indique si une année passée en paramètre est bissextile ou non en renvoyant un booléen.
☐ **c.** Écrire un module `bissextile.py` qui contiendra tout ce qu'il faut pour tester le caractère bissextile d'une année.

2 Débogage

→ FICHE 2

1. Quel message d'exception s'affiche si on tente d'exécuter le code suivant ?

```
1 a = 1
2 for i in range(3):
3     print("i = {}, a = {}".format(i, a)
4     a = 2 * a
```

☐ **a.** `NameError`
☐ **b.** `SyntaxError`
☐ **c.** `IndexError`
☐ **d.** `IndentationError`

2. À quel moment l'exception qui précède est-elle levée ?

☐ **a.** avant l'exécution
☐ **b.** pendant l'exécution

3 Message d'erreur

→ FICHE 2

1. Quel message d'exception s'affiche si on tente d'exécuter le code suivant ?

```
1 v = 1
2 while v < 100:
3     if v % 7 == 0:
4         print(v, "est un multiple de 7")
5         else :
6         print(v, "n'est pas un multiple de 7")
7     v = v + 1
```

☐ **a.** `NameError`
☐ **b.** `SyntaxError`
☐ **c.** `IndexError`
☐ **d.** `IndentationError`

2. À quel moment l'exception qui précède est-elle levée ?

☐ **a.** avant l'exécution ☐ **b.** pendant l'exécution

4 Une faute courante

→ FICHE 2

Le code suivant calcule les 10 premiers termes de la suite de Fibonacci. Quel message d'exception s'affiche si on tente de l'exécuter ?

```
f = [0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
for i in range(1, 10):
    f[i + 1] = f[i] + f[i - 1]
print(f)
```

☐ **a.** `NameError` ☐ **c.** `IndexError`
☐ **b.** `SyntaxError` ☐ **d.** `IndentationError`

5 Tests

→ FICHE 3

1. Pour tester une fonction ayant un seul paramètre, choisir une valeur du paramètre et vérifier que le résultat est correct pour cette valeur est toujours suffisant.

☐ **a.** Vrai ☐ **b.** Faux

2. On considère le code suivant qui, étant donné un terme u_n de la suite de Syracuse, renvoie le terme suivant u_{n+1}.

```
def syracuse(un: int) -> int:
    if un % 2 == 0:
        return un // 2
    else:
        return 3 * un + 1
```

Le test suivant est proposé :
`assert syracuse(32) == 16`
Parmi ces autres tests, lequel vous semble le plus urgent à ajouter ?

☐ **a.** `assert syracuse(16) == 8`
☐ **b.** `assert syracuse(0) == 0`
☐ **c.** `assert syracuse(3) == 10`

6 Guide de style

→ FICHE 4

Quel nom, selon la PEP 8, est plus approprié pour une fonction qui calcule la suite de Fibonacci ?

☐ **a.** `f`
☐ **b.** `fibonacci`
☐ **c.** `Fibonacci`
☐ **d.** `SuiteFibonacci`

S'ENTRAÎNER

7 Documenter un module d'arithmétique

→ FICHE 3

Un fichier `arithmetique.py` contient des fonctions sur l'arithmétique des nombres entiers. En particulier :
– `expo` qui prend en paramètres les entiers `a`, `b` et `n` et calcule le reste de la division par `n` de a^b ;
– `pgcd` qui prend en paramètres les entiers `a` et `b` et renvoie le plus grand diviseur commun de `a` et `b` ;
– `decomposition` qui prend un entier positif `n` en paramètre et renvoie la liste de ses facteurs premiers ainsi que leur multiplicité.

a. Proposer des dosctrings pour le module et les trois fonctions décrites (annoter les fonctions avec les annotations de type, voir → PRÉPABAC NSI 1re, FICHE 10). Il n'est pas demandé de donner le code des fonctions.

b. Ajouter quelques tests aux docstrings de la question **a**.

8 Écrire des tests couvrants

→ FICHE 3

Dans un module `outils_listes`, on dispose d'une fonction qui indique si les éléments d'une liste sont tous différents (si c'est le cas la fonction renvoie `True`, `False` sinon).

La première ligne de la fonction est :

```
def tous_differents(lst: list) -> bool:
```

Écrire des tests couvrants à l'aide d'assertions.

9 Calculer la valeur d'un mot : arithmancie

→ FICHE 3

On souhaite associer une valeur numérique à une chaîne de caractères. La valeur associée à la chaîne est la somme des valeurs associées à chacun des caractères qui la composent. Aux lettres non accentuées de A à Z, qu'elles soient en minuscules ou en majuscules, on associe leur numéro d'ordre dans l'alphabet, de 1 à 26. À tous les autres symboles (lettres accentuées, chiffres, espaces, ponctuations), on associe la valeur 0.

On rappelle que `ord(x)` est le code numérique associé à la lettre passée en paramètre → PRÉPABAC NSI 1re, FICHE 7.

La chaîne `string.ascii_uppercase` est définie dans le module `string` et vaut `"ABCDEFGHIJKLMNOPQRSTUVWXYZ"`.

1. Pour chacune des propositions suivantes, proposer un test qui montre que la fonction ne répond pas au cahier des charges de l'énoncé.

a.

```
def valeur1(chaine):
    s = 0
    for c in chaine:
        if c >= "A" and c <= "Z":
            s = s + ord(c) - ord('A') + 1
    return s
```

b.

```
def valeur2(chaine) :
    s = 0
    for c in chaine.upper():
        if c >= "A" and c < "Z":
            s = s + ord(c) - ord('A') + 1
    return s
```

c.

```
import string
def valeur3(chaine):
    s = 0
    for c in chaine.upper():
        if c in string.ascii_uppercase:
            s = s + string.ascii_uppercase.index(c)
    return s
```

2. Proposer un ensemble complet de tests pour la fonction que l'on veut créer (on nomme cette fonction `valeur`).

3. Proposer le code de cette fonction (il est possible de s'inspirer d'une des versions vues à la question **1.** en la corrigeant). Ne pas oublier de vérifier que la fonction écrite passe bien les tests de la question **2**.

10 Enquêter sur une erreur dans un calcul de π

→ FICHE 2

La somme des inverses des carrés des nombres entiers converge vers $\frac{\pi^2}{6}$:

$$\sum_{k=1}^{k=+\infty} \frac{1}{k^2} = \frac{\pi^2}{6}.$$

On utilise cette formule pour trouver une approximation de π :

```
1  import math
2  def terme(k: int) -> float:
3      return 1 / (k ** 2)
4
5  def approxpi(n: int) -> float:
6      s = 0
7      # utilise les termes jusqu'à 1/n**2 inclus
8      for k in range(n):
9          s = s + terme(k)
10     return math.sqrt(s * 6)
```

1. Lors de l'exécution de `approxpi(1000)` une exception est levée à l'exécution de la ligne 3. Quel est le type d'exception ?

☐ **a.** `ValueError` ☐ **b.** `SyntaxError` ☐ **c.** `TypeError`

☐ **d.** `ZeroDivisionError` ☐ **e.** `IndexError` ☐ **f.** `NameError`

2. Bien que l'exécution soit interrompue ligne 3, la source de l'erreur est ailleurs. Expliquer d'où provient l'erreur et proposer un correctif.

11 Enquêter sur une erreur de calcul de produit scalaire

→ FICHE 2

Le produit scalaire de deux vecteurs $\vec{u}$ et $\vec{v}$ de l'espace de coordonnées respectives $(u_1 ; u_2 ; u_3)$ et $(v_1 ; v_2 ; v_3)$ est le réel $u_1v_1 + u_2v_2 + u_3v_3$.

Le code suivant permet de calculer des produits scalaires. Il est testé sur des vecteurs de l'espace choisis au hasard :

```
1  import random
2
3  def scalaire(v1: tuple, v2: tuple) -> float:
4      """ Calcule le produit scalaire des deux
5          vecteurs v1 et v2
6      """
7      s = 0
8      for k in range(len(v1)):
9          s = s + v1[k] * v2[k]
10     return s
11
12 def random_vect(n: int) -> tuple:
13     """ Génère un vecteur de taille n contenant
14         des entiers non nuls entre -10 et 10
15     """
16     v = []
17     for i in range(n):
18         val = random.randint(-10, 10)
19         if val != 0:
20             v.append(val)
21     return tuple(v)
22
23 def main():
24     for i in range(20):
25         print(scalaire(random_vect(3), random_vect(3)),
26
27 main()
```

Prises séparément, les deux fonctions semblent opérationnelles :

```
>>> scalaire((1, 2, 1), (-1, 3, -2))
3
>>> random_vect(3)
(-6, 2, 8)
```

Pourtant, lorsqu'on exécute la fonction `main()`, le programme n'affiche que quelques produits scalaires (ici 12, 118 et –104) et *plante* :

```
>>> main()
12
118
-104
Traceback (most recent call last):
  File "scal.py", line 27, in <module>
    main()
  File "scal.py", line 25, in main
    print(scalaire(random_vect(3), random_vect(3)))
  File "scal.py", line 9, in scalaire
    s = s + v1[k] * v2[k]
IndexError: tuple index out of range
```

1. En analysant le traceback donné plus haut, indiquer quel est le type d'exception qui a été levée, et sur quelle ligne de code.

2. Sur la ligne en question, qu'est-ce qui pourrait avoir provoqué l'erreur ?

3. Proposer d'ajouter un affichage `print(...)` à un endroit du programme pour essayer de mettre en évidence la nature du problème, puis réexécuter le code.

4. Expliquer l'erreur. Pourquoi ne se produit-elle pas toujours au même moment ?

5. Proposer un correctif.

OBJECTIF BAC

12 Module de chiffrement

45 min

LE SUJET

Le petit module suivant (`macrypto.py`) propose quelques outils de chiffrement :
- la méthode de César (https://fr.wikipedia.org/wiki/Chiffrement_par_décalage) ;
- la méthode de Vigenère (https://fr.wikipedia.org/wiki/Chiffre_de_Vigenère).

Le développeur a malheureusement oublié les docstrings, les tests, et n'a pas suivi les recommandations d'écriture de la PEP 8.

```
import string
import itertools

def decale(lettre,dec):
    alph=string.ascii_uppercase
    new_index =(alph.index(lettre)+ dec)%26
    return alph[new_index]
```

```
def cesar(message, dec) :
    res = []
    for l in message: res.append(decale(l, dec))
    return "".join(res)
def Vigenere(message,passwd):
    res = []
    for lettre,decl in zip(message,
        itertools.cycle(passwd)):
        dec = string.ascii_uppercase.index(decl)
        res.append(decale(lettre , dec))
    return "".join(res)
```

1. Corriger le code afin qu'il suive les recommandations de la PEP 8.

2. En utilisant la fonction `help`, faire afficher la docstring du module `string`. Que vaut `string.ascii_uppercase` ?

3. En utilisant la fonction `help`, faire afficher la docstring de la fonction `cycle` du module `itertools`, et de la fonction `zip` si vous ne la connaissez pas.
Dans la fonction `vigenere`, expliquer ce que fait la ligne suivante :
`for lettre, decl in zip(message, itertools.cycle(passwd)):`

4. Écrire la dosctring du module de chiffrement et de chacune de ses 3 fonctions. Inclure un test représentatif, au format `doctest`, pour chacune des 3 fonctions.

5. Proposer des tests couvrants à l'aide d'assertions, pour les trois fonctions.

6. Le chiffrement Atbash consiste à remplacer A par Z, B par Y, C par X... (on inverse l'alphabet). Proposer une fonction qui permet de chiffrer avec cette méthode, en suivant la PEP 8, en intégrant une docstring (contenant un test), des annotations de fonction, ainsi que quelques assertions pour vérifier votre code.

▶▶▶ LA FEUILLE DE ROUTE

1. Suivre les recommandations d'écriture de la PEP 8 → FICHE 4
Vérifiez les noms de variables, les espaces autour des « : », « , » et des opérateurs.

2. Se documenter sur un module → FICHE 1
Pensez à importer le module avant d'essayer d'accéder à sa documentation.

3. Se documenter sur une fonction → FICHE 1
Vous pouvez demander directement la documentation d'une fonction particulière.

4. Réaliser la documentation d'une fonction → FICHES 1 et 3
La docstring doit être une chaîne de caractères placée en première ligne de la fonction, juste après la ligne qui commence par `def`.

5. Écrire des tests couvrants → FICHE 3
Pensez à tous les cas d'usage des fonctions, en particulier aux cas limites.

6. Écrire, documenter et tester une fonction → FICHES 1, 3 et 4
Inspirez-vous du code des fonctions déjà écrites pour calculer le chiffrement Atbash. Il peut être intéressant d'écrire 2 fonctions (comme pour `cesar`).