1 Écrire une assertion qui vérifie qu'un paramètre `n` est un entier strictement positif.

2 Quelle est l'erreur produite par le code suivant et comment la corriger ?

```
def f(x):
    assert isinstance(x, float)
    return x*x
f(2)
```

3 **a.** Qu'essaie de faire le code suivant et pourquoi est-il incorrect ?

```
def f(t):
    for i in range(len(t), 0, -1):
        print(t[i])
```

b. Trouver au moins deux façons différentes (et correctes) de coder cette fonction.

4 **a.** Le programme suivant produit-il une erreur ?

```
t = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
def zero(n):
    for i in range(n):
        t[i] = 0
    return t
zero(4)
```

b. Sinon, produit-il un effet de bord sur le tableau `t` ?

c. Quelle et la valeur de `t` à l'issue du programme ?

5 À quoi sert l'assertion à la fin du morceau de code suivant ?

```
    if alignement == 'gauche':
        ...
    elif alignement == 'centre':
        ...
    elif alignement == 'droit':
        ...
    assert True
```

6 Une année est bissextile si elle est divisible par 4, mais pas par 100, sauf si elle est divisible par 400.

a. Écrire deux fonctions qui indiquent si une année est bissextile de deux façons différentes : l'une avec une seule conditionnelle et l'autre avec une conditionnelle séparée pour chacun des cas.

b. Écrire un jeu de tests qui couvre tous les cas possibles et tester les deux fonctions.

7 Étant donnée une fonction `f` qui prend un nombre et retourne un nombre, la boucle suivante est-elle correcte ? Sinon, comment corriger l'erreur ?

```
x = 0.0
while x != 10.0:
    print(x, f(x))
    x += 0.1
```

8 **a.** Que se passe-t-il si l'on exécute le programme suivant ?

```
class C:
    def f(x):
        print(x)
c = C()
c.f(10)
```

b. S'il y a une erreur, comment la corriger ?

9 **a.** Utiliser `pytest` pour écrire un jeu de tests « boîte noire » d'une pile et tester les implémentations par tableau et par tuple vues au chapitre 1.

b. Toujours avec `pytest`, créer un jeu de tests « boîte blanche » pour l'implémentation par tableau de taille fixe.

10 **a.** Spécifier, coder et documenter un module Python avec les trois fonctions suivantes :

- `texte(mots)` retourne une chaîne de caractères formée de la concaténation des éléments du tableau `mots`, séparés par des espaces.
- `enum(mots)` est similaire à `texte` mais les mots sont séparés par des virgules, sauf les deux derniers qui sont séparés par « et ».
- `pluriel(n, mot, pl=None)` retourne la chaîne formée du nombre `n` et de `mot` ou sa forme plurielle selon la valeur de `n`. Si le paramètre optionnel `pl` n'est pas spécifié, la forme plurielle consiste à ajouter un « s » au mot, sauf s'il se termine par « s » ou « x ».

b. Écrire un jeu de tests pour ces fonctions dans un fichier séparé. Que valent `texte(['il fait', 10, 'degrés'])`, `enum([])` et `pluriel(-1, 'degré')` ?

c. Utiliser cette fonction dans l'activité d'introduction pour afficher les températures dans différentes villes sous la forme : « Il fait 10 degrés à Paris, 1 degré à Londres et 5 degrés à Berlin ».

11 TinyURL est un service qui crée des URLs courtes à partir d'URLs de longueur quelconque.

- Compléter le squelette de programme `tiny_url.py` pour écrire une fonction qui prend en paramètre une URL et retourne sa version courte, puis la tester.

12 Fonction mystère (1)

On considère la fonction suivante :

```
def mystere1(t):
    x = 0
    y = 0
    for z in t:
        x += z
        y += 1
    return x/y
```

a. Que fait cette fonction ? La réécrire en donnant des noms expressifs aux variables utilisées.

b. Écrire un test qui provoque une erreur d'exécution.

c. Corriger la fonction pour qu'elle passe ce test.

13 Fonction mystère (2)

On considère la fonction suivante :

```
def mystere2(x):
    y = 0
    for z in x:
        if z > y:
            y = z
    return y
```

a. Que fait cette fonction ? La réécrire en donnant des noms expressifs aux variables utilisées.

b. Écrire au moins un test qui produit un résultat erroné.

c. Corriger la fonction pour qu'elle passe ce test.

14 Objets et aliassage

On considère la classe suivante qui contient un attribut `dico` qui associe à chaque clé (par exemple une ville) une liste de valeurs (par exemple des températures). La méthode `ajouter` ajoute une valeur à la liste associée à une clé.

```
class C:
    def __init__(self):
        self.dico = {}
    def ajouter(self, cle, val):
        if cle not in self.dico:
            self.dico[cle] = []
        self.dico[cle].append(val)
        return self
    def copier(self):
        copie = C()
        copie.dico = self.dico
        return copie

c1 = C().ajouter('Paris',10).ajouter('Paris',30)
c2 = c1.copier()
c2.ajouter('Londres',20).ajouter('Paris',10)
```

a. Écrire un test qui échoue pour montrer que la méthode `copier` ne résout pas le problème d'aliassage entre `c1` et `c2`.

b. Corriger la méthode `copier` pour qu'elle passe le test.

15 Jours du mois

On souhaite implémenter la fonction `jours_mois(mois, annee)` qui retourne le nombre de jours du mois en fonction de l'année. On utilise la fonction `bissextile` de l'exercice 6 que l'on suppose correcte.

a. Écrire un jeu de tests exhaustif de cette fonction, c'est-à-dire testant tous les cas possibles. Combien de tests sont nécessaires ?

b. Implémenter la fonction en utilisant uniquement des conditionnelles (pas de tableau représentant le nombre de jours de chaque mois) et en vérifiant qu'elle passe les tests.

c. Écrire un jeu de tests « boîte blanche » qui teste chacune des conditions de la fonction. Combien de tests sont nécessaires ?

16 Partage de tableau

On souhaite définir la fonction `partage(t)` qui partage le tableau `t` en deux tableaux contenant respectivement les éléments de `t` de rang pair et de rang impair et retourne ces deux tableaux sous forme de tuple.

a. À partir de cette spécification, écrire un jeu de tests « boîte noire » de cette fonction.

b. Un programmeur a écrit l'implémentation suivante de cette fonction. Est-ce qu'elle passe tous les tests ? Si oui, écrire un test « boîte blanche » qui échoue.

```
def partage(t):
    assert isinstance(t, list)
    pairs = []
    impairs = []
    i = 0
    while i != len(t):
        pairs.append(t[i])
        impairs.append(t[i+1])
        i += 2
    return pairs, impairs
```

c. Corriger la fonction pour qu'elle passe tous les tests.

17 Test des files implémentées par une double pile

L'implémentation d'une file par une double pile nécessite une implémentation des piles. On considère dans un premier temps l'implémentation des piles par des tuples, validée par l'exercice 9.

a. Utiliser `pytest` pour écrire et exécuter un jeu de tests « boîte noire » de la file.

b. Sans changer l'implémentation de la file, remplacer l'implémentation des piles par une implémentation par tableaux de taille fixe et écrire un test « boîte blanche » de la file qui échoue à cause du débordement de pile.

c. Ajouter une fonction `file_pleine(f)` et modifier le jeu de tests en conséquence.

18 Test de stabilité d'un algorithme de tri

Un algorithme de tri est dit *stable* si deux éléments dont la clé de comparaison est identique sont dans le même ordre dans le tableau trié que dans le tableau initial.

Par exemple, si on trie des cartes selon leur valeur (en ignorant leur couleur), le tableau [(10,'pique'), (3,'trefle'), (10,'coeur'), (3,'carreau')] sera trié ainsi : [(3,'trefle'), (3,'carreau'), (10,'pique'), (10,'coeur')].

a. Écrire un jeu de tests « boîte noire » pour vérifier qu'un algorithme de tri est stable.

> ● NOTE
> On rappelle que la méthode Python `t.sort(key=f)` trie le tableau `t` en utilisant la fonction `f` pour obtenir la clé de tri des éléments de `t`. `f` prend un élément en paramètre et retourne la valeur à utiliser comme clé de tri. Ce tri est stable.

b. Tester si les tris vus en première (tri par insertion, tri par sélection) sont stables.

19 Pile typée

On veut s'assurer que les éléments d'une pile sont tous de même type.

a. Modifier l'une des implémentations des piles par une classe vue au chapitre 2 afin que l'empilement d'un élément respecte cette condition.

b. Écrire un ou plusieurs tests pour valider cette modification.

c. Que se passe-t-il si la pile, qui contenait par exemple des entiers, devient vide et que l'on empile ensuite des chaînes de caractères ? Écrire un test qui échoue dans cette situation.

d. Si nécessaire, modifier l'implémentation pour que la pile passe ce test.

20 Pile immuable

a. Écrire un test « boîte noire » pour savoir si une pile se comporte comme une structure de données immuable.

b. Montrer que l'implémentation par tuple satisfait ce test mais pas l'implémentation par tableau.

c. Modifier l'implémentation par tableau pour qu'elle passe ce test.

21 Extraire la documentation d'un module

Étant donné un module `m` importé par `import m`, `m.__dict__` contient un dictionnaire des noms définis dans ce module.

- Écrire un programme qui affiche la documentation du module `math`, puis la liste des fonctions définies dans ce module et leur documentation si elle existe. On omettra les noms qui commencent par un souligné (_).

22 Module d'affichage de graphes

a. À l'aide de la bibliothèque `matplotlib`, créer un module Python contenant deux fonctions `barres(t)` et `camembert(t)` qui affichent respectivement les données du tableau `t` sous forme de barres verticales et de camembert.

b. Ajouter à ce module une fonction `graphe(x, y)` qui affiche un graphe reliant les points (x_i, y_i) des tableaux `x` et `y`.

On utilisera la documentation (simplifiée) en français de `matplotlib` sur cette page Web : http://www.python-simple.com/python-matplotlib/pyplot.php, en particulier les rubriques « Barplot », « Pie » et « Lineplot ».

23 Utilisation de bibliothèques

a. À l'aide de la documentation en français de la bibliothèque Python `random` sur le site http://www.python-simple.com/python-modules-math/random.php, choisir les fonctions aléatoires les plus adaptées pour implémenter les deux fonctions suivantes :

- `pierre_papier_ciseaux()` : réalise un tirage aléatoire et désigne le gagnant selon les règles du jeu bien connu ;
- `poker()` : tire au hasard 5 cartes à jouer et les affiche.

On utilisera le module de l'exercice 10 pour l'affichage des résultats.

b. Trouver dans le site http://www.python-simple.com une fonction ou une méthode permettant de compter le nombre d'occurrences d'un élément dans un tableau. Utiliser celle-ci pour compléter la fonction `poker` et afficher les paires, brelans et carrés éventuels.

c. Utiliser le module de l'exercice 22 pour afficher les résultats d'une série de parties de pierre-papier-ciseaux sous forme de camembert représentant les proportions de chaque résultat possible d'une partie. Faire de même pour afficher les résultats d'une série de tirages de mains de poker sous forme de barres représentant le nombre de paires, de brelans et de carrés.

24 Recherche de chaîne

On veut définir une fonction `sous_chaine(s, ch)` qui retourne la position de la chaîne `s` dans la chaîne `ch` ou –1 si `s` n'apparait pas dans `ch`.

a. Écrire un jeu de tests « boîte noire » de cette fonction sur la base de sa spécification.

b. Implémenter la fonction de façon qu'elle passe ces tests.

c. Télécharger le fichier `test_recherche.py` et ajouter votre fonction au début de celui-ci. Exécuter ce fichier avec `pytest` et corriger le code de la fonction si elle ne passe pas tous les tests.

d. On veut maintenant implémenter et tester la fonction `derniere_sous_chaine(s, ch)` qui retourne la position de la *dernière* occurrence de `s` dans `ch`, ou –1 si elle est absente. Écrire un jeu de tests en s'inspirant des tests des questions **a** et **c**. Télécharger le fichier `recherche.py` qui contient une implémentation de cette fonction, la tester et la corriger si besoin.

e. Ajouter une variable globale qui compte le nombre de fois que l'on compare deux caractères. Écrire une autre implémentation qui passe les tests mais nécessite moins de comparaisons.

25 Boucle infinie ?

a. Peut-on être sûr que la boucle suivante n'est pas une boucle infinie ?

```
while n != 1:
    if n % 2 == 0:
        n = n // 2
    else:
        n = (3*n + 1) // 2
```

b. Sinon, comment peut-on modifier le code pour interrompre l'exécution si la boucle dépasse un nombre d'exécutions donné ?

26 Module d'affichage de graphes (suite)

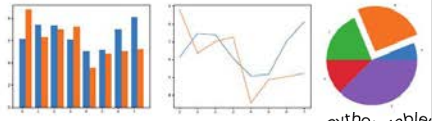
L'exercice 22 définit une API simplifiée d'affichage de trois types de graphes. On souhaite redéfinir cette API sous forme de trois classes d'objets, une pour chaque type de graphe, avec des méthodes pour ajouter des données au graphe, spécifier certains paramètres (comme les couleurs ou la légende) et afficher le graphe.

On utilisera la documentation de cette page web : http://www.python-simple.com/python-matplotlib/pyplot.php, en particulier les rubriques « Barplot », « Pie » et « Lineplot ».

a. Spécifier une API constituée de trois classes d'objets `Pie`, `Barplot` et `Lineplot`, avec des méthodes pour ajouter des données au graphe, spécifier certains paramètres d'affichage et afficher le graphe.

En particulier, `Lineplot` et `Barplot` devront permettre d'afficher plusieurs séries de données sur un seul graphe et `Pie` devra permettre de décaler des secteurs du camembert pour les faire ressortir.

b. Implémenter, documenter et tester ces trois classes.



27 Modulariser un programme

Télécharger le programme `program.py` et le fichier `covid.csv` et exécuter le programme, qui doit afficher plusieurs graphes des données.

a. Étudier le code du programme et le réorganiser en trois modules et un programme principal, de façon que ces modules puissent être utilisés par d'autres programmes. Le premier module s'occupe de la lecture des données et leur mise en forme dans une structure de données Python. Le second module permet de calculer des statistiques sur les données et de les lisser. Enfin, le troisième module permet d'afficher des données. C'est le module de l'exercice 26 (ou à défaut celui de l'exercice 22).

b. Télécharger le fichier `demographie.csv` et modifier le programme principal pour afficher ces données. S'il est nécessaire de modifier les modules, il faut qu'ils puissent continuer d'être utilisés par le programme de la question **a**.

28 Masquage de noms importés

Télécharger les fichiers `m.py` et `p.py` et exécuter le programme `p.py`.

a. Comment expliquer que la fonction `g` appelée par `f` et par `h` n'est pas la même lorsque l'on redéfinit `g` ?

b. Cette situation se produirait-elle si on avait importé le module (`import m`) au lieu des fonctions (`from m import f, g`) ?

29 Importation croisée de modules

Télécharger les fichiers `m1.py` et `m2.py` qui définissent chacun deux fonctions et le programme `prog.py` qui importe le module `m1` et appelle la fonction `f1`.

a. Exécuter le programme et identifier la cause de l'erreur.

b. Remplacer les instructions de la forme `from m import f` par `import m` et qualifier les appels de fonctions importées en conséquence. Exécuter le programme. Comment expliquer ce que l'on observe ?