Exercices de Python

Stéphane Nicolet
Université Paris 2 – Panthéon-Assas
stephane.nicolet@assas-universite.fr

Table des matières

Chapitre 1 : Variables, types de base, fonctions d'entrée et de sortie

Exercice 1 (les nombres : types int et float)**.**

Initialisez trois variables `x`, `y` et `z` à des nombre entiers ou flottants, puis familiarisez-vous avec les opérateurs arithmétiques de base en les appliquant sur ces variables. On rappelle que :

- l'addition se note `+`
- la soustraction se note `-`
- la multiplication se note `*`
- l'exponentiation se note `**`
- le modulo se note `%`
- la division se note `/`

En Python 2 (contrairement à Python 3), la division par défaut est la division entière : cela signifie que si les deux nombres sont des entiers, c'est la division entière qui est appliquée. Si au moins un des deux nombres est réel, c'est la division réelle qui est appliquée.

Exercice 2 (les chaînes de caractères : type string)**.**

Initialisez deux variables `s` et `t` à des chaînes de caractères, puis familiarisez-vous avec les opérateurs de base sur les chaînes en les appliquant sur ces variables. On rappelle que :

- la concaténation de chaînes de caractères s'obtient par l'opérateur `+`
- la répétition de chaînes de caractères s'obtient par l'opérateur `*`
- l'extraction de l'élément d'indice `n` d'une chaîne `s` s'obtient par la syntaxe `s[n]` (les indices commencent à 0)
- l'extraction de l'élément d'indice `n`, mais en partant de la fin de la chaîne `s`, s'obtient par la syntaxe `s[-n]` (dans ce cas, les indices commencent à -1)
- l'extraction de la sous-chaîne de `s` d'indices `m` compris jusqu'à l'indice `n` non compris s'obtient par la syntaxe `s[m:n]`
- la longueur d'une chaîne `s` s'obtient par la syntaxe `len(s)`

Pour la syntaxe `s[m:n]`, si l'indice `m` n'est pas spécifié, il vaut 0 par défaut ; si l'indice `n` n'est pas spécifié, il vaut `len(n)` par défaut.

Exercice 3 (les Booléens : type bool)**.**

En Python, les valeurs booléennes se disent `True` et `False`. Les opérateurs logiques sont les suivants :

- la négation se note `not`
- le "et" logique, appelé aussi disjonction, se note `and`
- le "ou" logique, appelé aussi conjonction, se note `or`

Initialisez quelques variables aux valeurs Booléennes `True` et `False` et testez les opérateurs logiques sur vos variables en tapant quelques expressions logiques de votre choix.

De plus, les différents opérateurs de comparaison entre nombres (ou autre types également) se note `<`, `<=`, `>`, `>=`, `==`, `!=`. Les résultat des évaluations d'expressions contenant ces opérateurs sont des valeurs Booléennes, `True` ou `False`. Par exemple, l'expression `2 < 3` est évaluée en `True` et l'expression `2 == 3` est évaluée en `False`. Tapez quelques expressions de votre choix contentant ces opérateurs de comparaison.

> **Point important**
>
> Faîtes attention de ne pas confondre le symbole d'égalité "=", qui sert à instancier une variable (par exemple `x = 2`), de celui de double égalité "==", qui sert à tester l'égalité entre deux opérands (par exemple `2 == 3`).

Exercice 4 (Chaînes de caractères)**.**

4.1. Initialisez deux variables appelées `prenom` et `nom` contenant votre prénom et votre nom, respectivement.

4.2. En utilisant l'opérateur de concaténation sur les chaînes de caractères (`+`), initialisez une variable `nom_complet` contenant votre prénom et votre nom séparés par un espace.

4.3. En utilisant l'opérateur de multiplication sur les chaînes de caractères (`*`) et la fonction `print`, imprimez vos nom et prénom une centaine fois à l'écran séparés par " * " (espace - étoile - espace).

4.4. En utilisant les opérateurs d'extraction de sous-chaînes, initialisez à partir de la variable `nom_complet` une autre variable `initiales` contenant vos initiales séparées par un espace.

4.5. Imprimez vos initiales une centaine fois à l'écran séparés par " * " (espace - étoile - espace).

Exercice 5 (Chaînes de caractères)**.**

En utilisant la fonction `input()`, écrivez un script qui demande à l'utilisateur d'entrer son prénom, puis d'entrer son nom, puis affiche automatiquement le prénom et le nom, les initiales et la première lette du nom de famille, comme illustré ci-dessous :

```
>>> Quel est votre prenom? Astor
>>> Quel est votre nom? Piazzolla
Astor Piazzolla
A. P.
Premiere lettre du nom de famille: P
```

Exercice 6 (int et float)**.**

Créez un script permettant de convertir des températures de degrés Celsius en degrés Fahrenheit et vice-versa. La formule de conversion est la suivante :

$$F = \frac{9}{5} \cdot C + 32$$

Votre script devra demander à l'utilisateur la température et afficher les deux conversions comme suit :

```
Quelle est la temperature? 20.2
20.2 C = 68.36 F
20.2 F = -6.555555555 C
```

Exercice 7 (Booléens)**.**

Créez un programme qui, premièrement, demande à l'utilisateur d'entrer des valeurs de vérité (`True` ou `False`) pour trois variables propositionnelle A, B et C, puis, deuxièmement, demande à l'utilisateur d'entrer "en français" une expression de logique Booléenne sur ces trois variables propositionnelles, et enfin, affiche l'évaluation de cette expression en fonction des valeurs de vérité de A, B et C. Le déroulement du programme devra ressembler à l'exécution ci-dessous.

```
Entrer une valeur de verite pour A: True
Entrer une valeur de verite pour B: False
Entrer une valeur de verite pour C: True
Entrer une expression Boolenne (exprimee avec des "non", "ou", "et" et des
    parentheses):
non (A et B) ou (C et non B)
L'evaluation de votre expression Booleenne est: True
```

Indication

Si `s` est une chaîne de caractères, la fonction `s.replace(old, new)` retourne une autre chaîne de caractères où toutes les occurrences de la sous-chaîne `old` dans `s` sont remplacées par la la sous-chaîne `new`. Ainsi, il vous faudra remplacer les sous-chaînes `"non"`, `"et"` et `"ou"` par `"not"`, `"and"` et `"or"`, puis évaluer votre expression Booléenne à l'aide de la fonction `bool()` pour obtenir le résultat désiré.

Chapitre 2 : Types de données

Rappel sur les chaînes de caractères

Les chaînes de caractères sont données entre guillemets. Par exemple : `s = "bonjour"`.

Comme déjà mentionné, l'accès aux éléments et aux sous-chaînes d'une chaîne `s` s'effectue via les instructions `s[n]`, `s[-n]`, `s[m:n]`, `s[m:]` et `s[:n]`.

Les chaînes de caractères sont des structures de données non modifiables. Cela signifie que nous ne pouvons pas procéder à des réaffectation de leurs éléments :

```
>>> s =  "abba"
>>> s[2] = "z"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

Le parcours d'une chaîne de caractères s'effectue via les instructions :

```
for variable in chaine:
  instructions...
```

Il existe diverses méthodes sur les chaînes de caractères, nous en aborderons quelques unes.

Rappel sur les tests conditionnels `if... else...`

En Python, un tel test conditionnel `if... else...` est obtenu via la syntaxe suivante (les blocs `elif` et `else` sont facultatifs) :

```
if condition:
  instructions...
elif:
  instructions...
...
else:
  instructions...
```

Exercice 1 (chaînes de caractères)**.**

1.1. Initialisez une variables `c` contenant la chaîne de caractères contenant des chiffres et des lettres, du type `"X44bf38j23jdjgfjh737nei47"`.

1.2. Écrivez un programme qui construit deux chaînes de caractères `c_alpha` et `c_num` telles que `c_alpha` représente la suite des lettres de `c` et `c_num` représente la suite des chiffres de `c`. Effectuez un parcours de votre chaîne et utilisez un test conditionnel ainsi que les méthodes `str.isalpha()` et `str.isdigit()` qui permettent de déterminer si la chaîne `str` est composée de caractères alphabétiques ou numériques.

1.3. Écrivez un programme qui détermine si la sous-chaîne `j23` apparaît dans votre chaîne, et, si c'est le cas, qui la remplace par `j24`. La chaîne de départ `c` devra donc être modifiée dans ce cas. Utilisez un test conditionnel ainsi que les méthodes `str.find()` et `str.replace()` qui permettent de retrouver et remplacer des sous-chaînes dans la chaîne `str`.

1.4. Écrivez un programme qui détermine si la sous-chaîne `f37` apparaît dans votre chaîne, mais pas forcément de manière consécutive. En utilisant la méthode `str.find()`, recherchez le premier indice d'apparition du caractère `f`, puis celui de `3`, puis celui de `7` et vérifiez que ces indices forment bien une suite croissante.

Exercice 2 (chaînes de caractères)**.**

2.1. Initialisez une variables `texte` contenant la chaîne de caractères suivante :
`"We introduce here the Python language"`

(a) Créez un script qui compte le nombre de caractères de cette chaîne. Vous initialiserez une variable "compteur" à 0, puis, à chaque caractère parcouru, incrémenterez cette variable. Vérifiez que votre résultat corresponde bien à celui de l'instruction `len(texte)`.

(b) Créez un autre script qui compte le nombre de caractères de cette chaîne qui ne sont pas des espaces. Pensez à utiliser un test conditionnel dont la syntaxe est décrite ci-dessus.

(c) Sachant que dans cette chaîne de caractères, les mots sont séparés par des espaces, créez un script qui compte le nombre de mots de cette chaîne. Pensez encore une fois à utiliser un test conditionnel.

2.2. Initialisez une variables `texte2` contenant le texte suivant :

```
"We introduce here the Python language. To learn more about the language,
  consider going through the excellent tutorial https://docs.python.org/
    tutorial. Dedicated books are also available, such as http://www.
                          diveintopython.net/."
```

Dans cette chaîne de caractères, les mots ne sont plus uniquement séparés par des espaces, mais également par des symboles de ponctuation. Testez votre script qui compte le nombre de mots sur cette chaîne de caractères. Votre script est-il toujours valable ? Si oui, pourquoi ? Sinon, comment devez-vous le modifier ? On aimerait que chaque adresse web ne soit comptée que comme un seul mot.

Rappel sur les listes

Les listes sont données entre crochets, les éléments étant séparés par des virgules. Par exemple : `ma_liste = [1, 2, 3]`

L'accès aux éléments et aux sous-listes d'une liste `l` s'effectue par la même syntaxe que pour les chaînes : on utilise les instructions `l[n]`, `l[-n]`, `l[m:n]`, `l[m:]` et `l[:n]`.

Contrairement aux chaînes, les listes sont des structures de données modifiables. Cela signifie que nous pouvons procéder à des réaffectation de leurs éléments :

```
>>> l =  [2, "abba", 3.7, True]
>>> l[2] = 101
>>> print l
[2, 'abba', 101, True]
```

Le parcours d'une liste s'effectue via les instructions :

```
for variable in liste:
  instructions...
```

Il existe diverses méthodes sur les listes, nous en aborderons quelques unes.

Exercice 3 (Listes).

3.1. Créez un programme qui demande à l'utilisateur d'entrer trois mots à la suite, puis renvoie les trois mots triés par ordre alphabétique. Utilisez une liste pour stocker les trois mots. Construisez et triez la liste grâce aux méthodes `append()` et `sort()`, respectivement.

3.2. Modifiez votre programme de manière à ce que l'utilisateur puisse entrer autant de mots qu'il le souhaite. Le processus de saisie s'arrête lorsque l'utilisateur entre le mot "FIN". Utilisez une boucle `while`.

Exercice 4 (Listes).

On considère les deux listes suivantes :

```
couleurs = ["Pique", "Trefle", "Carreaux", "Coeur"]
valeurs = [2, 3, 4, 5, 6, 7, 8, 9, 10, "valet", "dame", "roi", "as"]
```

4.1. À partir de ces deux listes, générer une liste `cartes` contenant toutes les 52 cartes sous forme de chaînes de caractères. Utilisez un double parcours de ces listes, c'est-à-dire une double boucle for, ainsi que la méthode `list.append()`.

4.2. Importez la fonction `shuffle` de la librairie `random` grâce à l'instructions `from random import shuffle`. Cette fonction `shuffle()` permet de mélanger une liste. Mélangez alors la liste de vos cartes.

4.3. Créez ensuite quatre listes `joueur1`, `joueur2`, `joueur3` et `joueur4` qui correspondent à la distribution votre jeu de cartes mélangé à quatre joueurs différents. Les cartes doivent être distribuées une par une et à tour de rôle. Utilisez un compteur qui compte modulo 4 pour simuler la distribution aux joueurs à tour de rôle. Utilisez un test conditionnel avec des conditions "elif".

Exercice 5 (Listes, fichiers et listes de listes).

Le fichier `diamonds.csv` contient des données d'environ 54000 diamants. Nous allons récolter ces données sous forme de listes afin de pouvoir les manipuler.

5.1. Copiez le fichier `diamonds.csv` dans le répertoire dans lequel se trouve votre script actuel. En Python, il est possible de lire ce fichier et de créer une liste des lignes de ce fichier. Pour cela, utilisez instructions suivantes :

```
with open("diamonds.csv", "r") as f:
  diamants = f.readlines()
```

La liste créée s'appelle `diamants`. Les éléments de cette liste sont des chaînes de caractères qui correspondent aux lignes du fichier. Quelle est la longueur de cette liste ? Afficher les premier, deuxième et troisième éléments de cette liste. Remarquer que le premier élément de la liste correspond aux variables mesurées sur les diamants. Les autres lignes correspondent aux données proprement dites.

5.2. On aimerait maintenant que chaque élément de la liste soit une liste plutôt qu'une chaîne de caractères. Pour cela, on utilise la méthode `split()` qui permet de couper une chaîne en une liste d'éléments. Cette méthode est illustrée ci-dessous :

```
>>> diamants[2]               # chaine de caracteres
'0.21,"Premium","E",59.8,326,3.89,3.84,2.31\n'
>>> diamants[2].split(",")    # chaine transformee en liste
['0.21', '"Premium"', '"E"', '59.8', '326', '3.89', '3.84', '2.31\n']
```

Testez cette commande. Écrivez un programme qui applique la méthode `split()` à touts les éléments de votre liste `diamants`. Utilisez un parcours de la liste `diamants` et un compteur qui rerpésente le numéro de ligne courant.

5.3. Créez une liste `diamants_100` qui contient les 100 éléments de la liste `diamants` qui succèdent au tout premier élément. Affichez les 20 premiers éléments de `diamants_100`.

5.4. Créez une liste `diamants_prix` qui contient les prix des 54000 diamants convertis en nombres réels (type `float`). Utilisez un parcours de la liste `diamants_prix`. Affichez les 20 premiers éléments de `diamants_100`.

.

Rappel sur les tuples

Les tuples sont données entre parenthèses, les éléments étant séparés par des virgules. Par exemple : `ma_tuple = (1, 2, 3)`

L'accès aux éléments et aux sous-tuples d'un tuple `t` s'effectue par la même syntaxe que pour les listes : on utilise les instructions `t[n]`, `t[-n]`, `t[m:n]`, `t[m:]` et `t[:n]`.

Contrairement aux listes, les tuples sont des structures de données non modifiables. Cela signifie que nous ne pouvons pas procéder à des réaffectation de leurs éléments, ni modifiez leur logueur, etc. :

```
>>> t = (1, 2, 3, "good", True)
>>> t[3] = "bad"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> t.append(28.45)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'append'
```

Le parcours d'un tuple s'effectue via les instructions :

```
for variable in tuple:
  instructions...
```

Exercice 6 (Listes de tuples)**.**

6.1. Créez un programme qui demande à l'utilisateur d'entrer le prénom, le nom et le numéro de matricule d'un étudiant, puis stocke ces informations dans un tuple à trois éléments. Affichez ce tuple à l'écran. L'intérêt d'utiliser un tuple dans ce cas réside dans le fait de ne pas pouvoir modifier les informations d'un étudiant, ce qui est plus sécurisé.

6.2. Modifiez votre programme de manière à ce que l'utilisateur puisse entrer autant de saisies qu'il le souhaite. Le processus de saisie s'arrête lorsque l'utilisateur entre le mot "FIN". La liste des étudiants saisis sera stockée dans une liste de tuples. Utilisez une boucle `while`.

6.3. Affichez de manière conviviale tous les étudiants de votre liste construite au point précédent. Utilisez un parcours de liste.

Rappel sur les dictionnaires

Les dictionnaires sont des couples d'éléments "clé-valeur" donnés entre accolades ; les clés sont séparées de leurs valeurs correspondantes par des double points, et les couples "clé-valeur" sont séparés entre eux par des virgules. Par exemple :

```
mon_dico = {"FR" : 643801, "DE" : 357168, "GB" : 229848}
```

L'accès à la valeur correspondante à la clé `key` d'un dictionnaire `d` s'effectue via l'instruction `d[key]`. L'affectation d'une valeur `value` à une clé `key` d'un dictionnaire `d` s'effectue via l'instruction `d[key] = value`.

Tout comme les listes, les dictionnaires sont des structures de données modifiables. Cela signifie que nous pouvons procéder à des réaffectation de leurs éléments :

```
>>> mon_dico = {"FR" : 643801, "DE" : 357168, "GB" : 229848}
>>> mon_dico["DE"] = 257200
>>> mon_dico
{'DE': 257200, 'GB': 229848, 'FR': 643801}
>>> mon_dico["BE"] = 30528
>>> mon_dico
{'DE': 257200, 'GB': 229848, 'BE': 30528, 'FR': 643801}
>>> del mon_dico["FR"]
>>> mon_dico
{'DE': 257200, 'GB': 229848, 'BE': 30528}
```

Le parcours des clés, des valeurs, ou des couples "clé-valeur" d'un dictionnaire s'effectue via les quatre types d'instructions suivantes :

```
# par defaut, le parcours d'un dico correspond parcours de ses cles
# dans le cas ci-dessous, "variable" prend les valeurs de cles du dico
for variable in dico:
  instructions...
# la syntaxe suivante est plus precise mais equivalente
for variable in dico.keys():
  instructions...
# pour le parcours des valeurs d'un dico
for variable in dico.values():
  instructions...
# pour le parcours des couples "cle-valeur" d'un dico
for variable_cle, variable_valeur in dico.items():
  instructions...
```

Il existe diverses méthodes sur les dictionnaires, nous en aborderons quelques unes.

Exercice 7 (Dictionnaires)**.**

Cet exercice correspond à une généralisation de l'exercice 6 dans le cas des dictionnaires.

7.1. Créez un dictionnaire qui contient comme clés un certain nombres de mots en français et comme valeurs la traduction de ces mots en anglais. Ce dictionnaire fait donc office de traducteur français-anglais.

7.2. Ajouter à votre dictionnaire le couple clé-valeur `"cerveau" : "brain"`.

7.3. Recherchez si votre dictionnaire contient la traduction du mot "cerveau", et si tel est, le cas, affichez sa traduction anglaise. Effectuez un parcours par clés de votre dictionnaire.

7.4. Créez un nouveau dictionnaire dont les clés et valeurs correspondent aux valeurs et aux clés de votre dictionnaire précédent. Ainsi, votre nouveau dictionnaire fera office de traducteur

anglais-français au lieu de français-anglais. Pour construire ce nouveau dictionnaire, utilisez un parcours clé-valeur de votre dictionnaire initial.

7.5. Recherchez si votre dictionnaire contient la traduction du mot “brain”.

7.6. Recherchez si votre dictionnaire contient la valeur “cerveau” et si tel est le cas, afficher sa clé correspondante. Effectuez un parcours clés-valeurs de votre dictionnaire.

7.7. Modifiez votre dictionnaire de départ de sorte que les valeurs ne soient plus des simples mots anglais, mais des listes de mots correspondant à autant de traductions possibles de vos clés.

7.8. Ajouter à votre nouveaux dictionnaire le couple clé-valeur `"chemin" : ["path", "way"]`.

7.9. Recherchez la deuxième traduction du mot “chemin”.

7.10. Effacez la clé “chemin” et sa valeur correspondante de votre dictionnaire (fonction `del`).

Exercice 8 (Dictionnaire de tuples)**.**

Cet exercice correspond à une généralisation de l’exercice 6 dans le cas des dictionnaires.

8.1. Créez un programme qui demande à l’utilisateur d’entrer le prénom, le nom et le numfo de matricule d’un étudiant, puis stocke ces informations dans un dictionnaire. Les clés du dictionnaires correspondront aux noms des étudiants et ses valeurs seront des tuples à trois éléments (prénom, nom, matricule). Affichez ce dictionnaire à l’écran. L’intérêt d’utiliser un dictionnaire dans ce cas réside dans le fait de pouvoir accéder aux informations des étudiants à leur noms.

8.2. Modifiez votre programme de manière à ce que l’utilisateur puisse entrer autant de saisies qu’il le souhaite. Le processus de saisie s’arrête lorsque l’utilisateur entre le mot “FIN”. Le dictionnaire des étudiants saisis sera stockée dans une liste de tuples. Utilisez une boucle `while`.

8.3. Affichez de manière conviviale tous les étudiants de votre dictionnaire construit au point précédent. Utilisez un parcours par clés de votre dictionnaire.

8.4. En utilisant la syntaxe `in` ou la méthode `has_key` [1], déterminer si un étudiant du nom de “Obama” appartient à votre dictionnaire, et, si tel est le cas, renvoyez les informations cet létudiant.

8.5. Créez un script qui détermine si le numéro de matricule 12345678 existe dans votre dictionnaire, et si tel est le cas, renvoyez les informations de létudiant correspondant. Utilisez un parcours “clé-valeur” de votre dictionnaire.

8.6. Question subsidiaire. Dans l’état actuel de votre programme, il est impossible d’enregistrer deux étudiants qui portent le même nom. Pourquoi cela ? Modifier votre programme du point 2 de manière à pouvoir enregistrer des étudiants portant le même nom. Pour cela, les clés de votre dico ne devront plus correspondre uniquement au nom des étudiants : mais attention, les clés ne peuvent pas être des listes, tuples, etc.

1. Cette méthode ne fonctionne plus en Python 3.

Chapitre 3 : Expressions arithmétiques, entrées/sorties

Le concile de Nicée

La définition de la date de Pâques a été fixée en 325 lors du concile de Nicée : Pâques est le dimanche qui suit le quatorzième jour de la lune qui atteint cet âge au 21 mars ou immédiatement après.
Autrement dit, c'est le premier dimanche qui suit ou qui coïncide avec la première pleine lune apres le 21 mars (marquant le début du printemps).

Algorithme de Oudin pour calculer la date de Pâques

Calculer la date de Pâques est loin d'être une chose facile. On connait plusieurs méthodes, dont la suivante due à Oudin qui a l'avantage de demander peu d'opérations. Quoiqu'il existe une version généraliste de l'agorithme de Oudin (sans limite de siècle), nous présentons ici une forme simplifiée uniquement valable pour le calendrier grégorien, donc pour **toute année postérieure à 1583**.

Dans l'algorithme ci-dessous, chaque ligne introduit une nouvelle variable qui dépend des précédentes. Les divisions doivent toujours être entières. L'exemple en bleu indique les calculs pour l'année 2007.

1. G représente l'écart d'or diminué de 1 : diviser l'année par 19, en prendre le reste ; (2007/19 = 105, or 105 × 19 = 1995 et il nous faut 2007, donc l'écart vaut G = 12)
2. C et C4 permettent le suivi des années bissextiles : diviser l'année par 100 puis encore par 4 ; (2007/100 = C = 20 et 20/4 = C4 = 5)
3. E : diviser 8×C +13 par 25 sans les décimales ; (8 × 20 + 13 = 173/25 = E = 6)
4. H qui dépend de l'épacte : diviser 19 × G + C - C4 - E + 15 par 30, en prendre le reste ; (on prend le reste d'une division selon le même principe que pour G : 252/30 = 8, or 8 × 30 = 240 et il nous faut 252, donc l'écart vaut H = 12)
5. K : diviser H par 28 ; (12/28 = K = 0)
6. P : diviser 29 par H + 1 ; (29/13 = P = 2)
7. Q : diviser 21-G par 11 ; (21 - 12 = 9/11 = Q = 0)
8. I représente le nombre de jours entre la pleine lune pascale et le 21 mars : (-1 + K × P × Q) × K + H ; (0 × 2 × 0 - 1 = -1 × -0 = 0 + 12 = I = 12)
9. B : diviser l'année par 4 et enlever les décimales, y ajouter l'année ; (2007/4 = 501 + 2007 = 2508)
10. J1 : Additionner B + I + 2 + C4 et retrancher C ; (J1 =2507)
11. J2 calcule le jour de la lune pascale (0=dimanche 1=lundi...6=samedi) : diviser J1 par 7 et en prendre le reste ; (on calcule toujours le reste d'une division selon le même principe qu'avec G et H, le résultat est J2 = 1)
12. R = 28 + I - J2. C'est le résultat final, enfin ! (R = 39)
13. R représente la date de Pâques dans le mois de mars, s'il dépasse 31 cela signifie que l'on déborde sur avril (... 30 correspond au 30 mars, 31 au 31 mars, 32 au 1er avril, 33 au 2 avril, ...). Retrancher 31 le cas échéant pour obtenir la date d'avril. (Pâques 2007 tombe donc le 8 avril)

Exercice 1 (Date de Pâques)**.**

1.1. Calculez la date de Pâques pour l'année 2007 en suivant la méthode de Oudin.

1.2. Debugguez au fur et à mesure les valeurs des variables avec la fonction `print` pour vérifier votre implémentation de l'algorithme à l'aide de l'exemple bleu.

```
G = ...
C = ...
C4 = ...
print( f"G = {G}" )
print( f"C = {C}" )
print( f"C4 = {C4}" )
```

Exercice 2 (Utilisation de fonctions et de fichiers)**.**

2.1. Modifiez le script pour mettre le calcul de Pâques dans une fonction `date_de_paques(N)`. Votre fonction doit renvoyer une chaine de caractères, par exemple `"Dimanche 23 mars 2008"`

2.2. Réalisez un programme qui demande une année à l'utilisateur dans le programme principal, puis affiche la date de Pâques pour cette année.

2.3. Quelle sera la prochaine année (à partir de 2025) où Pâques sera fêté un 1er avril ?

2.4. Réalisez un programme qui affiche toutes les années du XXIe siècle durant lesquelles Pâques est fêté une 1er avril.

2.5. Déterminez les années du XXIe siècle pour lesquelles Pâques est fêté le plus tard.

2.6. Écrivez toutes les dates de Pâques du XXIe siècle dans un fichier texte.

```
with open("toto.txt", "w") as f:
    f.write("I have written something in the file !!!")
```

2.7. Déterminez le jour de votre anniversaire (lundi, mardi, mercredi, etc) pour une année N donnée (indication : Pâques est un dimanche).

Chapitre 4 : Boucles et tests conditionnels

Rappel sur les boucles `for`

La fonction `range(start,stop,[step])` produit un objet qui retourne une liste de nombres entiers telle que : la première valeur est `start` (`start` vaut 0 par défaut) ; les valeurs suivantes sont incrémentées par pas de `step` (`step` vaut 1 par défaut) ; et ce jusqu'à atteindre la valeur `stop` non incluse.

Une boucle `for` signifie "pour `var` allant de `start` à `stop` par pas de `step`, effectuer les instructions...". La syntaxe d'une boucle `for` utilise la fonction `range()` et se présente comme suit :

```
for var in range(start,stop[,step]):
  instructions...
```

On peut forcer l'interruption d'une boucle `for` avec l'instruction `break`.

On rappelle également la syntaxe pour parcourir les chaînes de caractères, les listes, les tuples et les dictionnaires à l'aide de boucles `for`.

```
# parcours de chaines, listes ou tuples
for variable in chaine_liste_ou_tuple:
  instructions...
# parcours des cles d'un dico
for variable in dico.keys(): # (ou for variable in dico:)
  instructions...
# parcours des valeurs d'un dico
for variable in dico.values():
  instructions...
# parcours des couples "cle-valeur" d'un dico
for variable_cle, variable_valeur in dico.items():
  instructions...
```

Rappel sur les boucles `while`

Une boucle `while` signifie "tant que la condition est vraie, effectuer les instructions...". La syntaxe d'une boucle `while` se présente comme suit :

```
while condition:
  instructions...
```

On peut forcer l'interruption d'une boucle `while` avec l'instruction `break`.

Rappel sur les tests conditionnels `if... else...`

On rappelle la syntaxe d'un test conditionnel `if... else...` (les blocs `elif` et `else` sont facultatifs) :

```
if condition:
  instructions...
elif:
  instructions...
...
else:
  instructions...
```

Exercice 1 (Boucles “for”)**.**

1.1. Écrivez un script qui demande à l'utilisateur d'entrer un entier N et affiche ensuite la table de multiplication de N.

1.2. Modifiez votre script de sorte que la table de multiplication soit affichée sur une seule ligne. Il suffit d'ajouter une virgule après votre instruction `print`.

1.3. Modifiez votre script de telle manière qu'il affiche, line par ligne, les tables de multiplications de tous les entiers plus petit ou égaux à N. Utilisez une double boucle `for`.

1.4. Écrivez un script qui demande à l'utilisateur d'entrer un entier N et affiche ensuite un petit dessin comme ci-dessous de hauteur N.

```
# exemple pour N = 4
*
**
***
****
```

1.5. Essayez de modifier votre script de telle sorte que le dessin affiché soit comme ci-dessous. Déterminez règle qui donne le nombre d'espaces et d'étoiles de chaque ligne.

```
# exemple pour N = 5
    *
   * *
  * * *
 * * * *
* * * * *
```

Exercice 2 (Boucles “for”)**.**

On considère les deux listes suivantes :

```
jours = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
mois = ["January", "February", "March", "April", "May", "June", "July", "August
", "September", "October", "November", "December"]
```

2.1. Écrivez un script qui crée la liste de 12 tuples suivante :

```
mj = [("January", 31), ("February", 28), ("March", 31), ("April", 30), ("
May", 31), ("June", 30), ("July", 31), ("August", 31), ("September", 30),
("October", 31), ("November", 30), ("December", 31)]
```

2.2. Écrivez un script qui crée une liste de 365 chaînes de caractères correspondant aux 365 jours de l'année, comme ci-dessous. Utilisez une double boucle `for` sur votre liste `mj`.

```
annee = ["1 January", "2 January",..., "1 February", "2 February",...]
```

2.3. Initialisez la liste suivante :

```
jours_semaine = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "
Saturday", "Sunday"]
```

Supposons que le premier janvier soit un lundi. Écrivez un script qui crée une deuxième liste des 365 jours de l'annés comprenant les jours de la semaine en plus, comme ci-dessous. Effectuez une double boucle de taille 365, utilisez les listes `annee` et `jours_semaine` et pensez à utiliser l'opérateur modulo sur les indices de la liste `jours_semaine`.

```
annee2 = ["Monday 1 January", "Tuesday 2 January",...]
```

2.4. À partir de vos listes `annee` et `jours_semaine` , créez un dictionnaire dont les clés sont les éléments de `annee` et les valeurs sont les jours de la semaine correspondants, comme ci dessous.

```
dico_annee = ["1 January" : "Monday", "2 January" : "Tuesday",...]
```

2.5. À quel jour de la semaine correspond au 28 octobre ?

Exercice 3 (Boucles “while”)**.**

3.1. Écrivez un programme qui demande à l'utilisateur d'entrer 3 notes entre 0 et 20. Le programme affichera ensuite le minimum, le maximum et la moyenne des notes entrées. Utilisez une boucle `while`. Les notes entrées seront stockées progressivement dans une liste. Utilisez un parcours de votre liste de notes pour calculer la moyenne de celles-ci.

3.2. Modifiez le programme de sorte que celui-ci commence par demander à l'utilisateur le nombre de notes N qu'il désire entrer, puis procède comme précédemment.

3.3. Modifiez le programme de sorte que l'utilisateur puisse entrer autant de notes qu'il le désire et termine sa saisie par une certaine instruction, comme “fin” par exemple.

Exercice 4 (Boucles “while”)**.**

Écrivez un programme qui choisit un nombre entier au hasard entre 1 et 100, puis demande à l'utilisateur de le deviner. Si l'utilisateur entre un nombre trop petit ou trop grand, le programme devra afficher “plus haut” ou “plus bas”, respectivement. Le programme s'arrête lorsque le nombre a été trouvé. Pour générer un nombre entier au hasard, importez la librairie `random` et utilisez la fonction `randint()`.

Chapitre 5 : Fonctions

Rappel sur les fonctions et les procédures

La syntaxe pour définir une fonction :

```
def nom_fonction(arg_1,...,arg_N):
  instructions...
  return valeur   # dans le cas d'une procedure, simplement return
```

Paramètres par défaut dans les fonctions et les procédures

La syntaxe pour définir une fonction avec des arguments qui possèdent des valeurs par défaut est la suivante :

```
def nom_fonction(arg_1 = val_1,...,arg_N = val_N):
  instructions...
  return valeur # dans le cas d'une procedure, simplement return
```

Exercice 1 (Calcul d'intégrale par la méthode des rectangles)**.**

1.1. Montrez que l'équation $x^2 + y^2 = 1$ est l'équation du cercle de centre $(0, 0)$ et de rayon 1. En déduire l'équation développée du cercle de centre $(1, 0)$ et de rayon 1 en faisant le changement de variable $x' = x - 1$.

1.2. Soit f la fonction $f(x) = \sqrt{2x - x^2}$. Quels sont l'ensemble de définition et le graphe de f ? Vérifiez vos résultats en utilisant la calculatrice graphique du site `http://desmos.com`

1.3. Soit g la fonction définie par $g(x) = f(|x|)$. Quel est l'ensemble de définition de g ? Tracez son graphe sur `http://desmos.com`

1.4. Expliquez pourquoi

$$\int_{-2}^{2} g(x)dx = \pi$$

1.5. Programmez en Python les deux fonctions $f(x)$ et $g(x)$.

1.6. Définissez une fonction `integrale(phi, a, b, n)` qui renvoie une approximation de l'intégrale de la fonction φ sur $[a, b]$ par la méthode des rectangles avec n subdivisions. On pourra consulter `https://fr.wikipedia.org/wiki/Somme_de_Riemann` et utiliser la méthode du point médian.

1.7. Appliquez la question précédente à la fonction g sur $[-2, 2]$ et en déduire une approximation numérique de π.

Exercice 2 (fonctions, paramètres par défaut)**.**

2.1. Définissez une fonction `surf_cercle()` qui calcule la surface d'un cercle dont le rayon est donné en argument. Pour avoir accès à la valeur de π, ajouter l'instruction `from math import pi`.

2.2. Modifiez votre fonction `surf_cercle` de sorte que votre argument possède une valeur par défaut de 1.

2.3. Définissez une fonction `vol_boite()` qui renvoie le volume d'une boîte parallélépipédique dont on fournit les trois dimensions en arguments.

2.4. Modifiez votre fonction de telle sorte que si un seul argument est fourni, la boîte est considérée comme cubique (l'argument étant l'arête de ce cube) ; si deux arguments sont fournis, la boîte est considérée comme un parallélépipède à base carrée (le premier argument est le côté du carré, et le second la hauteur du parallélépipède) ; si trois arguments sont fournis, la boîte est considérée comme un parallélépipède général. Pour cela, donnez des valeurs par défaut à vos arguments et utiliser un test conditionnel pour savoir si tel ou tel paramètre possède telle ou telle valeur par défaut...

Exercice 3 (fonctions sur les chaines de caractères)**.**

3.1. Définissez une fonction `remplacement()` qui prend trois arguments `c1`, `c2` et `ch` et qui remplace tous les caractères `c1` par des caractères `c2` dans la chaîne caractères `ch`.

3.2. Modifiez votre fonction de telle sorte que si les arguments `c1`, `c2` et `ch` ne sont pas spécifiés, alors ils prendront les valeurs `" "`, `"*"` et `""`, respectivement.

Exercice 4 (comptage de lexèmes)**.**

4.1. En vous inspirant de l'exercice 2 du TD 2, créez deux fonctions `nb_car(chaine)` et `nb_mots(chaine)` qui comptent le nombre de caractères et le nombre de mots d'une chaîne de caractères `chaine` donnée en argument.

4.2. Comparez votre solution avec l'usage de la fonction `str.split()` disponible dans la librairie standard de Python ; dont la documentation est par exemple dans `https://docs.python.org/3/library/stdtypes.html`.

Chapitre 6 : Programmation structurée

Exercice 1 (table de multiplication)**.**

1.1. Définissez une fonction `TableMult(n, debut = 1, fin = 10)` qui imprime la table de multiplication de `n` entre les multiplicateurs `debut` et `fin`.

1.2. Améliorez votre fonction de telle sorte que si l'argument `debut` et plus grand que `fin`, la table de multiplication soit affichée correctement, dans l'ordre décroissant.

1.3. Créez une fonction similaire où les paramètres `ndebut` et `fin` sont demandés à l'utilisateur plutôt que passés en argument.

Exercice 2 (tour de magie)**.**

Le but de cet exercice est d'implémenter un tour de magie sur un jeu de 52 cartes.

2.1. Parmi les structures de données : entier, flottant, couple, chaine de caractères, liste et dictionnaire, quelle est celle qui vous parait la plus adaptée pour représenter l'ensemble des couleurs d'un jeu de cartes dans l'ordre Trèfle, Carreau, Coeur, Pique ? Même question pour les hauteurs des cartes de l'as au roi.

2.2. Après avoir défini les ensembles de hauteurs et de couleurs en Python, définissez les deux fonctions `hauteur(carte)` et `couleur(carte)` qui renvoient respectivement l'index de la hauteur et de la couleur d'une carte dans vos ensembles. Faites attention que les index commencent à zéro en Python. Par exemple, on a :

```
hauteur("valet de carreau") = 10
couleur("valet de carreau") = 1
```

2.3. Définissez la fonction `numero(carte)` qui renvoie le numéro d'une carte (un entier entre 0 et 51) dans l'ordre canonique : tous les Trèfles, tous les Carreaux, tous les Coeurs puis tous les Piques.

```
numero("valet de carreau") = 23
```

2.4. Définir la bijection réciproque `carte(numero)` qui prend un numéro en argument et renvoie la carte correspondante. Testez votre implémentation en vérifiant que l'on a bien la relation de composition `n = numero(carte(n))` pour tout n.

```
carte(23) = "le valet de carreau"
```

2.5. Programmez la fonction entree(prompt) qui demande dans la console une carte à l'utilisateur et renvoie la carte tapée par l'utilisateur, sous forme de chaîne de caractères. Votre fonction pourra être robuste et redemander la carte si l'ordinateur n'a pas compris.

2.6. Implémentez le tour de magie "The Fitch Cheney Five-Card Trick"

```
saisir la premiere carte : le 10 de pique
saisir la deuxieme carte : le 4 de coeur
saisir la troisieme carte : le 10 de carreau
saisir la quatrieme carte : le roi de trefle

... hmm, je pense que la cinquieme carte est le 3 de pique
```

Exercice 3 (le jeu “Motus”)**.**

Soit la liste de mots de cinq lettres suivantes :

```
mots = ["arbre", "grave", "piece", "nuage", "crane", "sonne", "table", "herbe",
 "ecrou", "mulet"]
```

Créz un programme qui implémente le jeu “Motus” avec cette liste de mots. Plus précisément, au départ, l'ordinateur choisit un mot au hasard dans cette liste (utilisez la fonction `randint()` de la librairie `random`). Ensuite, l'utilisateur dispose de 10 essais pour deviner le mot. Pour cela, il proposera des lettres tour à tour. À chaque fois, l'ordinateur l'informera de si cette lettre est présente ou non dans le mot, et, si tel est le cas, mettra à jour les lettres déjà devinées. L'exécution de votre programme devra ressembler à quelque chose comme ci-dessous :

```
Essai 6
Entrer une lettre: t

...et

Essai 7
Entrer une lettre: m

m..et
```

3.1. Commencez par définir une fonction `remplace(i, c, ch)` qui renvoie une chaîne dans laquelle le `i`-ème caractère de la chaîne `ch` est remplacé par `c`.

3.2. Utilisez ensuite cette fonction dans votre programme. À chaque lettre proposée par l'utilisateur, effectuez un parcours du mot à deviner, et, parallèlement, construisez un nouveau mot dans lequel les lettres non encore devinées apparaissent comme des points alors que celles déjà devinées apparaissent clairement.

3.3. Modifiez votre programme de telle sorte que votre jeu soit implémenté dans une procédure `motus()` à un argument ; cet argument devra correspondre au nombre d'essais maximum auquel l'utilisateur a le droit et aura une valeur par défaut de 10.

Exercice 4 (le jeu “Puissance 4” (long et plus difficile ; très bon exercice récapitulatif))**.**

Programmez un jeu “Puissance 4” sur une grille de taille N. Le jeu devra avoir l'apparence comme montré ci-dessous :

```
.   .   .   .   .   .   .   .

.   .   .   .   X   .   .   .

.   .   .   O   .   .   .   .

O   .   .   .   .   .   X   .

.   .   .   .   .   .   .   .

.   .   .   O   .   .   .   .

.   .   .   .   .   X   .   .

.   .   .   .   .   .   .   .

Joueur 1: quelle position (ligne, colonne)?
```

4.1. Programmez une fonction `generer_grille(N = 4)` qui génère une grille d'une certaine taille `N` valant 4 par défaut. Chaque ligne de la grille est une liste de taille `N` dont les éléments sont des chaînes `"."`. La grille est alors représentée comme la liste de ses lignes, comme illustré ci-dessous.

```
# exemple d'une grille de taille 3 x 3
# on a 3 listes de taille 3 remplies avec des "."
[[".", ".", "."], [".", ".", "."], [".", ".", "."]]
```

4.2. Programmez une fonction `placer_pion(joueur, grille, position)` qui permet de placer le pion du joueur `joueur` à la position `position` dans la grille `grille`. L'argument `joueur` vaut `1` ou `2`. S'il vaut, `1`, le pion placé sera un rond (O majuscule) `"O"` ; s'il vaut, `2`, le pion placé sera une croix (X majuscule) `"X"`. L'argument `grille` est une liste de listes. L'argument `position` est une liste ou un tuple de la forme `(i,j)`.

4.3. Programmez une fonction `chercher_alignement(joueur, liste, n)` qui cherche si un alignement de `n` pions du joueur `joueur` existe dans la liste `liste` (cette liste représentera plus tard, une ligne, une colonne ou une diagonale). Suivant que l'argument `joueur` vaut `1` ou `2`, on cherchera un alignement de `n` `"O"` ou `n` `"X"`, respectivement.

4.4. Programmez une fonction `generer_lignes(grille)` qui retourne la liste de toutes les lignes de la grille `grille`. Remarquez que dans ce cas, il suffit de retourner la grille elle même, puisque celle-ci est donnée comme la liste de ces lignes.

4.5. Programmez une fonction `generer_colonnes(grille)` qui retourne la liste de toutes les colonnes de la grille `grille`. L'idée est de transposer la matrice `grille`...

4.6. Programmez une fonction `generer_diagonales1(grille)` qui retourne la liste de toutes les diagonales "montantes" de la grille `grille`. Pour cela, remarquez que chaque diagonale de ce type est constituée des éléments de la grille dont les indices donnent une même somme. Par exemple, la diagonale `[(3,1), (2,2), (1,3)]` est formée de tous les éléments dont les indices ont une somme de 4.

4.7. Programmez une fonction `generer_diagonales2(grille)` qui retourne la liste de toutes les diagonales "descendantes" de la grille `grille`. Pour cela, remarquez que chaque diagonale de ce type est constituée des éléments de la grille dont les indices donnent une même différence. Par exemple, la diagonale `[(1,3), (2,4), (3,5)]` d'une grille de taille 5 est formée de tous les éléments dont les indices ont une différence de -2.

4.8. Programmez une fonction `imprimer_grille(grille)` qui imprime la grille `grille` dans votre terminal de manière conviviale, comme illustré ci-dessus. On rappelle que les chaînes `"\t"`, `"\n"` codent un espace de tabulation et un retour de ligne, respectivement.

4.9. En utilisant toutes les fonctions que vous avez programmées jusque là, définissez une fonction `jouer(n)` qui permet de jouer à Puissance 4 sur une grille de taille `n`. Utilisez une grande boucle `while` qui demande à chaque joueur de placer un pion à tour de rôle, jusqu'à ce qu'un alignement de 4 pions soit trouvé dans une ligne, une colonne ou une diagonale. Tout le programme peut se faire en 250 lignes de code environ.

Chapitre 7 : Graphisme (MatPlotLib) - simulations de Monte-Carlo

Exercice 1 (Tracer des courbes avec MatPlotLib)**.**

1.1. Implémenter deux fonctions réelles f et g de votre choix.

1.2. Définir trois listes X, Y et Z, où la liste X contient les valeurs de x dans $[-5, 5]$ par pas de 0.05, la liste Y les valeurs de $y = f(x)$ et Z les valeurs de $z = g(x)$.

1.3. Vérifier que les trois listes X, Y et Z ont la même longueur.

1.4. Tracer les courbes $Y = f(X)$ et $Z = g(X)$ sur le même graphique à l'aide de la librairie MatPlotLib.

Exercice 2 (Simulation de Monte-Carlo)**.**

Le but de cet exercice est d'utiliser la méthode de Monte-Carlo pour obtenir une approximation statistique de l'aire de la surface à l'intérieur d'une courbe fermée. On obtiendra en particulier une valeur approchée de π.

2.1. Utiliser la fonction `eval()` de Python pour évaluer une chaîne de caractères en remplaçant les variables par leur valeur dans l'environnement actuel.

```
x = 0.7
y = -0.8
formule = " x**2 + y**2 <= 1.0 "
test = eval(formule)
print(test)
```

2.2. On veut tirer N point aléatoires dans le carré $[-1, 1] \times [-1, 1]$. A l'aide de la librairie NumPy, générer un couple (X, Y), où X et Y sont deux listes de longueur N de tirages de variables aléatoires uniformes réelles dans l'intervalle $[-1, 1]$.

2.3. Si je tire N points aléatoires uniformément dans le carré $[-1, 1] \times [-1, 1]$, quelle est la probabilité que chaque point soit à l'intérieur du disque de centre $(0, 0)$ et de rayon 1 ?

2.4. Implémenter la fonction `montecarlo(listeX, listeY, formule)` qui prend en argument une formule et les coordonnées de N points aléatoires. La fonction tracera en rose le nuage des points qui valident la formule, et en vert le nuage des points qui invalident la formule.

> **Indication**
>
> Pour tracer un nuage de points avec MatPlotLib, utiliser `plt.scatter()`.

2.5. Changer le titre du graphique tous les mille points pour afficher l'évolution de la valeur approchée de la surface à l'intérieur de la courbe (utiliser la fonction `plt.title()` de MatPlotLib).

2.6. Pour avoir une animation plus fluide, ralentir l'affichage en faisant une pause de 0.2 secondes tous les mille points affichés à l'aide de la fonction `plt.pause()`.

2.7. Afficher la simulation de Monte-Carlo avec 10000 points, et le calcul approché des aires, pour les trois courbes suivantes :

$$x^2 + y^2 < 1$$

$$\sqrt{|x|} + \sqrt{|y|} < 1$$

$$x^2 + (0.3 + 1.5y - |y|^{0.6})^2 < 1$$

Chapitre 8 : Graphisme (MatPlotLib) - mouvement brownien

Exercice 1 (Simulation de mouvements browniens)**.**

1.1. To be written... (voir le fichier /td/ito.py)

Exercice 2 (Attracteur de Lorentz)**.**

2.1. To be written... (voir le fichier /td/lorentz.py)

Chapitre 9 : Manipulation de fichiers

Exercice 1 (Opérations de base sur les fichiers)**.**

1.1. Écrivez un script qui crée un fichier nommé "`test.txt`" (utilisez le mode écriture `"w"` et la syntaxe `with open(f_physique,"w") as f_logique`).

1.2. Écrivez maintenant plusieurs lignes de texte dans ce fichier (utilisez une nouvelle fois le mode écriture `"w"` ; on rappelle que le saut de ligne est codé par `\n`).

1.3. Ouvrez le ficher en mode lecture et d'imprimez son contenu à l'écran (pour cela, utilisez le mode lecture `"r"` ; vous récupérerez le contenu sous forme de chaîne avec la méthode `f.read()`, puis imprimerez ce contenu).

1.4. Rouvrez votre fichier en mode lecture et imprimer la deuxième ligne de celui ci (utilisez la méthode `f.readlines()`).

1.5. Ouvrez votre fichier en mode écriture et ajouter-lui une ligne de texte (utilisez le mode append `"a"`).

1.6. Rouvrez votre fichier en mode lecture et imprimer l'avant dernière ligne de celui ci (utilisez une nouvelle fois la méthode `f.readlines()`).

Exercice 2 (Affichage d'un fichier texte)**.**

2.1. Ecrivez un script qui demandera à l'utilisateur un nom de fichier et, lorsque ce fichier existe, affiche le contenu de celui-ci à l'écran.

> **Indications**
>
> Si vous désirez rechercher le fichier directement dans le répertoire courant, vous pouvez importer la libraire `os` avec la commande `from os import *`, puis, utiliser la commande `cwd = os.getcwd()` pour enregistrer le "path" du répertoire courant (sous forme de string) dans la variable cwd.

2.2. Modifiez votre script de manière à ce que :

(a) lors de l'affichage, chaque ligne soit précédée par un numéro ;

(b) après l'affichage du fichier, on informera du nombre total de lignes et de caractères du fichier.

Par exemple, un fichier qui contient les 3 lignes suivantes :

```
Comparison operations are supported by all objects.
They all have the same priority.
Comparisons can be chained arbitrarily.
```

devra être affiché par le programme de la façon suivante :

```
Le contenu du fichier test.txt est:
1 : Comparison operations are supported by all objects.
2 : They all have the same priority.
3 : Comparisons can be chained arbitrarily.
Il y a 3 lignes et 124 caracteres.
```

Exercice 3 (Listes et fichiers)**.**

3.1. Écrivez un programme qui remplit une liste de N éléments par des nombres entiers aléatoires entre -100 et $+100$. La taille de la liste doit être demandée à l'utilisateur.

> **Indications**
>
> Pour générer un nombre entier aléatoire entre `a` et `b`, il vous faut importer la librairie `random` en utilisant la commande `from random import *`, puis, utiliser la fonction `randint(a,b)`.

3.2. Affichez la liste ainsi que la somme de ses éléments.

3.3. Ajoutez maintenant une fonctionnalité qui permet d'écrire la liste, élément par élément, chacun sur une seule ligne, dans un fichier spécifié par l'utilisateur.

Exercice 4 (Fichiers et listes)**.**

4.1. Écrivez un programme qui lit un fichier spécifié par l'utilisateur (par exemple, le fichier de l'exercice 3) et remplit une liste par les nombres trouvés dans les lignes du fichier.

4.2. Ensuite, affichez le contenu de la liste lue ainsi que la somme de ses éléments.

Exercice 5 (Fichiers et dictionnaires)**.**

Imaginez que vous avez un fichier sous format CSV (comma separated values – valeurs séparées par des virgules) qui contient des informations concernant les résultats d'un test intermédiaire, c'est-à-dire qu'il y a plusieurs lignes du type :

```
prenom,nom,faculte,note,bonus
```

Par exemple :

```
Eric,Clark,economie,13.5,1.0
Lisa,Clark,droit,8.0,0.0
```

Vous pouvez créer un tel fichier de données par vous-même dans un éditeur de texte.

5.1. Écrivez une fonction qui prend en paramètres le nom du fichier, ouvre le fichier, le lit et renvoie un dictionnaire contenant les données du fichier. Les clés du dictionnaire devront correspondre au prénom et nom complet de la personne et leurs valeurs devront correspondre aux données complètes. Par exemple, les données de l'exemple précédent seront converties comme suit :

```
donnees = {'Lisa Clark': ['Lisa','Clark','droit','8.0','0.0'],
'Eric Clark': ['Eric','Clark','economie','13.5','1.0'] }
```

5.2. Écrivez une deuxième fonction qui prend en paramètres la liste des données d'une personne et les affiche de manière lisible.

5.3. Écrivez une troisième fonction qui va afficher les résultats du test pour les étudiants ayant obtenu une note au dessus de 9.5. Elle doit prendre deux paramètres : les données sous la forme d'un dictionnaire (cf. point 1) et la note au dessus de laquelle il faut afficher l'information, 9.5 par défaut. Tester vos fonctions en affichant des données d'un fichier cré pour l'exemple.

Exercice 6 (Fichiers et dictionnaires (suite))**.**

Écrivez un script qui demande à l'utilisateur le nom et le prénom d'une personne et, ensuite, affiche à l'écran le résultat de cette personne au test, pour autant que celle-ci existe dans le fichier de données. Le script devra redemander le nom et le prénom jusqu'à ce qu'une chaine vide soit entrée par l'utilisateur (i.e., l'utilisateur a tapé Enter). Pensez à réutiliser les fonctions déjà écrites.

Exercice 7 (Dictionnaires et fichiers)**.**

7.1. Écrivez une fonction qui prend les données sous forme d'un dictionnaire et les enregistre dans un fichier. Les données et le nom du fichier devront passer en paramètres de la fonction. Vous pouvez utiliser la fonction `join()` pour convertir une liste de chaines de caractères en une chaine. Par exemple le code suivant :

```
info = ['Lisa','Clark','droit','8.0','0.0']
chaine = ",".join( info )
```

vous donne la chaine `"Lisa,Clark,droit,8.0,0.0"`

7.2. Modifiez votre script de manière à pouvoir enregistrer les résultats des étudiants dans des fichiers différent pour chaque faculté (par exemple `test-droit.csv` pour la faculté de droit et `test-economie.csv` pour la faculté d'économie, etc.). Le format des fichiers enregistrés doit être le même que celui de `etudiants.csv`.

Chapitre 10 : Librairie Pandas

Exercice 1 (Librairie Pandas : fichiers CSV et dataframe)**.**

1.1. Ouvrez le fichier `diamonds.csv` dans un éditeur de code et analysez sa structure. Ecrivez une fonction Python qui ouvre le fichier `diamonds.csv` et importe les données sous forme d'une matrice (dataframe) de la librairie Pandas. Extrayez la colonne des prix, et calculez ses statistiques élémentaires (moyenne, écart type, quantiles, etc.). Appliquez la fonction $f(x) = 1.3x + 5$ à la colonne des prix. Tracez le nuage de points des prix par rapport au nombre de carats.

> **Indications**
>
> Pour utiliser la librairie `pandas` il faut inclure la librairie en utilisant la commande `import pandas as pd`. Une documentation rapide de la librairie se trouve dans le répertoire `http://cassio.free.fr/assas/info/MBF2/doc/` .

1.2. Triez les valeurs de la matrice par nombre de carats décroissants, puis modifiez votre script de manière à pouvoir enregistrer les diamants dans des fichiers différents pour chaque type de taille (par exemple `ideal.csv` pour la taille "Ideal" et `very-good.csv` pour la taille "Very Good", etc.). Le format des fichiers enregistrés doit être le même que celui de `diamonds.csv`.

Exercice 2 (Utilisation de données météo)**.**

2.1. Ouvrez le fichier `AnomalieTempe_RCP2.6_mensuel.txt` dans un éditeur de code et analysez sa structure. Essayez de lire le fichier avec la fonction `pandas.read_csv()` : quel(s) problème(s) rencontre-t-on ?

2.2. Nettoyez les données, renommez le fichier en `anomalies.txt` et lisez les données dans un dataframe correct. Vous pouvez chercher sur le Web et lire la documentation de la fonction `pandas.read_csv()` pour connaitre ses options et l'adapter à vos données.