Introduction à la programmation Web

Dans l'article précédent, nous avons transformé la fonctionnalité de la suite de Fibonacci en une version orientée objet, en implémentant une interface en ligne de commande.

`server.py` :

```
class BaseHandler:
  def handle(self, request:str):
    pass

class Server:
  def __init__(self, handlerClass):
    self.handlerClass = handlerClass

  def run(self):
    while True:
      request = input()
      self.handlerClass().handle(request)
```

`fib_handle.py` :

```
from fib import f
from server import BaseHandler, Server

class FibHandler(BaseHandler):
  def handle(self, request:str):
    n = int(request)
    print('f(n)=', f(n))
    pass

server = Server(FibHandler)
server.run()
```

Serveur Web simple

Alors, comment le transformer en une interface `Web` ?

Il suffit de remplacer le `Server` ci-dessus par un `Server` utilisant le `protocole HTTP`. Voyons d'abord à quoi ressemble un `serveur HTTP` en Python.

La bibliothèque standard de Python fournit un serveur web.

```
python -m http.server
```

Exécutez-le dans le terminal.

```
$ python -m http.server
Serveur HTTP en écoute sur :: port 8000 (http://[::]:8000/) ...
```

Ouvrez-le dans votre navigateur pour voir l'effet.

Cela liste le répertoire actuel. Ensuite, lorsque vous naviguez sur cette page web, revenez au terminal. Cette fois, c'est très intéressant.

```
$ python -m http.server
Serveur HTTP en écoute sur :: port 8000 (http://[::]:8000/) ...
::1 - - [07/Mar/2021 15:30:35] "GET / HTTP/1.1" 200 -
::1 - - [07/Mar/2021 15:30:35] code 404, message Fichier non trouvé
::1 - - [07/Mar/2021 15:30:35] "GET /favicon.ico HTTP/1.1" 404 -
::1 - - [07/Mar/2021 15:30:35] code 404, message Fichier non trouvé
::1 - - [07/Mar/2021 15:30:35] "GET /apple-touch-icon-precomposed.png HTTP/1.1" 404 -
::1 - - [07/Mar/2021 15:30:35] code 404, message Fichier non trouvé
::1 - - [07/Mar/2021 15:30:35] "GET /apple-touch-icon.png HTTP/1.1" 404 -
::1 - - [07/Mar/2021 15:30:38] "GET / HTTP/1.1" 200 -
```

Voici le journal d'accès au site web. Ici, `GET` représente une opération d'accès aux données dans le service web. `HTTP/1.1` indique que la version `1.1` du protocole `HTTP` a été utilisée.

Comment l'utiliser pour créer notre service de séquence de Fibonacci. D'abord, cherchons des exemples de code en ligne, modifions-les légèrement, et écrivons un serveur Web très simple :

```
from http.server import SimpleHTTPRequestHandler, HTTPServer

class Handler(SimpleHTTPRequestHandler):
    def do_GET(self):
        self.send_response(200)
```

```
        self.send_header('Content-type', 'text')
        self.end_headers()
        self.wfile.write(bytes("hi", "utf-8"))
```

server = HTTPServer((“127.0.0.1”, 8000), Handler)

```
server.serve_forever()
```

Cela ne vous semble-t-il pas familier ? C’est presque identique à ce que nous avons fait avec `Server` ci-dessus. Notez que `SimpleHTTPRequestHandler` n’est pas une classe de base, il existe également une classe appelée `BaseHTTPRequestHandler`. `SimpleHTTPRequestHandler` gère un peu plus de choses par rapport à celle-ci. Ajouter la fonctionnalité de traitement de la suite de Fibonacci à cela est facile.

Ici, `127.0.0.1` représente l’adresse de la machine locale, et `8000` représente le port de la machine locale. Comment comprendre le port ? C’est comme une fenêtre dans une maison, c’est un point de communication entre la maison et l’extérieur. `bytes` signifie convertir une chaîne de caractères en octets. `utf-8` est une méthode d’encodage de chaînes de caractères. `send_response`, `send_header` et `end_headers` sont tous utilisés pour envoyer du contenu, afin de produire ce que le protocole `HTTP` exige, de manière à ce qu’il puisse être compris par le navigateur. Ainsi, nous voyons `hi` sur la page web.

Ensuite, essayons de récupérer les paramètres à partir de la requête.

```
from http.server import SimpleHTTPRequestHandler, HTTPServer
from fib import f
from urllib.parse import urlparse, parse_qs

class Handler(SimpleHTTPRequestHandler):
    def do_GET(self):
        self.send_response(200)
        self.send_header('Content-type', 'text')
        self.end_headers()
        parsed = urlparse(self.path)
        qs = parse_qs(parsed.query)
        result = ""
        if len(qs) > 0:
            ns = qs[0]
```

```
            if len(ns) > 0:
                n = int(ns)
                result = str(f(n))
        self.wfile.write(bytes(result, "utf-8"))

server = HTTPServer(("127.0.0.1", 8000), Handler)

server.serve_forever()
```

C’est un peu complexe, n’est-ce pas ? Ici, on est en train de parser quelques paramètres.

```
self.path=/?n=3
parsed=ParseResult(scheme='', netloc='', path='/', params='', query='n=3', fragment='')
qs={'n': ['3']}
ns=['3']
n=3
```

Récursion Avancée

Faisons un peu de refactorisation du code.

```
from http.server import SimpleHTTPRequestHandler, HTTPServer
from fib import f
from urllib.parse import urlparse, parse_qs

class Handler(SimpleHTTPRequestHandler):

    def parse_n(self, s):
      parsed = urlparse(s)
      qs = parse_qs(parsed.query)
      if len(qs) > 0:
        ns = qs['n']
        if len(ns) > 0:
          n = int(ns[0])
          return n
      return None
```

```
def do_GET(self):
  self.send_response(200)
  self.send_header('Content-type', 'text')
  self.end_headers()
```

Traduction en français :

```
def parse_n(self, s):
  parsed = urlparse(s)
  qs = parse_qs(parsed.query)
  if len(qs) > 0:
    ns = qs['n']
    if len(ns) > 0:
      n = int(ns[0])
      return n
  return None

def do_GET(self):
  self.send_response(200)
  self.send_header('Content-type', 'text')
  self.end_headers()
```

Le code reste en anglais car il s'agit de code source, mais voici une explication en français :

- `parse_n` : Cette fonction analyse une URL pour extraire un paramètre `n` de la chaîne de requête. Si le paramètre est trouvé et qu'il est valide, il est converti en entier et retourné. Sinon, la fonction retourne `None`.
- `do_GET` : Cette méthode gère les requêtes HTTP GET. Elle envoie une réponse HTTP 200 (OK) et définit l'en-tête `Content-type` à `text`. Les en-têtes sont ensuite finalisés avec `end_headers()`.

```
  result = ""
  n = self.parse_n(self.path)
  if n is not None:
    result = str(f(n))

  self.wfile.write(bytes(result, "utf-8"))
  self.wfile.write(bytes(result, "utf-8"))
```

```
server = HTTPServer(("127.0.0.1", 8000), Handler)

server.serve_forever()
```

Introduisons la fonction `parse_n` pour encapsuler l'extraction de `n` à partir du chemin de la requête.

Le programme actuel présente le problème suivant. Xiao Wang a demandé le 10000ème terme de la suite de Fibonacci, et quelques jours plus tard, Xiao Ming a également demandé le 10000ème terme de la suite de Fibonacci. À deux reprises, Xiao Wang et Xiao Ming ont dû attendre longtemps avant d'obtenir le résultat. Comment pouvons-nous améliorer l'efficacité de ce `service Web` ?

Pour résoudre ce problème, nous pouvons envisager plusieurs approches :

1. **Mise en cache des résultats** : Une fois que le 10000ème terme de la suite de Fibonacci a été calculé, nous pouvons stocker ce résultat dans un cache. Ainsi, lorsque quelqu'un d'autre demande le même terme, nous pouvons simplement renvoyer le résultat stocké au lieu de le recalculer.
2. **Calcul asynchrone** : Si le calcul prend beaucoup de temps, nous pouvons le déléguer à un processus asynchrone. Cela permettra à l'utilisateur de recevoir une réponse immédiate indiquant que la demande a été reçue et que le calcul est en cours. Une fois le calcul terminé, le résultat peut être envoyé à l'utilisateur.
3. **Optimisation de l'algorithme** : Nous pouvons également envisager d'optimiser l'algorithme de calcul de la suite de Fibonacci pour qu'il soit plus rapide. Par exemple, en utilisant des techniques de programmation dynamique ou des formules mathématiques pour calculer les termes de manière plus efficace.
4. **Limitation des demandes** : Si le calcul est très coûteux en termes de ressources, nous pouvons limiter la fréquence des demandes ou mettre en place un système de file d'attente pour gérer les demandes de manière plus efficace.

En combinant ces approches, nous pouvons grandement améliorer l'efficacité du service Web et réduire le temps d'attente pour les utilisateurs.

On remarque que si `n` est identique, la valeur de `f(n)` reste toujours la même. Nous avons mené quelques expériences.

```
127.0.0.1 - - [10/Mar/2021 00:33:01] "GET /?n=1000 HTTP/1.1" 200 -
```

```
----------------------------------------
Une exception s'est produite lors du traitement de la requête depuis ('127.0.0.1', 50783)
Traceback (dernier appel en dernier) :
    ...
    if v[n] != -1:
IndexError: indice de liste hors limites
```

Si le tableau n'est pas assez grand, alors modifions le tableau `v` pour qu'il ait une taille de 10000.

```
v = []
for x in range(10000):
   v.append(-1)
```

Cependant, lorsque `n` est égal à `2000`, une erreur de dépassement de la profondeur de récursion est survenue :

```
127.0.0.1 - - [10/Mar/2021 00:34:00] "GET /?n=2000 HTTP/1.1" 200 -
----------------------------------------
Une exception s'est produite lors du traitement de la requête depuis ('127.0.0.1', 50821)
Traceback (dernier appel en dernier):
    ...
    if v[n] != -1:
RecursionError: profondeur de récursion maximale dépassée lors de la comparaison
```

Cependant, tout cela s'est déroulé assez rapidement.

Pourquoi ? Parce que de `f(1)` à `f(1000)`, chaque valeur ne doit être calculée qu'une seule fois. Cela signifie que lors du calcul de `f(1000)`, l'opération `+` n'est peut-être exécutée qu'environ 1000 fois. Nous savons que la profondeur de récursion en `Python` est d'environ 1000. Cela signifie que nous pouvons optimiser le programme de cette manière : si nous devons calculer `2000`, nous calculons d'abord `1000`. Non, cela pourrait encore entraîner une `erreur de dépassement de la profondeur de récursion`. Si nous devons calculer 2000, calculons d'abord 1200. Si nous devons calculer 1200, calculons d'abord 400.

Après avoir calculé 400 et 1200 de cette manière, puis en calculant 2000, la profondeur de récursion sera d'environ 800, ce qui évitera l'erreur de débordement de la profondeur de récursion.

```
v = []
for x in range(1000000):
   v.append(-1)

def fplus(n):
   if n > 800:
      fplus(n-800)
      return f(n)
   else:
      return f(n)

def f(n):
   if v[n] != -1:
      return v[n]
   else:
      a = 0
      if n < 2:
         a = n
      else:
         a = f(n-1) + f(n-2)
      v[n] = a
      return v[n]
```

La fonction `fplus` a été ajoutée.

Cependant, cela nous amène à nous demander ce qui se passerait si `fplus` était appelé de manière récursive `1000` fois. 1000 * 800 = 800000. Lorsque j'ai défini `n` à 800 000, j'ai de nouveau rencontré une erreur de profondeur de récursion. Après quelques essais supplémentaires, j'ai réalisé que la situation était plus complexe. Néanmoins, après cette optimisation, calculer 2000 est devenu très facile.

Lecture et écriture de fichiers

Il semble que nous nous soyons écartés du sujet. Revenons au développement web. La première requête est `f(400)`, la seconde est `f(600)`. Lors de la deuxième requête, nous pouvons utiliser les valeurs du tableau `v` générées par la première requête. Cependant, lorsque nous quittons le programme et le redémarrons, ces valeurs ne sont plus disponibles. Avec notre

méthode, le calcul de la suite de Fibonacci est rapide. Mais imaginons que ce soit lent. Surtout, comme lorsque nous n'avions pas introduit le tableau `v`, il y avait beaucoup de calculs répétés. Dans ce cas, nous aimerions pouvoir sauvegarder les résultats obtenus avec tant d'efforts.

C'est là qu'intervient le concept de `cache`. Le tableau `v` ici agit comme un cache. Cependant, il n'existe que pendant la durée de vie du programme. Une fois le programme fermé, il disparaît. Que faire alors ? Naturellement, on pense à le sauvegarder dans un fichier.

Comment sauvegarder le tableau `v` dans un fichier ?

```
0 0
1 1
2 1
3 2
4 3
...
```

Notre tableau `v` peut être sauvegardé de cette manière. Chaque ligne est enregistrée sous la forme `n f(n)`. Étant donné que `n` croît naturellement, nous pourrions peut-être simplement sauvegarder les valeurs de `f(n)`.

```
0
1
1
2
3
...
```

Venez essayer.

```
f = open("demofile2.txt", "a")
f.write("Maintenant le fichier a plus de contenu !")
f.close()
```

Ouvrir et lire le fichier après l'ajout :

f = open("demofile2.txt", "r") print(f.read())

Le deuxième paramètre de `open` peut être `a`, ce qui signifie que le contenu sera ajouté à la fin du f

```python
file = open('fib_v', 'a')
file.write('hi')
file.close()
```

En exécutant cela, on constate effectivement qu'il y a un fichier `fib_v`.

`fib_v` :

```
hi
```

Lorsque nous exécutons à nouveau, cela devient comme ceci.

```
hihi
```

Comment faire un saut de ligne ?

En Markdown, vous pouvez créer un saut de ligne de plusieurs manières :

1. **Utiliser deux espaces à la fin de la ligne**
   Ajoutez deux espaces à la fin de la ligne avant de passer à la suivante.
   Exemple :

   ```
   Ligne 1
   Ligne 2
   ```

2. **Utiliser une balise HTML `<br>`**
   Vous pouvez insérer une balise `<br>` pour forcer un saut de ligne.
   Exemple :

   ```
   Ligne 1<br>
   Ligne 2
   ```

3. **Laisser une ligne vide entre les paragraphes**
   Laissez une ligne vide entre deux lignes pour créer un nouveau paragraphe.
   Exemple :

```
    Ligne 1


    Ligne 2
```

Choisissez la méthode qui convient le mieux à votre cas d'utilisation !

```
file = open('fib_v', 'a')
file.write('hi\n')
file.close()
```

Cela imprimera une fois, affichant `hihihi`, sans voir de saut de ligne. Cependant, lors de la deuxième impression, un saut de ligne apparaît. Cela montre que le saut de ligne a été imprimé la première fois, mais il était à la fin, donc invisible.

Comment lire cela.

```
file = open('fib_v', 'r')
print(file.read())
```

```
$ python fib.py
hihihi
hi
```

Ensuite, modifions notre programme de Fibonacci.

```
v = []
for x in range(1000000):
   v.append(-1)

def read():
   file = open('fib_v', 'r')
   s = file.read()
   if len(s) > 0:
      lines = s.split('\n')
      if (len(lines) > 0):
        for i in range(len(lines)):
           v[i] = int(lines[i])
```

```
def save():
   file = open('fib_v', 'w')
   s = ''
   start = True
   for vv in v:
      if vv == -1:
         break
      if start == False:
         s += '\n'
      start = False
      s += str(vv)
   file.write(s)
   file.close()

def fcache(n):
   x = fplus(n)
   save()
   return x

def fplus(n):
   if n > 800:
      fplus(n-800)
      return f(n)
   else:
      return f(n)

def f(n):
   if v[n] != -1:
      return v[n]
   else:
      a = 0
      if n < 2:
         a = n
      else:
         a = f(n-1) + f(n-2)
      v[n] = a
      return v[n]
```

read() fcache(10) save()

Enfin, nous avons terminé l'écriture du programme. Après l'exécution du programme, le fichier `fib_v` r

`fib_v` :

```shell
0
1
1
2
3
5
8
13
21
34
55
```

Vous trouvez que l'analyse ci-dessus est un peu fastidieuse. `\n` est un caractère de nouvelle ligne. Existe-t-il une manière plus simple et unifiée pour l'analyse ? Les gens ont inventé le format de données `JSON`.

JSON

Le nom complet de JSON est `JavaScript Object Notation`. Voici un exemple de `JSON`.

```
{"name":"John", "age":31, "city":"New York"}
```

Voici comment représenter une correspondance.

JSON possède les éléments de base suivants :

1. Nombres ou chaînes de caractères
2. Listes
3. Mappages

Ces éléments de base peuvent également être imbriqués de manière arbitraire. Par exemple, une liste peut contenir une autre liste, et un mapping peut également contenir une liste, et ainsi de suite.

```
{
  "name":"John",
  "age":30,
  "cars":[ "Ford", "BMW", "Fiat" ]
}
```

Écrire sur une seule ligne ou bien formater joliment le code fait une différence en termes de sens. Peut-être pouvons-nous imaginer leurs graphes de calcul. Les espaces n'affectent pas leurs graphes de calcul.

Ensuite, nous devons transformer le tableau `v` en une chaîne de caractères au format `json`.

```
import json
```

v = [] for x in range(1000000): v.append(-1)

```
def fplus(n):
   if n > 800:
      fplus(n-800)
      return f(n)
   else:
      return f(n)

def f(n):
   if v[n] != -1:
      return v[n]
   else:
      a = 0
      if n < 2:
         a = n
      else:
         a = f(n-1) + f(n-2)
      v[n] = a
      return v[n]
```

```
fplus(100)
s = json.dump(v)
file = open('fib_j', 'w')
file.write(s)
file.close()
```

Lorsque nous écrivons ceci, une erreur se produit : `TypeError: dump() missing 1 required positional argument: 'fp'`. Dans `vscode`, vous pouvez voir la définition de la fonction de cette manière.

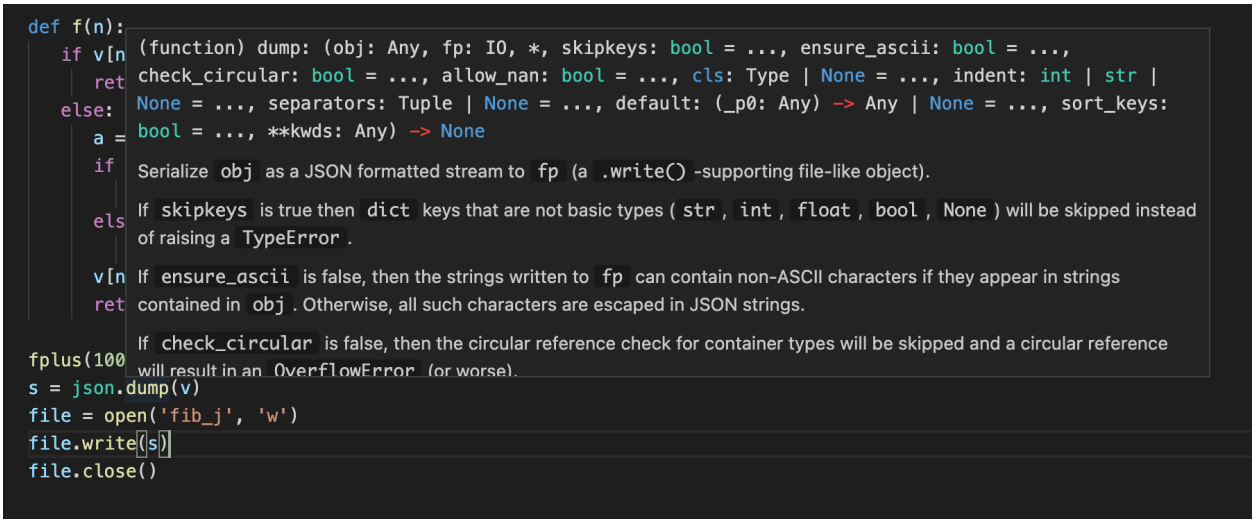

Figure 1: json

Vous pouvez simplement déplacer votre souris sur `dump`. C'est pratique, n'est-ce pas ?

```
fplus(10)
file = open('fib_j', 'w')
json.dump(v, file)
file.close()
```

Le calcul jusqu'à 100 affiche un peu trop de nombres, donc ici on le change à 10. Il suffit de passer un objet `file` comme deuxième paramètre à `dump`.

Ainsi, vous pouvez voir le fichier :

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, -1, -1, -1]
```

Notez que de nombreux `-1` ont été omis par la suite.

```
def read():
    file = open('fib_j', 'r')
    s = file.read()
    sv = json.loads(s)
    for i in range(len(sv)):
        if sv[i] != -1:
            v[i] = sv[i]
def save():
    file = open('fib_j', 'w')
    json.dump(v, file)
    file.close()
```

read()

```
for vv in v:
    if vv != -1:
        print(vv)
```

Lorsque cela se produit, on peut voir que cela imprime :

```
0
1
1
2
3
5
8
13
21
34
55
```

Voici une traduction en français de la phrase :

“Vérifions ensemble ces fonctions :”

Si vous avez besoin de plus de détails ou d’une traduction plus spécifique, n’hésitez pas à me le dire !

```
def read():
    file = open('fib_j', 'r')
    s = file.read()
    sv = json.loads(s)
    for i in range(len(sv)):
        v[i] = sv[i]

def save():
    sv = []
    for i in range(len(v)):
        if v[i] != -1:
            sv.append(v[i])
        else:
            break
    file = open('fib_j', 'w')
    json.dump(sv, file)
    file.close()

lire()
fplus(100)
sauvegarder()
```

Ensuite, en vérifiant le fichier, on constate que les valeurs correctes ont bien été enregistrées, et de manière très ordonnée.

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946, 17711
```

Base de données

Si les données sont volumineuses et de structure complexe, que faire ? Utiliser des fichiers pour les sauvegarder deviendra lent et fastidieux. C'est là qu'interviennent les `bases de données`. Elles équivalent à des `Excel` programmables. Ce sont des `feuilles Excel` que l'on peut facilement manipuler avec du code pour ajouter, supprimer, modifier et rechercher des données.

J'ai trouvé l'exemple dans la documentation officielle.

```
import sqlite3
con = sqlite3.connect('example.db')
```

```
cur = con.cursor()
```

Créer une table

cur.execute(‘‘‘CREATE TABLE stocks (date text, trans text, symbol text, qty real, price real)” ’)

Insérer une ligne de données

cur.execute(“INSERT INTO stocks VALUES (‘2006-01-05’,‘BUY’,‘RHAT’,100,35.14)”)

Sauvegarder (valider) les modifications

con.commit()

Nous pouvons également fermer la connexion si nous en avons terminé avec elle.

Assurez-vous simplement que toutes les modifications ont été validées, sinon elles seront perdues.

con.close()

```python
for row in cur.execute('SELECT * FROM stocks ORDER BY price'):
        print(row)
```

`cursor` représente un curseur, un peu comme un curseur de texte. Ce qui précède signifie la connexion à la base de données, la création de table, l’insertion de données, la soumission des modifications et la fermeture de la connexion. L’exemple à la fin est un exemple de requête de données.

```
import sqlite3
```

```
v = []
for x in range(1000000):
   v.append(-1)

def create_table(cur: sqlite3.Connection):
    cur.execute('CREATE TABLE vs(v text)')

def read():
    pass

def save():
    con = sqlite3.connect('fib.db')
    cur = con.cursor()
    create_table(cur)
    for vv in v:
        if vv != -1:
            cur.execute('INSERT INTO vs VALUES(' + str(vv) + ')')
        else:
            break
    con.commit()
    con.close()
```

fplus(10) save()

C'est écrit. Essayons de voir.

J'ai déjà `sqlite3` installé sur mon ordinateur.

```shell
$ sqlite3
SQLite version 3.32.3 2020-06-18 14:16:19
Entrez ".help" pour des conseils d'utilisation.
Connecté à une base de données temporaire en mémoire.
Utilisez ".open FILENAME" pour rouvrir sur une base de données persistante.

sqlite> .help
.auth ON|OFF             Afficher les rappels de l'autorisation
```

```
.backup ?DB? FILE        Sauvegarder la base de données DB (par défaut "main") dans FILE
.bail on|off             Arrêter après une erreur. Par défaut OFF
.binary on|off           Activer ou désactiver la sortie binaire. Par défaut OFF
.cd DIRECTORY            Changer le répertoire de travail en DIRECTORY
.changes on|off          Afficher le nombre de lignes modifiées par SQL
.check GLOB              Échouer si la sortie depuis .testcase ne correspond pas
.clone NEWDB             Cloner les données dans NEWDB à partir de la base de données existante
.databases               Lister les noms et fichiers des bases de données attachées
.dbconfig ?op? ?val?     Lister ou modifier les options de sqlite3_db_config()
.dbinfo ?DB?             Afficher les informations d'état sur la base de données
.dump ?TABLE?            Rendre le contenu de la base de données sous forme de SQL
.echo on|off             Activer ou désactiver l'écho des commandes
.eqp on|off|full|...     Activer ou désactiver l'EXPLAIN QUERY PLAN automatique
.excel                   Afficher la sortie de la commande suivante dans un tableur
.exit ?CODE?             Quitter ce programme avec le code de retour CODE
.expert                  EXPÉRIMENTAL. Suggérer des index pour les requêtes
.explain ?on|off|auto?   Changer le mode de formatage EXPLAIN. Par défaut : auto
.filectrl CMD ...        Exécuter diverses opérations sqlite3_file_control()
.fullschema ?--indent?   Afficher le schéma et le contenu des tables sqlite_stat
.headers on|off          Activer ou désactiver l'affichage des en-têtes
.help ?-all? ?PATTERN?   Afficher l'aide pour PATTERN
.import FILE TABLE       Importer des données de FILE dans TABLE
.imposter INDEX TABLE    Créer une table imposte TABLE sur l'index INDEX
.indexes ?TABLE?         Afficher les noms des index
.limit ?LIMIT? ?VAL?     Afficher ou modifier la valeur d'une limite SQLITE_LIMIT
.lint OPTIONS            Signaler les problèmes potentiels de schéma
.log FILE|off            Activer ou désactiver la journalisation. FILE peut être stderr/stdout
.mode MODE ?TABLE?       Définir le mode de sortie
.nullvalue STRING        Utiliser STRING à la place des valeurs NULL
.once ?OPTIONS? ?FILE?   Sortir la prochaine commande SQL uniquement dans FILE
.open ?OPTIONS? ?FILE?   Fermer la base de données existante et rouvrir FILE
.output ?FILE?           Envoyer la sortie vers FILE ou stdout si FILE est omis
.parameter CMD ...       Gérer les liaisons de paramètres SQL
.print STRING...         Imprimer la chaîne littérale STRING
.progress N              Invoquer le gestionnaire de progression après chaque N opcodes
.prompt MAIN CONTINUE    Remplacer les invites standard
```

```
.quit                    Quitter ce programme
.read FILE               Lire l'entrée depuis FILE
.recover                 Récupérer autant de données que possible d'une base de données corrompue
.restore ?DB? FILE       Restaurer le contenu de DB (par défaut "main") depuis FILE
.save FILE               Écrire la base de données en mémoire dans FILE
.scanstats on|off        Activer ou désactiver les métriques sqlite3_stmt_scanstatus()
.schema ?PATTERN?        Afficher les instructions CREATE correspondant à PATTERN
.selftest ?OPTIONS?      Exécuter les tests définis dans la table SELFTEST
.separator COL ?ROW?     Changer les séparateurs de colonne et de ligne
.session ?NAME? CMD ...  Créer ou contrôler des sessions
.sha3sum ...             Calculer un hash SHA3 du contenu de la base de données
.shell CMD ARGS...       Exécuter CMD ARGS... dans un shell système
.show                    Afficher les valeurs actuelles de divers paramètres
.stats ?on|off?          Afficher les statistiques ou activer/désactiver les statistiques
.system CMD ARGS...      Exécuter CMD ARGS... dans un shell système
.tables ?TABLE?          Lister les noms des tables correspondant au motif LIKE TABLE
.testcase NAME           Commencer à rediriger la sortie vers 'testcase-out.txt'
.testctrl CMD ...        Exécuter diverses opérations sqlite3_test_control()
.timeout MS              Essayer d'ouvrir les tables verrouillées pendant MS millisecondes
.timer on|off            Activer ou désactiver le chronomètre SQL
.trace ?OPTIONS?         Afficher chaque instruction SQL lors de son exécution
.vfsinfo ?AUX?           Informations sur le VFS de premier niveau
.vfslist                 Lister tous les VFS disponibles
.vfsname ?AUX?           Afficher le nom de la pile VFS
.width NUM1 NUM2 ...     Définir la largeur des colonnes pour le mode "column"
```

Vous pouvez voir qu'il y a de nombreuses commandes. Parmi elles, `.quit` signifie quitter.

Si vous ne l'avez pas, vous pouvez le télécharger sur le site officiel ou exécuter la commande `brew install sqlite` pour l'installer.

```
$ sqlite3 fib.db

sqlite> show tables
   ...> ;
Erreur : près de "show" : erreur de syntaxe
sqlite> tables;
Erreur : près de "tables" : erreur de syntaxe
```

```
sqlite> .schema
CREATE TABLE vs(v text);
```

Au début, je pensais que c'était comme avec `MySQL`. Je croyais pouvoir utiliser `show tables` pour voir quelles tables existaient. Plus tard, j'ai découvert que dans `SQLite`, c'est différent. `MySQL` est un autre type de base de données, que nous allons également apprendre dans le futur.

```
sqlite> select * from vs;
0
1
1
2
3
5
8
13
21
34
55
```

En effet, nous avons correctement écrit les données. Notez que nous avons utilisé `text`, car nos nombres sont très grands et pourraient ne pas être conservés dans un type entier de la base de données.

```
import sqlite3

v = []
for x in range(1000000):
   v.append(-1)

def fplus(n):
   if n > 800:
      fplus(n-800)
      return f(n)
   else:
      return f(n)
```

```
def f(n):
   if v[n] != -1:
      return v[n]
   else:
      a = 0
      if n < 2:
         a = n
      else:
         a = f(n-1) + f(n-2)
      v[n] = a
      return v[n]

def create_table(cur: sqlite3.Connection):
    cur.execute('CREATE TABLE vs(v text)')

def read():
    con = sqlite3.connect('fib.db')
    cur = con.cursor()
    create_table(cur)
    i = 0
    for row in cur.execute('SELECT * from vs'):
         v[i] = int(row)
    con.close()

def save():
    con = sqlite3.connect('fib.db')
    cur = con.cursor()
    create_table(cur)
    for vv in v:
        if vv != -1:
            cur.execute('INSERT INTO vs VALUES(' + str(vv) + ')')
        else:
            break
    con.commit()
    con.close()

read()
```

```
for i in range(10):
    print(v[i])
```

Nous continuons en ajoutant la fonction `read`. Cependant, après l'exécution, une erreur s'est produite.

```
$ python fib_db.py
  ...
  File "fib_db.py", line 27, in create_table
    cur.execute('CREATE TABLE vs(v text)')
sqlite3.OperationalError: la table vs existe déjà
```

Nous ne pouvons plus créer la table, car elle existe déjà. Modifions légèrement la syntaxe.

```
def create_table(cur: sqlite3.Connection):
    cur.execute('CREATE TABLE IF NOT EXISTS vs(v text)')
```

Cependant, une erreur est survenue.

```
    v[i] = int(row)
TypeError: l'argument de int() doit être une chaîne de caractères, un objet de type bytes ou un nombre,
```

`tuple` est une structure de données en Python qui est similaire à une liste, mais immuable (c'est-à-dire qu'elle ne peut pas être modifiée après sa création). Lorsque vous dites que la fonction `row` retourne un `tuple`, cela signifie qu'elle renvoie une séquence ordonnée d'éléments qui ne peut pas être modifiée.

Pour mieux comprendre, imprimons le contenu du `tuple` retourné par `row` :

```
print(row)
```

Cela affichera les éléments du `tuple` retourné par la fonction `row`.

```
    for row in cur.execute('SELECT * from vs'):
        print(row)
        v[i] = int(row)
```

Le résultat est :

```
('0',)
```

En fait, un `tuple` est assez similaire à un tableau. La différence est que ses éléments peuvent être de types différents, contrairement à un tableau où tous les éléments doivent être du même type.

```
def read():
    con = sqlite3.connect('fib.db')
    cur = con.cursor()
    create_table(cur)
    i = 0
    for row in cur.execute('SELECT * from vs'):
        v[i] = int(row[0])
    con.close()
```

Voici la modification. Cependant, c'est assez étrange. La sortie est la suivante :

```
55
-1
-1
-1
-1
-1
-1
-1
-1
-1
```

Il s'avère que notre `i` ne s'incrémente pas.

```
    for row in cur.execute('SELECT * from vs'):
        v[i] = int(row[0])
        i += 1
```

C'est parfait ainsi.

```
0
1
```

```
1
2
3
5
8
13
21
34
```

Cependant, nous avons remarqué que lorsque les nombres sont très grands, ils sont stockés dans la base de données de cette manière :

```
4660046610375530309
7540113804746346429
1.22001604151219e+19
1.97402742198682e+19
3.19404346349901e+19
```

En relançant, voici ce que l'on obtient.

```
$ python fib_db.py
Traceback (most recent call last):
  File "fib_db.py", line 35, in read
    v[i] = int(row[0])
ValueError: valeur littérale invalide pour int() en base 10 : '1.22001604151219e+19'
```

Modifiez-le :

```
cur.execute("INSERT INTO vs VALUES('" +str(vv) + "')")
```

J'ai remarqué que nous avons modifié les guillemets simples autour de l'instruction `INSERT` en guillemets doubles, tout en ajoutant des guillemets à notre chaîne de caractères numérique. Si nous avions écrit cela ainsi auparavant, la base de données aurait interprété notre chaîne comme un nombre. Maintenant, en utilisant ces guillemets, cela indique qu'il s'agit d'une chaîne de caractères.

Ensuite, c'est correct. Cependant, comment effacer les données erronées précédentes ?

```
$ sqlite3 fib.db
SQLite version 3.32.3 2020-06-18 14:16:19
Entrez ".help" pour des indications d'utilisation.
sqlite> delete * from vs;
```

Ensuite, vous pouvez essayer d'autres instructions. `Créer`, `Supprimer`, `Mettre à jour`, `Lire` (CRUD). Nous avons ici donné des exemples pour `Créer`, `Supprimer`, `Lire`.

Exercice

- Les étudiants explorent de manière similaire comme indiqué ci-dessus.