वेब प्रोग्रामिंग का परिचय

पिछले भाग में हमने □□□□□□□□□□ □□□□□□□□□ (फिबोनाची अनुक्रम) के कार्य को □□□□□□□-□□□□□□□□□ (ऑब्जेक्ट-ओरिएंटेड) संस्करण में बदल दिया था और एक टर्मिनल इंटरफेस को लागू किया था।

server.py□

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

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

यह एक □□□□□□□ क्लास Server है जिसमें एक कंस्ट्रक्टर __init__ है। यह कंस्ट्रक्टर handlerClass नामक एक पैरामीटर लेता है और इसे क्लास के इंस्टेंस वेरिएबल self.handlerClass में स्टोर करता है।

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

यह कोड एक अनंत लूप में चलता है, जहां यह उपयोगकर्ता से इनपुट लेता है और फिर handlerClass के handle मेथड को कॉल करता है। यह प्रक्रिया तब तक दोहराई जाती है जब तक प्रोग्राम को बंद नहीं किया जाता।

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()
```

सरल वेब सर्वर

□□□ इंटरफ़ेस में कैसे बदलें?

हम ऊपर दिए गए `Server` को `HTTP` के `Server` से बदल देंगे। आइए पहले देखें कि □□□□□□ में `HTTP` कैसा दिखता है।

□□□□□□ के मानक पुस्तकालय में एक वेब सर्वर प्रदान किया गया है।

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

इसे टर्मिनल में चलाएं।

```
$ python -m http.server
HTTP   ::    8000            (http://[::]:8000/) ...
```

ब्राउज़र में खोलें और आप प्रभाव देख सकते हैं।

यह वर्तमान निर्देशिका को सूचीबद्ध करता है। फिर जब आप इस वेबपेज को ब्राउज़ करते हैं, तो टर्मिनल पर वापस जाएं। यह काफी दिलचस्प है।

```
$ python -m http.server
HTTP   ::    8000            (http://[::]:8000/) ...
::1 - - [07/Mar/2021 15:30:35] "GET / HTTP/1.1" 200 -
::1 - - [07/Mar/2021 15:30:35]    404,
::1 - - [07/Mar/2021 15:30:35] "GET /favicon.ico HTTP/1.1" 404 -
::1 - - [07/Mar/2021 15:30:35]    404,
::1 - - [07/Mar/2021 15:30:35] "GET /apple-touch-icon-precomposed.png HTTP/1.1" 404 -
::1 - - [07/Mar/2021 15:30:35]    404,
::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 -
```

यह वेब पेज एक्सेस लॉग है। इसमें `GET` वेब सर्विस के लिए एक प्रकार का डेटा एक्सेस ऑपरेशन को दर्शाता है। `HTTP/1.1` यह दर्शाता है कि `HTTP` के `1.1` संस्करण का प्रोटोकॉल उपयोग किया गया है।

इसका उपयोग करके हम अपनी फिबोनाची अनुक्रम सेवा कैसे बना सकते हैं। पहले ऑनलाइन उदाहरण कोड ढूंढें, थोड़ा संशोधित करें, और एक सरलतम वेब सर्वर लिखें:

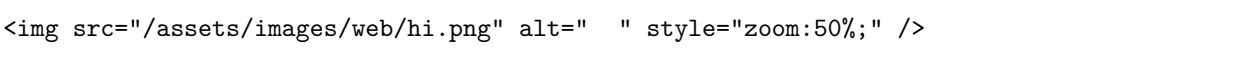
```
from http.server import SimpleHTTPRequestHandler, HTTPServer

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()
```

(,)

  ? `Server` `SimpleHTTPRequestHandler` ,

 `127.0.0.1` , `8000` ? ,



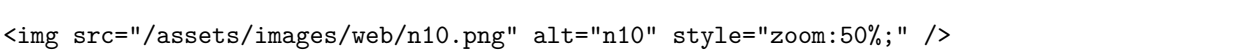
 ,

```python
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"))
```

यह कोड एक `Handler` क्लास को परिभाषित करता है जो `SimpleHTTPRequestHandler` से विरासत में मिलता है। यह क्लास □□□□ □□□ अनुरोधों को संभालता है। जब कोई □□□ अनुरोध प्राप्त होता है, तो यह 200 स्थिति कोड भेजता है और '□□□□□□□□-□□□□' हेडर को '□□□□' के रूप में सेट करता है। फिर यह □□□ के क्वेरी स्ट्रिंग को पार्स करता है और यदि क्वेरी स्ट्रिंग में कोई मान होता है, तो उसे पूर्णांक में परिवर्तित करता है और एक फ़ंक्शन `f` को कॉल करता है। अंत में, यह परिणाम को क्लाइंट को वापस भेजता है।

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

server.serve_forever()
```

```
(        ,                    )
```



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

इस कोड में, `self.path` एक □□□ पथ को दर्शाता है जो `/?n=3` है। `parsed` वेरिएबल `ParseResult` ऑब्जेक्ट को दर्शाता है जो □□□ के विभिन्न भागों को पार्स करता है। `qs` एक डिक्शनरी है जो क्वेरी स्ट्रिंग को दर्शाता है, जहां `n` की वैल्यू `['3']` है। `ns` एक लिस्ट है जो `n` की वैल्यू को दर्शाती है, और `n` वेरिएबल `3` को दर्शाता है।

रिकर्सन का उन्नत स्तर

आइए कोड को थोड़ा सा रिफैक्टर करें।

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

यह कोड □□□□□□ में एक सरल □□□□ सर्वर बनाने के लिए है। इसमें `SimpleHTTPRequestHandler` और `HTTPServer` क्लासेस का उपयोग किया गया है, जो □□□□□□ के `http.server` मॉड्यूल से आते हैं। साथ ही, `fib` मॉड्यूल से `f` फ़ंक्शन को इम्पोर्ट किया गया है, और `urllib.parse` मॉड्यूल से `urlparse` और `parse_qs` फ़ंक्शन्स को इम्पोर्ट किया गया है।

```
class Handler(SimpleHTTPRequestHandler):
```

यह एक □□□□□□ कोड स्निपेट है जो `SimpleHTTPRequestHandler` क्लास को इनहेरिट करके एक नई `Handler` क्लास बनाता है। इसे ट्रांसलेट करने की आवश्यकता नहीं है क्योंकि यह कोड है और कोड को ट्रांसलेट नहीं किया जाता है।

```
    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()
```

व्याख्या:

- `parse_n` फ़ंक्शन एक □□□ स्ट्रिंग `s` को पार्स करता है और उसके क्वेरी पैरामीटर से `n` का मान निकालता है। यदि `n` मौजूद है और वैध है, तो इसे पूर्णांक में परिवर्तित करके वापस करता है। अन्यथा, `None` वापस करता है।
- `do_GET` फ़ंक्शन एक □□□□ □□□ अनुरोध को संसाधित करता है। यह 200 □□ स्थिति कोड भेजता है और `Content-type` हेडर को `text` के रूप में सेट करता है, फिर हेडर को समाप्त करता है।

```
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"))
```

इस कोड में:

- `result` एक खाली स्ट्रिंग के रूप में शुरू होता है।
- `n` को `self.path` से पार्स करके प्राप्त किया जाता है।
- यदि `n` `None` नहीं है, तो `result` को `f(n)` के मान को स्ट्रिंग में बदलकर सेट किया जाता है।
- अंत में, `result` को UTF-8 एन्कोडिंग में बाइट्स में बदलकर `self.wfile` में दो बार लिखा जाता है।

□□□□□□□ = □□□□□□□□□□□□((“127.0.0.1”, 8000), □□□□□□□□)

```
server.serve_forever()
```

```
(         ,                    )
```

```
`parse_n`           ,             `n`
```

```
                Xiao Wang   Fibonacci sequence   10000            ,         , Xiao Ming     Fibonacci sequ
```

```
          `n`     ,    `f(n)`
```

```
```shell
127.0.0.1 - - [10/Mar/2021 00:33:01] "GET /?n=1000 HTTP/1.1" 200 -
----------------------------------------
                    ('127.0.0.1', 50783)
Traceback (          ):
    ...
    if v[n] != -1:
IndexError:
```

अगर मूल सरणी (□□□□□□) पर्याप्त बड़ी नहीं है, तो □ सरणी को 10000 तक बदल दें।

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

यह कोड एक खाली सूची `v` बनाता है और फिर 0 से 9999 तक के सभी पूर्णांकों के लिए `v` में `-1` जोड़ता है। इसका परिणाम एक सूची होगी जिसमें 10000 बार `-1` होगा।

हालांकि, जब `n` का मान `2000` होता है, तो रिकर्सन की गहराई (□□□□□□□□□□ □□□□□□) से संबंधित एक त्रुटि (□□□□□□) उत्पन्न होती है:

```
127.0.0.1 - - [10/Mar/2021 00:34:00] "GET /?n=2000 HTTP/1.1" 200 -
----------------------------------------
```
```

```
                    ('127.0.0.1', 50821)
Traceback (           ):
    ...
    if v[n] != -1:
RecursionError:
```

हालांकि, यह सब काफी तेज़ी से हुआ।

क्यों। क्योंकि `f(1)` से `f(1000)` तक, हर एक को केवल एक बार ही गणना करने की आवश्यकता है। इसका मतलब है कि जब `f(1000)` की गणना की जा रही होती है, तो + ऑपरेशन शायद केवल 1000 बार ही किया जाता है। हम जानते हैं कि `Python` में रिकर्सन की गहराई लगभग 1000 होती है। इसका मतलब है कि हम प्रोग्राम को इस तरह से ऑप्टिमाइज़ कर सकते हैं, अगर हमें 2000 की गणना करनी है, तो पहले 1000 की गणना करें। नहीं, ऐसा करने पर भी हो सकता है। अगर 2000 की गणना करनी है, तो पहले 1200 की गणना करें। अगर 1200 की गणना करनी है, तो पहले 400 की गणना करें।

400 और 1200 को इस तरह से गणना करने के बाद, 2000 की गणना करें, और रिकर्सन की गहराई लगभग 800 के आसपास होगी, जिससे रिकर्सन गहराई ओवरफ्लो त्रुटि नहीं होगी।

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

यह कोड एक खाली सूची `v` बनाता है और फिर एक लूप का उपयोग करके 0 से 999,999 तक के सभी नंबरों के लिए सूची में `-1` जोड़ता है। इसका परिणाम एक सूची होगी जिसमें 1,000,000 बार `-1` होगा।

```
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` फ़ंक्शन को जोड़ा गया है।

हालांकि, यह सोचने पर मजबूर करता है कि अगर `fplus` को 1000 बार पुनरावर्ती रूप से कॉल किया जाए तो क्या होगा। 1000 * 800 = 800000। जब मैंने □ को 800,000 पर सेट किया, तो फिर से पुनरावर्ती गहराई त्रुटि आ गई। थोड़ा और परीक्षण करने के बाद, पता चला कि मामला और भी जटिल है। हालांकि, इस ऑप्टिमाइज़ेशन के बाद, 2000 की गणना करना बहुत आसान हो गया है।

फ़ाइल पढ़ना और लिखना

लगता है कि विषय से भटक गया हूँ। वापस □□□ □□□□□□□□□□□ के विषय पर आते हैं। पहली बार `f(400)` का अनुरोध किया गया, दूसरी बार `f(600)` का। तो दूसरी बार अनुरोध करते समय, पहले अनुरोध से उत्पन्न `v` □□□□□ के मानों का उपयोग कर सकते हैं। हालांकि, जब हम प्रोग्राम को बंद कर देते हैं और फिर से शुरू करते हैं, तो उन मानों का उपयोग नहीं कर सकते। हमारी विधि के अनुसार, □□□□□□□□□ □□□□□□□□ की गणना बहुत तेज है। हालांकि, कल्पना कीजिए कि अगर यह बहुत धीमी हो तो क्या होगा। खासकर जब हमने `v` □□□□□ को शामिल नहीं किया होता है, तो बहुत सारी दोहराई जाने वाली गणनाएँ होती हैं। ऐसे में हम चाहते हैं कि जो मुश्किल से प्राप्त हुए परिणाम हैं, उन्हें सहेज लिया जाए।

इस समय, की अवधारणा को पेश किया जाता है। `v` सरणी यहाँ एक कैश है। हालांकि, यह केवल प्रोग्राम के जीवनकाल में मौजूद होता है। प्रोग्राम बंद होने के बाद, यह गायब हो जाता है। तो क्या करें? स्वाभाविक रूप से, हम इसे फ़ाइल में सहेजने के बारे में सोचेंगे।

□ ऐरे को फ़ाइल में कैसे सहेजें।

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

हमारा `v` ऐरे इस तरह से सहेजा जा सकता है। प्रत्येक पंक्ति को `n f(n)` के रूप में सहेजा जाता है। चूंकि `n` प्राकृतिक रूप से बढ़ता है, शायद हम केवल `f(n)` मानों को सहेज सकते हैं।

```
0
1
1
2
```

```
3
...
```

इसे आज़माएं।

```
f = open("demofile2.txt", "a")
f.write("              !")
f.close()
```

फ़ाइल को खोलें और उसे पढ़ें, जोड़ने के बाद:

□ = □□□□("□□□□□□□□□2.□□□", "□") □□□□□□(□.□□□□())

`open` `a` , ; `w` ,

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

(यह कोड हिंदी में अनुवाद नहीं किया जा सकता क्योंकि यह एक प्रोग्रामिंग कोड है जो पायथन भाषा में लिखा गया है।)

इसे चलाएं, और निश्चित रूप से `fib_v` नामक फ़ाइल मिल जाएगी।

`fib_v`:

```
hi
```

जब हम इसे फिर से चलाते हैं, तो यह इस तरह बदल जाता है।

```
hihi
```

कैसे नई लाइन शुरू करें।

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

(यह कोड हिंदी में अनुवाद नहीं किया जा सकता क्योंकि यह एक प्रोग्रामिंग कोड है और इसमें उपयोग किए गए फ़ंक्शन और सिंटैक्स को बदला नहीं जा सकता।)

यह एक बार प्रिंट होगा, `hihihi` दिखाई देगा, लेकिन नई लाइन नहीं दिखाई देगी। हालांकि, दूसरी बार प्रिंट करने पर, नई लाइन दिखाई देगी। इससे पता चलता है कि पहली बार प्रिंट करते समय नई लाइन का कैरेक्टर प्रिंट हो चुका था, लेकिन वह अंत में होने के कारण दिखाई नहीं दिया।

कैसे पढ़ें।

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

(यह कोड हिंदी में अनुवाद नहीं किया जा सकता क्योंकि यह एक प्रोग्रामिंग कोड है और इसमें उपयोग की गई भाषा और सिंटैक्स को बदलना संभव नहीं है।)

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

अगला कदम, हमारे फिबोनाची प्रोग्राम को संशोधित करें।

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

यह कोड एक खाली सूची `v` बनाता है और फिर एक लूप के माध्यम से 1,000,000 बार `-1` को इस सूची में जोड़ता है। अंत में, `v` सूची में 1,000,000 `-1` मान होंगे।

```
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])
```

यह कोड एक फ़ाइल `fib_v` को पढ़ता है और उसकी सामग्री को एक सूची `v` में संग्रहीत करता है। यहाँ `v` एक सूची है जिसे पहले से परिभाषित किया गया होना चाहिए। फ़ाइल की प्रत्येक पंक्ति को पढ़कर उसे पूर्णांक में परिवर्तित किया जाता है और `v` सूची में संग्रहीत किया जाता है।

```
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()
```

हिंदी अनुवाद:

```
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()
```

व्याख्या:

- `file = open('fib_v', 'w')`: 'fib_v' नाम की एक फ़ाइल को लिखने के लिए खोलता है।
- `s = ''`: एक खाली स्ट्रिंग `s` को इनिशियलाइज़ करता है।
- `start = True`: एक फ्लैग `start` को `True` पर सेट करता है।
- `for vv in v:`: लिस्ट `v` के प्रत्येक एलिमेंट `vv` के लिए लूप चलाता है।
  - `if vv == -1:`: यदि `vv` का मान `-1` है, तो लूप से बाहर निकल जाता है।

- `if start == False:`: यदि `start` `False` है, तो `s` में एक नई लाइन जोड़ता है।
- `start = False`: `start` को `False` पर सेट करता है।
- `s += str(vv)`: `vv` को स्ट्रिंग में बदलकर `s` में जोड़ता है।

 `file.write(s)`: स्ट्रिंग `s` को फ़ाइल में लिखता है।

 `file.close()`: फ़ाइल को बंद करता है।

```
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]
```

□□□□() □□□□□□□(10) □□□□()

, `fib_v`

`fib_v`:

```shell
```

0
1
1
2
3
5
8
13
21
34
55

ऊपर दिए गए विश्लेषण को देखकर थोड़ा परेशानी हो सकती है। `\n` एक नई लाइन (□□□□□□□□) का प्रतीक है। क्या इसका और सरल और एकीकृत तरीके से विश्लेषण करने का कोई तरीका है? लोगों ने `JSON` नामक एक डेटा प्रारूप का आविष्कार किया है।

□□□□

□□□□ (□□□□□□□□□□ □□□□□□ □□□□□□□□) एक हल्का डेटा इंटरचेंज फॉर्मेट है जो मानव-पठनीय और मशीन-पार्स करने योग्य होता है। यह □□□□□□□□□□ में ऑब्जेक्ट्स के सिंटैक्स पर आधारित है, लेकिन यह भाषा-स्वतंत्र है और कई प्रोग्रामिंग भाषाओं में इसका समर्थन किया जाता है।

□□□□ **सिंटैक्स**

□□□□ डेटा को □□□-□□□□□ पेयर्स के रूप में संग्रहीत करता है। यहाँ एक सरल □□□□ उदाहरण है:

```
{
  "name": "John Doe",
  "age": 30,
  "isStudent": false,
  "courses": ["Math", "Science", "History"],
  "address": {
    "street": "123 Main St",
    "city": "Anytown",
    "state": "CA"
  }
}
```

□□□□ **के मुख्य तत्व**

- **ऑब्जेक्ट्स**: {} के अंदर □□□-□□□□□ पेयर्स के रूप में संग्रहीत।
- **एरेज़**: [] के अंदर मानों की सूची।
- **मान**: स्ट्रिंग्स, नंबर्स, ऑब्जेक्ट्स, एरेज़, `true`, `false`, या `null` हो सकते हैं।

□□□□ **का उपयोग**

□□□□ का उपयोग आमतौर पर वेब एप्लिकेशन्स में सर्वर और क्लाइंट के बीच डेटा ट्रांसफर करने के लिए किया जाता है। यह □□□ रिस्पॉन्सेस और कॉन्फ़िगरेशन फाइल्स के लिए भी लोकप्रिय है।

□□□□ **और** □□□

□□□□ अक्सर □□□ की तुलना में हल्का और पढ़ने में आसान होता है, जिसके कारण यह आधुनिक वेब डेवलपमेंट में अधिक प्रचलित हो गया है।

□□□□ एक शक्तिशाली और लचीला डेटा फॉर्मेट है जो आधुनिक सॉफ्टवेयर डेवलपमेंट में व्यापक रूप से उपयोग किया जाता है।

□□□□ का पूरा नाम `JavaScript Object Notation` है। नीचे `JSON` का एक उदाहरण दिया गया है।

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

उपरोक्त तरीके से एक मैपिंग को दर्शाया जाता है।

□□□□ में निम्नलिखित मूल तत्व होते हैं:

1. संख्या या स्ट्रिंग
2. सूची
3. मैपिंग

और ये मूल तत्व किसी भी तरह से नेस्टेड किए जा सकते हैं। यानी कि एक सूची में दूसरी सूची हो सकती है। एक मैपिंग में भी सूची हो सकती है। और इसी तरह।

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

एक पंक्ति में लिखने और इस तरह से सुंदर ढंग से लिखने के बीच अर्थपूर्ण अंतर हो सकता है। शायद हम उनके कम्प्यूटेशनल ग्राफ की कल्पना कर सकते हैं। स्पेस उनके कम्प्यूटेशनल ग्राफ को प्रभावित नहीं करेगा।

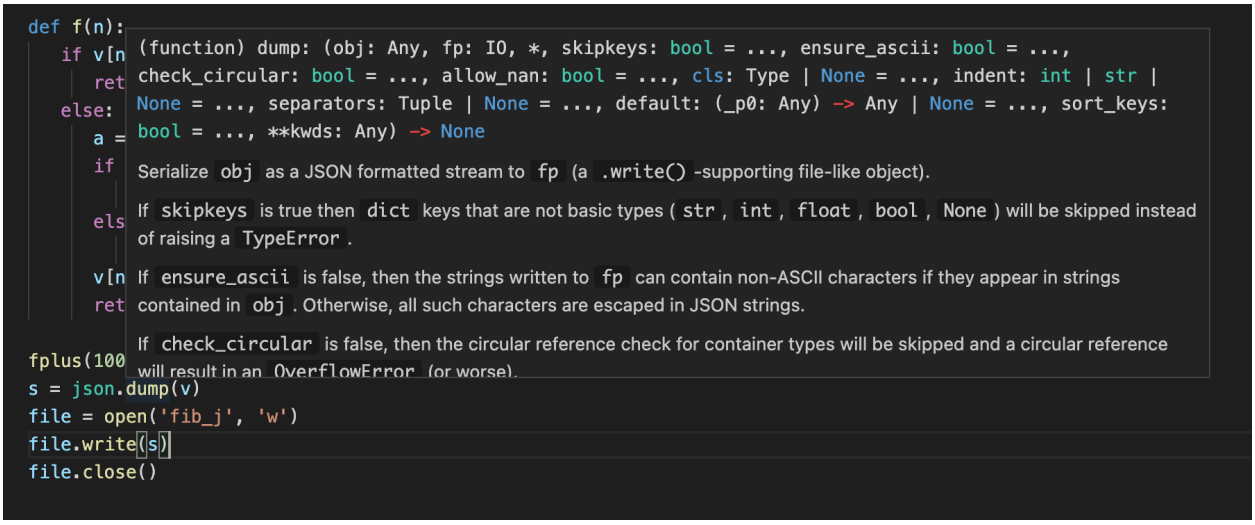
अगले चरण में हमें □ ऐरे को `json` फॉर्मेट की स्ट्रिंग में बदलना होगा।

```
import json

□ = [] □□□ □ □□ □□□□□(1000000): □.□□□□□□□(-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]
```

यह कोड एक फ़ंक्शन `f(n)` को परिभाषित करता है जो फिबोनैचि अनुक्रम की □वीं संख्या की गणना करता है। यह मेमोइज़ेशन (□□□□-□□□□□□□) तकनीक का उपयोग करता है ताकि पहले से गणना किए गए मानों को स्टोर किया जा सके और उन्हें दोबारा गणना करने से बचा जा सके। `v` एक सूची है जो पहले से गणना किए गए मानों को स्टोर करती है, और `-1` इंगित करता है कि उस स्थान का मान अभी तक गणना नहीं किया गया है।

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

जब हम इस तरह लिखते हैं, तो एक त्रुटि आती है: `TypeError: dump() missing 1 required positional argument: 'fp'` । vscode में हम फ़ंक्शन की परिभाषा को इस तरह देख सकते हैं।



□□□□□□□ 1: □□□□

आप `dump` पर माउस ले जा सकते हैं। यह बहुत सुविधाजनक है, है ना?

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

(ध्यान दें: कोड ब्लॉक को अनुवादित नहीं किया जाता है क्योंकि यह प्रोग्रामिंग भाषा का हिस्सा है और इसे अपरिवर्तित रहना चाहिए।)

100 तक की गणना करने पर प्रदर्शित संख्याएँ थोड़ी अधिक हैं, इसलिए यहाँ इसे 10 कर दिया गया है। मूल रूप से, `dump` के दूसरे पैरामीटर में `file` ऑब्जेक्ट पास करना पर्याप्त है।

इससे आप फ़ाइल देख सकते हैं:

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

(नोट: कोड ब्लॉक्स को अनुवादित नहीं किया जाता है, इसलिए यह वही रहता है।)

ध्यान दें कि बाद में बहुत सारे `-1` छोड़ दिए गए हैं।

```
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()
```

(नोट: कोड ब्लॉक को हिंदी में ट्रांसलेट नहीं किया गया है क्योंकि यह प्रोग्रामिंग कोड है और इसे अपरिवर्तित रखना उचित है।)

```
read()

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

जब ऐसा होता है, तो प्रिंट आउट दिखाई देता है:

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

इन कुछ फ़ंक्शनों को एक साथ जाँचें:

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

यह कोड एक फ़ाइल `fib_j` को पढ़ता है, उसकी सामग्री को □□□□ के रूप में पार्स करता है, और फिर उस डेटा को एक सूची `v` में स्टोर करता है।

```
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()
```

□□□□() □□□□□(100) □□□□()

, -

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

डेटाबेस

अगर डेटा बहुत बड़ा और संरचना बहुत जटिल है तो क्या करें? फ़ाइलों में सहेजने का तरीका धीमा और जटिल हो जाता है। यहीं पर का परिचय होता है। यह एक प्रोग्रामेबल `Excel` शीट की तरह है। यह एक ऐसी `Excel` शीट है जिसमें कोड के माध्यम से आसानी से डेटा जोड़ा, हटाया, संशोधित और खोजा जा सकता है।

आधिकारिक वेबसाइट के दस्तावेज़ में उदाहरण मिला।

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

(यह कोड ब्लॉक है, इसलिए इसे अनुवादित नहीं किया गया है।)

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

टेबल बनाएं

□□□.□□□□□□□□(’’’□□□□□□□ □□□□□□ □□□□□□□ (□□□□□ □□□□□, □□□□□□ □□□□□, □□□□□□□ □□□□□, □□□□ □□□□□, □□□□□□ □□□□□)” ’)

डेटा की एक पंक्ति डालें

□□□.□□□□□□□□(“□□□□□□□ □□□□□ □□□□□□□ □□□□□□□ (‘2006-01-05’,‘□□□’,‘□□□□□’,100,35.14)”)

परिवर्तनों को सहेजें (कमिट करें)

□□□.□□□□□□□()

हम कनेक्शन को बंद भी कर सकते हैं अगर हम इसके साथ काम कर चुके हैं।

बस यह सुनिश्चित कर लें कि कोई भी परिवर्तन कमिट हो चुके हैं, नहीं तो वे खो जाएंगे।

□□□.□□□□□□()

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

(यह कोड ब्लॉक को हिंदी में अनुवाद करने की आवश्यकता नहीं है क्योंकि यह प्रोग्रामिंग कोड है और इसे अपने मूल रूप में ही रहना चाहिए।)

cursor का मतलब होता है कर्सर, जो कि कर्सर की तरह ही होता है। ऊपर दिए गए कोड में डेटाबेस से कनेक्ट करना, टेबल बनाना, डेटा इन्सर्ट करना, परिवर्तनों को सबमिट करना, और कनेक्शन को बंद करने का मतलब है। अंत में दिया गया उदाहरण डेटा को क्वेरी करने का एक उदाहरण है।

```
import sqlite3

□ = [] □□□ □ □□ □□□□□□(1000000): □.□□□□□□□(-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()
```

हिंदी अनुवाद:

```
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()
```

व्याख्या:

- `save()` फ़ंक्शन एक □□□□□□□ डेटाबेस `fib.db` से कनेक्ट होता है।
- `create_table(cur)` फ़ंक्शन का उपयोग करके एक टेबल बनाई जाती है।
- `v` लिस्ट में मौजूद प्रत्येक मान `vv` के लिए, यदि `vv` -1 नहीं है, तो उसे `vs` टेबल में इन्सर्ट किया जाता है।
- यदि `vv` -1 है, तो लूप से बाहर निकल जाता है।
- अंत में, डेटाबेस में परिवर्तनों को सुरक्षित करने के लिए `con.commit()` किया जाता है और कनेक्शन को बंद कर दिया जाता है।

□□□□□(10) □□□□()

`sqlite3`

```shell
$ sqlite3
SQLite version 3.32.3 2020-06-18 14:16:19
".help"
       -
                  ".open FILENAME"

sqlite> .help
.auth ON|OFF
.backup ?DB? FILE        DB (    "main")   FILE
.bail on|off                              OFF
.binary on|off                                  OFF
.cd DIRECTORY                           DIRECTORY
.changes on|off          SQL
.check GLOB                 .testcase
.clone NEWDB                      NEWDB
.databases
.dbconfig ?op? ?val?     sqlite3_db_config()
.dbinfo ?DB?
.dump ?TABLE?                      SQL
.echo on|off
.eqp on|off|full|...          EXPLAIN QUERY PLAN
.excel
.exit ?CODE?                        -   CODE
.expert
.explain ?on|off|auto?   EXPLAIN              : auto
.filectrl CMD ...            sqlite3_file_control()
.fullschema ?--indent?        sqlite_stat
.headers on|off
.help ?-all? ?PATTERN?   PATTERN
.import FILE TABLE       FILE       TABLE
.imposter INDEX TABLE       INDEX          TABLE
.indexes ?TABLE?
```

```
.limit ?LIMIT? ?VAL?     SQLITE_LIMIT
.lint OPTIONS
.log FILE|off                          FILE stderr/stdout
.mode MODE ?TABLE?
.nullvalue STRING        NULL           STRING
.once ?OPTIONS? ?FILE?      SQL             FILE
.open ?OPTIONS? ?FILE?                FILE
.output ?FILE?                FILE    stdout        FILE
.parameter CMD ...       SQL
.print STRING...             STRING
.progress N                N
.prompt MAIN CONTINUE
.quit
.read FILE               FILE
.recover
.restore ?DB? FILE       DB (    "main")          FILE
.save FILE                           FILE
.scanstats on|off        sqlite3_stmt_scanstatus()
.schema ?PATTERN?        PATTERN          CREATE
.selftest ?OPTIONS?      SELFTEST
.separator COL ?ROW?
.session ?NAME? CMD ...
.sha3sum ...                        SHA3
.shell CMD ARGS...              CMD ARGS...
.show
.stats ?on|off?
.system CMD ARGS...             CMD ARGS...
.tables ?TABLE?          LIKE    TABLE
.testcase NAME                'testcase-out.txt'
.testctrl CMD ...            sqlite3_test_control()
.timeout MS                           MS
.timer on|off            SQL
.trace ?OPTIONS?            SQL
.vfsinfo ?AUX?             -    VFS
.vfslist                       VFSes
.vfsname ?AUX?           VFS
```

```
.width NUM1 NUM2 ...     "  "
```

आप देख सकते हैं कि यहाँ बहुत सारे कमांड हैं। इनमें से `.quit` का मतलब है बाहर निकलना।

यदि आपके पास □□□□□□ नहीं है, तो आप इसे आधिकारिक वेबसाइट से डाउनलोड कर सकते हैं, या `brew install sqlite` चलाकर इंस्टॉल कर सकते हैं।

```
$ sqlite3 fib.db

sqlite> show tables
   ...> ;
  : "show"    :
sqlite> tables;
  : "tables"    :
sqlite> .schema
CREATE TABLE vs(v text);
```

शुरुआत में मैंने सोचा कि यह `MySQL` की तरह होगा। `show tables` का उपयोग करके देख सकते हैं कि कौन-कौन से टेबल हैं। बाद में पता चला कि `SQLite` में यह अलग तरह से काम करता है। `MySQL` एक अलग प्रकार का डेटाबेस है, जिसे हम भविष्य में सीखेंगे।

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

सही है, हमने डेटा को सही ढंग से लिखा है। ध्यान दें कि हमने `text` का उपयोग किया है, क्योंकि हमारे पास बड़ी संख्याएं हैं और डेटाबेस के पूर्णांक प्रकार इसे संभाल नहीं सकते।

```
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)
```

यह कोड एक पुनरावर्ती (□□□□□□□□□□) फ़ंक्शन `fplus` को परिभाषित करता है। यदि `n` का मान 800 से अधिक है, तो यह फ़ंक्शन `fplus(n-800)` को कॉल करता है और फिर `f(n)` को वापस लौटाता है। अन्यथा, यह सीधे `f(n)` को वापस लौटाता है।

```
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]
```

यह कोड एक फ़ंक्शन `f(n)` को परिभाषित करता है जो फाइबोनैचि अनुक्रम की □वीं संख्या की गणना करता है। यहां `v` एक सूची है जिसका उपयोग पहले से गणना की गई संख्याओं को संग्रहीत करने के लिए किया जाता है। यदि `v[n]` का मान -1 नहीं है, तो यह पहले से ही गणना की गई संख्या को वापस कर देता है। अन्यथा, यह फाइबोनैचि अनुक्रम की गणना करता है और उसे `v[n]` में संग्रहीत करता है।

```
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])
```

हमने `read` फ़ंक्शन को जोड़ना जारी रखा। हालांकि, इसे चलाने के बाद, एक त्रुटि उत्पन्न हुई।

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

हम अब टेबल नहीं बना सकते, टेबल पहले से मौजूद है। सिंटैक्स को थोड़ा बदल दें।

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

(यह कोड ब्लॉक को हिंदी में अनुवाद करने की आवश्यकता नहीं है क्योंकि यह प्रोग्रामिंग कोड है और इसे अपने मूल रूप में ही रखना चाहिए।)

हालांकि, एक त्रुटि उत्पन्न हुई।

```
    v[i] = int(row)
TypeError: int()           ,    -                , 'tuple'
```

tuple क्या है। इसका मतलब है कि □□□ ने tuple वापस किया है। आइए इसे प्रिंट करके देखें।

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

(यह कोड ब्लॉक को हिंदी में अनुवाद करने की आवश्यकता नहीं है क्योंकि यह प्रोग्रामिंग कोड है और इसे अपरिवर्तित छोड़ दिया जाना चाहिए।)

परिणाम:

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

वास्तव में, tuple और □□□□□ लगभग समान होते हैं। हालांकि, इसके □□□□□□□□ एक दूसरे से अलग हो सकते हैं, जबकि □□□□□ के सभी □□□□□□□□ एक ही प्रकार के होने चाहिए।

```
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()
```

(यह कोड पायथन में लिखा गया है और इसे हिंदी में अनुवाद करने की आवश्यकता नहीं है, क्योंकि यह एक प्रोग्रामिंग कोड है और इसे उसी रूप में रखना चाहिए।)

इसे इस तरह बदलें। हालांकि, यह बहुत अजीब है। आउटपुट इस प्रकार है:

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

```
-1
-1
-1
-1
```

हमारा `i` अपने आप नहीं बढ़ रहा था।

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

(यह कोड ब्लॉक को हिंदी में अनुवाद करने की आवश्यकता नहीं है क्योंकि यह प्रोग्रामिंग कोड है और इसे अपरिवर्तित छोड़ दिया जाना चाहिए।)

यह सही है।

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

हालांकि, हमने देखा कि जब संख्याएँ बहुत बड़ी होती हैं, तो डेटाबेस में उन्हें इस तरह से सहेजा जाता है:

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

फिर से चलाने पर यह ऐसा दिखता है।

```
$ python fib_db.py
```

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

बदल दो:

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

(यह कोड ब्लॉक को हिंदी में अनुवाद करने की आवश्यकता नहीं है क्योंकि यह एक प्रोग्रामिंग कोड है और इसे अपरिवर्तित रखा जाना चाहिए।)

यहाँ हमने देखा कि हमने INSERT स्टेटमेंट के दोनों तरफ के सिंगल कोट्स को डबल कोट्स में बदल दिया है, और साथ ही हमारे नंबर स्ट्रिंग को कोट्स में लपेट दिया है। अगर पहले ऐसा लिखा जाता, तो डेटाबेस हमारे स्ट्रिंग को नंबर के रूप में समझता, लेकिन अब, इसे कोट्स में लपेटने से यह स्ट्रिंग के रूप में माना जाएगा।

फिर सब कुछ सही हो जाएगा। हालांकि, पहले की गलत डेटा को कैसे साफ़ किया जाए।

```
$ sqlite3 fib.db
SQLite version 3.32.3 2020-06-18 14:16:19
        ".help"
sqlite> delete * from vs;
```

अगले चरण में अन्य कथनों को आज़मा सकते हैं। , , , । हमने यहां , , के उदाहरण दिए हैं।

अभ्यास

- छात्रों को ऊपर दिए गए तरीके से इसी तरह की खोज करनी चाहिए।