ウェブサイトの内容をスクレイピングする

既に多くの既存ツールがウェブサイトのコンテンツをスクレイピングするために利用可能です。しかし、それらを使用するだけでは、背後にあるプロセスを深く理解することはできません。仕事で複雑なサイトや特殊なサイトに遭遇した場合、これらのツールでは望む結果が得られないこともあります。私たちは、それらをよりよく学び、より効果的に活用するために、自ら「車輪を再発明」する必要があるのです。

また、既存のツールも見てみましょう。

データマイナー

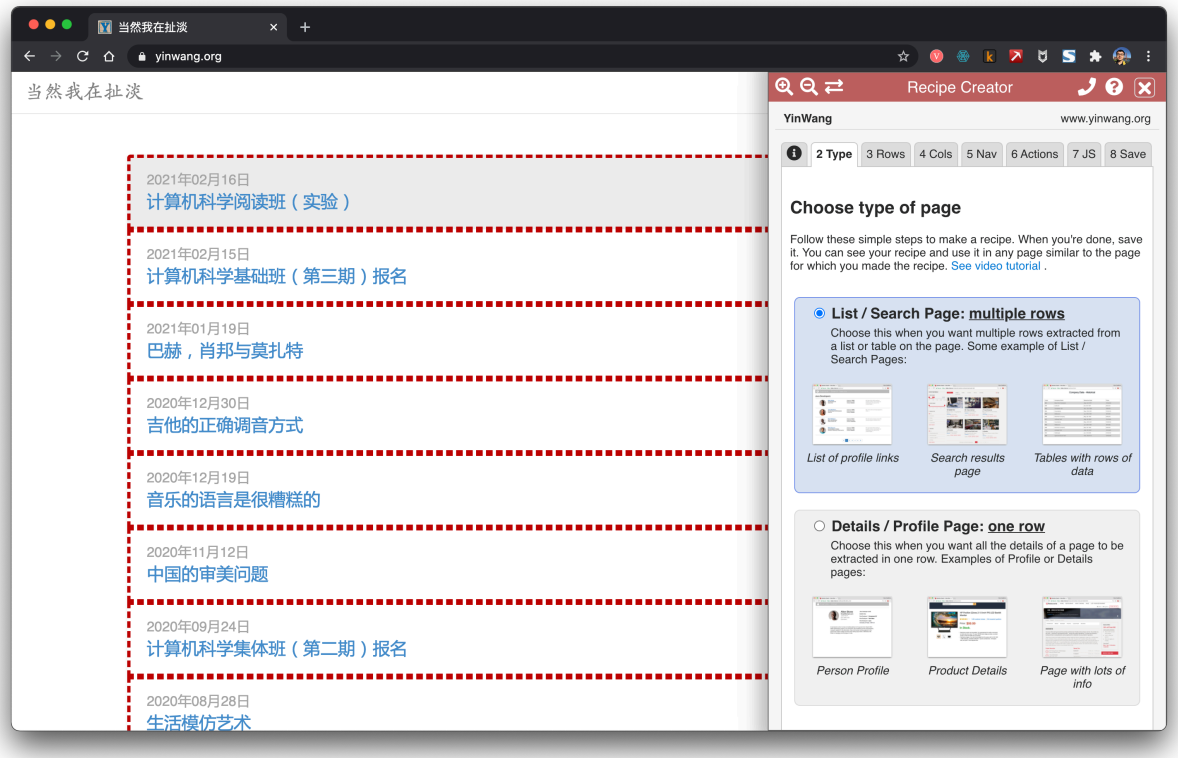

Figure 1: miner

`Data Miner` は、Chrome 上の非常に便利な拡張機能です。リンクやコンテンツを簡単にスクレイピングするのに役立ちます。

getbook

getbook は、電子書籍を作成するための非常に便利なツールです。

```
pip install getbook
```

```
{
  "title": " 本のタイトル",
  "author": " 著者名",
  "description": " 本の説明",
  "language": "ja",
  "plugins": ["plugin1", "plugin2"]
}
```

```
{
  "uid": "book",
  "title": "Hello World",
  "author": "Armin",
  "chapters": [
    "http://lucumr.pocoo.org/2018/7/13/python/",
    "http://lucumr.pocoo.org/2017/6/5/diversity-in-technology",
  ]
}
```

```
getbook -f ./book.json --mobi
```

このコマンドは、指定された book.json ファイルから書籍を取得し、Mobi 形式で出力するものです。

これにより、いくつかのリンクを電子書籍として簡単にまとめることができます。Data Miner と getbook を使用することで、一つはリンクをクロールし、もう一つはリンクを電子書籍に変換するため、電子書籍の作成が非常に便利になります。

ファインマン物理学講義

「プロジェクト実践：ファインマン物理学講義のウェブページを電子書籍にする」の章では、mathjax でレンダリングされた html ウェブページを電子書籍にする方法を学びました。ここでは、このプロジェクトを続けて、すべてのウェブページを取得する方法を見ていきます。ファインマン物理学講義は全 3 巻です。上の画像は第 1 巻の目次です。

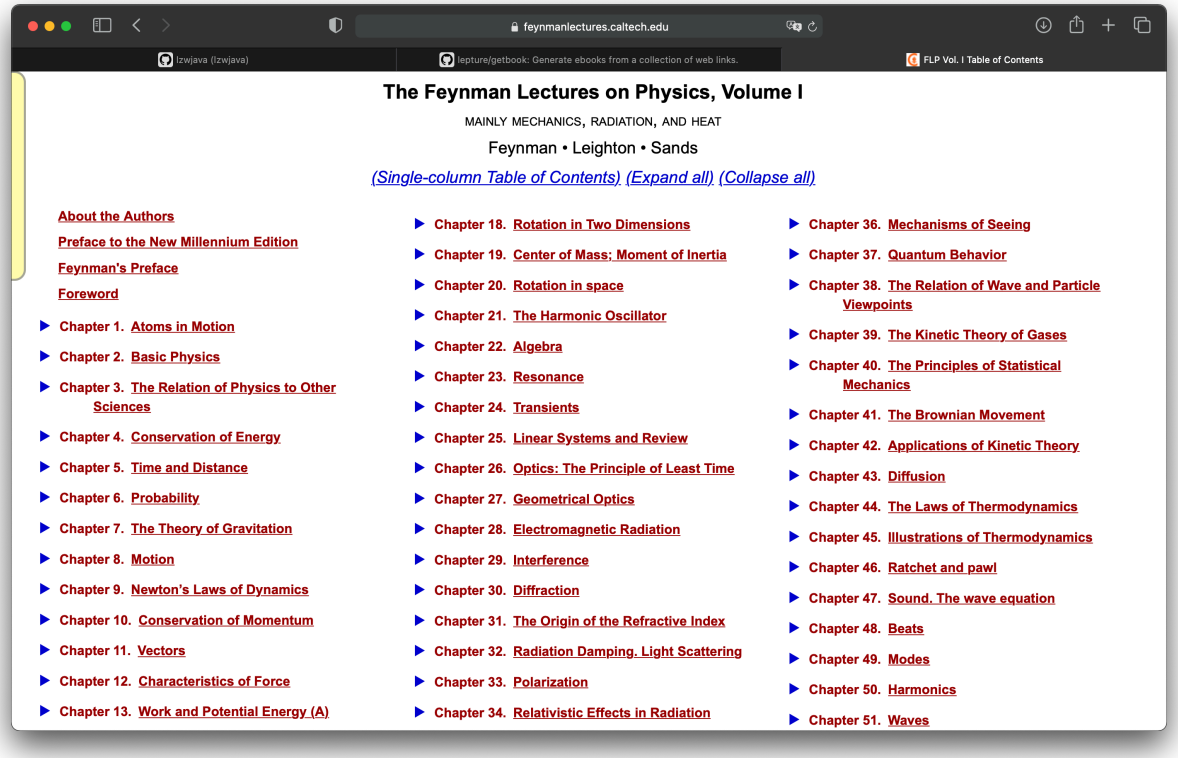
feynmanlectures.caltech.edu
lzwjava (lzwjava)
lepture/getbook: Generate ebooks from a collection of web links.
FLP Vol. I Table of Contents
The Feynman Lectures on Physics, Volume I
MAINLY MECHANICS, RADIATION, AND HEAT
Feynman • Leighton • Sands
(Single-column Table of Contents) (Expand all) (Collapse all)
About the Authors
Preface to the New Millennium Edition
Feynman's Preface
Foreword
Chapter 1. Atoms in Motion
Chapter 2. Basic Physics
Chapter 3. The Relation of Physics to Other Sciences
Chapter 4. Conservation of Energy
Chapter 5. Time and Distance
Chapter 6. Probability
Chapter 7. The Theory of Gravitation
Chapter 8. Motion
Chapter 9. Newton's Laws of Dynamics
Chapter 10. Conservation of Momentum
Chapter 11. Vectors
Chapter 12. Characteristics of Force
Chapter 13. Work and Potential Energy (A)
Chapter 18. Rotation in Two Dimensions
Chapter 19. Center of Mass; Moment of Inertia
Chapter 20. Rotation in space
Chapter 21. The Harmonic Oscillator
Chapter 22. Algebra
Chapter 23. Resonance
Chapter 24. Transients
Chapter 25. Linear Systems and Review
Chapter 26. Optics: The Principle of Least Time
Chapter 27. Geometrical Optics
Chapter 28. Electromagnetic Radiation
Chapter 29. Interference
Chapter 30. Diffraction
Chapter 31. The Origin of the Refractive Index
Chapter 32. Radiation Damping. Light Scattering
Chapter 33. Polarization
Chapter 34. Relativistic Effects in Radiation
Chapter 36. Mechanisms of Seeing
Chapter 37. Quantum Behavior
Chapter 38. The Relation of Wave and Particle Viewpoints
Chapter 39. The Kinetic Theory of Gases
Chapter 40. The Principles of Statistical Mechanics
Chapter 41. The Brownian Movement
Chapter 42. Applications of Kinetic Theory
Chapter 43. Diffusion
Chapter 44. The Laws of Thermodynamics
Chapter 45. Illustrations of Thermodynamics
Chapter 46. Ratchet and pawl
Chapter 47. Sound. The wave equation
Chapter 48. Beats
Chapter 49. Modes
Chapter 50. Harmonics
Chapter 51. Waves

Figure 2: fl

> http.client — HTTP プロトコルクライアント
>
> ソースコード: Lib/http/client.py
>
> このモジュールは、HTTP および HTTPS プロトコルのクライアント側を実装するクラスを定義しています。通常、直接使用されることはありません — `urllib.request` モジュールが HTTP および HTTPS を使用する URL を処理するためにこれを使用します。
>
> 参考: より高レベルの HTTP クライアントインターフェースには、Requests パッケージが推奨されます。

`requests` はより高レベルのインターフェースであることがわかります。

```
import requests

def main():
    r = requests.get('https://api.github.com/user', auth=('user', 'pass'))
    print(r.status_code)
```

main()

```shell
401
```

```
import requests

def main():
    r = requests.get('https://github.com')
    print(r.status_code)
    print(r.text)
```

main()

```html
200
<html>
  ...
</html>
```

上記のコードブロックは、HTML の基本的な構造を示しています。200 は HTTP ステータスコードで、リクエストが成功したことを示します。<html> タグは HTML ドキュメントのルート要素であり、その中に他のすべての要素が含まれます。... の部分には、実際の HTML コンテンツが入ります。

試してみたところ、requests のインターフェースが使えることが確認できました。

```
<div class="toc-chapter" id="C03">
   <span class="triangle">
    â¶
   </span>
   <a class="chapterlink" href="javascript:Goto(1,3)">
    <span class="tag">
     第 3 章.
    </span>
    物理学と他の科学との関係
   </a>
   <div class="sections">
    <a href="javascript:Goto(1,3,1)">
     <span class="tag">
      3-1
     </span>
     序論
    </a>
    <a href="javascript:Goto(1,3,2)">
     <span class="tag">
      3-2
     </span>
     化学
    </a>
    <a href="javascript:Goto(1,3,3)">
     <span class="tag">
      3-3
     </span>
     生物学
    </a>
    <a href="javascript:Goto(1,3,4)">
```

```
 <span class="tag">
  3-4
 </span>
 天文学
</a>
<a href="javascript:Goto(1,3,5)">
 <span class="tag">
  3-5
 </span>
 地質学
</a>
<a href="javascript:Goto(1,3,6)">
 <span class="tag">
  3-6
 </span>
 心理学
</a>
<a href="javascript:Goto(1,3,7)">
 <span class="tag">
  3-7
 </span>
 どうしてそうなったのか？
</a>
</div>
</div>
```

これは目次ページ内の第三章節の `html` コードです。ここから各章節のリンクを取得したいと考えています。`<a href="javascript:Goto(1,3,7)">` とあるように、これは `javascript` のリンクです。

```
https://www.feynmanlectures.caltech.edu/I_03.html
```

次に、各章のパスが非常に規則的であることに気づきました。`I_03.html` は、第 1 巻の第 3 章を表しています。

```
import requests
from bs4 import BeautifulSoup
from multiprocessing import Process
```

```
def scrape(chapter):
    if chapter < 1 or chapter > 52:
        raise Exception(f'chapter {chapter}')
    chapter_str = '{:02d}'.format(chapter)
    url = f'https://www.feynmanlectures.caltech.edu/I_{chapter_str}.html'
    print(f'scraping {url}')
    r = requests.get(url)
    if r.status_code != 200:
        raise Exception(r.status_code)
    soup = BeautifulSoup(r.text, features='lxml')
    f = open(f'./chapters/I_{chapter_str}.html', 'w')
    f.write(soup.prettify())
    f.close()
```

このコードは、指定された章番号に基づいて Feynman Lectures のウェブページをスクレイピングし、その内容を HTML ファイルとして保存する関数です。以下にその動作を説明します。

1. **章番号のチェック**: 章番号が 1 から 52 の範囲内にあるかどうかを確認します。範囲外の場合は例外を発生させます。
2. **URL の生成**: 章番号を 2 桁の文字列に変換し、Feynman Lectures の該当する章の URL を生成します。
3. **ウェブページの取得**: `requests` ライブラリを使用して、生成した URL からウェブページの内容を取得します。
4. **ステータスコードの確認**: 取得したウェブページのステータスコードが 200（成功）でない場合、例外を発生させます。
5. **HTML の解析**: `BeautifulSoup` を使用して、取得した HTML を解析します。
6. **ファイルへの保存**: 解析した HTML を整形して、指定されたパスに HTML ファイルとして保存します。

この関数を使用することで、指定された章の内容を簡単に取得し、ローカルに保存することができます。

```
def main():
    for i in range(52):
        p = Process(target=scrape, args=(i+1))
        p.start()
        p.join()
```

```
main()
```

続いて、スクレイピングのコードについて書いていきます。ここでは `Process` を使用しています。

```
    raise RuntimeError('''
RuntimeError:
        現在のプロセスのブートストラップフェーズが完了する前に、
        新しいプロセスの開始が試みられました。

    これはおそらく、子プロセスを開始するために `fork` を使用しておらず、
    メインモジュールで適切なイディオムを使用するのを忘れていることを意味します:

if __name__ == '__main__':
    freeze_support()
    ...

    プログラムが実行ファイルを生成するためにフリーズされる予定がない場合、
    "freeze_support()" の行は省略できます。

```python
def main():
    for i in range(52):
        p = Process(target=scrape, args=(i+1,))
        p.start()
    p.join()
```

このコードは、Python の `multiprocessing` モジュールを使用して、52 個のプロセスを並列に実行するものです。各プロセスは `scrape` 関数を実行し、引数として `i+1` を渡します。最後に `p.join()` を呼び出すことで、すべてのプロセスが終了するのを待ちます。

```
if __name__ == "__main__":
    main()

def main():
    start = timeit.default_timer()
    ps = [Process(target=scrape, args=(i+1,)) for i in range(52)]
```
```

```
    for p in ps:
        p.start()
    for p in ps:
        p.join()
    stop = timeit.default_timer()
    print('Time: ', stop - start)

if __name__ == "__main__":
    main()
```

```
scraping https://www.feynmanlectures.caltech.edu/I_01.html
scraping https://www.feynmanlectures.caltech.edu/I_04.html
...
scraping https://www.feynmanlectures.caltech.edu/I_51.html
scraping https://www.feynmanlectures.caltech.edu/I_52.html
時間:  9.144841699秒
```

the most information in the fewest words? I believe it is the *atomic hypothesis* (or the atomic *fact* , or whatever you wish to call it) that *all things are made of atomsâlittle particles that move around in perpetual motion, attracting each other when they are a little distance apart, but repelling upon being squeezed into one another* . In that one sentence, you will see, there is an *enormous* amount of information about the world, if just a little imagination and thinking are applied.

Figure 1â1

To illustrate the power of the atomic idea, suppose that we have a drop of water a quarter of an inch on the side. If we look at it very closely we see nothing but waterâsmooth, continuous water. Even if we magnify it with the best optical microscope availableâroughly two thousand timesâthen the water drop will be roughly forty feet across, about as big as a large room, and if we looked rather closely, we would *still* see relatively smooth waterâbut here and there small football-shaped things swimming back and forth. Very interesting. These are paramecia. You may stop at this

Figure 3: 図

```
<div class="figure" id="Ch1-F1">
        <img src="img/FLP_I/f01-01/f01-01_tc_big.svgz">
        <div class="caption empty">
```

```
          <span class="tag">
           図 1-1
          </span>
         </div>
</div>
```

```
import requests
from bs4 import BeautifulSoup
from multiprocessing import Process
import timeit

def scrape(chapter):
    if chapter < 1 or chapter > 52:
        raise Exception(f'chapter {chapter}')
    chapter_str = '{:02d}'.format(chapter)
    url = f'https://www.feynmanlectures.caltech.edu/I_{chapter_str}.html'
    print(f'scraping {url}')
    r = requests.get(url)
    if r.status_code != 200:
        raise Exception(r.status_code)
    soup = BeautifulSoup(r.text, features='lxml')
    f = open(f'./chapters/I_{chapter_str}.html', 'w')
    f.write(soup.prettify())
    f.close()
```

このコードは、指定された章番号に基づいて、Feynman Lectures のウェブページをスクレイピングし、その内容を HTML ファイルとして保存する関数です。以下にその動作を説明します。

1. **引数のチェック**: `chapter` が 1 から 52 の範囲外の場合、例外を発生させます。
2. **URL の生成**: 章番号を 2 桁の文字列に変換し、それに基づいて URL を生成します。
3. **ウェブページの取得**: `requests` ライブラリを使用して、指定された URL からウェブページの内容を取得します。
4. **ステータスコードの確認**: レスポンスのステータスコードが 200（成功）でない場合、例外を発生させます。
5. **HTML の解析**: `BeautifulSoup` を使用して、取得した HTML を解析します。
6. **ファイルへの保存**: 解析した HTML を整形して、指定されたファイル名で保存します。

この関数を使用することで、指定された章の内容を簡単に取得し、ローカルに保存することができます。

```
def main():
    start = timeit.default_timer()
    ps = [Process(target=scrape, args=(i+1,)) for i in range(52)]
    for p in ps:
        p.start()
    for p in ps:
        p.join()
    stop = timeit.default_timer()
    print('Time: ', stop - start)

if __name__ == "__main__":
    main()
```

リンクを確認してください。

```
    imgs = soup.find_all('img')
    for img in imgs:
        print(img)
```

このコードは、BeautifulSoup を使って HTML からすべての画像タグ (<img>) を検索し、それぞれの画像タグを出力するものです。soup.find_all('img') は、HTML ドキュメント内のすべての <img> タグをリストとして返します。その後、for ループを使って各画像タグを順番に出力しています。

```
https://www.feynmanlectures.caltech.edu/I_01.html をスクレイピング中
<img id="TwitLink" src=""/>
<img id="FBLink" src=""/>
<img id="MailLink" src=""/>
<img id="MobileLink" src=""/>
<img id="DarkModeLink" src=""/>
<img id="DesktopLink" src=""/>
<img src="img/camera.svg"/>
<img src="img/FLP_I/f01-00/f01-00.jpg"/>
<img data-src="img/FLP_I/f01-01/f01-01_tc_big.svgz"/>
```

```
<img data-src="img/FLP_I/f01-02/f01-02_tc_big.svgz"/>
<img data-src="img/FLP_I/f01-03/f01-03_tc_big.svgz"/>
<img data-src="img/FLP_I/f01-04/f01-04_tc_big.svgz"/>
<img data-src="img/FLP_I/f01-05/f01-05_tc_big.svgz"/>
<img data-src="img/FLP_I/f01-06/f01-06_tc_big.svgz"/>
<img class="first" data-src="img/FLP_I/f01-07/f01-07_tc_iPad_big_a.svgz"/>
<img class="last" data-src="img/FLP_I/f01-07/f01-07_tc_iPad_big_b.svgz"/>
<img data-src="img/FLP_I/f01-08/f01-08_tc_big.svgz"/>
<img data-src="img/FLP_I/f01-09/f01-09_tc_big.svgz"/>
<img data-src="img/FLP_I/f01-10/f01-10_tc_big.svgz"/>
```

https://www.feynmanlectures.caltech.edu/img/FLP_I/f01-01/f01-01_tc_big.svgz

禁止されています

このリソースにアクセスする権限がありません。

Apache/2.4.38 (Debian) サーバー at www.feynmanlectures.caltech.edu Port 443

```shell
% pip install selenium
Collecting selenium
  Using cached selenium-3.141.0-py2.py3-none-any.whl (904 kB)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.9/site-packages (from selenium) (1.24.1
Installing collected packages: selenium
Successfully installed selenium-3.141.0
```

上記のコマンドは、Python のパッケージ管理ツールである `pip` を使用して、Selenium ライブラリをインストールするプロセスを示しています。Selenium は、ウェブブラウザの自動化に広く使用されるツールです。このコマンドを実行すると、Selenium の最新バージョンがダウンロードされ、システムにインストールされます。`urllib3` という依存パッケージが既にインストールされている場合、そのパッケージは再インストールされません。最後に、Selenium が正常にインストールされたことが表示されます。

```
export CHROME_DRIVER_HOME=$HOME/dev-env/chromedriver
export PATH="${PATH}:${CHROME_DRIVER_HOME}"
```

このコードは、シェル環境で ChromeDriver のパスを設定するためのものです。`CHROME_DRIVER_HOME` という環境変数に ChromeDriver のインストールディレクトリを指定し、そのディレクトリを

PATH に追加しています。これにより、ターミナルから chromedriver コマンドを直接実行できるようになります。

```
% chromedriver -h
使用方法: chromedriver [オプション]
```

オプション –port=PORT リッスンするポート –adb-port=PORT adb サーバーのポート –log-path=FILE サーバーログを標準エラー出力ではなくファイルに書き込み、ログレベルを INFO に上げる –log-level=LEVEL ログレベルを設定: ALL, DEBUG, INFO, WARNING, SEVERE, OFF –verbose 詳細にログを記録する（–log-level=ALL と同等）–silent 何もログに記録しない（–log-level=OFF と同等）–append-log ログファイルを上書きせずに追記する –replayable （実験的）詳細にログを記録し、長い文字列を切り詰めないようにして、ログを再生できるようにする –version バージョン番号を表示して終了する –url-base コマンドのベース URL パスプレフィックス、例: wd/url –readable-timestamp ログに読みやすいタイムスタンプを追加する –enable-chrome-logs ブラウザからのログを表示する（他のログオプションを上書き）–allowed-ips ChromeDriver に接続を許可するリモート IP アドレスのカンマ区切り許可リスト

```
# コードブロックの開始を示すマークダウン記法
# ここに Python コードを記述します
print("Hello, World!")

from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support.expected_conditions import presence_of_element_located
```

このコードは、Selenium を使用してウェブブラウザを自動化するための基本的なインポート文です。以下に各インポートの説明を示します：

- webdriver: Selenium のコアモジュールで、ブラウザのインスタンスを作成し操作するために使用されます。
- By: 要素を検索するためのメソッドを提供します（例: By.ID, By.NAME, By.CLASS_NAME など）。
- Keys: キーボード操作をシミュレートするためのキー（例: Keys.ENTER, Keys.TAB など）を提供します。
- WebDriverWait: 特定の条件が満たされるまで待機するためのユーティリティを提供します。
- presence_of_element_located: 指定された要素が DOM に存在するかどうかを確認するための条件を提供します。

これらのインポートを使用して、ウェブページの要素を操作し、自動化されたテストやスクレイピングを行うことができます。

```
with webdriver.Chrome() as driver:
    wait = WebDriverWait(driver, 10)
    driver.get("https://google.com/ncr")
    driver.find_element(By.NAME, "q").send_keys("cheese" + Keys.RETURN)
    first_result = wait.until(presence_of_element_located((By.CSS_SELECTOR, "h3>div")))
    print(first_result.get_attribute("textContent"))

# このコードブロックは Python のコードを示しています。
# ここに Python のコードを書くことができます。

from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support.expected_conditions import presence_of_element_located
import urllib
```

このコードは、Selenium を使用してウェブブラウザを自動化し、ウェブページ上の要素を操作するための基本的なインポート文です。以下に各インポートの説明を示します：

- `webdriver`: Selenium のコアモジュールで、ブラウザのインスタンスを作成し、操作するために使用されます。
- `By`: 要素を検索するための戦略（例: ID、クラス名、CSS セレクタなど）を提供します。
- `Keys`: キーボード操作（例: Enter キー、Tab キーなど）をシミュレートするために使用されます。
- `WebDriverWait`: 特定の条件が満たされるまで待機するためのユーティリティです。
- `presence_of_element_located`: 指定された要素が DOM に存在するかどうかを確認するための条件です。
- `urllib`: URL を操作するための標準ライブラリです（このコードでは使用されていませんが、インポートされています）。

```
def main():
    driver = webdriver.Chrome()
    wait = WebDriverWait(driver, 10)
```

```
    driver.get("https://www.feynmanlectures.caltech.edu/I_01.html")
    elements = driver.find_elements(By.TAG_NAME, "img")
    # print(dir(elements[0]))
    print(driver.page_source)
    i = 0
    for element in elements:
        # src = element.get_attribute('src')
        element.screenshot(f'images/{i}.png')
        i +=1
    driver.close()
main()

from bs4 import BeautifulSoup
from multiprocessing import Process
import timeit
from pathlib import Path
from selenium import webdriver
from selenium.webdriver.common.by import By

def img_path(chapter):
    return f'./chapters/{chapter}/img'

def img_name(url):
    splits = url.split('/')
    last = splits[len(splits) - 1]
    parts = last.split('.')
    name = parts[0]
    return name

def download_images(driver: webdriver.Chrome, chapter):
    path = img_path(chapter)
    Path(path).mkdir(parents=True, exist_ok=True)

    elements = driver.find_elements(By.TAG_NAME, "img")
    for element in elements:
        src = element.get_attribute('src')
        name = img_name(src)
        element.screenshot(f'{path}/{name}.png')
```

このコードは、指定された章の画像をダウンロードするための関数です。以下にその動作を説明します。

1. `path = img_path(chapter)`：指定された章に対応する画像の保存先パスを取得します。
2. `Path(path).mkdir(parents=True, exist_ok=True)`：保存先ディレクトリが存在しない場合、親ディレクトリも含めて作成します。
3. `elements = driver.find_elements(By.TAG_NAME, "img")`：ウェブページ内のすべての `<img>` タグを取得します。
4. `for element in elements:`：各画像要素に対して以下の処理を行います。
    - `src = element.get_attribute('src')`：画像のソース URL を取得します。
    - `name = img_name(src)`：ソース URL から画像の名前を生成します。
    - `element.screenshot(f'{path}/{name}.png')`：画像をスクリーンショットとして保存します。

この関数を使用することで、指定された章の画像を自動的にダウンロードし、指定されたパスに保存することができます。

```
USER_AGENT = 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_6) AppleWebKit/605.1.15 (KHTML, like Gecko) '
```

このコードは、ウェブスクレイピングや API リクエストを行う際に使用されるユーザーエージェント文字列を定義しています。この特定のユーザーエージェントは、Mac OS X 10.15.6 上で動作する Safari 14.0.3 を模倣しています。これにより、ウェブサーバーに対して、リクエストが Mac の Safari ブラウザから来ているように見せることができます。

```
def scrape(chapter):
    if chapter < 1 or chapter > 52:
        raise Exception(f'chapter {chapter}')
    chapter_str = '{:02d}'.format(chapter)
    url = f'https://www.feynmanlectures.caltech.edu/I_{chapter_str}.html'
    driver = webdriver.Chrome()
    driver.get(url)
    page_source = driver.page_source
    Path(f'./chapters/{chapter_str}').mkdir(parents=True, exist_ok=True)
    print(f'scraping {url}')

    download_images(driver, chapter_str)
```

```
    soup = BeautifulSoup(page_source, features='lxml')
    imgs = soup.find_all('img')
    for img in imgs:
        if 'src' in img.attrs or 'data-src' in img.attrs:
            src = ''
            if 'src' in img.attrs:
                src = img.attrs['src']
            elif 'data-src' in img.attrs:
                src = img.attrs['data-src']
                del img.attrs['data-src']
            name = img_name(src)
            img.attrs['src'] = f'img/{name}.png'

    f = open(f'./chapters/{chapter_str}/I_{chapter_str}.html', 'w')
    f.write(soup.prettify())
    f.close()

    driver.close()
```

このコードは、指定された章番号に基づいて Feynman Lectures のウェブページをスクレイピングし、画像をダウンロードして HTML ファイルを保存する関数です。以下にその動作を説明します。

1. **章番号のチェック**: 章番号が 1 から 52 の範囲内にあるかどうかを確認します。範囲外の場合は例外を発生させます。
2. **URL の生成**: 指定された章番号に基づいて、Feynman Lectures の該当する章の URL を生成します。
3. **WebDriver の起動**: Chrome の WebDriver を起動し、生成した URL にアクセスします。
4. **ページソースの取得**: アクセスしたページの HTML ソースを取得します。
5. **ディレクトリの作成**: 章ごとのディレクトリを作成します。既に存在する場合は何もしません。
6. **画像のダウンロード**: ページ内の画像をダウンロードします。
7. **HTML の解析と修正**: BeautifulSoup を使用して HTML を解析し、画像の `src` 属性を修正します。`data-src` 属性がある場合は、それを `src` に置き換えます。
8. **HTML ファイルの保存**: 修正した HTML をファイルとして保存します。
9. **WebDriver の終了**: WebDriver を閉じて、リソースを解放します。

この関数を使用することで、指定された章の内容をローカルに保存し、オフラインで閲覧することが可能になります。

```
def main():
    start = timeit.default_timer()
    ps = [Process(target=scrape, args=(i+1,)) for i in range(2)]
    for p in ps:
        p.start()
    for p in ps:
        p.join()
    stop = timeit.default_timer()
    print('Time: ', stop - start)

if __name__ == "__main__":
    main()
```

```
スクレイピング中 https://www.feynmanlectures.caltech.edu/I_01.html
スクレイピング中 https://www.feynmanlectures.caltech.edu/I_02.html
時間:  21.478510914999998
```

```
 errpipe_read, errpipe_write = os.pipe()
OSError: [Errno 24] 開いているファイルが多すぎます
```

```
 % ulimit a
ulimit: 無効な数値: a
lzw@lzwjava feynman-lectures-mobi % ulimit -a
-t: CPU時間（秒）                    無制限
-f: ファイルサイズ（ブロック）        無制限
-d: データセグメントサイズ（キロバイト）  無制限
-s: スタックサイズ（キロバイト）       8192
-c: コアファイルサイズ（ブロック）     0
-v: アドレス空間（キロバイト）         無制限
-l: メモリにロックされたサイズ（キロバイト）  無制限
-u: プロセス数                       2784
-n: ファイルディスクリプタ数          256
```

```
12
```

```
download_images
12
mathjax2svg
latexs 128
make_svg 0
insert_svg 0
make_svg 1
insert_svg 1
make_svg 2
insert_svg 2
make_svg 3
insert_svg 3
convert
```

このコードブロックは、特定のタスクを実行するためのシェルスクリプトの一部のようです。各コマンドは、画像のダウンロード、MathJax から SVG への変換、LaTeX の処理、SVG の生成と挿入、そして最終的な変換を行うためのステップを示しています。具体的な内容や目的は、このコードが属するプロジェクトやコンテキストに依存します。

```
12
download_images
12
mathjax2svg
latexs 0
latexs 0
convert
Time:  11.369145162
```

(注：上記のコードブロックは、特定のコマンドやプロセスの実行結果を示すものであり、翻訳の必要はありません。そのままの形で使用してください。)

```
% grep --include=\*.html -r '\$' *
43/I_43.html:長い時間 $T$ の間に、一定の数 $N$ の衝突があるとします。もし私たちが
43/I_43.html:衝突の数は時間 $T$ に比例します。私たちは
43/I_43.html:比例定数を $1/\tau$ と書きました。ここで
43/I_43.html:$\tau$ は時間の次元を持つ定数です。この定数 $\tau$ は
43/I_43.html:60回の衝突があるとすると、$\tau$ は1分です。私たちは
43/I_43.html:$\tau$（1分）が
```

エラー E21018: ファイルの内容を解析中に、改良されたMobiドメインの作成に失敗しました。内容：<In earlier chapters
警告 W28001: Kindleリーダーは、コンテンツ内で指定されたCSSスタイルをサポートしていません。CSSプロパティを削除中
警告 W29004: 強制的に閉じられた開いているタグ： ファイル： /private/var/folders/_3
警告 W29004: 強制的に閉じられた開いているタグ： <p amzn-src-id="975"> ファイル： /private/var/folders/_3/n3

警告 W14001: ハイパーリンクに問題があり、未解決です: /private/var/folders/_3/n3b7dq8x6652drmx6_d3t3bh0000
警告 W14001: ハイパーリンクに問題があり、未解決です: /private/var/folders/_3/n3b7dq8x6652drmx6_d3t3bh0000
警告 W14001: ハイパーリンクに問題があり、未解決です: /private/var/folders/_3/n3b7dq8x6652drmx6_d3t3bh0000

```
<span class="disabled" href="#Ch1-F1">
          1-1
</span>
```

この HTML コードは、`disabled` クラスが適用された `span` 要素を含んでいます。この要素は、リンク先として `#Ch1-F1` を指定していますが、`disabled` クラスが適用されているため、リンクとして機能しない可能性があります。`1-1` というテキストが表示されますが、クリックしても何も起こらないか、またはリンクが無効化されている状態です。

```
'OEBPS/84b8b4179175f097be1180a10089107be75d7d85.svg' を 1264x1011 にラスタライズ中
'OEBPS/23a4df37f269c8ed43f54753eb838b29cff538a1.svg' を 1264x259 にラスタライズ中
Traceback (most recent call last):
  File "runpy.py", line 194, in _run_module_as_main
  File "runpy.py", line 87, in _run_code
  File "site.py", line 39, in <module>
  File "site.py", line 35, in main
  File "calibre/utils/ipc/worker.py", line 216, in main
  File "calibre/gui2/convert/gui_conversion.py", line 41, in gui_convert_override
  File "calibre/gui2/convert/gui_conversion.py", line 28, in gui_convert
  File "calibre/ebooks/conversion/plumber.py", line 1274, in run
  File "calibre/ebooks/conversion/plugins/mobi_output.py", line 214, in convert
  File "calibre/ebooks/conversion/plugins/mobi_output.py", line 237, in write_mobi
  File "calibre/ebooks/oeb/transforms/rasterize.py", line 55, in __call__
  File "calibre/ebooks/oeb/transforms/rasterize.py", line 142, in rasterize_spine
  File "calibre/ebooks/oeb/transforms/rasterize.py", line 152, in rasterize_item
  File "calibre/ebooks/oeb/transforms/rasterize.py", line 185, in rasterize_external
  File "calibre/ebooks/oeb/base.py", line 1092, in bytes_representation
  File "calibre/ebooks/oeb/base.py", line 432, in serialize
TypeError: 'NoneType' オブジェクトをバイトに変換できません
```

```
% kindlepreviewer feynman-lectures-on-physics-volumn-1.epub -convert
指定された引数を確認しています。
前処理を進行中です。
1/1 の本を処理中です。
本が警告付きで変換されました！: /Users/lzw/projects/feynman-lectures-mobi/feynman-lectures-on-physics-volu
後処理を進行中です。
出力/ログファイルを /Users/lzw/projects/feynman-lectures-mobi/output に書き込んでいます。

マニフェストのクリーンアップ中...
マニフェストから未使用のファイルを削除中...
AZW3出力を作成中...
リソースをシリアライズ中...
ページの区切りやフローの制限に基づいてマークアップを分割中（該当する場合）...
KF8出力を作成中
    KF8マークアップを生成中...
タグテーブルにaidがなく、チャンクサイズが大きすぎます。それでも追加します。
タグテーブルにaidがなく、チャンクサイズが大きすぎます。それでも追加します。
タグテーブルにaidがなく、チャンクサイズが大きすぎます。それでも追加します。
    マークアップを圧縮中...
    インデックスを作成中...
```