

# Python で Web から台本を収集する

---

さた@satamame

株式会社リーディング・エッジ社  
劇団 GAIA\_crew

# 概要

---

## Python で Web スクレイピングの基本をやります

- ・ 台本公開サイトからテキストデータをダウンロードします

### 2段階でやります

1. ダウンロードしたいデータについての情報収集
  - `scrapy` を使います
2. その情報を使ってダウンロード
  - `requests` を使います

# 動機

---

世の中の台本がデータとして扱えるようになっていない



データとして扱えればいろいろメリットがあるのに…



どういうフォーマットなら良さそうか



そもそも世の中の台本はどうなっているのか



データを集めよう

(解析とかフォーマット化については考えてないけど、集める)

# 条件

---

## テキストデータであること

- ・私が今までに見た台本はほぼ全て Word または PDF

## 参考

テキストベースのフォーマットも存在する

- ・Fountain
- ・Θέσπης 記法
- ・O's Editor 2 の「台本」スタイル \*
- ・VerticalEditor の「シナリオ形式」\*

(\* 書式はアプリの設定によります)

# ソースの選定

---

「はりこのトラの穴」というサイトに4,000本ある

| *<https://haritora.net/>*

- ダウンロードして読むだけなら問題なさそう
- 形式がプレーンテキスト (重要)

# 準備

---

- Python (3.8)
- 以下のパッケージ
  - scrapy (2.0.0)
  - requests (2.23.0)
  - pandas (1.0.2)

# 情報収集フェーズ

---

# scrapy のプロジェクトを作る

scrapy では、プロジェクトという単位で作業を管理します

scrapy の `startproject` コマンドを実行すると、プロジェクト用のサブディレクトリができます

```
> scrapy startproject hariko
```

以降の作業はこのサブディレクトリの中でやります

```
> cd hariko
```

# 収集する情報を決める(1)

サイトの台本リストのページはこんな感じ

<a href="#">01</a>	<a href="#">2</a>	<a href="#">3</a>	<a href="#">4</a>	<a href="#">5</a>	<a href="#">6</a>	<a href="#">7</a>	<a href="#">8</a>	<a href="#">9</a>	<a href="#">10-</a>	<a href="#">20</a>	<a href="#">30</a>	<a href="#">40</a>	<a href="#">50</a>	<a href="#">60</a>	<a href="#">70</a>	<a href="#">80</a>	<a href="#">90</a>	<a href="#">100</a>	<a href="#">110</a>	<a href="#">120</a>	<a href="#">次へ-</a>
<a href="#">130</a>	<a href="#">140</a>	<a href="#">150</a>	<a href="#">160</a>	<a href="#">170</a>	<a href="#">180</a>	<a href="#">190</a>	<a href="#">200</a>	<a href="#">210</a>	<a href="#">219</a>												<a href="#">≥</a>
題名											作者										
<a href="#">DeaD &amp; ArrivE</a>											<a href="#">詳細</a>	<a href="#">熔鉱炉</a>									
<a href="#">ネバーエンディング・千秋楽</a>											<a href="#">詳細</a>	<a href="#">あかざとう</a>									
<a href="#">胡蝶の夢</a>											<a href="#">詳細</a>	<a href="#">蟲守ユウ</a>									
<a href="#">コドクなひとたち 1</a>											<a href="#">詳細</a>										
<a href="#">夢見る羊たちのレクイエム</a>											<a href="#">詳細</a>	<a href="#">山上祐輝</a>									
<a href="#">白居三姉妹は今日も元気です</a>											<a href="#">詳細</a>	<a href="#">組木行路</a>									
<a href="#">奈々未のひとり旅</a>											<a href="#">詳細</a>	<a href="#">鈴木良尚</a>									
<a href="#">少女、誘拐事件</a>											<a href="#">詳細</a>	<a href="#">菅原悠人</a>									

- ・ページネーションがあり、ページあたり20件（デフォルト）

# 取得する情報を決める(2)

ソースを見ると、リストは `<table>` になっています

- 各行はこんな形です

```
<tr>
<td></td>
<th bgcolor="#FF3030"><font color="white" size="+1">題名</font>
</th>
<td><a href="look.cgi?script=16224">詳細</a></td>
<th><a href="script.cgi?writer=7791">著者名</a>
</th>
</tr>
```

# 取得する情報を決める(3)

「詳細」を押して表示されるページにダウンロードフォームがあります

[冒頭だけ読む](#) | [この作品の感想を読む](#) | [この作品の評価](#) | [上演記録](#) |

---

面白いと思ったら、続きを読む全文ダウンロードで！

Windows  Macintosh  
 E-mail

E-mail送付希望の方は、アドレス御記入ください。

[全文ダウンロード](#)



# 取得する情報を決める(4)

ダウンロードフォームのソースがこうなので…

```
<form method="POST" action="scriptdl.cgi/16224.txt">
<input type="hidden" name="script" value="16224">
<table>( ...中略... )</table>
<br>
<input type="submit" value="全文ダウンロード">
</form>
```

詳細ページへのリンクのパラメタが、ダウンロードにも使えそう  
↓これ (script= のところ)

```
<td><a href="look.cgi?script=16224">詳細</a></td>
```

# 取得する情報を決める (5)

---

## 方針

情報収集フェーズでは詳細ページの中は見ないで、台本リストから各台本の以下の情報を得る

1. 詳細ページの URL
2. その URL のパラメタ (script=???) の値
3. タイトル
4. 著者名

※ダウンロードのリクエストに必要な情報は 2 だけ

# scrapy の Item を作る

プロジェクトディレクトリ (hariko) の中に `items.py` があるので、編集してサイトから取ってくるデータを定義します

```
class HarikoItem(scrapy.Item):  
    url = scrapy.Field()  
    script_id = scrapy.Field()  
    title = scrapy.Field()  
    author = scrapy.Field()
```

※なぜクラス変数を定義するのか等が気になる方は、[scrapy の公式ドキュメント](#)をご覧ください

# scrapy のスパイダーを作る (1)

スパイダーはサイトを巡回して情報を取ってくるやつです  
`genspider` コマンドで作ります

```
> scrapy genspider (スパイダーネーム) (開始 URL)
```

今回は名前を `index` として、台本リストの URL を指定して作ります

```
> scrapy genspider index https://haritora.net/script.cgi
```

# scrapy のスパイダーを作る (2)

hariko/spiders ディレクトリの中に `index.py` ができます

```
import scrapy

class IndexSpider(scrapy.Spider):
    name = 'index'
    allowed_domains = ['https://haritora.net/script.cgi']
    start_urls = ['http://https://haritora.net/script.cgi/']

    def parse(self, response):
        pass
```

# scrapy のスパイダーを作る (3)

allowed\_domains と start\_urls が変なので修正します

```
allowed_domains = ['https://haritora.net/script.cgi']
start_urls = ['http://https://haritora.net/script.cgi/']
```



```
allowed_domains = ['haritora.net']
start_urls = ['https://haritora.net/script.cgi']
```

# scrapy のスパイダーを作る (4)

---

## parse() メソッド

スパイダーの `parse()` メソッドは、サイトからのレスポンスを受け取って、以下のいずれかを `yield` で返します

### 1. 取得した情報

- `scrapy.Item` オブジェクトまたは `dict`
  - 出力に追加されます

### 2. さらなるリクエスト

- `scrapy.http.Request` オブジェクト
  - これにより、芋づる式にページを渡り歩きます

※ `yield` なので何個返しても OK

# scrapy のスパイダーを作る (5)

たとえば以下のようにすると、台本リストの全てのページの URL を出力します

```
def parse(self, response):
    yield {'url': response.url}
    next_links = response.xpath("//a[text()='次へ->']")
    if next_links:
        url = response.urljoin(next_links[0].attrib['href'])
        yield scrapy.Request(url, callback=self.parse)
```

- 「次へ」要素があれば、その `href` をリクエストして再び `parse()` で受け取る、という処理になっています

# scrapy のスパイダーを作る (6)

## XPathについて

XPathは、XML や HTML 内の要素を見つけるための、パターンマッチングの記法です

### 例

台本リストの詳細ページへのリンク部分は以下のようでした

```
<td><a href="look.cgi?script=16224">詳細</a></td>
```

この `<a>` 要素を見つけるためのパターンは、こうなります

```
xpath = "//a[contains(@href, 'look.cgi?script=')]"
```

# scrapy のスパイダーを作る(7)

見つけた要素の親/兄弟要素をマッチングすることも可能です

```
# 各台本の詳細ページへのリンク
xpath = "//a[contains(@href, 'look.cgi?script=')]"

# 詳細ページへのリンクの周辺からデータ取得
for detail_link in response.xpath(xpath):
    # 詳細ページへのリンク (相対パス)
    link = detail_link.attrib['href']
    # そのフルパス
    url = response.urljoin(link)
    # 台本番号
    script_id = int(link.split('=')[-1])
    # 題名
    title = detail_link.xpath(
        "parent::td/preceding-sibling::th/font/text()").get()
    # 著者名
    author = detail_link.xpath(
        "parent::td/following-sibling::th/a/text()").get()
```

# scrapy のスパイダーを作る (8)

詳細ページへのリンクごとに、周辺情報をまとめて `yield` で返します (`items.py` で作ったクラスを使います)

```
for detail_link in response.xpath(xpath):  
    # (中略)  
  
    # インスタンス生成  
    item = HarikoItem(  
        url = url,  
        script_id = script_id,  
        title = title,  
        author = author  
    )  
  
    yield item
```

# scrapy のスパイダーを作る (9)

---

## parse() メソッドでやることの整理

1. `response` から「詳細」リンク要素(複数)を取得
2. 各リンク要素について
  - 周辺要素から `HarikoItem` 情報を作り `yield` で返す
3. `response` から「次へ」リンク要素を取得
4. 要素が見つかったらリクエストを作って `yield` で返す

GitHub にコードを置いてあります

- [https://github.com/satamame/pscscrape/  
blob/master/hariko/hariko/spiders/index.py](https://github.com/satamame/pscscrape/blob/master/hariko/hariko/spiders/index.py)

# スパイダーを実行する前に

hariko/settings.py というファイルがあるので、サイトに負荷をかけないよう以下の変数を設定しておきます

```
# Configure maximum concurrent requests performed by Scrapy (default: 16)
CONCURRENT_REQUESTS = 1

# Configure a delay for requests for the same website (default: 0)
# See https://docs.scrapy.org/en/latest/topics/settings.html#download-delay
# See also autothrottle settings and docs
DOWNLOAD_DELAY = 2
```

- ここでは同時接続数を1本に、リクエスト間隔を2秒に設定しています

# スパイダーを実行する

- ・ スパイダーを実行するには `crawl` コマンドを使います
- ・ 結果をファイルに保存するには `-o` (ファイル名) をつけます
- ・ ファイル名を `???.csv` とすると、自動的に csv で保存します
- ・ 追記になるので、同名のファイルがあれば削除しておきます

既存ファイルを削除する

```
> del index.csv
```

スパイダー `index` を実行し、`index.csv` に出力

```
> scrapy crawl index -o index.csv
```

- ・ 注意：この実行には時間がかかります

# スパイダーからの出力

## index.csv

author	script_id	title	url
熔鉱炉	16224	DeaD & ArrivE	<a href="https://haritora.net/look.cgi?script=16224">https://haritora.net/look.cgi?script=16224</a>
あかざとう	16223	ネバーエンディング・千秋楽	<a href="https://haritora.net/look.cgi?script=16223">https://haritora.net/look.cgi?script=16223</a>
蟲守ユウ	16222	胡蝶の夢	<a href="https://haritora.net/look.cgi?script=16222">https://haritora.net/look.cgi?script=16222</a>
山上祐輝	16221	夢見る羊たちのレクイエム	<a href="https://haritora.net/look.cgi?script=16221">https://haritora.net/look.cgi?script=16221</a>
組木行路	16220	白居三姉妹は今日も元気です	<a href="https://haritora.net/look.cgi?script=16220">https://haritora.net/look.cgi?script=16220</a>
鈴木良尚	16219	奈々未のひとり旅	<a href="https://haritora.net/look.cgi?script=16219">https://haritora.net/look.cgi?script=16219</a>
菅原悠人	16218	少女、誘拐事件	<a href="https://haritora.net/look.cgi?script=16218">https://haritora.net/look.cgi?script=16218</a>

- サイトのエンコーディングに関わらず UTF-8 で出力されるようです

# ダウンロードフェーズ

---

# 方針

---

- `scrapy` でダウンロードすることも出来ますが…
- URL パラメタを変えて繰り返すだけの処理なので…
- `requests` を使って(なるべくシンプルに) やります

リクエスト先の URL

```
https://haritora.net/scriptdl.cgi/(台本番号).txt
```

# メインループ

---

pandas で csv を読み込んで、行ごとにループします  
ざっくり、以下のような形になります

```
import pandas as pd
import requests

# インデックスファイルを読み込む
df = pd.read_csv('index.csv')

# 行ごとにループ
for row in df.itertuples():

    # 1. 保存先ファイル名を作る

    # 2. ダウンロード用リクエストを作る

    # 3. POST して Response をファイルに保存
```

# 保存先ファイル名を作る

- pandas の DataFrame では、row.script\_id のように、csv のヘッダを使って列の値を取れます
- ファイル名の形をそろえるためにゼロ埋めしています
- サブフォルダの中に保存するようパス名を作ります

```
import os

for row in df.itertuples():

    # 台本番号
    script_id = int(row.script_id)

    # 保存先ファイル名
    save_name = f'{script_id:06}.txt'
    save_path = os.path.join('scripts/', save_name)
```

# ダウンロード用リクエストを作る(1)

リクエストを作るにあたって、フォームがどうなっていたか確認します

```
<FORM METHOD="POST" ACTION="scriptdl.cgi/16224.txt">
<input type="hidden" name="script" value="16224">
<input type="radio" name="type" value="win" checked>Windows
<input type="radio" name="type" value="mac">Macintosh
<input type="radio" name="type" value="mail">E-mail
<input type="text" name="email" size="40">
<input type="submit" value="全文ダウンロード">
</FORM>
```

リクエストにくっつけるデータとしては、`script` と `type` をセットすれば良さそうです

# ダウンロード用リクエストを作る(2)

- リクエスト先の URL は決め打ちでも良いのですが、せっかくなので詳細ページの URL を使って生成しています
- `type` はフォームでデフォルトだった `win` を指定しました
  - 文字コードや改行コードを決めていると思われます

```
import urllib

for row in df.itertuples():

    # (中略)

    # ダウンロード用の POST リクエストを準備する
    cgi_path = f'scriptdl.cgi/{script_id}.txt'
    action_url = urllib.parse.urljoin(row.url, cgi_path)
    data = {'script': str(script_id), 'type': 'win'}
```

# POSTしてResponseをファイルに保存

- `requests` はレスポンスのエンコーディングを自動判定するのですが、間違えるので直に `sjis` をセットしてあげます
- 保存は `utf-8` です

```
for row in df.itertuples():  
    # (中略)  
  
    # リクエストして結果をテキストで得る  
    response = requests.post(action_url, data=data)  
    # encoding が正しく判定されないので直に指定する  
    response.encoding = 'sjis'  
  
    # テキストを保存する  
    with open(save_path, mode='w', encoding='utf-8') as f:  
        f.write(response.text)
```

# ダウンロードを実行する

---

さらに以下の点を考慮して、コードを完成させます

- ・保存用ディレクトリがなければ作成する
- ・同名ファイルが存在すれば、ダウンロードをスキップする
- ・サーバ負荷を考慮して待ち時間を入れる
- ・一回の実行でダウンロードするファイル数に上限を設ける

GitHub にコードを置いてあります

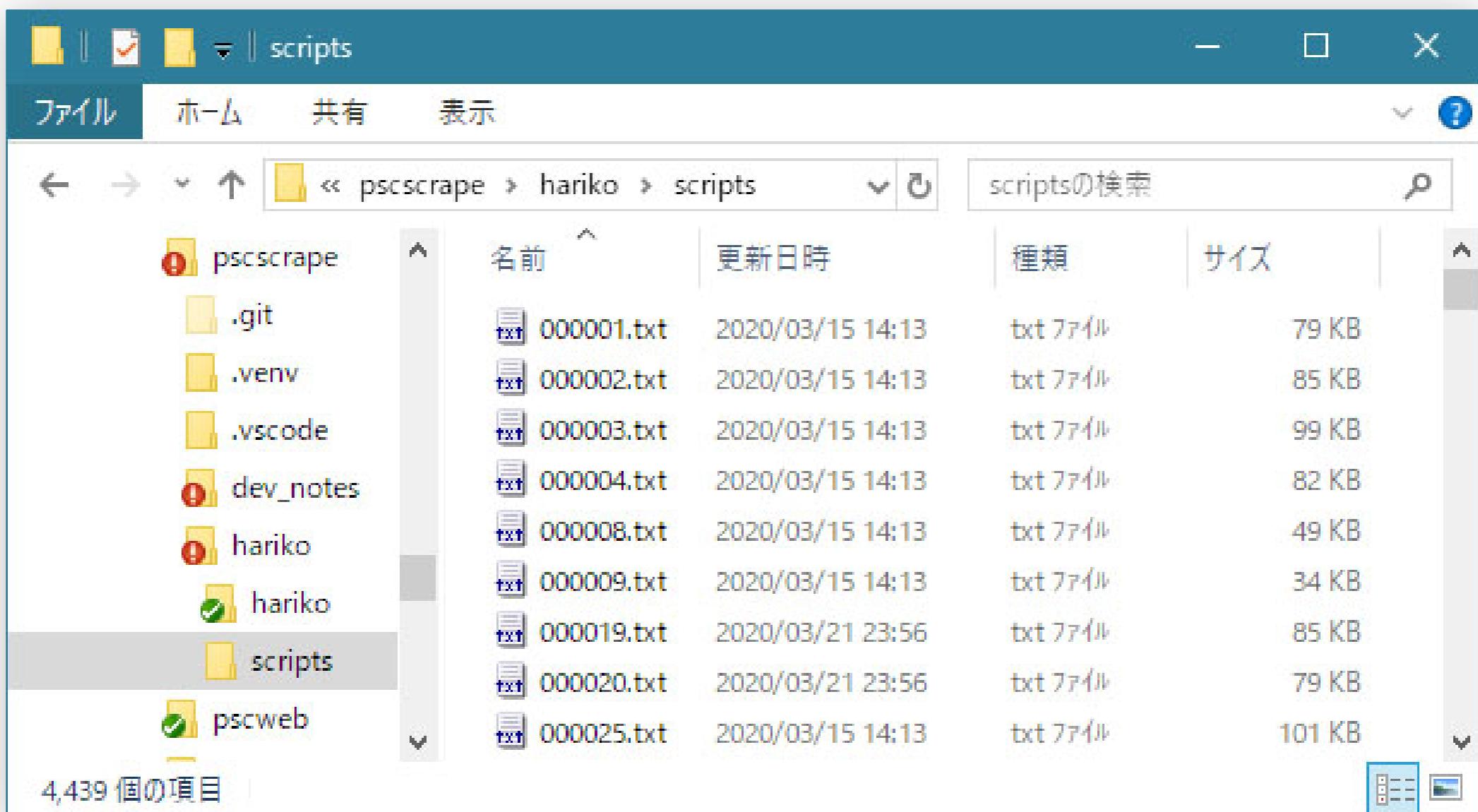
- ・[https://github.com/satamame/pscscrape/  
blob/master/hariko/download.py](https://github.com/satamame/pscscrape/blob/master/hariko/download.py)

実行

```
> python download.py
```

# 実行結果

約4,000件の台本をダウンロードしました



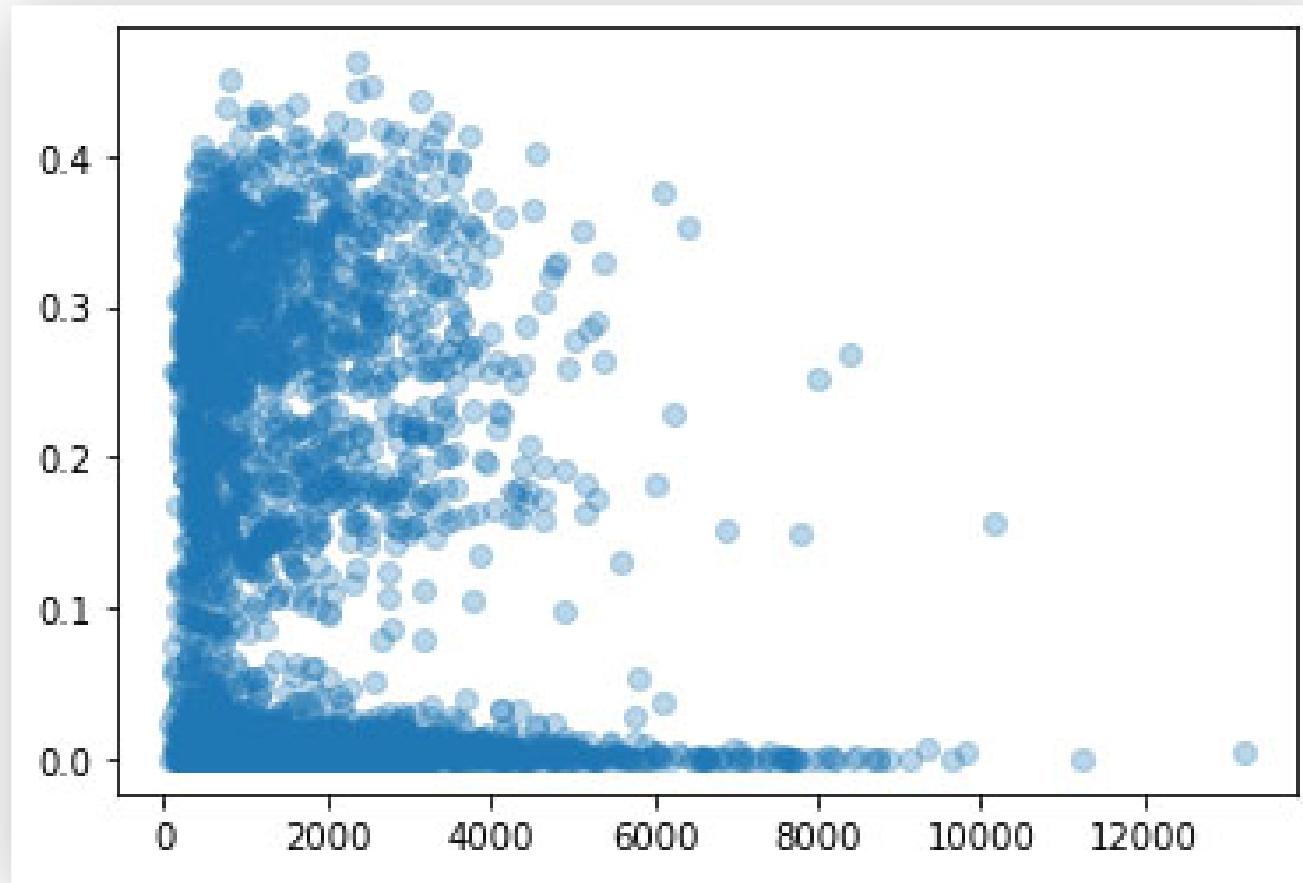
# おまけ

---

# 括弧を数えてみました

ダウンロードした各台本について、「括弧を含む行」の割合がどれくらいか、数えてみました

横軸が全行数、縦軸が「括弧を含む行」の割合です



ト書きの量や、一個のセリフが何行にまたがるか等にもよりますが、「セリフに括弧を使っているか」の判断材料にできそうです