

動かしながら学ぶ Python

購入者特典

Ver 1.0

🚩 Try 違う文字を出力してみよう

「Hello World」以外の文字を出力するにはどうすればよいでしょう？ 例えば「Hello Japan」なら？「こんにちは」ならどうすればよいでしょうか？

「Hello World」以外の文字を出力するには、`print("文字列")` の文字列を変更します。例えば「Hello Japan」と「こんにちは」を出力する場合は以下の通りです。

実行結果

```
>>> print("Hello Japan")
Hello Japan
>>> print(" こんにちは ")
こんにちは
```

🚩 Try 違う文字を出力するプログラムを作ろう

1. 「Hello World」と出力するプログラムをファイルに書いて実行してみましょう。
ファイルに以下のコードを記述し、保存します。

```
try.py

print("Hello World")
```

ファイル名は `try.py` として保存します。Python のプログラムは「.py」という拡張子を持つファイルに保存します。また、プログラム実行時にはそのファイルが保存されているフォルダに移動して実行する必要があります。以下の手順で実行します。

▶ Windows の場合

コマンドプロンプトを開きます。`try.py` が保存されているフォルダに移動してファイルを実行します。

```
cd { 移動したいフォルダへのパス }
python try.py
```

実行結果

```
Hello World
```

▶ macOS の場合

ターミナルを開きます。try.py が保存されているフォルダに移動します。
次のコマンドでプログラムを実行します。

```
cd { フォルダへのパス }  
python3.12 try.py
```

実行結果

```
Hello World
```

2. そのプログラムを「Hello Japan」と出力するプログラムに修正して実行してみましょう。

try.py ファイルの内容を以下のように変更し、再び保存します。

```
try.py  
  
print("Hello Japan")
```

再度同じ方法でプログラムを実行すると、以下の結果が得られます。

実行結果

```
Hello Japan
```

フォルダの移動は `cd {フォルダへのパス}` を使用します。フォルダのパスがわからない場合、ドラッグ & ドロップや「パスのコピー」機能を使うと便利です。一方で、実行コマンドは Windows では `python {実行ファイル名}`、macOS では `python{バージョン} {実行ファイル名}` を使用します。

🚩 Try 式を再利用して計算してみよう①

1. 100km の距離を 5 時間で移動した場合の平均速度はいくつですか？

実際に数字を入力して計算をしてみましょう。

実行結果

```
>>> distance = 100 # 距離 (km)
>>> time = 5 # 時間 (h)
>>> speed = distance / time # 平均速度の計算
>>> print(speed) # 出力
20.0
```

2. 50km の距離を 2 時間で移動した場合の平均速度はいくつですか？

同じく、実際に数字を入力して計算をしてみましょう。

実行結果

```
>>> distance = 50 # 距離 (km)
>>> time = 2 # 時間 (h)
>>> speed = distance / time # 平均速度の計算
>>> print(speed) # 結果を出力
25.0
```

3. 120km の距離を 4 時間で移動した場合の平均速度はいくつですか？

同じく、実際に数字を入力して計算をしてみましょう。

実行結果

```
>>> distance = 120 # 距離 (km)
>>> time = 4 # 時間 (h)
>>> speed = distance / time # 平均速度の計算
>>> print(speed) # 結果を出力
30.0
```

🚩 Try 式を再利用して計算してみよう②

1. 休憩時間が3時間の場合はどうでしょうか？

ここでは `break_time = 3` に書き換えます。これにより、計算式がそのまま適用され、残りの移動時間が計算できます。

実行結果

```
>>> total_distance = 100
>>> time = 2
>>> speed = total_distance / time
>>> travelved_distance = 25
>>> break_time = 3 # 休憩時間を3時間に変更
>>> remaining_distance = total_distance - travelved_distance
>>> remaining_time = remaining_distance / speed + break_time
>>> print(remaining_time)
4.5
```

2. 移動速度は同じで、距離が130kmの場合はどうですか？

今度は `total_distance = 130` に書き換えます。他の変数はそのままにすることで、速度を変更せずに計算が可能です。

実行結果

```
>>> total_distance = 130 # 距離を130kmに変更
>>> time = 2
>>> speed = total_distance / time
>>> travelved_distance = 25
>>> break_time = 1
>>> remaining_distance = total_distance - travelved_distance
>>> remaining_time = remaining_distance / speed + break_time
>>> print(remaining_time)
2.6153846153846154
```

3. 同じ距離を往復する場合は、何時間かかりますか？

往復の場合は総距離を2倍にして計算します。

実行結果

```
>>> total_distance = 100* 2
>>> time = 2
>>> speed = total_distance / time
>>> travelved_distance = 25
>>> break_time = 1
>>> remaining_distance = total_distance - travelved_distance
>>> remaining_time = remaining_distance / speed + break_time
>>> print(remaining_time)
2.75
```

4. 同じだけ休憩し、同じ速度で移動した場合、5時間以内に到着できる距離は何 km ですか？

変数の特性を活かし、以下のように工夫して計算をしていきましょう。まずはこれまでと同じ条件で速度を求めておきます。続いて、`time` と `total_distance` を再定義します。数学の変数とは異なり、後から変数の値を上書きできる点に注意しましょう。

実行結果

```
>>> total_distance = 100
>>> time = 2
>>> speed = total_distance / time
>>> time = 5
>>> travelved_distance = 25
>>> break_time = 1
>>> total_distance = travelved_distance + speed * (time - break_time)
>>> print(total_distance)
225.0
```

🚩 Try 文字列と変数をつかってみよう

1. 「Python」という文字列を変数 `language` に入れてみよう。

文字列は `"`（ダブルクォーテーション）で囲う必要がありますので注意しましょう。変数に入れたら `print()` で確認します。

実行結果

```
>>> language = "Python"
>>> print(language)
Python
```

2. 変数 `language` に Ruby という文字列を入れてみよう。

変数の中身を上書きするのも同じ方法です。実際に置き換わったか `print()` で確認します。

実行結果

```
>>> language = "Ruby"
>>> print(language)
Ruby
```

3. 文字列を作るときに `'` を使ってみよう。

文字列を作るときには `'`（シングルクォーテーション）を使うこともできます。例えば `"Python"` と `'Python'` は同じ意味になります。また、文字列内で `"` を使いたい場合は、外側を `'` で囲むと良いでしょう。

実行結果

```
>>> language = 'Ruby'
>>> print(language)
Ruby
>>> message = '今アツいプログラミング言語は "Python" だ '
>>> print(message)
今アツいプログラミング言語は "Python" だ
```

🚩 Try input を使ったプログラムを動かしてみよう

1. 利用者にキーボードから文字列を入力する際に「あなたの名前は？」と表示してみよう。

`input()` を使って利用者に「あなたの名前は？」が表示されるようにします。

実行例

```
>>> input("あなたの名前は？")
あなたの名前は？
```

2. 利用者が入力した文字列を変数 `title` に入れてみよう。

続いては `input()` を使ってユーザーから文字列を入力してもらい、その結果を `title` という変数に格納します。

実行結果

```
>>> title = input("あなたの名前は？")
あなたの名前は？ワタル
>>> print(title)
ワタル
```

3. 利用者が入力した文字列と、あらかじめ定義した文字列を組み合わせ、文字列を出力してみよう。

入力された文字列と組み合わせる文字列を、あらかじめ定義しておきます。例えば応援メッセージなんてどうでしょうか。また、入力された文字列を他の文字列と組み合わせるには、演算子 `+` を使用します。

実行結果

```
>>> message = title + "さん。がんばって！"
>>> title = input("あなたの名前は？")
あなたの名前は？ワタル
>>> print(message)
ワタルさん。がんばって！
```


🚩 Try Discord Bot のメッセージを変更する

1. Discord Bot に `>hey` というコマンドを入力すると、「おはよう」と返答するようにしてみましょう。

app.py のコードを使用して、Discord Bot の挙動を変更します。

app.py の一部

```
10 @bot.command()
11 async def hey(ctx):
12     await ctx.send("おはよう")
```

2. Discord Bot に `>hey 山田くん` というコマンドを入力すると、「山田くん、おはよう！」と返答するようにしてみましょう。

同じく app.py のコードを使用して、Discord Bot の挙動を変更します。ここでは、Discord を通じてユーザーの入力した文字列を取得して、変数に格納して利用します。

app.py の一部

```
10 @bot.command()
11 async def hey(ctx, name):
12     greeting = "、おはよう"
13     await ctx.send(name + greeting)
```

3. Discord Bot に `>hey 佐藤くん` というコマンドを入力すると、「佐藤くん、今日はいい天気ですね！」と返答するようにしてみましょう。

こちらも同じく Discord を通じてユーザーの入力した文字列を取得して、変数に格納して利用します。それぞれ別名のファイルとして保存して、Discord Bot を起動し実行して確かめてみましょう。

app.py の一部

```
10 @bot.command()
11 async def hey(ctx, name):
12     greeting = "、今日はいい天気ですね！"
13     await ctx.send(name + greeting)
```

🚩 Try if 文を使ったプログラムを動かしてみよう

1. `weather` に格納されている文字列を別の文字列に変更してみよう

変数 `weather` に `"曇り"` を格納してみます。ただし、`if weather == "晴れ":` の条件に合致しないため、何も出力されません。

実行結果

```
>>> weather = "曇り"
>>> if weather == "晴れ":
...     print("遊びに行く ")
... 
```

2. if の条件に指定する文字列を別の文字列に変更してみよう

続いて変数 `weather` の `"曇り"` をに合わせて、`if weather == "曇り":` の条件に変更します。条件に合致させているので、`print()` が実行され「遊びに行く」が出力されます。

実行結果

```
>>> if weather == "曇り":
...     print("遊びに行く ")
... 
```

遊びに行く

3. 条件に利用する変数を `weather` 以外の文字列として if 文を書いてみましょう

例えば新しい変数として `condition` を作成します。

実行結果

```
>>> condition = "元気"
>>> if condition == "元気":
...     print("遊びに行く ")
... 
```

遊びに行く

if 文は、条件式が `True` の場合にブロック内のコードを実行します。条件式は、`==` 演算子を使用して 2 つの値が等しいかどうかを比較します。条件式の後は必ずコロン `:` が必要です。忘れると `SyntaxError` が発生します。また、Pytho ではインデントを使ってブロックの範囲を示します。基本的にはスペース 4 つ分で揃えます。tab キーを使うと 1 回の入力ですペース 4 つ分が入力できます。

🚩 Try if 文と for 文を使ったプログラムを動かしてみよう

1. `"Python"` が文字列に含まれていない場合に、「この本は Python の本ではありません」と表示しましょう。

`else` を使って、もとのプログラムの条件に一致しない場合の処理を追加します。

実行結果

```
>>> for book in books:
...     if "Python" in book:
...         print(f"Python の本として「{book}」があります。")
...     else:
...         print(f"「{book}」は Python の本ではありません ")
...
Python の本として「Python プログラミングブック」があります。
Python の本として「Python のエンジニアになるには」があります。
「プログラマーのお仕事」は Python の本ではありません
```

2. リストの中に追加で要素を入れて動かしてみましよう。

続いては以下のリストに `append()` を使って要素を追加してみましょう。また、`print()` でリスト `books` の中身を確認しておきます。

実行結果

```
>>> books = ["Python プログラミングブック ", "Python のエンジニアになるには ", "
プログラマーのお仕事 "]
>>> books.append(" 動かしながら学ぶ Python")
>>> print(books)
["Python プログラミングブック ", "Python のエンジニアになるには ", " プログラマーの
お仕事 ", " 動かしながら学ぶ Python"]
```

3. "プログラマー" という文字列が含まれる本を探してみましょう。

今度は探す文字列を "Python" から "プログラマー" に変更します。

実行結果

```
>>> for book in books:
...     if "プログラマー" in book:
...         print(f"プログラマーに関する本として「{book}」があります。")
...
プログラマーに関する本として「プログラマーのお仕事」があります。
```

🚩 Try Discord Bot をさらに修正してみよう。

1. >talk おやすみと送信すると「おやすみ」と返ってくるようにしてみましょう。

app5.py のコードに elif を使って条件分岐を追加します。

app5.py の一部

```
10 @bot.command()
11 async def talk(ctx, message):
12     if message == "こんにちは":
13         await ctx.send("こんにちは")
14     elif message == "おはよう":
15         await ctx.send("おはよう")
16     elif message == "おやすみ":
17         await ctx.send("おやすみ")
18     else:
19         await ctx.send("なんですか？")
```

2. >talk 文字列 を送信すると、送信した文字列が返って来るようにしてみましょう。

このプログラムでは、入力された文字列は変数 >message に格納されるので、それを利用します。これまで "なんですか？" と返していたところに message を返すように変更します。

app5.py の一部

```
10 @bot.command()
11 async def talk(ctx, message):
12     if message == "こんにちは":
13         await ctx.send("こんにちは")
14     elif message == "おはよう":
15         await ctx.send("おはよう")
16     else:
17         await ctx.send(message)
```

🚩 Try 辞書をつかってみよう①

1. フルーツの名前をキー、値段を値として格納した辞書を定義し、それぞれのフルーツの値段を取得してみましょう。

まずはフルーツの名前をキー、値段を値とする辞書を定義します。ここでは例としてリンゴ、バナナ、ミカンの3種類を使ってみます。その後、`print()` を使ってそれぞれの値段を取得します。

実行結果

```
>>> fruit_price = {
...     "リンゴ": 100,
...     "バナナ": 150,
...     "ミカン": 80,
... }
>>> print(fruit_price["リンゴ"])
100
>>> print(fruit_price["バナナ"])
150
>>> print(fruit_price["ミカン"])
80
```

2. 動物の名前をキー、鳴き声を値として格納した辞書を定義し、それぞれの動物の鳴き声を取得してみましょう。

先ほどと同様に辞書を定義してから、`print()` を使ってそれぞれの値を取得します。

実行結果

```
>>> animal_sound = {
...     "ねこ": "にゃー",
...     "いぬ": "わんわん",
...     "うし": "もー"
... }
>>>
>>> print(animal_sound["ねこ"])
にゃー
>>> print(animal_sound["いぬ"])
わんわん
>>> print(animal_sound["うし"])
もー
```

リストと異なり、辞書ではキーを指定して要素へアクセスします。`辞書名[キー]` で対応する値を取得できます。キーは任意の文字列を利用できるため、辞書を使うとデータに意味を持たせて格納することができます。指定したキーが存在しない場合、エラー (`KeyError`) が生じるので注意が必要です。

🚩 Try 辞書をつかってみよう②

1. `status` のキーに「未読」という文字列を入れてみましょう。

まずは以下のリストに文字列 `status` をキーとする読書ステータスの情報を追加します。最初の値は `"未読"` とします。その後 `print()` を使って情報を確認しましょう。

実行結果

```
>>> book = {
...     "name": "Python プログラミング入門",
...     "price": 1800,
...     "publication_date": "2024-04",
... }
>>>
>>> book["status"] = "未読"
>>> print(book)
{'name': 'Python プログラミング入門', 'price': 1800, 'publication_date': '2024-04', 'status': '未読'}
```

2.statusのキーを「読了」という文字列で更新してみましょう。

辞書の値の更新は内容の追加と同じ方法で実行できます。既存のキーの値が更新されます。今回はキー `status` の値を `"読了"` に更新して、`print()` で更新が行われたか確認しましょう。

実行結果

```
>>> book["status"] = "読了"
>>> print(book)
{'name': 'Python プログラミング入門', 'price': 1800, 'publication_date': '2024-04', 'status': '読了'}
```

🚩 Try 辞書を使ったプログラムをもっと使ってみよう

1. 読書ステータスの情報を格納してみましょう。

リスト `books` の中の辞書にキー `status` を追加して、その中に読書ステータスの情報を格納しましょう。ここでは繰り返し処理を使って各辞書に `"未読"` のステータスを追加していきます。さらにその後に `print()` で確認しましょう。

実行結果

```
>>> books = [
...     {"name": "Python プログラミング入門", "price": 1800, "publication_
...     date": "2021-04"},
...     {"name": "Python によるデータ分析", "price": 2200, "publication_
...     date": "2020-09"},
...     {"name": "Python による機械学習", "price": 2500, "publication_
...     date": "2019-12"},
... ]
>>>
>>> for book in books:
...     book["status"] = "未読"
>>>
>>> for book in books:
...     print(book)
{'name': 'Python プログラミング入門', 'price': 1800, 'publication_date': '2021-04', 'status': '未読'}
```

```
{'name': 'Python によるデータ分析', 'price': 2200, 'publication_date':  
'2020-09', 'status': '未読'}  
{'name': 'Python による機械学習', 'price': 2500, 'publication_date': '2019-  
12', 'status': '未読'}
```

2. 複数のフルーツの名前と値段を格納した辞書をリストに格納し、それぞれのフルーツの値段を取得してみましょう。

まずはリストに格納された辞書を作成します。その後に `print()` を利用して値段を取得してみましょう。ここでは繰り返しと f-string を使って出力させてみます。

実行結果

```
>>> fruits = [  
...     {"name": "リンゴ", "price": 100},  
...     {"name": "バナナ", "price": 150},  
...     {"name": "ミカン", "price": 80}  
... ]  
>>> for fruit in fruits:  
...     print(f"{'fruit['name']} の値段は {'fruit['price']} 円です")  
りんごの値段は 100 円です  
バナナの値段は 150 円です  
みかんの値段は 80 円です
```

3. 複数の動物の名前と鳴き声を格納した辞書をリストに格納し、鳴き声が「ワン」という動物の名前を取得してみましょう。

まずはリストに格納された辞書を作成します。その後に繰り返しと if 文を利用して条件の検索を行います。最後に `print()` を利用して出力させてみます。

実行結果

```
>>> animals = [  
...     {"name": "いぬ", "sound": "ワン"},  
...     {"name": "ねこ", "sound": "ニャー"},  
...     {"name": "うし", "sound": "モー"}  
... ]  
>>> for animal in animals:  
...     if animal["sound"] == "ワン":  
...         print(f"『ワン』と鳴く動物は {animal['name']} です。")  
『ワン』と鳴く動物はいぬです。
```


🚩 Try Spotify API で情報を検索してみよう

1. 好きなキーワードでアーティストを検索してみよう。

Spotify API で検索キーワードを変更してアーティスト情報を取得します。search_artist.py の 29 行目のキー `q` の値を任意のキーワードに変更しましょう。例えば「東京」をキーワードにする場合、以下のようになります。

search_artist.py の一部

```
24 # 3. 指定されたキーワードで Spotify を検索
25 response = requests.get(
26     "https://api.spotify.com/v1/search",
27     headers={"Authorization": f"Bearer {token}"},
28     params={
29         "q": "東京",
30         "type": "artist",
31         "market": "JP",
32     },
33 )
```

2. アーティストではなく、アルバムや楽曲を検索してみよう。

また、30 行目の検索対象も変更することができます。例えば、「青春」というキーワードで楽曲から検索したい場合は以下のように修正します。

search_artist.py の一部

```
24 # 3. 指定されたキーワードで Spotify を検索
25 response = requests.get(
26     "https://api.spotify.com/v1/search",
27     headers={"Authorization": f"Bearer {token}"},
28     params={
29         "q": "青春",
30         "type": "track",
31         "market": "JP",
32     },
33 )
```

また、type の利用可能なオプションは、`"artist"`（アーティスト）、`"album"`（アルバム）、`"track"`（楽曲）が存在しています。詳しくは Spotify API のリファレンスを確認しましょう。

🚩 Try Discord Bot にさらに機能を追加してみよう

1. 楽曲検索コマンドも追加してみよう

まずは spotify.py に楽曲を検索できる関数を定義します。新たに `search_track` 関数を定義しておきます。コードの全文は、サンプルコードの `spotify_try.py` で確認できます。やや発展的な内容になるためここでは省略しますが、`search` 関数と `search_track` 関数では重複している部分がありますので、そこを新たな関数にしてまとめることもできます。

spotify_try.py の一部

```
35 def search_track(keyword):
36     # 1. Spotifyで認証
37     auth_response = requests.post(
38         "https://accounts.spotify.com/api/token",
39         data={
40             "grant_type": "client_credentials",
41             "client_id": client_id,
42             "client_secret": client_secret,
43         },
44     )
45
46     # 2. アクセストークン取り出し
47     response_json = auth_response.json()
48     token = response_json["access_token"]
49
50     # 3. 指定されたキーワードで Spotify を検索
51     response = requests.get(
52         "https://api.spotify.com/v1/search",
53         headers={"Authorization": f"Bearer {token}"},
54         params={
55             "q": keyword,
56             "type": "track",
57             "market": "JP",
58         },
59     )
60
61     return response.json()
```

さらに `app.py` を変更して、Discord 上で `>search_track 任意の文字列` という入力で楽

曲の検索ができるようにします。そのためには app.py に新たにコマンドを定義します。このコードは app.py の 11 行目から 30 行目のコードとほぼ同一で、コマンドは `search_track` とし、呼び出す関数も spotify_try.py の `search_track` に変更しましょう。また、このファイルは app_try.py と名前を変えておきます。コードの全文は、サンプルコードの appy_try.py で確認できます。

app_try.py の一部

```
32 @bot.command()
33 async def search_track(ctx, keyword):
34     if len(keyword) < 2:
35         await ctx.send("2 文字以上の検索キーワードを入力してね ")
36         return
37     elif len(keyword) > 20:
38         await ctx.send(" 検索キーワードは 20 文字以内にしておね ")
39         return
40
41     result = spotify_try.search_track(keyword)
42
43     if len(result["tracks"]["items"]) == 0:
44         await ctx.send(" 検索結果が 0 件だったよ。キーワードを調整してみてね ")
45     elif len(result["tracks"]["items"]) == 1:
46         url = result["tracks"]["items"][0]["external_urls"]["spotify"]
47         await ctx.send(url)
48     elif len(result["tracks"]["items"]) > 1:
49         url = result["tracks"]["items"][0]["external_urls"]["spotify"]
50         message = f" 複数見つかったけど、これ？ {url}"
51         await ctx.send(message)
```

実際に app_try.py を実行すると `>search` と `>search_track` のコマンドを使い分けて Bot で検索することができます。実際に実行して確かめてみましょう。

2. 検索で取得したアーティストの ID を使って、アーティスト詳細を取得する API を実行し、日本語のアーティスト名を取得してみよう。それを Discord Bot に組み込んで表示してみよう

まずは以下を順番に実行していきましょう。

- ①検索でアーティストの ID を取得する。
- ②アーティストの ID を使って日本語のアーティスト名を含む、アーティスト詳細を取得する。

③上記を Discord Bot に組み込んで表示する。

①検索でアーティストの ID を取得する

まず、アーティストの ID は `spotify.py` での検索で取得している情報の中に含まれています。出力される JSON の中を確認してみましょう。入れ子になっている構造の中にキー `'id'` として格納されているのでこれを利用しましょう。

②アーティストの ID を使って日本語のアーティスト名を含む、アーティスト詳細を取得する

続いて、アーティストの詳細を取得するために利用する API のリファレンスを確認しましょう。以下のリファレンスで詳細を確認できます。アーティスト詳細を取得するには、Spotify API が提供するアーティストのエンドポイント (Get Artist) にリクエストを送信する必要があります。

参考 Web API Reference | Spotify for Developers

<https://developer.spotify.com/documentation/web-api/reference/get-an-artist>



ここでは API の出力の仕様も確認できます。まずはここでどのような形で情報が返ってくるか確認しておくといいでしょう。また、この際に日本語のアーティスト名を取得するには、Spotify API からの出力が日本語になるように指示が必要です。ここでは、`Accept-Language` リクエストヘッダーを利用します。これは HTTP リクエストにおいて一般的に使用される重要なヘッダーの 1 つです。詳細は以下から確認してみてください。

参考 RFC 7231 - Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content

<https://datatracker.ietf.org/doc/html/rfc7231#section-5.3.5>



`search_artist.py` をベースとして上記の条件を組み込みます。ファイル名は新たに `search_artist_challenge.py` とします。まずはこれまで同様に Search エンドポイントへリクエストを送信し検索結果を取得します。さらに今回はその結果からアーティスト ID を取得しておき、さらに Get Artist エンドポイントへリクエストを送信する方法になります。また、このリクエストを送信する際に日本語情報を指定しています。このコードの全文は `search_artist_challenge.py` で確認できます。

search_artist_challenge.py の一部

```

22 # 3. 指定されたキーワードで Spotify を検索
23 search_response = requests.get(
24     "https://api.spotify.com/v1/search",
25     headers={
26         "Authorization": f"Bearer {token}",
27         "Accept-Language": "ja" # 日本語情報を指定
28     },
29     params={
30         "q": "XXXX", # 検索したい任意のアーティスト名
31         "type": "artist",
32         "market": "JP",
33     },
34 )
35
36 # 4. 検索結果のアーティスト ID を取得
37 search_results = search_response.json()
38 artist_id = search_results['artists']['items'][0]['id']
39
40 # 5. アーティストの詳細情報を取得
41 artist_response = requests.get(
42     f"https://api.spotify.com/v1/artists/{artist_id}",
43     headers={
44         "Authorization": f"Bearer {token}",
45         "Accept-Language": "ja" # 日本語情報を指定
46     },
47 )
48
49 # 6. 結果の表示
50 pprint.pprint(artist_response.json())

```

この search_artist_challenge.py を実行すると次のような出力を得られます。これは、API の仕様書通りの形式であり、今回取得すべき目的はキー `name` に割り当てられている情報です。これが日本語で正しく取得できていれば問題ありません。なお、結果の一部をマスキングしています。

artist_response.json

```
{'external_urls': {'spotify': 'https://open.spotify.com/artist/*****'},
 'followers': {'href': None, 'total': 6439411},
 'genres': ['anime', 'j-pop'],
 'href': 'https://api.spotify.com/v1/artists/*****?locale=ja',
 'id': '*****',
 'images': [{'height': 640,
              'url': 'https://...',
              'width': 640},
             {'height': 320,
              'url': 'https://...',
              'width': 320},
             {'height': 160,
              'url': 'https://...',
              'width': 160}],
 'name': 'XXXX',
 'popularity': 75,
 'type': 'artist',
 'uri': 'spotify:artist:...'}
```

③上記を Discord Bot に組み込んで表示する。

続いてこのコードを Discord bot のコマンドで実行できるように修正します。修正した spotify_challenge.py は以下の通りです。コードの全文は spotify_challenge.py で確認できます。

spotify_challenge.py の一部

```
7 def search_artist(keyword):
8     # 1. Spotify で認証
9     auth_response = requests.post(
10         "https://accounts.spotify.com/api/token",
11         data={
12             "grant_type": "client_credentials",
13             "client_id": client_id,
14             "client_secret": client_secret,
15         },
16     )
17
```



```

18     # 2. アクセストークン取り出し
19     response_json = auth_response.json()
20     token = response_json["access_token"]
21
22     # 3. 指定されたキーワードで Spotify を検索
23     search_response = requests.get(
24         "https://api.spotify.com/v1/search",
25         headers={
26             "Authorization": f"Bearer {token}",
27             "Accept-Language": "ja" # 日本語情報を指定
28         },
29         params={
30             "q": keyword,
31             "type": "artist",
32             "market": "JP",
33         },
34     )
35
36     # 4. 検索結果のアーティスト ID を取得
37     search_results = search_response.json()
38     artist_id = search_results['artists']['items'][0]['id']
39
40     # 5. アーティストの詳細情報を取得
41     artist_response = requests.get(
42         f"https://api.spotify.com/v1/artists/{artist_id}",
43         headers={
44             "Authorization": f"Bearer {token}",
45             "Accept-Language": "ja" # 日本語情報を指定
46         },
47     )
48
49     return artist_response.json()

```

では最後に Discord Bot に組み込んで表示できるようにしましょう。今回は spotify_challenge.py で最終的に取得するアーティスト詳細のデータから名前を Discord Bot 上に表示する必要があります。取得する Json 形式の情報を確認して、それが正しく出力されるように app.py のコードを修正します。値が格納されていないキーを指定してしまうとエラーが発生します。ここではキー `name` を中心に利用して、アーティストの名前と取得した URL が Discord bot 上に表示できるようにコードを修正しました。修正した app_challenge.py は以下のようになります。

app_challenge.py の一部

```
11 @bot.command()
12 async def search(ctx, keyword):
13     if len(keyword) < 2:
14         await ctx.send("2 文字以上の検索キーワードを入力してね ")
15         return
16     elif len(keyword) > 20:
17         await ctx.send(" 検索キーワードは 20 文字以内にしておね ")
18         return
19
20     result = spotify_challenge.search_artist(keyword)
21
22     # 検索結果がない場合の対応
23     if not result:
24         await ctx.send(" 検索結果が 0 件だったよ。キーワードを調整してみてね ")
25         return
26
27     # アーティスト情報を取得して出力
28     name = result["name"]
29     url = result["external_urls"]["spotify"]
30     await ctx.send(f" アーティスト名 : {name}\nURL: {url}")
31
32 bot.run(TOKEN)
```

これで取得した情報から日本語のアーティスト名を Discord bot を使って表示できるようになりました。コードの全文は app_challenge.py で確認できます。22 行目からは if 文の条件が偽である際に処理を実行する `if not` を利用して、結果が返ってこない場合にメッセージを出力する処理を設定しています。また、取得しているアーティストの詳細の中には、`name`（名前）以外にもキーが割り当てられている情報が格納されています。興味があればそれらもちょっとコードを変更するだけで表示できますので挑戦してみてください。