第 10 章

Flask による Web アプリの構築

いわゆる「ホームページ」や「Web サイト」「Web ページ」と呼ばれているものは，ネット内でいつでもアクセスできる Web サーバ (Server) 上で提供される一種のアプリケーションソフトウェア（アプリ）を意味します。Web アプリを作成するための開発環境は様々なものがあり，日々，新しいものが提供されています。本章ではその中でも Python をベースとした「Flask」というモジュールの使い方を簡単に紹介します。

10.1　Web アプリケーションの概要

Web(「うぇぶ」と発音，蜘蛛の巣の意味) とは，インターネット上で公開されている Web サーバ (Web server) にある情報を (Web) ブラウザ (Web browser, Chrome，Safafi, Edge 等) で閲覧するシステムを意味します。Web サーバもブラウザも，基本は一台のマシン (スマートフォン，パソコン等の機器) 上で動作するネイティブアプリケーション (native application) ですが (10.1 の左図)，インターネットを介して Web サーバとブラウザが強調して動作する Web システム全体がアプリケーションの一種と言えますので，これを Web アプリケーション (10.1 の右図) とも呼称します。

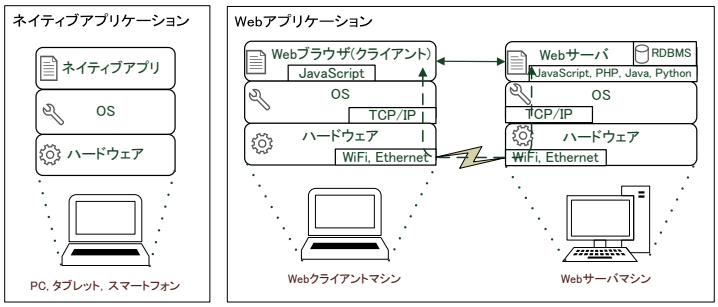

図 10.1　ネイティブアプリケーション（左）と Web アプリケーション

Web アプリケーションは，ユーザ側にブラウザさえあれば，いつでもアクセスできる Web サーバ上にある機構を使って様々なサービスを提供できるため，小さな組み込み機器 (IoT, Internet of Things) からスマートフォン，タブレット，パソコン等に至るまで，ハードウェアを選ばずに使用することができます。Web アプリケーションを作るためのソフトウェアも多様な組み合わせで開発でき，Python 環境でも，ここで取り上げる Flask 以外に，Django，FastAPI, Streamlit 等，様々なモジュールが Web アプリケーション開発用とし

て提供されています。

Flask はその中でも簡単な機能を作る事に向いており，Web サーバの側で動作する Web アプリケーションを手軽に作成することができます。実際にインターネット上で公開するにはサーバ環境を用意する必要がありますが，ここでは自分で作って自分で遊べるごく簡単な Web アプリケーションを作ってみることにしましょう。

10.2 HTML と CSS

まず，ブラウザで閲覧できる Web ページを HTML(Hyper Text Markup Language) ファイルとして作ってみましょう。今まで Python ソースコードを作ってきたテキストエディタで，あ，下記の内容を打ち込み，適切なフォルダに「first.html」という名前で保存して下さい。Python とは異なり，インデントはしてもしなくても問題ありませんが，構造を見やすくするためにあえて行っています。「自分の学籍番号」「Student Number」は自分の学籍番号に，「氏名」「Name」は自分の氏名に置き換えて記述して下さい。

ソースコード 10.1 first.html

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <meta charset="UTF-8" />
5          <title>The first HTML file</title>
6      </head>
7      <body>
8          <h1>最初のHTML ファイル</h1>
9          <h2>自分の学籍番号　氏名</h2>
10         <hr />
11         ここに本文を書く。
12         <hr />
13         <address>Copyright (c) 20xx Student Number, Name</address>
14     </body>
15 </html>
```

一読して分かることは，HTML ファイルの特徴として，タグ (tag) と呼ばれる**<タグ>...</タグ>**というブロックで文書が構造化されていることです。開始タグ (**<タグ>**) と終了タグ (**</タグ>**) がセットになっているものは

1. <html>...</html> HTML の中身 (省略可)
2. <head>...</head> Web ページのヘッダ情報 (ブラウザ本体には表示されない)
3. <title>...</title> Web ページタイトル (ブラウザのタブに表示される)
4. <body>...</body> Web ページの本文（ブラウザ本体に表示される）
5. <h1>...</h1> トップの見出し
6. <h2>...</h2> 2 番目の見出し
7. <address>...</address> アドレス (著作権表示などを書く)

です。

また，<!DOCTYPE html>(HTML5 の宣言)，<meta>（Web ページの情報），<hr>（水平線）は終了タグ (**</タグ>**) のない単独タグとなっています。このように文書の構造を表現するタグはたくさんありますので，興

味のある方は調べてみて下さい。

では，この `first.html` を，Chrome, Edge, Safari といったブラウザで直接開いてみましょう。図 10.2 のような表示がされるはずです。

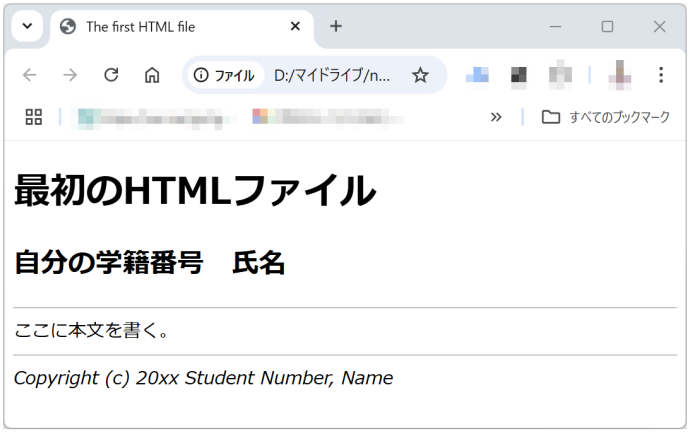

図 10.2　first.html を Chrome（ブラウザ）で閲覧したもの

これはあくまで「デフォルト」の表示ですので，ブラウザによっては異なる見栄えになるかもしれません。もう少しデザイン的に凝りたければ，CSS(Cascading Style Sheet) という機能を使って，フォントの書式や色，背景色や配置を自在に変更することができますので，興味のある方は調べてみて下さい。

以降ではこの HTML ファイルの構造を基本にして，Web サーバ側で動作する Web アプリケーションを作っていきます。

問題 10.1

本文に箇条書きを行う機能として`<ol>...</ol>`(Ordered List, 順序付き箇条書き)と`<ul>...</ul>`(Unordered List, 番号なし箇条書き）がある。どちらも項目は`<li>...</li>`(LIst, 箇条書きの項目) をこれらのタグの中に挿入して形成する。この 2 種類の箇条書きを使って図 10.3 という本文表示ができるよう，`first.html` を修正して `second.html` を作れ。

10.3　Flask で Hellow, Python!

Python 最初のスクリプト (1.1) がそうであったように，「ようこそ，Flask の世界へ！」という一文のみをコンソール（コマンドライン画面）とブラウザに表示する「hellow」アプリを作ってみましょう。先に，pip コマンド等を用いて Flask モジュールをインストールした Python 環境を用意しておきます。

次に，適当なフォルダ位置に「flask」フォルダを作成して下さい，以降の Web アプリはこの中にサブフォルダ単位で作成していくことにします。今回は「flask」フォルダ内に「hellow」フォルダを作成し，`flask\yen hellow` フォルダで作業を行っていきます。

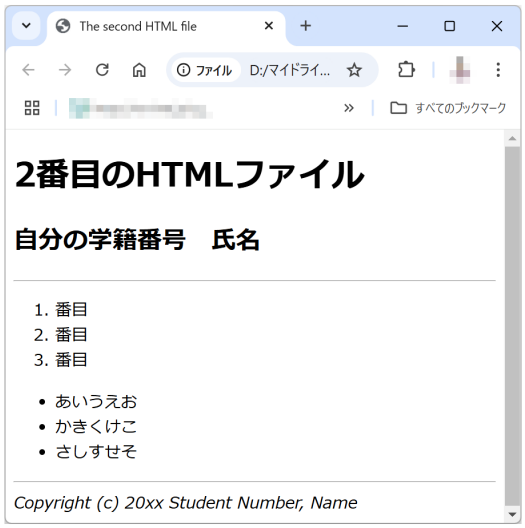

図 10.3　second.html を Chrome（ブラウザ）で閲覧したもの

ソースコード 10.2　flask¥hellow¥app.py

```
1  # flask\hellow\app.py: 最初のFlask アプリ
2  from flask import Flask # flask モジュールから Flask クラスをインポートする
3
4  app = Flask(__name__) # Flask のインスタンスを app に生成する
5
6  # トップページアクセス時の処理
7  @app.route('/')
8  def hellow_world():
9      mojiretsu = 'ようこそ，Flask の世界へ！'
10     print(mojiretsu) # コンソールに表示
11     return mojiretsu # ブラウザに出力
```

1. app.py をテキストエディタで作成し，flask¥hellow フォルダに保存する。
2. 次のコマンドで Flask を起動する。

   ```
   $ export FLASK_ENV='development' ←開発用モード（ソースコードの自動更新あり）の指定
   ```

3. ローカルマシンを Web サーバ（ポート番号は 5000 がデフォルト）としてアプリケーションを起動

   ```
   $ flask run
   ```

   上記が動かない場合は python -m flask run と指定
4. 起動したら，適当なブラウザ (Google Chrome か Edge を推奨) で

   ```
   http://127.0.0.1:5000/
   ```

   にアクセスし，図 10.4 のように retun で返される文字列 (mojiretsu 変数の中身) が表示されている

ことを確認する。

図 10.4　最初の Flask アプリ実行画面（ブラウザ）

また，この際，コマンドライン画面（コンソール）にも文字列が表示されていることも確認して下さい。これは print 文の処理によるものです。

問題 10.2
ブラウザのタブを複数開いて http://127.0.0.1:5000/にアクセスできることを確認せよ。

10.4　HTML テンプレートの使用

とりあえずブラウザに文字を表示させることはできましたが，これでは HTML ファイルに比べて情報が少なすぎて物足りません。そこで，テンプレート (template) 機能を用いて，必要な個所のみ書き換えを行って HTML ファイルを表示できるよう，|app.py—を書き換えてみましょう。変更箇所は下記の通りです。

ソースコード 10.3　flask¥hellow¥app.py

```
1  # app.py: 最初のFlaskアプリ(HTMLテンプレートの使用版)
2  from flask import Flask
3  from flask import render_template # 追加
4
5  app = Flask(__name__)
6
7  # トップ
8  @app.route('/')
9  def hellow_world():
10     mojiretsu = 'ようこそ，Flaskの世界へ！'
11     # print(mojiretsu) # コメントに
12     # return mojiretsu # コメントに
13     return render_template('
14 .html', mystr = mojiretsu) # 追加
```

flask モジュールから，render_template 関数をインポートし，これを用いて return で返す文字列を HTML ファイルに変更します。

次に，「hellow」フォルダ内に「templates」フォルダを生成し，この中に下記の template.html を作ります。前に作成した first.html をコピペして必要なところのみ書き換えて下さい。

ソースコード 10.4　flask¥hellow¥templates¥template.html

```
1  <!DOCTYPE html>
2  <html>
```

```
3     <head>
4         <meta charset="UTF-8" />
5         <title>First Flask Application by 学籍番号,自分の名前</title>
6     </head>
7     <body>
8         <h1>最初のFlask アプリケーション</h1>
9         <h2>自分の学籍番号　氏名</h2>
10        <hr />
11        {% block main_content %}
12        {% endblock %}
13        <hr />
14        <address>Copyright (c) 20xx 学籍番号，氏名</address>
15    </body>
16 </html>
```

特徴的なのは，本文に{% block ブロック名 %}{% endblock %}という記述があることです。このブロック部分のみ，アプリケーションの動作に応じて書き換えることができるようになります。「templates」フォルダ内に index.html を下記のように記述すると，{{ mystr }}のところが，アプリケーションから渡される変数 mystr で書き換えられます。

ソースコード 10.5　flask¥hellow¥templates¥index.html

```
1 {# index.html #}
2 {% extends 'template.html' %}
3 {% block main_content %}
4 <!-- 文字列を挿入 -->
5 {{ mystr }}
6 {% endblock %}
```

前回同様，app.py を起動させ，ブラウザで閲覧すると 10.5 のように，HTML ファイルとして評させることができるようになります。

図 10.5　HTML テンプレート使用時

問題 10.3

10.3 と同じ表記を行えるようにテンプレートを使用した「second」アプリを作成し，ブラウザを動作させて実行結果を確認せよ。

10.5　フォームを用いた数式の計算

次に，1 変数関数 $f(x)$ を x を用いた式としてユーザが与え，指定した x の値を用いて $f(x)$ の値を計算する Web アプリ「func_calc」を作ってみましょう。そのためには数式を文字列として与え，それを計算式とし

て評価 (evaluate) する式パーサ (formula parser) の機能が必要になります。

インタプリタ言語の特徴の一つに，スクリプトの中でスクリプト文を動作する事ができるということが挙げられます。Python の場合は eval 関数というものがあり，例えば下記のように，x**2 + 3 という文字列を str_formula 変数に与えて eval 関数の引数として渡して実行すると，実行時における変数 x の値を評価して計算してくれます。

```
>>> str_formula = 'x**2 + 3'
>>> x = 3
>>> eval(str_formula)
12
```

使いようによっては大変便利な関数ですが，Web プログラミングでは，ユーザからの入力は危険なものも含まれているという前提でスクリプトを組む必要があり，eval 関数の使用は望ましくありません。

そこで，NumPy と組み合わせての動作も可能な式パーサを Pytnon で使用できるようにした NumExpr を使用することとします。pip コマンドなどで NumExpr をインストールし，上記の計算が可能であることを確認しましょう。

```
>>> import numexpr as ne
>>> str_formula = 'x**2 + 3'
>>> x = 3
>>> float(ne.evaluate(str_formula))
12.0
```

eval 関数とは異なり，形指定をして値を取り出す必要があります。

この式パーサを使って, ユーザから与えられた数式を計算するフォーム (form) を作ってみましょう。フォームとは，HTML のタグの一つで，下記のようにブラウザ経由でユーザからの入力を受け付ける機構を提供してくれます。まずは最小限の機能を使って数式計算用のフォームを作ってみましょう。前節で作ったテンプレート (template.html) を使い index.html(図 10.6) を作ります。

ソースコード 10.6 flask¥func_calc¥template¥index.html

```
1  {# index.html: 関数計算 #}
2  {% extends 'template.html' %}
3  {% block main_content %}
4      <!-- 入力フォーム -->
5      <h2>関数入力フォーム</h2>
6      <form action="/calc" method="POST">
7      <ul>
8          <li>関数式（「x」を変数として指定）: <input type="text" name="input_formula" size
               ="30" /></li>
9          <li>x の値：<input type="text" name="input_x" size="10" /></li>
10     </ul>
11     <p><input type="submit" value="計算実行！"> <input type="reset" value="リセット
           "></p>
12     </form>
13     <hr />
14     <!-- 計算値出力 -->
15     <h2>計算結果</h2>
```

```
16          <p>x = {{ str_x }}</p>
17          <p>f(x) = {{ str_formula }} = {{ str_func_val }} </p>
18      {% endblock %}
```

フォームは<form>...</form>で囲まれている部分で，この中の<input>タグでユーザからの入力を受け付けます。ここで使用している入力タイプは下記の 3 つです。

text　・・・テキストボックス (文字列)
submit　・・・サブミットボタン (入力データの送信)
reset　・・・フォーム内入力のリセット

サブミットボタンがクリックされる際に入力されていたテキストボックスなどの入力値に，それぞれ name 属性で指定された文字列がセットされ，<form>の action 属性で指定された URL に，method で指定された手法 (この場合は POST メソッド) で入力データが転送されます。

例えば $x = 3$ の時，$x^2 + 3 = 3^2 + 3 = 12$ を計算したい場合は，図 10.6 のように実行します。

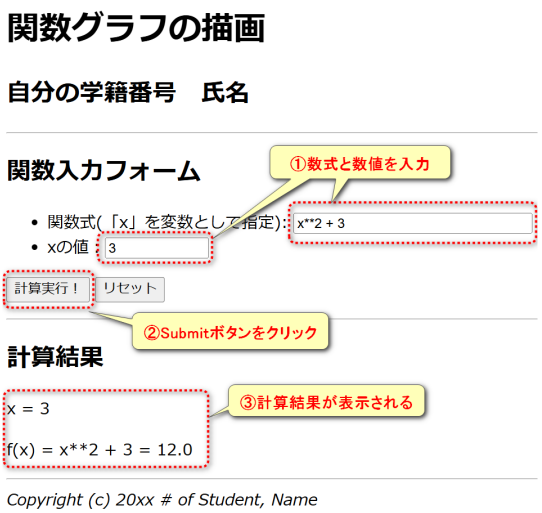

図 10.6　関数値計算アプリのフォーム

この環数値計算アプリの動作を実現するためには，フォームからの数式と x の値を受け取って処理する必要があります。「/calc」にアクセスされた際にはフォームの値が入力されているという前提で処理を行うようにすると，下記のように次のようになるでしょう。

ソースコード 10.7　flask¥func_calc¥app.py

```
1  # app.py : 関数式の計算
2  from flask import Flask
3  from flask import render_template
4  from flask import request # フォーム入力を受け取る
5  from numpy import * # NumPy 関数を直接呼出し
6  import numexpr as ne # NumExpr: 式パーサ
7
8  app = Flask(__name__)
```

```
9
10 # 関数フォームを開く
11 @app.route('/')
12 def index():
13     return render_template('index.html')
14
15 # 関数値の評価
16 @app.route('/calc', methods = ['POST']) # POST でのみ受付
17 def calc():
18     str_formula = request.form['input_formula']
19     str_x = request.form['input_x']
20     # 計算実行
21     x = float(str_x) # x を数値に変換
22     func_val = float(ne.evaluate(str_formula))
23     # 計算結果を戻す
24     return render_template('index.html', str_x = str_x, str_formula = str_formula,
           str_func_val=str(func_val))
```

フォームから渡されるデータは request.form リストに保存されており，<input>タグの name 属性値をインデックスとして参照できます。

問題 10.4

x の値と関数 f が次のように与えられるときの $f(x)$ の値を func_calc アプリを使って求め，実行時のブラウザ画面のスナップショットを画像として保存せよ。

1. $x = -5,\ f(x) = (x-3)(x+4)(x-1)$
2. $x = -3,\ f(x) = \exp(\cos(x^2)) = e^{\cos(x^2)}$

10.6 数式に基づくグラフ描画アプリ

関数値計算アプリを拡張し，指定区間 $x \in [a, b]$ における関数 $f(x)$ のグラフを描画する「graph」アプリを作ってみましょう。図 10.7 のようなフォームを index.html として作り，関数グラフを PNG 画像としてブラウザ上に表示するようにします。

flask¥graph¥templates¥template.html は省略し，下記の index.html を入力フォームとして使えるようにしておいて下さい。

ソースコード 10.8 flask¥graph¥templates¥index.html

```
1  {# index.html: 数式入力フォーム＆グラフ描画画面 #}
2  {% extends 'template.html' %}
3  {% block main_content %}
4      <h2>関数</h2>
5          {% if formula %}
6          <div name="graph">
7              function: {{ formula }}<br />
8              x : [{{ x_min }}, {{ x_max }}]
9              <!-- y = {{ y_vals }} -->
10             <img src="{{␣img_name␣}}" alt="{{␣formula␣}}" />
11         </div>
12         {% endif %}
13         <form action="/draw" method="POST">
```

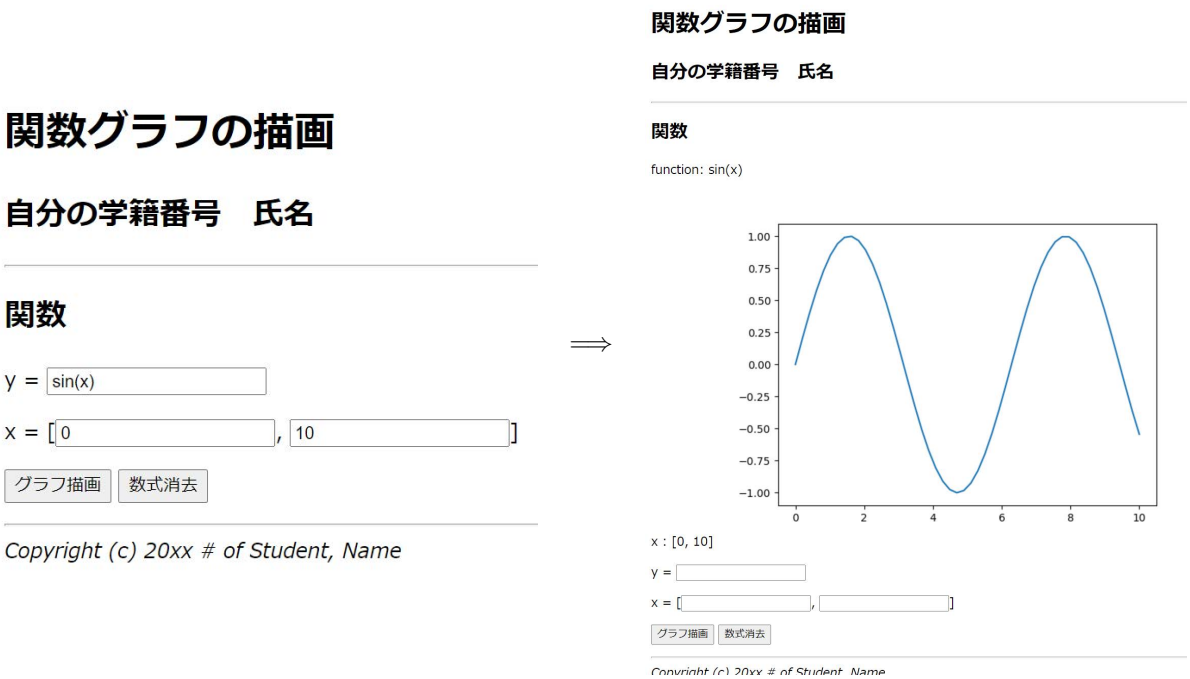

図 10.7　Flask による関数グラフ描画ツールの実行画面

```
14            <p>y = <input type="text" name="formula" length="20" /></p>
15            <p>x = [<input type="text" name="x_min" length="5" />, <input type="text"
                  name="x_max" length="5" />]</p>
16            <p><input type="submit" value="グラフ描画" /> <input type="reset" value="数式
                  消去" /></p>
17        </form>
18    {% endblock %}
```

描画した関数は PNG ファイルとして保存され，ファイルの位置が `img_name` として渡される，という前提で`<img>`タグの指定が行われています。

この入力フォームからの区間の端点を a =`x_min`，b =`x_max` として与え，関数式は `formula` として与えます。

グラフを描画するためには区間を分割し，各分割点で関数値を計算する必要があります。ここでは `div` で分割数を指定しています。

ソースコード 10.9　`flask¥graph¥app.py`

```
1   # graph/app.py : 関数グラフの描画
2   from flask import Flask
3   from flask import render_template
4   from flask import request # methods as argument
5   from flask import redirect
6   from flask import Response
7   from numpy import * # NumPy の関数を使用する
8   import numexpr as ne # 式パーサ
9   import matplotlib.pyplot as plt # グラフ描画ライブラリ
10
```

```
app = Flask(__name__)

# 固定ディレクトリ
PNG_DIR = 'static/'

# Cache コントロール用
# https://stackoverflow.com/questions/45583828/python-flask-not-updating-images
app.config['SEND_FILE_MAX_AGE_DEFAULT'] = 0

#@app.after_request
def add_header(response):
    response.headers['Cache-Control'] = 'no-store, no-cache, must-revalidate, post-
        check=0, pre-check=0, public, max-age=0'
    response.headers['Pragma'] = 'no-cache'
    response.headers['Expires'] = '0'
    return response


@app.route('/')
@app.route('/index')
def index():
    return render_template('index.html')


@app.route('/draw', methods = ['GET', 'POST'])
def draw():
    formula_text = request.form['formula']
    x_min = request.form['x_min']
    x_max = request.form['x_max']
    div = 50
    graph_img_name = PNG_DIR + 'graph.png'

    # 計算とグラフ描画
    x_array = linspace(float(x_min), float(x_max), div)
    y_array = [ne.evaluate(formula_text) for x in x_array]
    fig, ax = plt.subplots()
    ax.plot(x_array, y_array)
    fig.savefig(graph_img_name)
    return render_template('index.html', formula = formula_text, x_min = x_min, x_max
        = x_max, img_name = graph_img_name)
```

まずは図 10.7 の実行例のように動作するかどうか，$f(x) = \sin(x),\ x \in [0, 10]$ を入力して動作を確認してみましょう。

問題 10.5

次の関数グラフを描画せよ。なお，四則演算 (+, -, *, /) は省略せずに書くこと。例) ：例 $2x/3 + \cos(x^2)$ → `2 * x / 3 + cos(x * x)`

1. $(x-3)(x+4)(x-1),\ x \in [-5, 5]$
2. $\exp(\cos(x^2)),\ x \in [-3, 3]$

[できそうならチャレンジ！問題] 現状では x 区間の分割数を 50(`div=50`) に固定している。この分割数もユー

ザが入力して変更できるようにせよ。

10.7　統計ツールを Web アプリに

本章の最後に，前の章で作成した Excel ファイルを読み込み，各学生ごとの成績データをグラフ化する「statistic」アプリを作ってみましょう。実行時には図 10.8 のように，成績表示する学生を選べるようにフォームを使います。

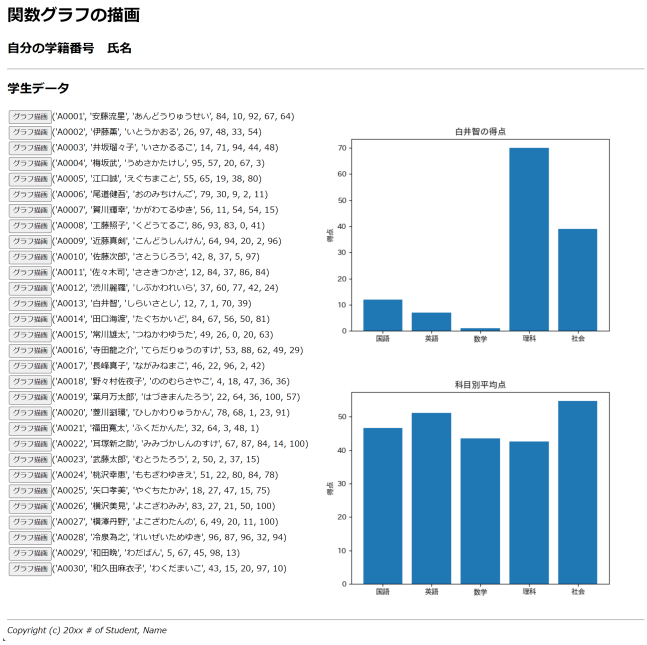

図 10.8　Flask による簡易統計ツール実行時の画面

テンプレートファイルは省略します。また，`index.html` で表示する学生選択のためのフォームは，学生数が多いため，Python スクリプトで自動生成しますので，ここでフォームは定義しないようにします。

ソースコード 10.10　`flask¥statistics¥templates¥index.html`

```
1    {# index.html: 学生データ表示＆グラフ描画画面 #}
2    {% extends 'template.html' %}
3    {% block main_content %}
4        <h2>学生データ</h2>
5        <div name="table" style="float:left">
6            {{ list|safe }}
7        </div>
8        <div name="graph">
9            <img src="{{ graph1 }}" alt="{{ graph1_comment }}" />
10           <img src="{{ graph2 }}" alt="{{ graph2_comment }}" />
11       </div>
12   {% endblock %}
```

Python スクリプトは下記のようになります。

ソースコード 10.11 flask¥statistics¥app.py

```
# flask\statistics\app.py : CSV ファイルの描画
from flask import Flask
from flask import render_template
from flask import request # methods as argument
from flask import redirect
from flask import Response
import matplotlib # グラフ描画ライブラリ
matplotlib.use('Agg') # 再プロット時のエラー防止 (GUI 禁止)
import matplotlib.pyplot as plt # グラフ描画ライブラリ
import japanize_matplotlib # Matplotlib 日本語対応
# pandas
import numpy as np # NumPy
import pandas as pd # Pandas

app = Flask(__name__)

# 固定ディレクトリ
PNG_DIR = 'static/'

# Cache コントロール用
# https://stackoverflow.com/questions/45583828/python-flask-not-updating-images
app.config['SEND_FILE_MAX_AGE_DEFAULT'] = 0

# 選択した生徒の成績
find_row_student = {}
find_row_scores = []

# Excel ファイル名
filename = 'student_grade.xlsx'
# Excel シート名
sheetname = '学籍番号・氏名・タイトルなし'
# 教科名
subjects = ['国語', '英語', '数学', '理科', '社会']
# フィールド数 (カラム数)
# 学籍番号,氏名,氏名読み仮名,教科名
max_num_column = 3 + len(subjects)
# 科目平均点グラフファイル名
subject_graph_name = 'subject_graph.png'
# 生徒別グラフファイル名
student_graph_name = 'student_graph.png'
# 選択した生徒の成績
#find_row_student = {}
# canvas_right, left_flag
canvas_right_flag = 0 # 存在なし
canvas_left_flag = 0 #

# 関数フォームを開く
@app.route('/')
@app.route('/index')
def index():
    # Excel ファイル読み込み確認
```

```
52      try:
53          pd.ExcelFile(filename)
54      except:
55          print(filename, 'が開けませんでした。')
56
57      # Excel ファイル読み込み時のみ動作
58      with pd.ExcelFile(filename) as xls:
59          sheet = pd.read_excel(xls, sheetname)
60          #print(sheet)
61
62          # 科目ごとの平均点,標準偏差,最高点,最低点
63          for i in range(len(subjects)):
64              print('{:s}の平均点,標準偏差,最高点,最低点:␣{:3.1f},␣{:5.3g},␣{:4d},␣{:4d}'.
                    format(
65                  subjects[i],
66                  np.average(sheet[subjects[i]]),
67                  np.std(sheet[subjects[i]]),
68                  np.amax(sheet[subjects[i]]),
69                  np.amin(sheet[subjects[i]])
70              ))
71
72          # 平均点のグラフ化: subject_graph.png
73          x = subjects # ['国語', '英語', '数学', '理科', '社会']
74          y = [np.average(sheet[x[i]]) for i in range(5)]
75          fig, ax = plt.subplots()
76          # 棒グラフ
77          ax.set_title('科目別平均点')
78          ax.set_ylabel('得点')
79          ax.bar(x, y)
80          # グラフ表示
81          #plt.show()
82          # ファイルに書き出し
83          plt.savefig(PNG_DIR + subject_graph_name)
84
85          # 一行ごとに読み込み
86          table = '<table>'
87          print('sheet.shape:␣', sheet.shape)
88          for i in range(sheet.shape[0]):
89              # table += '<tr><td>' + str(tuple(sheet.iloc[i, :])) + '</td></tr>'
90              table += '<tr><td>'
91              table += '<form␣action="/draw"␣method="POST">'
92              table += '<input␣type="submit"␣value="グラフ描画">'
93              table += str(tuple(sheet.iloc[i, :]))
94              table += '<input␣type="hidden"␣name="student_id"␣value="' + sheet.iloc[i,
                    0] + '">'
95              table += '</form>'
96              table += '</td></tr>'
97              print(tuple(sheet.iloc[i, :]))
98
99          table += '</table>'
100
101     return render_template('index.html', list = table)
102
103 # 関数グラフ描画
```

```
104 @app.route('/draw', methods = ['GET', 'POST'])
105 def draw():
106
107     # 学生id取得
108     student_id = request.form['student_id']
109
110     # Excel ファイル読み込み確認
111     try:
112         pd.ExcelFile(filename)
113     except:
114         print(filename, 'が開けませんでした。')
115
116     # Excel ファイル読み込み時のみ動作
117     with pd.ExcelFile(filename) as xls:
118         sheet = pd.read_excel(xls, sheetname)
119         #print(sheet)
120
121         # 科目ごとの平均点,標準偏差,最高点,最低点
122         for i in range(len(subjects)):
123             print('{:s}の平均点,標準偏差,最高点,最低点:␣{:3.1f},␣{:5.3g},␣{:4d},␣{:4d}'.
                    format(
124                 subjects[i],
125                 np.average(sheet[subjects[i]]),
126                 np.std(sheet[subjects[i]]),
127                 np.amax(sheet[subjects[i]]),
128                 np.amin(sheet[subjects[i]])
129             ))
130
131         # 平均点のグラフ化: subject_graph.png
132         x = subjects # ['国語', '英語', '数学', '理科', '社会']
133         y = [np.average(sheet[x[i]]) for i in range(5)]
134         fig, ax = plt.subplots()
135         # 棒グラフ
136         ax.set_title('科目別平均点')
137         ax.set_ylabel('得点')
138         ax.bar(x, y)
139         # グラフ表示
140         #plt.show()
141         # ファイルに書き出し
142         plt.savefig(PNG_DIR + subject_graph_name)
143
144         # 一行ごとに読み込み
145         table = '<table>'
146         print('sheet.shape:␣', sheet.shape)
147         for i in range(sheet.shape[0]):
148             # table += '<tr><td>' + str(tuple(sheet.iloc[i, :])) + '</td></tr>'
149             table += '<tr><td>'
150             table += '<form␣action="/draw"␣method="POST">'
151             table += '<input␣type="submit"␣value="グラフ描画">'
152             table += str(tuple(sheet.iloc[i, :]))
153             table += '<input␣type="hidden"␣name="student_id"␣value="' + sheet.iloc[i,
                        0] + '">'
154             table += '</form>'
155             table += '</td></tr>'
```

```
156             print(tuple(sheet.iloc[i, :]))
157             # 学生id の発見
158             if sheet.iloc[i, 0] == student_id:
159                 find_row_student = sheet.iloc[i, 1]
160                 find_row_scores = [sheet.iloc[i, j] for j in range(3, 8)]
161 
162         table += '</table>'
163 
164         # 学生別グラフの描画
165         #find_row_scores = []
166         student_name = ''
167         if len(find_row_student) > 0 :
168             student_name = find_row_student
169             #find_row_scores = [find_row_student[subjects[i]] for i in range(len(
                    subjects))]
170             print('find_row_scores␣=␣', find_row_scores)
171 
172         # 個人成績のグラフ化
173         x = subjects # ['国語', '英語', '数学', '理科', '社会']
174         y = find_row_scores # [find_row[x[i]] for i in range(5)]
175         #print(y)
176         fig_right, ax = plt.subplots()
177         # 棒グラフ
178         ax.set_title(student_name + 'の得点')
179         ax.set_ylabel('得点')
180         if len(student_name) > 0:
181             ax.bar(x, y)
182         # グラフ表示
183         #plt.show()
184         plt.savefig(PNG_DIR + student_graph_name)
185 
186     print('graph1,␣graph2␣=␣', student_graph_name, subject_graph_name)
187     return render_template('index.html', list = table, graph1 = PNG_DIR +
            student_graph_name, graph1_comment = '学生データ
            ', graph2 = PNG_DIR + subject_graph_name, graph2_comment = '教科別')
```