Bringing your Python script to more users!

Quick tour from CLI through GUI to Web app with image size reduction script

EuroPython 2020 (2020/07/23) Takuya Futatsugi (a.k.a. nikkie)

Hello EuroPython!

- Nice to meet you!😃 Please call me nikkie.
- Participating from Japan🇯🇵 (It's 6 p.m. in Japan.)
- Thank you for the opportunity to speak online!

Introduce myself (nikkie)

- Twitter @ftnext / GitHub @ftnext
- Data Scientist (NLP) at UZABASE, inc. Tokyo, Japan
- My favorites💖: To automate boring stuff with Python
  & Anime (Japanese cartoons)
- PyCon JP staff (2019/2020)

Do you know PyCon JP?

- Python Conference Japan🇯🇵 https://pycon.jp/2020/
- Conference: **August 28(Fri.) & 29(Sat.)** at **ONLINE**
- Tickets for attending a conference session by Zoom ON SALE!
  - Conference sessions will also be streamed on YouTube Live for free (without a ticket).

Why Bringing your Python script to more users?

Automate boring stuff with Python

- A first Python book (ex. “Automate the boring stuff with Python”) allow you to write a Python script to automate the boring stuff.
- It's beneficial to have Python do the boring stuff for me.

Automate boring stuff with Python for others

- Python script should help others who have similar boring stuff.
- Share **how to bring your useful script to others**.
- Try one of what I introduce after your first Python book.😃

Go on a quick tour🏃

Share 3 implementations to convert a script for others to use:

1. Command Line Interface (CLI app)
2. Graphical User Interface (GUI app)
3. Web app

What kind of boring stuff?

Boring stuff example: Resize images to small size

- I have lots of images of varying sizes.

Boring stuff example: Resize images to small size

- Resize all images to fit in 300px square, keeping the ratio.

300px

300px

Boring stuff example: Resize images to small size

- Resizing images one by one by hand😫
- 👉 Automate it with Python!
- Wrote shrink_image.py, referring to a first Python book.

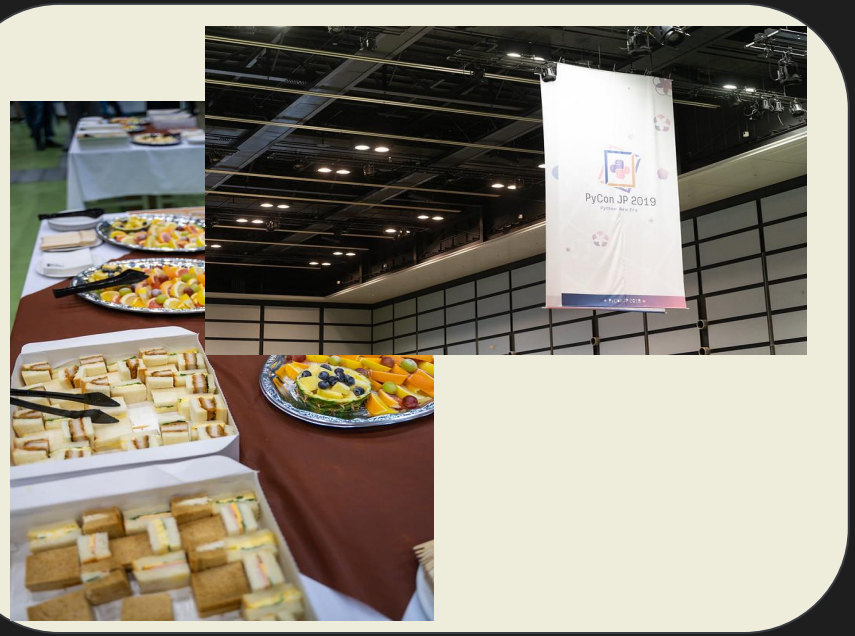
Overview of shrink_image.py

```
from pathlib import Path
from PIL import Image  # pip install Pillow
# The directory which includes images of varying sizes
image_dir_path = Path("../target_images/img_pyconjp")

for image_path in image_dir_path.iterdir():
    # Image.open(image_path), resize it and save at save_path
    has_resized = resize_image(image_path, save_path, 300)
```

How the script works (before)

```
├── target_images
│   └── img_pyconjp
└── start
    └── shrink_image.py
```

```
$ python shrink_image.py
```



How the script works (after)

```
├── target_images
│   └── img_pyconjp
└── start
    ├── shrink_image.py
    └── images
```

Resized images

Table of contents

Command Line Interface

Issue of the current script

- HARD-CODED target direcotry

```
# The directory which includes images of varying sizes
image_dir_path = Path("../target_images/img_pyconjp")
```

- Need to edit the script to target a different directory.

Resolve the issue: CLI

- Specify the target directory **from the command line**.
  - e.g. `$ python awesome.py spam ham`
- No need to edit the script every time you run it.
- Introduce **argparse**. (Document)

First example of argparse: hello_world.py

```
from argparse import ArgumentParser
# allow to take (required) arguments from the command line
parser = ArgumentParser()
parser.add_argument("name")
args = parser.parse_args()

# name attribute of args stores the (str) value specified in
print(f"Hello, {args.name}")  # command line
```

Run first example of argparse

```
# If name is not specified, a help message is displayed
$ python hello_world.py
usage: hello_world.py [-h] name
hello_world.py: error: the following arguments are required: name

# Call example
$ python hello_world.py EuroPython
Hello, EuroPython
```

Change shrink_image.py into CLI app

```
from argparse import ArgumentParser
# allow to take (required) arguments from the command line
parser = ArgumentParser()
parser.add_argument("target_image_path")
args = parser.parse_args()

# Before: image_dir_path = Path("../target_images/img_pyconjp")
image_dir_path = Path(args.target_image_path)
# No need to change anything other than what is shown here!
```

Run shrink_image.py (CLI app ver.)

```
├── target_images
│   └── img_pyconjp
└── cli
    ├── shrink_image.py
    └── images
```

Resized images

```
$ python shrink_image.py ../target_images/img_pyconjp
```

Brush up: Specify max length (int value and optional)

```
from argparse import ArgumentParser
parser = ArgumentParser()
parser.add_argument("target_image_path")
# Arguments which start -- are optional to specify. (Document)
# Arguments without -- are required and the order is important.
parser.add_argument("--max_length", default=300, type=int)
args = parser.parse_args()  # args.max_length
```

Brush up: Specify max length (int value and optional)

```
from argparse import ArgumentParser
parser = ArgumentParser()
parser.add_argument("target_image_path")
# type=int: convert entered str to int (Document)
# default: the default value if you do not specify it (Document)
parser.add_argument("--max_length", default=300, type=int)
args = parser.parse_args()
```

Run shrink_image.py with max length

```
├── target_images
│   └── img_pyconjp
└── cli
    ├── shrink_image.py
    └── images
```

Resized images (smaller)

```
$ python shrink_image.py ../target_images/img_pyconjp \
    --max_length 200
```

When a path to a non-existent directory is specified

```
# Users will be surprised when they see the traceback
$ python shrink_image.py ../target_images/img_pycon \
    --max_length 200
Traceback (most recent call last):
  File "shrink_image.py", line 75, in <module>
    for image_path in image_dir_path.iterdir():
FileNotFoundError: [Errno 2] No such file or directory:
'../target_images/img_pycon'
```

Tips: when a path to a non-existent directory is specified 1/2

```
from argparse import ArgumentParser
parser = ArgumentParser()
# Specify a function as the value of type parameter.
# Function existing_path converts entered str value to Path.
parser.add_argument("target_image_path", type=existing_path)
parser.add_argument("--max_length", default=300, type=int)
args = parser.parse_args()
```

Tips: when a path to a non-existent directory is specified 2/2

```
from argparse import ArgumentTypeError
def existing_path(path_str):
    """converts str to pathlib.Path"""
    path = Path(path_str)
    # if path does not point to any existing files or directories,
    if not path.exists():
        message = f"{path_str}: No such file or directory"
        raise ArgumentTypeError(message)  # raises an exception
    return path
```

Run shrink_image.py: when a path to a non-existent directory is specified

```
$ python shrink_image.py ../target_images/img_pycon \
    --max_length 200
usage: shrink_image.py [-h] [--max_length MAX_LENGTH]
target_image_path
shrink_image.py: error: argument target_image_path:
../target_images/img_pycon: No such file or directory
```

Recap: CLI

- Introduce argparse.
  - Specify arguments from the command line
  - No need to edit the script
- `add_argument("required"), add_argument("--optional")`
- Using `type` parameter in `add_argument`, convert specified str value from the command line to other types.

FYI: CLI

- How to execute as a command
- How to distribute
- Other packages

Table of contents

Graphical User Interface

Cons of CLI apps

Developers are familiar with CLI apps, but ...

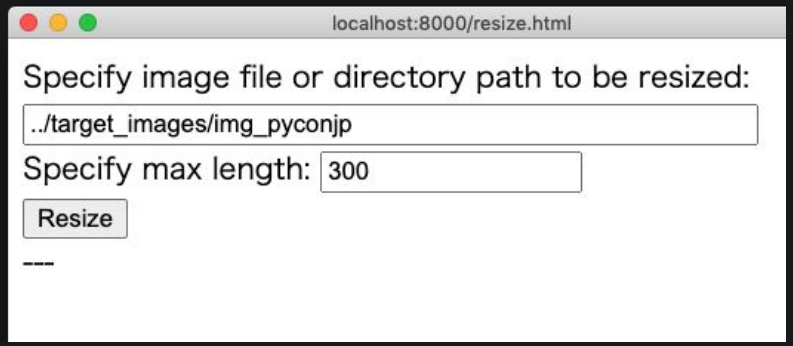
- People who aren't developers feel the same way? 🤔
- I want non-developers to use the app.

Make up for cons of CLI apps

- I want people who aren't developers to use the app.
  - make the app more user-friendly than CLI.
- **GUI** apps should be more **familiar** and **user-friendly** to non-developers than CLI.
  - many GUI apps in PCs!

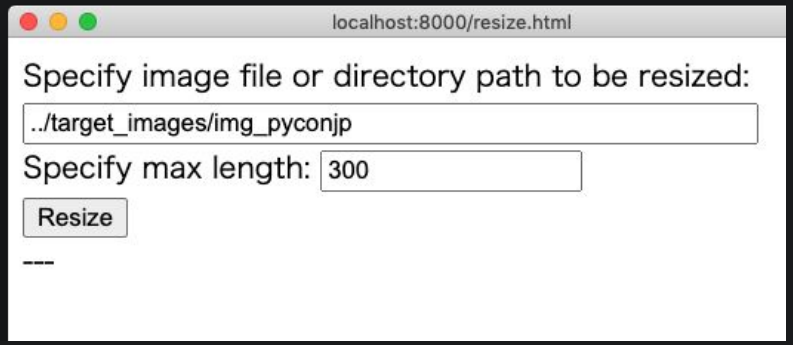
Goal: convert CLI into GUI

```
$ python shrink_image.py \
  ../target_images/img_pyconjp \
  --max_length 200
```



What you can do with the GUI app (Demo)

- Enter a path of the directory and max length in the input fields.
- Then click the “Resize” button.
- The app resizes images to smaller sizes.



GUI apps with Python

- introduce **Eel** (one of so many packages for GUI apps)
- Eel could make it easier to implement GUI apps.
- Prerequisite for Eel: Google Chrome needs to be installed.

Components of a GUI app made with Eel

- Python
- HTML
- JavaScript
- CSS (👉Appendix)

Eel components: HTML

- code written using tags.
  - input field: <input>
  - button: <button>
- Defines **structures** of GUI apps.
- Learn more: HTML basics - Learn web development | MDN

```
<input id="image-path-input"
placeholder="Type image path here"
value="/Users/" size="60">
<button type="button"
onclick="resize()">Resize</button>
```

Eel components: JavaScript 1/2

- Adds interaction to GUI apps
  - e.g. When a user click the button, JavaScript changes the screen by rewriting certain tags in the HTML

```
function resize() {  // Indentation is usually two spaces
  let targetImagePath = document.getElementById(
    "image-path-input").value;  // requires trailing ;
  // ... snip ...
}
```

Eel components: JavaScript 2/2

- Key point: Eel allows you to **call functions written in Python from JavaScript**.
  - enable to convert a Python script into a GUI app with just a little HTML and JavaScript.
- Learn more: JavaScript basics - Learn web development | MDN

Directory structure for Eel apps

```
gui
├── shrink_image.py  # Python
└── web
    └── resize.html  # HTML & JavaScript written in HTML
```

First example of Eel: Hello World

- Click “Greet” button, then the app displays a greeting

Directory structure for Hello World app

```
gui
├── hello_world.py
└── hello
    └── hello.html


# Start the app (Google Chrome will launch)
$ python hello_world.py
# Enter Ctrl+C when you exit the app
```

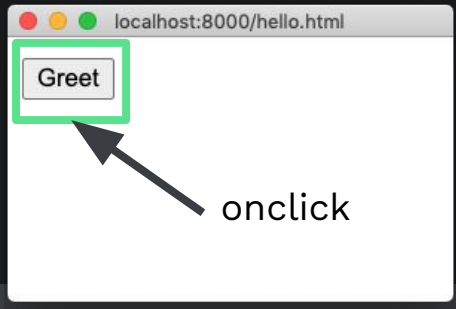
Impl. of hello_world.py

```
import random
import eel
@eel.expose  # Functions decorated @eel.expose can be
def say_hello():  # called by JavaScript
    return f"Hello World {random.choice(list(range(10)))}"

eel.init("hello")
# Specify to use hello.html under hello directory
eel.start("hello.html", size=(300, 200))
```

Impl. of hello.html (HTML)


```
<!DOCTYPE html>
<html>
  <head>
    <script type="text/javascript" src="/eel.js"></script>
    <script type="text/javascript">/* next slide */</script>
  </head>
  <body> <!-- When this button is clicked, greeting -->
    <button type="button" onclick="greeting()">Greet</button>
    <p id="greeting"></p> <-- (JavaScript function) is called -->
  </body>
</html>
```

Impl. of hello.html (JavaScript) 1/2



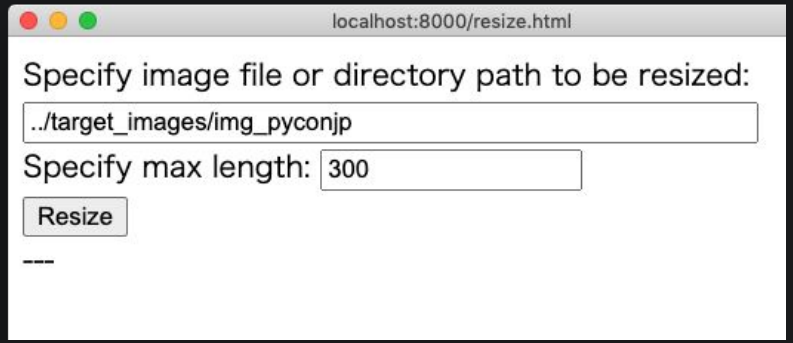

```
function greeting() {  // Called when the button is clicked
  // 1. eel.say_hello: Call the say_hello function in Python file
  // 2. Call the print_greeting function (next slide)
  // with the return value of the say_hello function
  //   e.g.) say_hello returns "Hello World 1"
  //       -> print_greeting("Hello World 1")
  eel.say_hello()(print_greeting);
}
```

Impl. of hello.html (JavaScript) 2/2


```
function print_greeting(message) {
  // operates the HTML element which id equals "greeting"
  // (or, operates <p id="greeting"></p>)
  let greeting = document.getElementById("greeting");
  // <p></p> -> <p>message</p>: displays message on the screen
  greeting.innerHTML = message;
}
function greeting() {
  eel.say_hello()(print_greeting);
}
```

Convert CLI to GUI

- Resize images in the entered directory to the entered size.



Directory structure for image resize app

```
gui
├── shrink_image.py
└── web
    ├── resize.html
    └── images  # put the resized images

# Start the app (Enter Ctrl+C when you exit)
$ python shrink_image.py
```

Overview of shrink_image.py

```
@eel.expose
def resize(target_image_path_str, max_length):
    target_image_path = existing_path(target_image_path_str)
    # You can manipulate files from Python without restriction.
    for image_path in target_image_path.iterdir():
        resize_image(image_path, save_path, max_length)
    # Returns paths of resized images, e.g. ["images/ham.png", ...]
    return save_paths
```

Impl. of resize.html (HTML)

```
<!-- Fields users can enter -->
<input id="image-path-input" placeholder="Type image path here"
value="/Users/" size="60">
<input id="max-length-input" value="300">

<button type="button" onclick="resize()">Resize</button>

<div id="resized-image"></div>
```

Impl. of resize.html (JavaScript) 1/2

```
function resize() {
  // get the value entered in an element using the id
  let targetImagePath = document.getElementById(
    "image-path-input").value;
  let maxLengthStr = document.getElementById(
    "max-length-input").value;
  let maxLength = parseInt(maxLengthStr, 10);  // convert to int
  eel.resize(targetImagePath, maxLength)(listUpImages);
}
```

Impl. of resize.html (JavaScript) 2/2

```
function listUpImages(imagePaths) {
  var imageHtml = `<p>No specified file or directory</p>`;
  if (imagePaths) {
   imageHtml = imagePaths.map(path => `<img src=${path}>`).join('');
 }
  let imageDiv = document.getElementById("resized-image");
  imageDiv.innerHTML = imageHtml;
}
```

Distribute eel app

- pip install PyInstaller
- e.g. from macOS to macOS
- not easy to distribute (some pitfalls) 😢
- ref: https://github.com/samuelhwilliams/Eel#building-distributable-binary-with-pyinstaller

Recap: GUI

Introduce Eel

- Components: Python, HTML, JavaScript(, CSS)
- Call Python functions from JavaScript (`@eel.expose`, callback)
- In HTML, set a JavaScript function (`onclick`)
- In JavaScript, get the entered values and rewrite the contents

FYI: GUI

- How to debug JavaScript code
- Other packages

Table of contents

Web app

Cons of GUI apps

GUI apps are user-friendly to non-developers, but ...

- Distribution is sometimes tough.
- Installation may be a bit difficult.

Make up for cons of GUI apps

- Avoid distribution and installation.
- Once users connect to the Internet, the app is immediately available.

Web application

- have GUI, easy to start using
- Web 
  - A mechanism for sharing information.
  - one of the ways we use the Internet.
- Web app ver. is here: https://bring-image-resize-to-users.herokuapp.com/resize

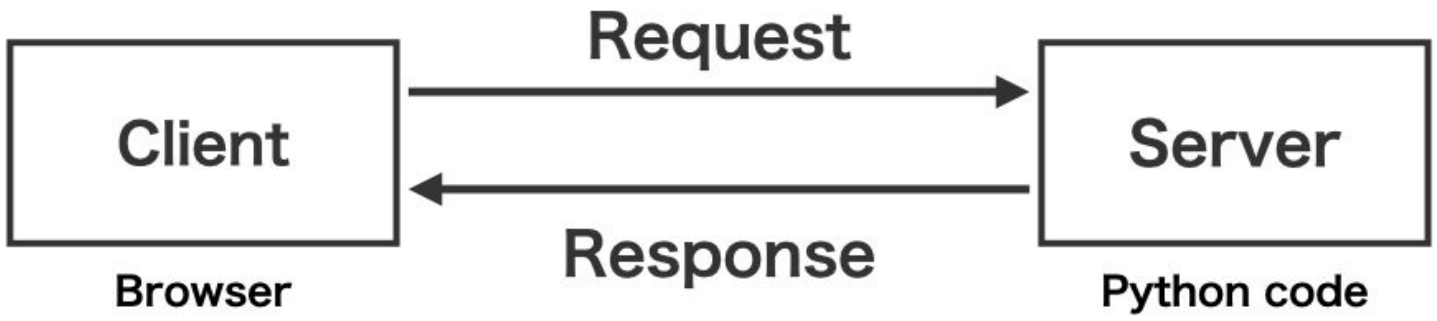
Web app: Multiple machines

- CLI & GUI: 1 machine (PC)
- Web app: more machines. communicate with each other.

2 roles: Server / Client

- Server:
  - where web app is running.
  - where we put the source code (deploy)
- Client:
  - use web apps
  - e.g. PCs and smartphones (often use via a web browser)

How server and client communicate?



Contents of request / response

- Request (client ➡️ server)
  - URL (e.g. https://ep2020.europython.eu/events/sessions/)

    which server *which process*
  - Information entered into your browser
- Response (server ➡️ client)
  - includes HTML
  - (we can recycle HTML files in the GUI part)

Web apps with Python

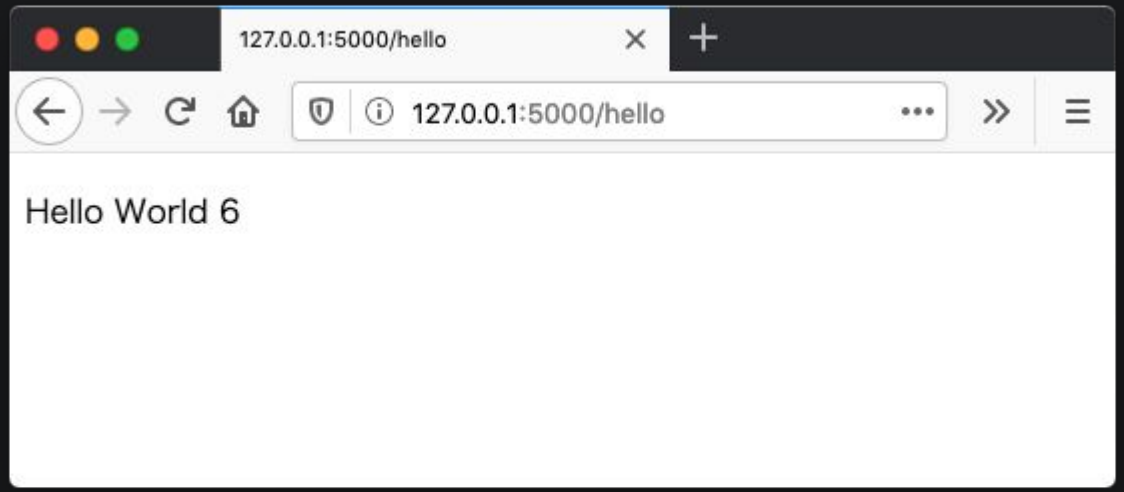
- introduce Flask (one of so many packages for Web apps)
- Flask could make it easier to implement simple Web apps.

Components of a Web app

- Python (Flask)
- HTML
- JavaScript (Not covered in this talk)
- CSS (Not covered in this talk)

First example of Flask: Hello World

- When open the URL in the browser, displays a greeting.

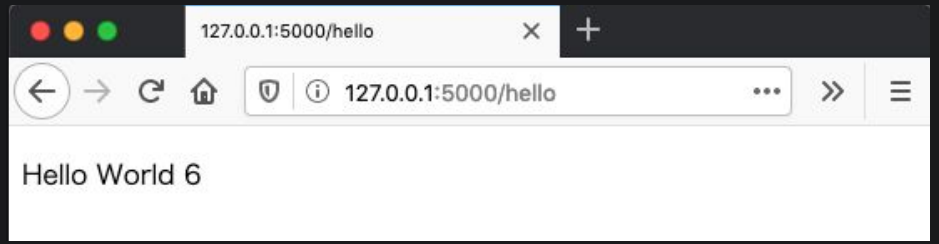


Directory structure for Hello World app

```
webapp
├── hello_world.py
└── templates
    └── hello.html


# Start the server (Enter Ctrl+C when you exit)
$ python hello_world.py
# Open http://127.0.0.1:5000/hello in your browser
# (Send a request to the server running in your PC)
```

How to handle request / response

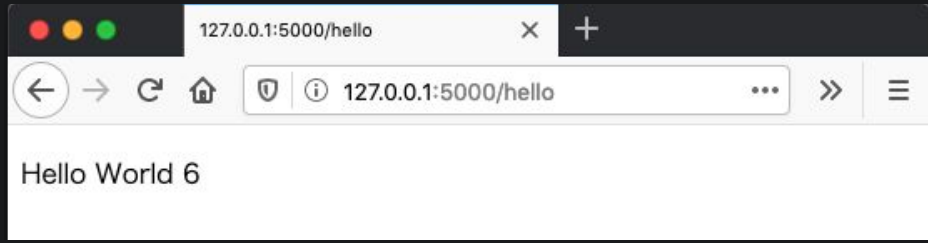
1. User opens the URL http://127.0.0.1:5000/hello in the browser (Client sends a request).
2. Server starts the process corresponding to /hello and returns a response (including HTML).
3. Client receives a response, browser renders the HTML, then user can see a greeting.



Impl. of hello_world.py

```
from flask import Flask, render_template
app = Flask(__name__)

@app.route("/hello")  # Called by requests to URLs (.../hello)
def hello():
    message = say_hello()  # same as say_hello in GUI part
    # Returns a response based on templates/hello.html
    return render_template("hello.html", message=message)

app.run(debug=True)  # Start a server for development
```

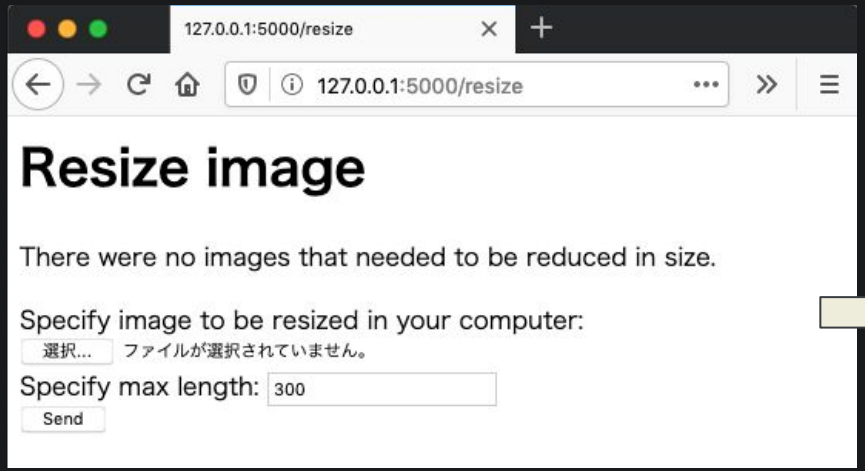
Impl. of hello.html

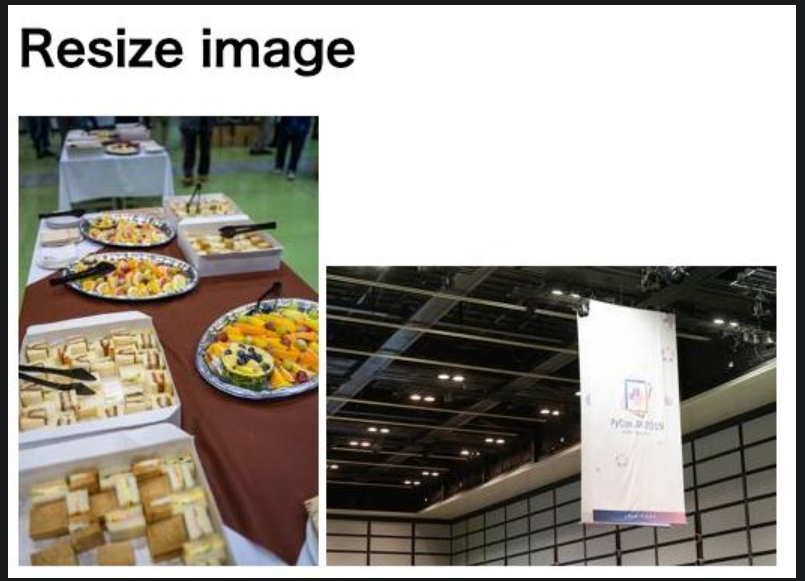



```
<!DOCTYPE html>
<html>
  <body>
    <!-- The {{ message }} is replaced by the value stored
         in the message variable -->
    <p>{{ message }}</p>
    <!-- passed as message=message in render_template function -->
  </body>
</html>
```

Goal of Web app: convert CLI to Web app using GUI asset

- Select images and enter max length.
- displays resized images.





How to handle images in web apps

- Copy of data of an image in a client is sent to a server.
- Web app loads the image from data, resizes it and saves on the server.
- Need to set up to publish images stored on the server.
- <img> tag works for public images on the server.

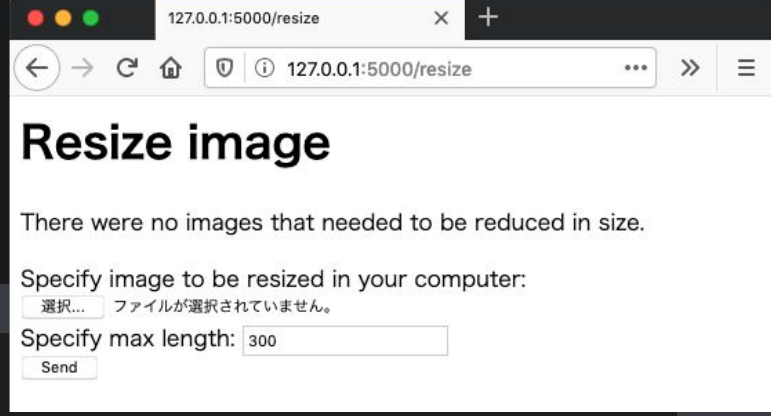
Directory structure for image resize app

```
webapp
├── shrink_image.py
├── templates
│   └── resize.html
└── images  # put the resized images (sent from clients)


# Start the server (Enter Ctrl+C when you exit)
$ python shrink_image.py
# Open http://127.0.0.1:5000/resize in your browser
```

Overview of shrink_image.py 1/2

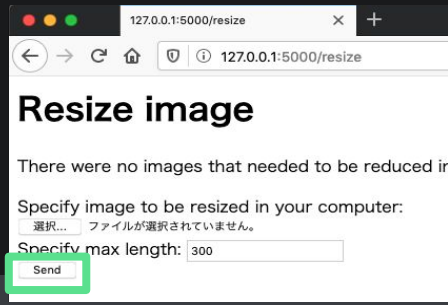
```
from flask import Flask, render_template, request
# Images placed in the "images" directory are published
app = Flask(__name__, static_folder="images")

@app.route("/resize", methods=["GET", "POST"])
def resize():
    # Open /resize in your browser (HTTP method: GET)
    if request.method == "GET":
        return render_template("resize.html")
    # explain later ...
app.run(debug=True)
```

Impl. of resize.html 1/2




```
<!-- body part: define input fields -->
<form method="post" enctype="multipart/form-data">
  <input id="image_file" type="file" name="image_file"
    accept="image/png, image/jpeg" required multiple>
  <input id="max_length" name="max_length" value="300" required>
  <button type="submit">Send</button>
</form>
```

Overview of shrink_image.py 2/2




```python
def resize():  # data is sent to /resize. (HTTP method: POST)
    # receives entered values sent by browser
    max_length = int(request.form["max_length"])
    image_files = request.files.getlist("image_file")
    for image_file in image_files:
        resize_image(image_file, save_path, max_length)
    # Pass a list of paths of resized images to render_template
    return render_template("resize.html", image_paths=image_paths)
```

Impl. of resize.html 2/2

Resize image

```
<!-- body part: create HTML which includes resized images -->
{% for image_path in image_paths %}
  <img src="{{ image_path }}">
{% else %}
  <p>There were no images that needed to be reduced in size.</p>
{% endfor %}
```

How to deploy (e.g. on heroku)

1. pip install gunicorn
2. create config files for heroku
    - Procfile, runtime.txt, requirements.txt, wsgi.py
3. push source code to heroku

ref:
https://www.geeksforgeeks.org/deploy-python-flask-app-on-heroku/

Recap: Web app

- Web app receives a request and returns a response

introduce Flask

- executes the Python function corresponding to the URL in the request (`@app.route`)
- creates a response using template tags (`render_template`)
- To display the images in HTML, need to set images to publish on the server (`static_folder`)

FYI: Web app

- Other packages

Wrap up: Bringing your Python script to more users!

Recap: Quick tour

- CLI: **Resolve the hard-coded** by specifying from the command line
- GUI: **more user-friendly** app
  - In addition to Python, add a little HTML and JavaScript
- Web app: **no** more **installation**
  - Python processes data sent via communication with a server and a client

Bring your script and automate someone’s boring stuff with your Python script

- Thank you very much for your attention.