

# 10장. 서버(Server) 및 네트워크



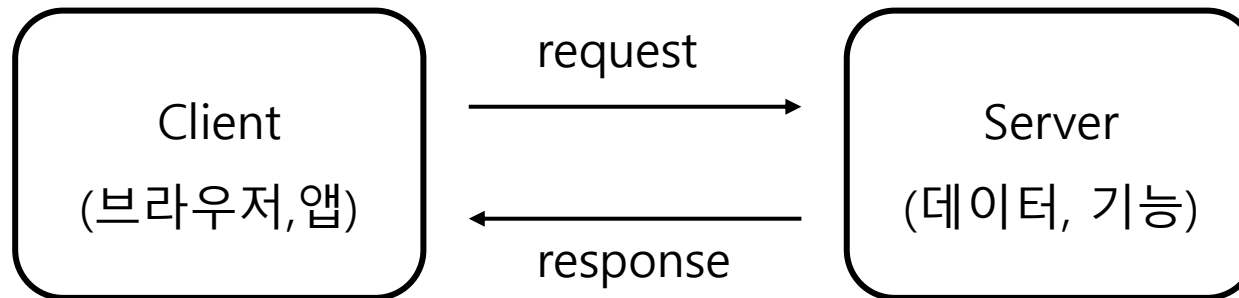
# 클라이언트 & 서버

## ■ 챗GPT 프롬프트 입력

+ 컴퓨터 용어인 서버와 클라이언트에 대해 설명해줘



## ■ 서버와 클라이언트 기본 구조



# 클라이언트 & 서버

## ■ 클라이언트란?

서버에 요청(Request)을 보내는 쪽이다. 서버에 "이거 주세요"라고 요청 서버가 준 결과를 화면에 보여주거나 사용함.

예시)

- 웹 브라우저(Chrome, Edge)
- 스마트폰 앱
- 게임 실행 프로그램

## ■ 서버란?

서비스를 제공하는 컴퓨터(또는 프로그램) 이다.

- 데이터, 파일, 웹페이지, 프로그램 기능 등을 제공
- 항상 켜져 있거나 요청을 기다리는 역할

예시)

- 웹 서버: 네이버, 구글 같은 웹사이트를 보여줌
- 게임 서버: 게임 접속, 점수, 캐릭터 정보 관리
- DB 서버: 데이터 저장·조회 담당

# flask(플라스크)

- 플라스크(flask)란?

파이썬으로 제작된 마이크로 웹 프레임워크의 하나이며, 웹 sever를 만들고, 웹 애플리케이션을 제작할 수 있다.

👉 <https://flask.palletsprojects.com/en/stable/>



# Flask

Flask 문서에 오신 것을 환영합니다. Flask는 경량 WSGI 웹 애플리케이션 프레임워크입니다. Flask는 누구나 쉽고 빠르게 시작할 수 있도록 설계되었으며, 복잡한 애플리케이션까지 확장할 수 있는 기능을 갖추고 있습니다.

# flask(플라스크)

- 플라스크 설치

**pip install Flask**

- 간단한 웹 서버 예제

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello_world():
    return "<p>Hello, World!</p>"
```

# flask 웹서버 만들기

## ■ 웹서버 만들기

```
from flask import Flask

# Flask 애플리케이션 인스턴스 생성
app = Flask(__name__)

# 루트 경로에 대한 라우트 설정
@app.route('/')
def home():
    return "<h1>Hello, Flask!"

if __name__ == "__main__":
    # 디버그 모드로 서버 실행
    # 운영 모드에서는 debug=False로 설정
    app.run(debug=True)
```

← → ↻ ⓘ 127.0.0.1:5000

**Hello, Flask!**

# flask 웹서버 만들기

## ▪ Flask 서버 실행

### 1. 서버 실행 방법 - 터미널

서버&네트워크> python server01.py

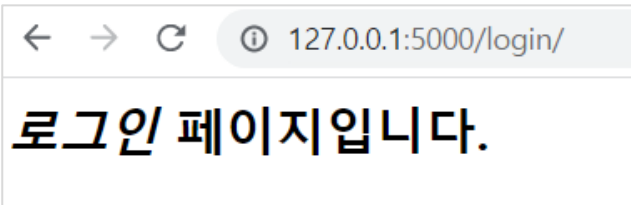
```
* Serving Flask app 'server01'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment.
on WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 106-824-902
```

### 2. 웹 브라우저 접속 - http://127.0.0.1:5000

- 127.0.0.1 -> 내 컴퓨터에서 실행 중인 웹 서버에 접속하는 주소
- 5000 -> 포트 번호(어떤 프로그램(서버)인지 구분하는 번호)

# flask 웹서버 만들기

## ■ 페이지 라우팅



```
from flask import Flask

app = Flask(__name__) #app 객체 생성

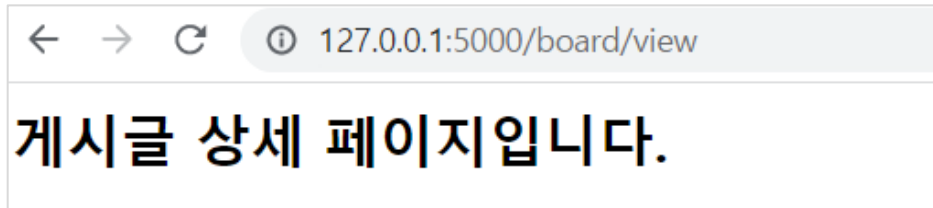
@app.route('/') #http://127.0.0.1:5000/
def index():
    return "Hello~ Flask"

@app.route('/login') #http://127.0.0.1:5000/login
def login():
    return "<h2><i>로그인</i> 페이지입니다.</h2>"
```



# flask 웹서버 만들기

## ■ 페이지 라우팅



```
@app.route('/register')
def register():
    return "<h2>회원가입 페이지입니다.</h2>"

@app.route('/board/view')
def view():
    return "<h2>게시글 상세 페이지입니다.</h2>"
```

# flask 웹서버 만들기

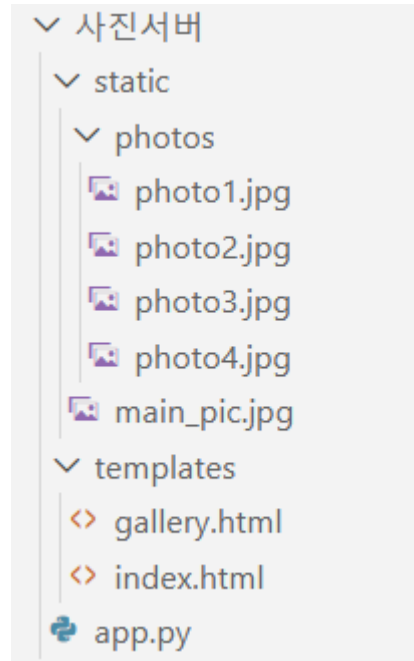
## ■ 웹 프레임워크 및 사진 서버 디렉터리 구조

### 웹 프레임워크 구조

**templates** 폴더 -> html 파일

**static** 폴더 -> css, image 파일

**app.py** -> 실행 파일



# flask 웹서버 만들기



# flask 웹서버 만들기

- 플라스크 사진 서버 만들기

app.py

```
from flask import Flask, render_template
import os

app = Flask(__name__)

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

@app.route("/gallery")
def gallery():
    photos = os.listdir("static/photos")
    return render_template("gallery.html", photos=photos)

if __name__ == "__main__":
    app.run(debug=True)
```

# flask 웹서버 만들기

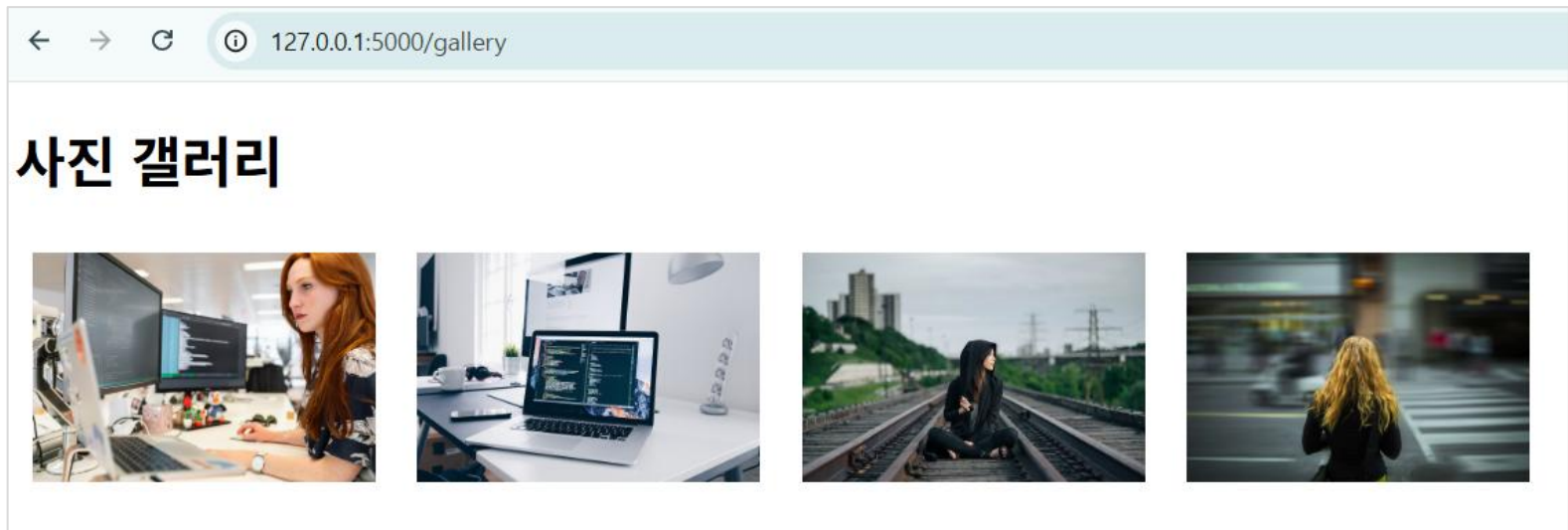
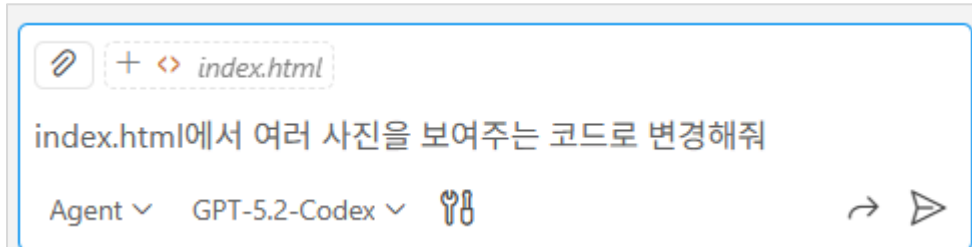
## ▪ index.html 파일 만들기

index.html

```
<!DOCTYPE html>
<html lang="ko">
<head>
  <meta charset="UTF-8">
  <title>사진 보기</title>
</head>
<body>
  <h1>어느 봄날 카페에서...</h1>
  
</body>
</html>
```

# flask 웹서버 만들기

- gallery.html 파일 만들기



# flask 웹서버 만들기

## ▪ gallery.html 파일 만들기

```
<!DOCTYPE html>
<html lang="ko">
<head>
  <meta charset="UTF-8">
  <title>사진 갤러리</title>
  <!-- CSS 스타일 추가 -->
  <style>
    img {
      width: 200px;
      margin: 10px;
    }
  </style>
</head>
<body>
  <h1>사진 갤러리</h1>
  <!-- 파이썬 템플릿 언어 - 제어문{% %}, 출력문 {{ }} -->
  {% for photo in photos %}
    
  {% endfor %}
</body>
</html>
```

# 네트워크(Network)

## ■ 컴퓨터의 내부 및 외부 IP

+ 파이썬으로 컴퓨터의 내부 IP를 확인하는 코드를 작성해줘



```
import socket

# 내 컴퓨터 이름과 내부 IP 주소 가져오기
hostname = socket.gethostname()
ip_address = socket.gethostbyname(hostname)

print("컴퓨터 이름:", hostname)
print("내부 IP 주소:", ip_address)

# 외부 IP 주소 가져오기
# 원격 서버에 요청을 보내는 방법 사용
import requests

url = 'https://api.ipify.org'
external_ip = requests.get(url).text
print("외부 IP 주소:", external_ip)
```



# 네트워크(Network)

## ▪ TCP(Transmission Control Protocol) 통신

데이터를 “확실하게, 순서대로” 보내는 방식이다.  
안전하고 신뢰성 높은 통신 규약이다.

## ▪ TCP 핵심 특징

- 연결(Connection)을 만든다.  
(통신 전에 서로 연결 확인)
- 데이터가 순서대로 도착한다.
- 데이터가 안오면 다시 보낸다.
- 오류를 스스로 처리한다.

# 네트워크(Network)

## ▪ TCP 동작 방식

- `socket()` → 통신 통로 생성
- `bind()` → 서버 주소 지정
- `listen()` → 연결 대기
- `accept()` → 클라이언트 접속 허용
- `connect()` → 서버 접속
- `sendall()` / `recv()` → 데이터 송수신

# 네트워크(Network)

## ■ TCP 통신 예제

### ➤ 서버 실행 결과

```
문제   출력   디버그 콘솔   터미널   포트

• PS D:\python_2026\서버 & 네트워크> cd 네트워크
• PS D:\python_2026\서버 & 네트워크\네트워크> python server.py
서버가 시작되었습니다. 클라이언트를 기다리는 중...
('127.0.0.1', 59164) 에서 연결되었습니다.
클라이언트로부터 받은 메시지: 안녕하세요, 서버!
```

### ➤ 클라이언트 실행 결과

```
+ v powershell - 네트워크

• PS D:\python_2026\서버 & 네트워크\네트워크> python client.py
서버 응답: 메시지 잘 받았습니다!
```

# 네트워크(Network)

## ▪ server.py

```
import socket

# 서버 주소와 포트
HOST = "127.0.0.1" # 내 컴퓨터
PORT = 5000

# 소켓 생성 (IPv4, TCP)
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# 주소 바인딩
server_socket.bind((HOST, PORT))

# 연결 대기
server_socket.listen()
print("서버가 시작되었습니다. 클라이언트를 기다리는 중...")
```

# 네트워크(Network)

## ▪ server.py

```
# 클라이언트 연결 수락
client_socket, addr = server_socket.accept()
print(f"{addr} 에서 연결되었습니다.")

# 데이터 수신
data = client_socket.recv(1024)
print("클라이언트로부터 받은 메시지:", data.decode())

# 응답 전송
client_socket.sendall("메시지 잘 받았습니다!".encode())

# 연결 종료
client_socket.close()
server_socket.close()
```

# 네트워크(Network)

## ▪ client.py

```
import socket

HOST = "127.0.0.1" # 서버 주소
PORT = 5000

# 소켓 생성
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# 서버 연결
client_socket.connect((HOST, PORT))

# 메시지 전송
client_socket.sendall("안녕하세요, 서버!".encode())

# 응답 받기
data = client_socket.recv(1024)
print("서버 응답:", data.decode())

# 연결 종료
client_socket.close()
```

# 네트워크(Network)

- **실행 방법**

터미널 2개 열기(터미널 분할)

첫번째의 터미널에서

> python server01.py

두번째의 터미널에서

> python client01.py