웹프로그래밍의 기초

Week8

Decorator;

Basic ideas of web environments

Decorator

Before we go any further

- In Python, everything is object and callable by other objects.
- Names that we define are simply identifiers bound to these objects, so do functions.

```
def first(msg):
    print(msg)

first("Hello")

second = first
second("Hello")
```

```
-----------------------------------
Hello
Hello
```

Step 1

- Let’s say we want to leave logs for a task that could be divided into 3 phases.

```
def phase1():
    print('Phase1 Started!')
    print('Phase1 Now')
    print('Phase1 Ended!\n')

def phase2():
    print('Phase2 Started!')
    print('Phase2 Now')
    print('Phase2 Ended!\n')

def phase3():
    print('Phase3 Started!')
    print('Phase3 Now')
    print('Phase3 Ended!\n')
```

```
phase1()
phase2()
phase3()
```

```
----------------------------------------
Phase1 Started!
Phase1 Now
Phase1 Ended!

Phase2 Started!
Phase2 Now
Phase2 Ended!

Phase3 Started!
Phase3 Now
Phase3 Ended!
```

Step 2

- A function decorator in Python is just a function that takes in another function as an argument, extending the decorated function's functionality without changing its structure.
- To implement decorators, define an outer function that takes a function argument.
- A decorator wraps another function, amplifies its behavior, and returns it.
- Nest a wrapper function within the outer decorator function, which also wraps the decorated function.

```
def trace(func):
    def wrapper():
        print(func.__name__.title(), 'Started!')
        func()
        print(func.__name__.title(), 'Ended!\n')
    return wrapper

def phase1():
    print('Phase1 Now')

def phase2():
    print('Phase2 Now')

def phase3():
    print('Phase3 Now')
```

```
trace_phase1 = trace(phase1)
trace_phase1()
trace_phase2 = trace(phase2)
trace_phase2()
trace_phase3 = trace(phase3)
trace_phase3()

----------------------------------------
Phase1 Started!
Phase1 Now
Phase1 Ended!

Phase2 Started!
Phase2 Now
Phase2 Ended!

Phase3 Started!
Phase3 Now
Phase3 Ended!
```

Step 3

- A simple decorator function is easily identified when it begins with the @ prefix, coupled with the decorated function underneath.

```
def trace(func):
    def wrapper():
        print(func.__name__.title(), 'Started!')
        func()
        print(func.__name__.title(), 'Ended!\n')
    return wrapper

@trace
def phase1():
    print('Phase1 Now')

@trace
def phase2():
    print('Phase2 Now')

@trace
def phase3():
    print('Phase3 Now')
```

```
phase1()
phase2()
phase3()

----------------------------------------
Phase1 Started!
Phase1 Now
Phase1 Ended!

Phase2 Started!
Phase2 Now
Phase2 Ended!

Phase3 Started!
Phase3 Now
Phase3 Ended!
```

Chaining decorators #1

```
def trace1(func):
    def wrapper():
        print('Initialized...')
        func()
        print('Closed...\n')
    return wrapper

def trace2(func):
    def wrapper():
        print(func.__name__.title(), 'Started!')
        func()
        print(func.__name__.title(), 'Ended!')
    return wrapper

@trace1
@trace2
def phase1():
    print('Phase1 Now')

@trace1
@trace2
def phase2():
    print('Phase2 Now')

@trace1
@trace2
def phase3():
    print('Phase3 Now')
```

```
phase1()
phase2()
phase3()

----------------------------------------
Initialized...
Phase1 Started!
Phase1 Now
Phase1 Ended!
Closed...

Initialized...
Phase2 Started!
Phase2 Now
Phase2 Ended!
Closed...

Initialized...
Phase3 Started!
Phase3 Now
Phase3 Ended!
Closed...
```

Chaining decorators #2

```
def increase_decorator(func):
    def increase_by_two():
        print('Multiplying number by 2')
        new_number = func()
        return new_number * 2

    return increase_by_two

def decrease_decorator(func):
    def decrease_by_one():
        print('Decreasing number by 1')
        new_number = func()
        return new_number - 1

    return decrease_by_one

@increase_decorator
@decrease_decorator
def get_number():
    print('Initial number is 3')
    return 3

print(get_number())
```

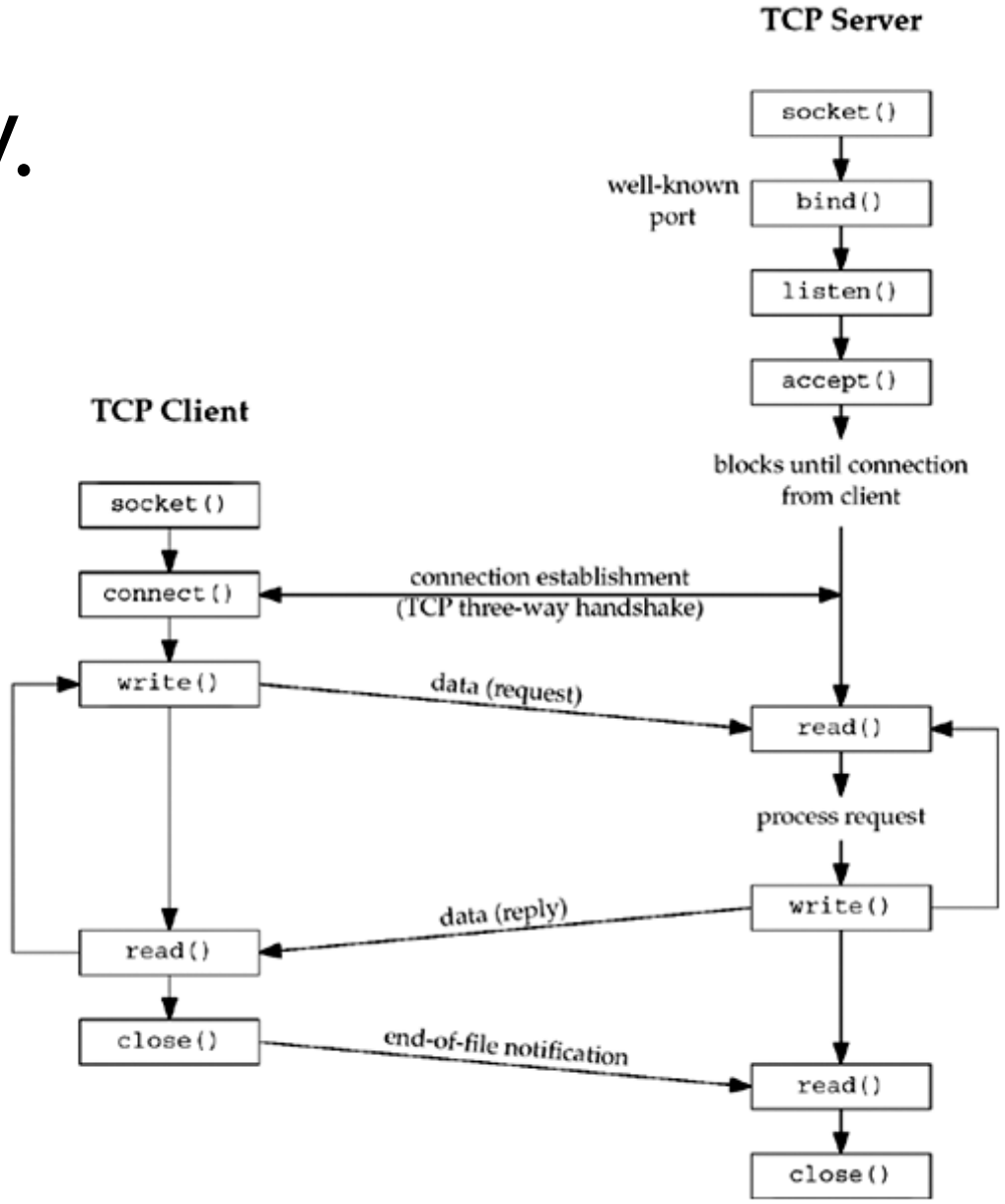
```
print(get_number())

------------------------------------------------
Multiplying number by 2
Decreasing number by 1
Initial number is 3
4
```

Front/back ends, Server/client sides

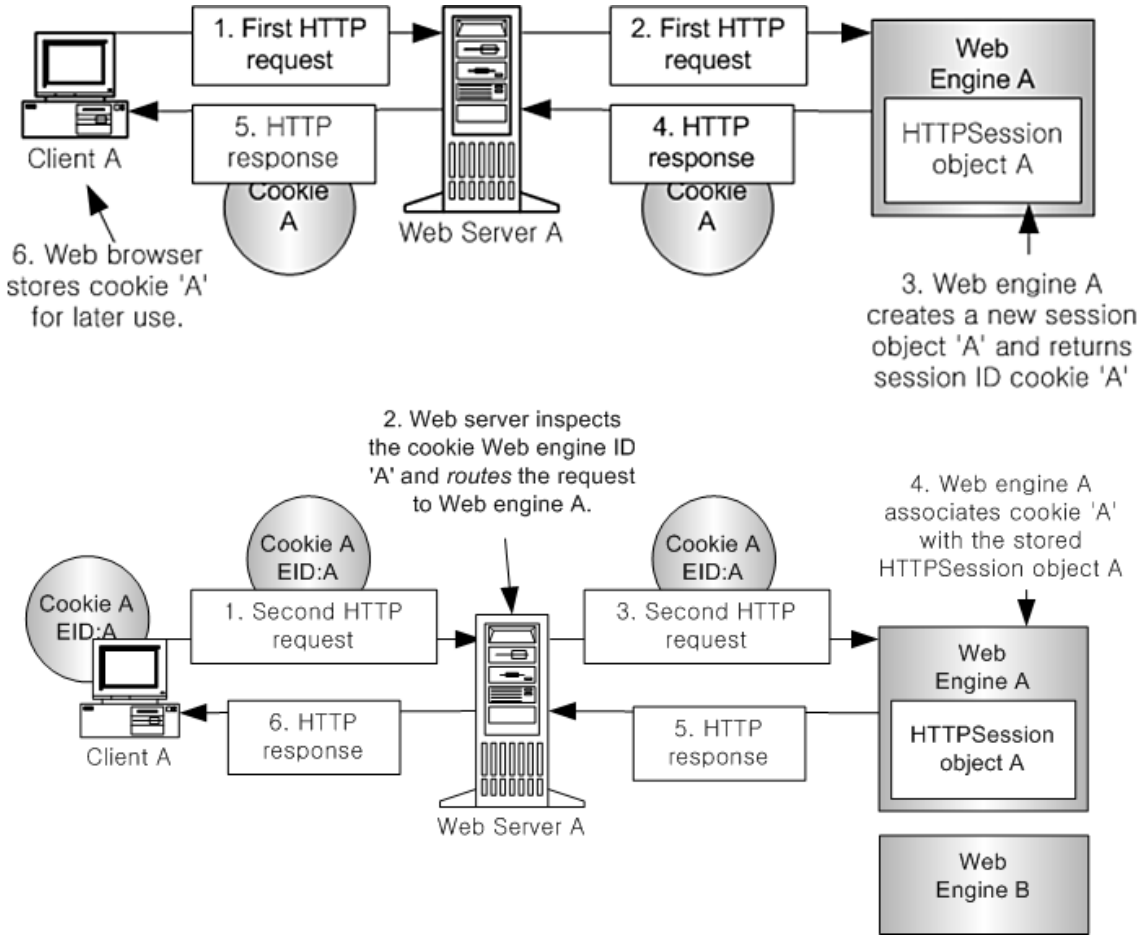
From Server-Client env.

- The terms client and server are common within the TCP/IP community, and many definitions exist. In the TCP/IP context, these terms are defined as follows:
  - Server
    - A process that waits passively for requests from clients, processes the work specified, and returns the result to the client that originated the request.
  - Client
    - A process that initiates a service request.



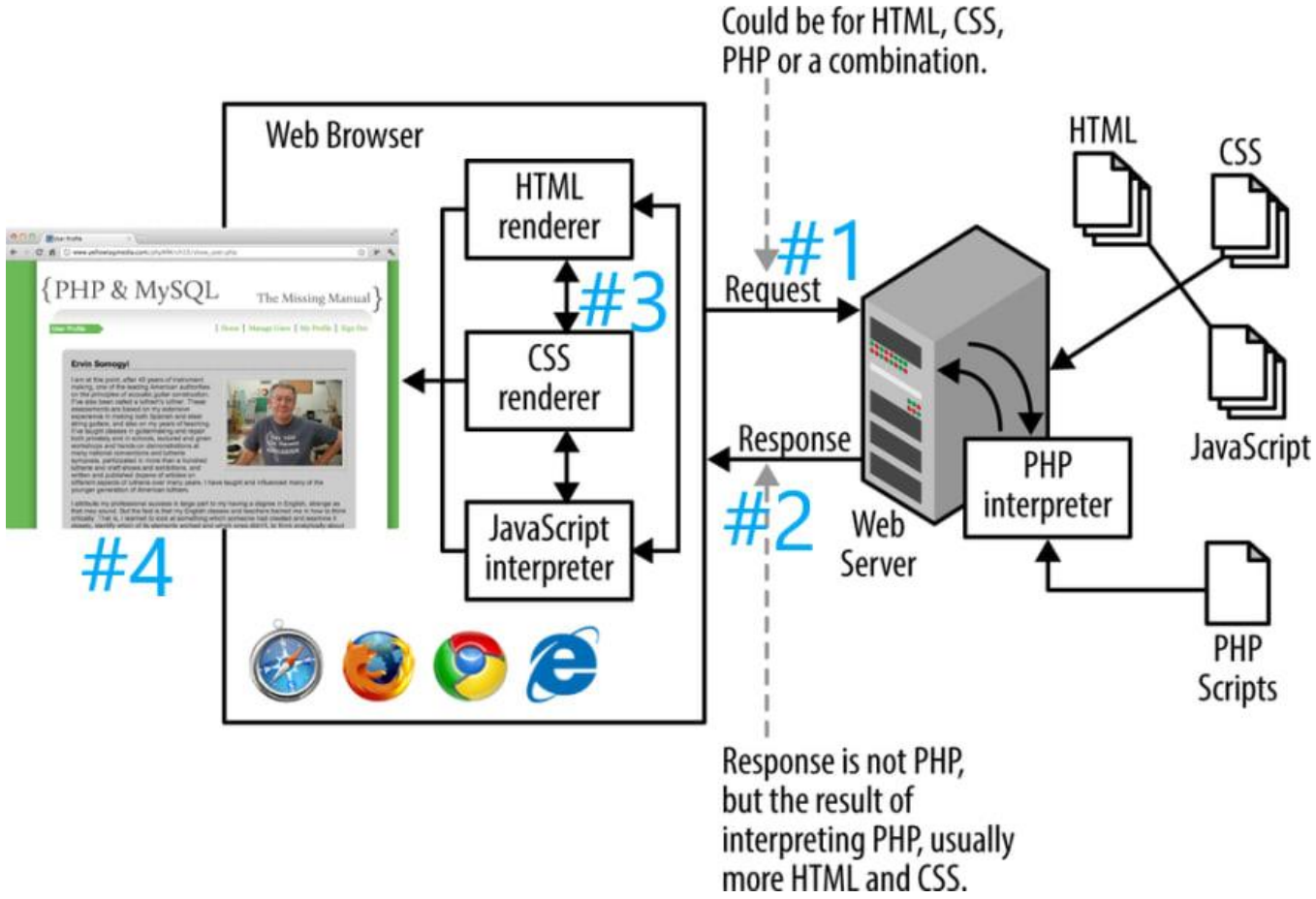
To HTTP env.

- **Creating a Session Cookie**
  - A client makes an initial request to the web server.
  - The web server sends the request to the web engine.
  - The web engine creates an HTTP session object for the client and a session cookie that contains the session ID of the HTTP session. The session ID is used to retrieve the HTTP session object for any subsequent requests from the client.
  - The response data and session cookie are sent to the web server.
  - The response data and session cookie are sent to the client's browser, and then the HTTP connection is disconnected.
  - The session cookie that contains the session ID is saved in the client's browser.
- **Sending a Second Request using the Session ID**
  - The client sends another request to the same web server by attaching the session cookie previously received from the web browser.
  - The web server receives the request with the session cookie and just like in the first request, sends the request to the same web server.
  - The web engine receives the request and session cookie. Then, it finds the HTTP session object corresponding to the session ID of the session cookie from its own memory. Then, the web engine uses the HTTP session data to process the request.
  - The response data and session cookie are sent to the web server.
  - The HTTP response is sent to the web browser and the HTTP connection is disconnected. The session cookie has to be included in the response only when the initial connection is created.

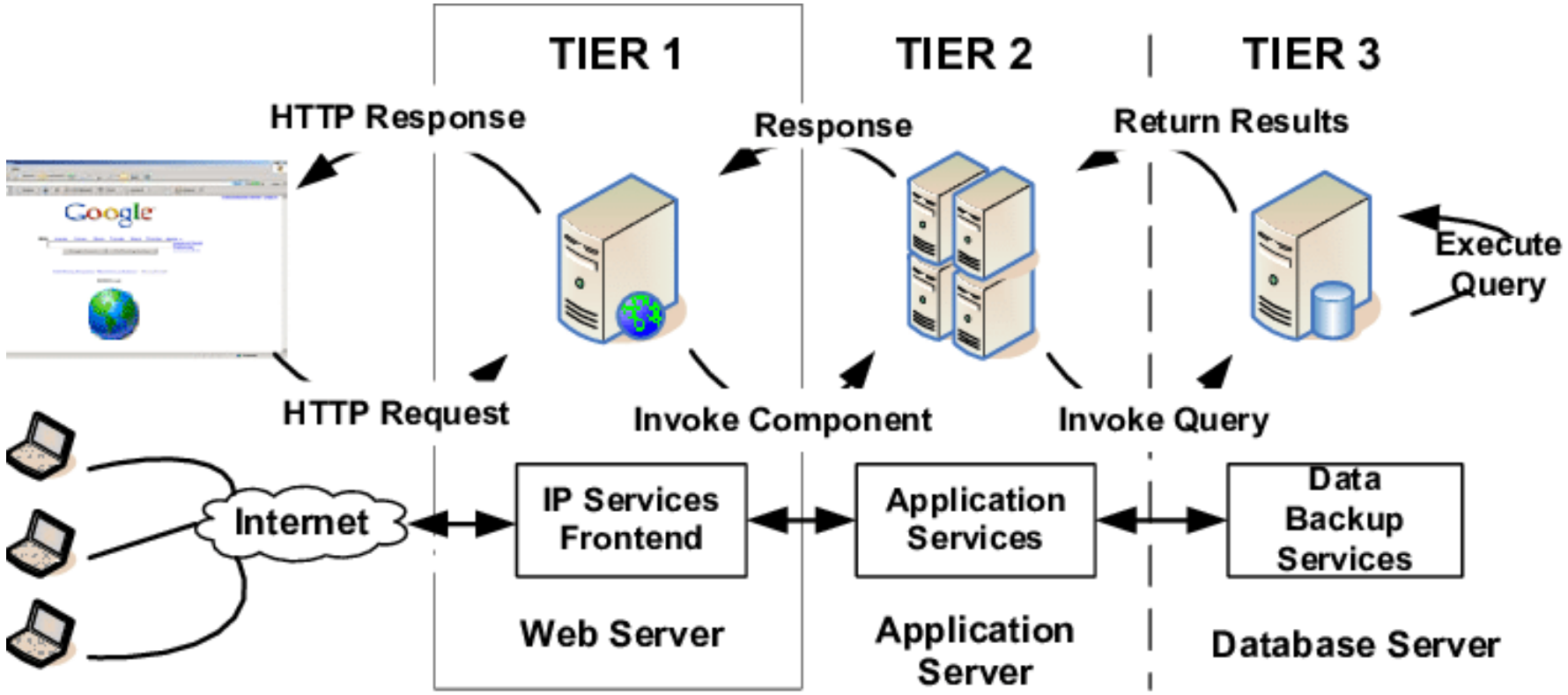


How Web Server reacts against clinets

- When you decided to visit a web site and made the request in your browser to do so, there are 4 main steps that happen:
  1. Request
  2. Response
  3. Build
  4. Render



3-tier architecture

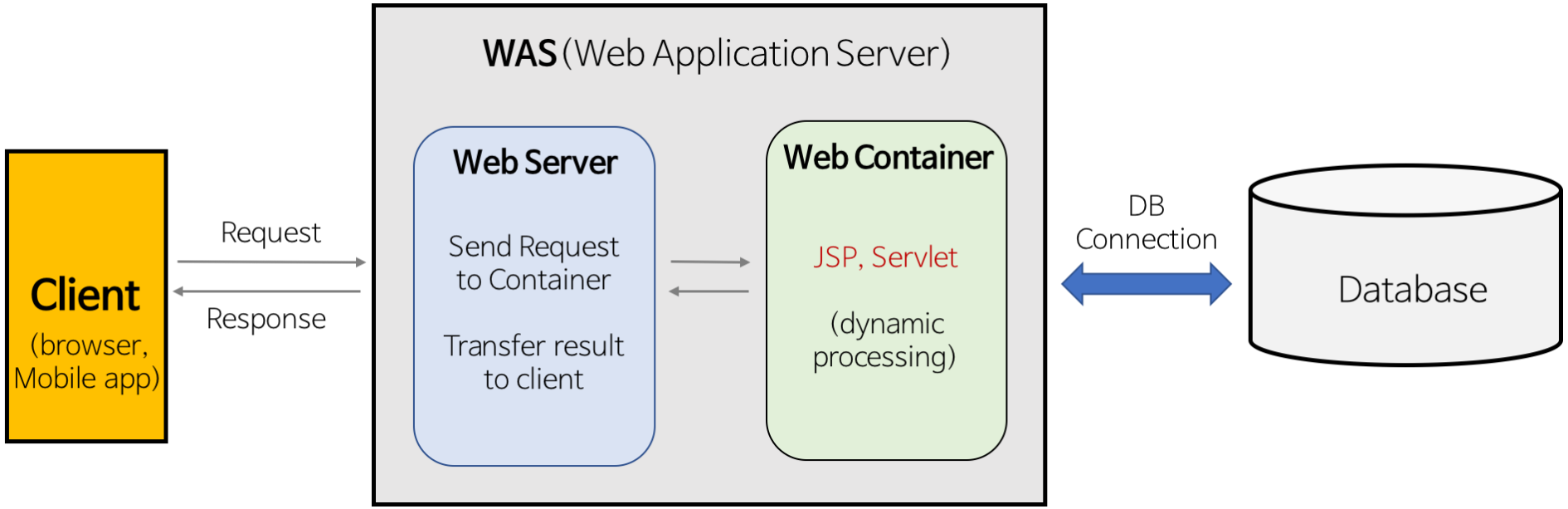


Web server vs. web application server

- By strict definition, a web server is a common subset of an application server.
- A web server **delivers static web content**—e.g., HTML pages, files, images, video—primarily in response to hypertext transfer protocol (HTTP) requests from a web browser.
- A web application server typically can deliver web content too, but its primary job is to enable interaction between end-user clients and server-side application code—the code representing what is often called business logic—**to generate and deliver dynamic content**, such as transaction results, decision support, or real-time analytics.
  - The client for an application server can be the application's own end-user UI, a web browser, or a mobile app, and the client-server interaction can occur via any number of communication protocols.

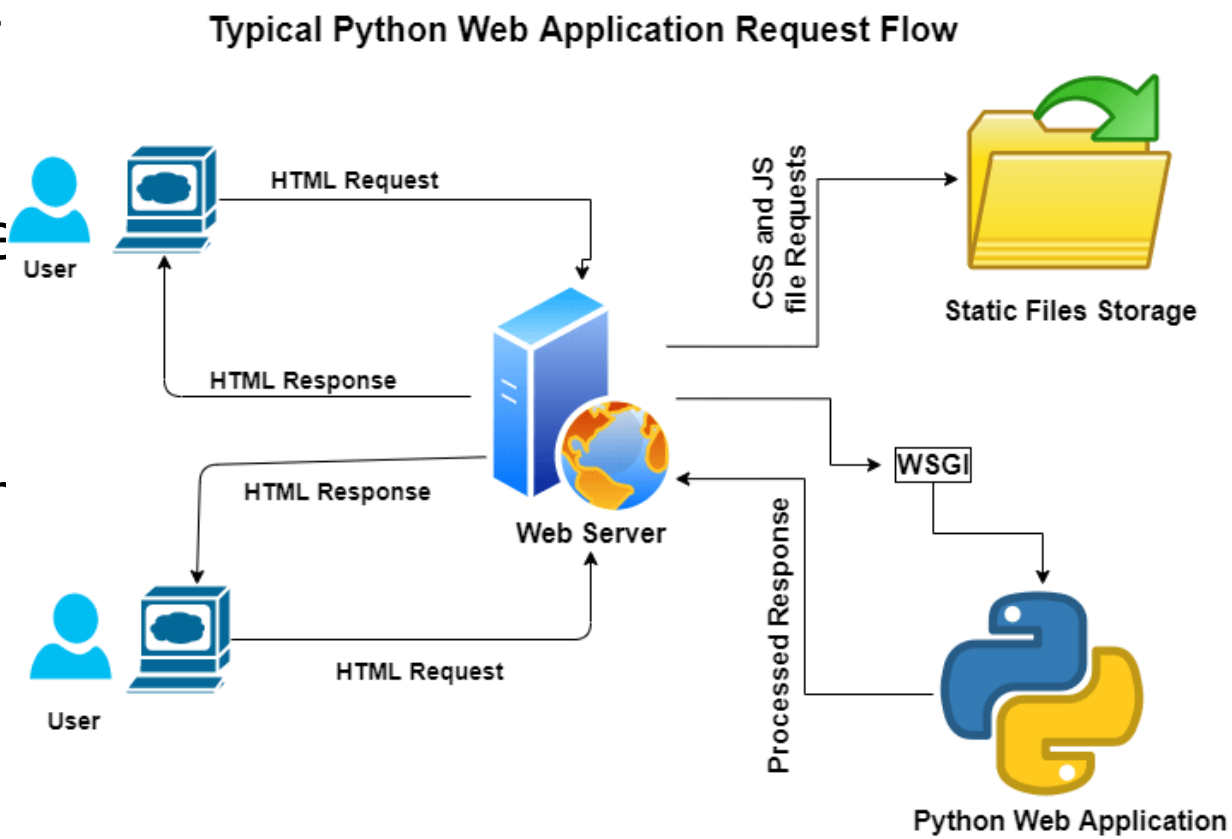
Web server vs. Web application server

- Real life web service architectures
  - Client -> Web Server -> DB
  - Client -> WAS -> DB
  - Client -> Web Server -> WAS -> DB



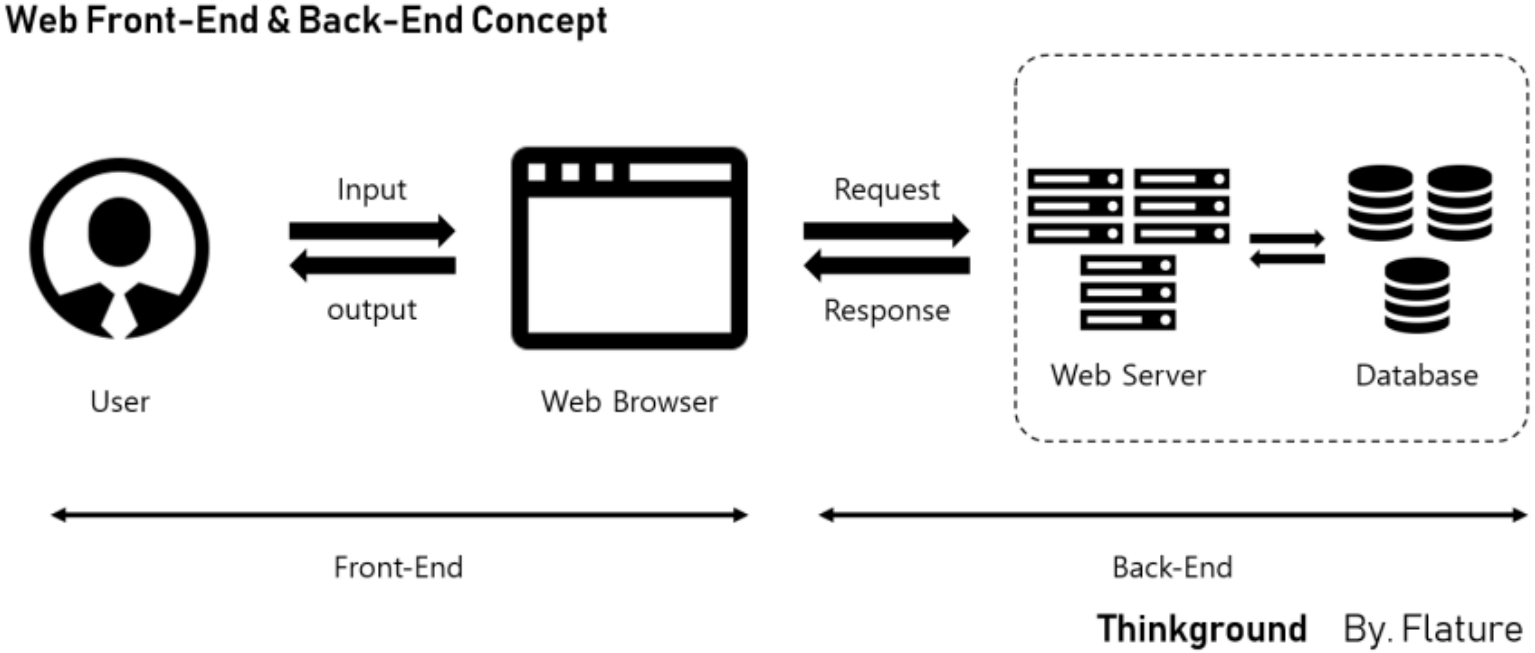
Flask built-in web server

- A typical web server does not have the ability to run Python web applications. WSGI (Web Server Gateway Interface) is a standard specification that the web server has to implement to run a Python web application.
- Flask derives WSGI application functionality from Werkzeug which is a comprehensive WSGI web application library.
- Flask framework does have a built-in server. But it is suitable only for development purposes.



Web Front-End & Back-End

- We will often divide the components of a webserver into two distinctive parts - Front-end and Back-end.
  - The front-end consists of the HTML, CSS, and any client-side programs((i.e., JavaScript).
  - the Back-end consists of server-side programs and the database.



And the people at each side

Front-End Developers

- People who create websites and web applications for a living, are called Front-End Developers.
- Front-end development refers to the client-side (how a web page looks).
  - Many Front-End Developers also have basic knowledge of different CSS and JavaScript frameworks and libraries, like Bootstrap, SASS (CSS pre-processor), jQuery and React, and the popular version control system, Git.

Back-End Developers

- Back-end development refers to the server-side (how a web page works).
- Front-end code is used to create static websites, where the purpose is to display the web page. However, if you want to make your website dynamic (manage files and databases, add contact forms, control user-access, etc.), you need to learn a back-end programming language, like PHP or Python, and use SQL to communicate with databases.