Python: Day 04

Advanced Programming

Agenda

01

Packaging

Handling Python Files

02

Multiple Tasks

Handling bottlenecks

03

Best Practices

Writing Pythonic code

04

Testing

Code correctness

05

Web Dev

Introduction to Flask

06

Lab Session

Culminating Exercise

01

Packaging

How to handle Python files properly

Modules and Packages

Module

Single Python file

```
.
└── module.py
```

Package

Folder with an `__init__.py`

```
.
└── package/
    ├── __init__.py
    └── module.py
```

Basic Import

./hello.py

```
1 def say_hello():
2     print("Hello from module hello")
```

./current_file.py

```
1 import hello
2
3 hello.say_hello()
```

Specific Import

./hello.py

```
1 def say_hello():
2     print("Hello from module hello")
```

./current_file.py

```
1 from hello import say_hello
2
3 say_hello()
```

Basic Import with Alias

./hello.py

```
def say_hello():
    print("Hello from module hello")
```

./current_file.py

```
import hello as ho

ho.say_hello()
```

Multiple Specific Import

./hello.py

```
1 def say_hello():
2     print("Hello from module hello")
3 greeting = “Yellow!”
```

./current_file.py

```
1 from hello import say_hello, greeting
2
3 say_hello()
4 print(greeting)
```

Basic Nested Import

./package/module_01.py

```
1  def say_hello():
2      print("Hello from module 1!")
```

./current_file.py

```
1  import package.module_01
2
3  package.module_01.say_hello()
```

Specific Nested Import

./package/module_01.py

```
1 def say_hello():
2     print("Hello from module 1!")
```

./current_file.py

```
1 from package.module_01 import say_hello
2
3 say_hello()
```

Nested Import with Alias

./package/module_01.py

```
1 def say_hello():
2     print("Hello from module 1!")
```

./current_file.py

```
1 import package.module_01 as pm1
2
3 pm1.say_hello()
```

Standard Packaging Format

Most Python projects follow this project structure:

```
project_name/
   ├── LICENSE
   ├── pyproject.toml
   ├── README.md
   ├── src/
   │    ├── example_package_1/
   │    │    ├── __init__.py
   │    │    └── example.py
   │    └── example_package_2/
   │         ├── __init__.py
   │         └── example.py
   ├── tests/
   ├── doc/
   └── script/
```

Try these Built-in Libraries!

Math

Common math constants and operations

Datetime

Dedicated package for handling calendar dates

Collections

Additional data structures

Time

Access to system time, delays, and conversions

SQlite

Quick setup for a light database system

Itertools

Efficient looping and combinatorials

H1

Random Counter

Using pre-built packages to do our work

Random Counter

Create one million random numbers from one to one thousand.

```
random_numbers = [...]
```

List down the number of occurrence for each number

```
random_number_count = ...
```

Finally, print out the number with the highest count and how many times it appeared

02

Multiple Tasks

A preview of Multiprocessing and Multithreading

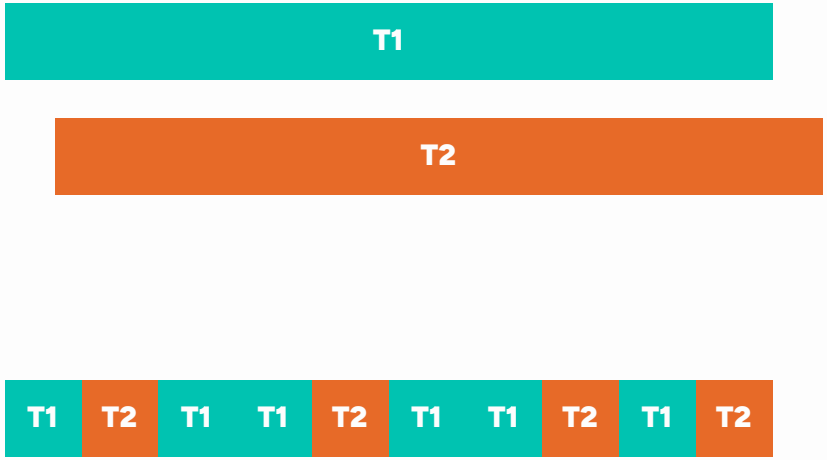
Parallelism versus Concurrency

Parallel Process

Tasks running simultaneously
or at the same time

Concurrent Process

Switching between tasks
when waiting for results



Concurrency

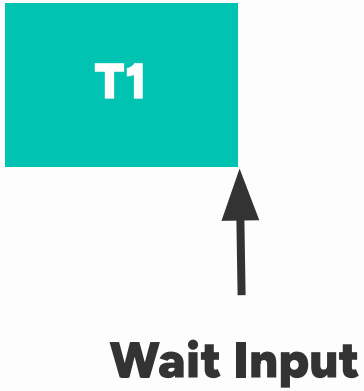
Working while waiting for other tasks

Concurrent Process

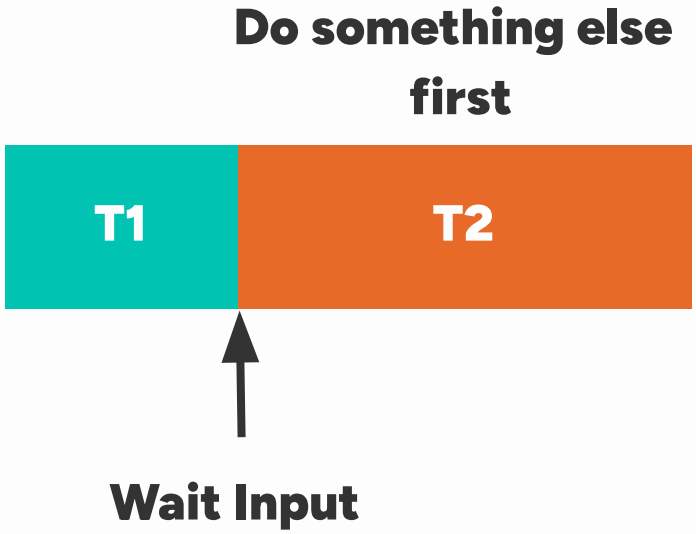
Current Task

T1

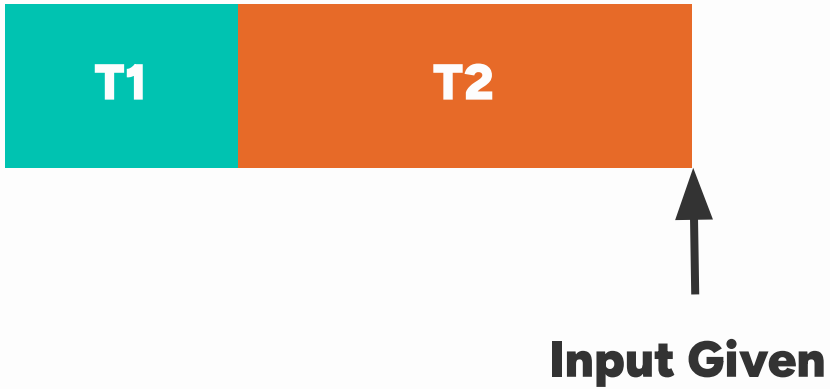
Concurrent Process



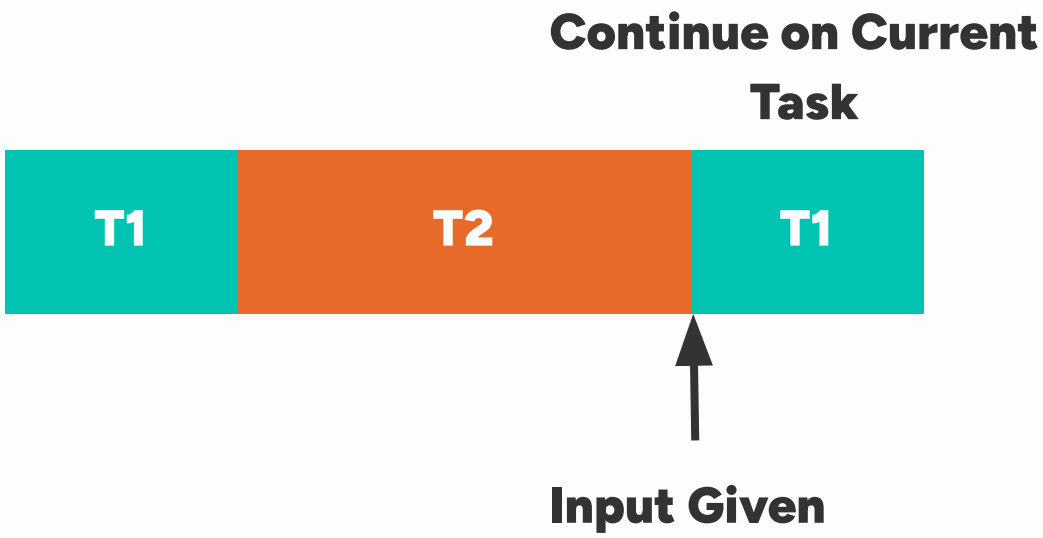
Concurrent Process



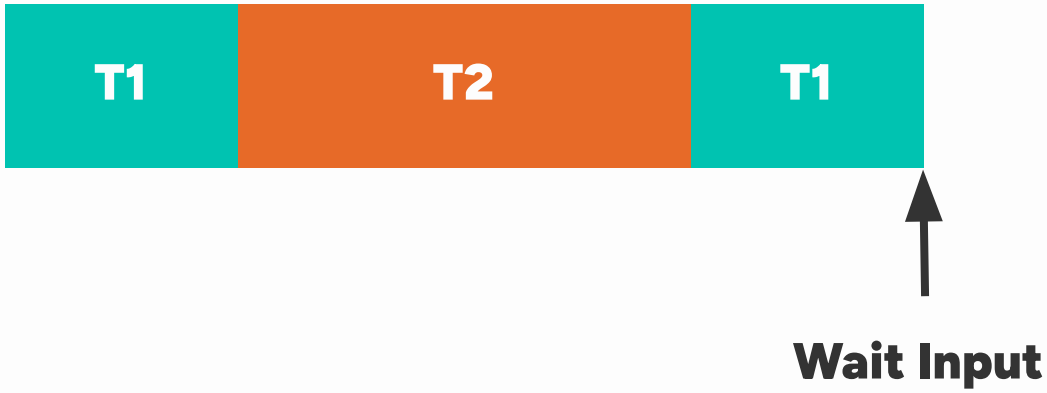
Concurrent Process



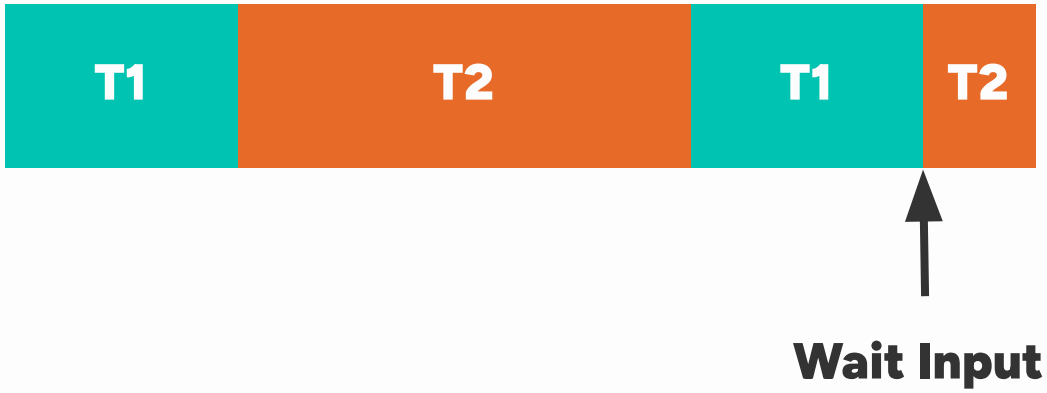
Concurrent Process



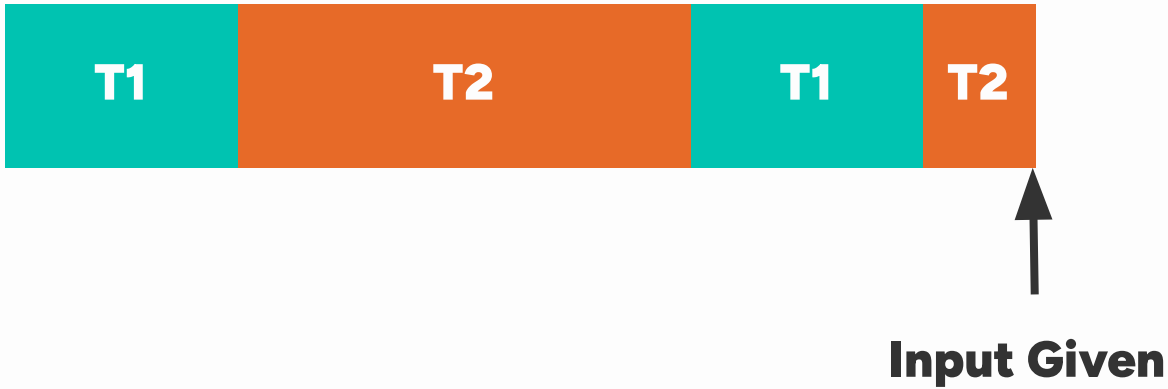
Concurrent Process



Concurrent Process



Concurrent Process



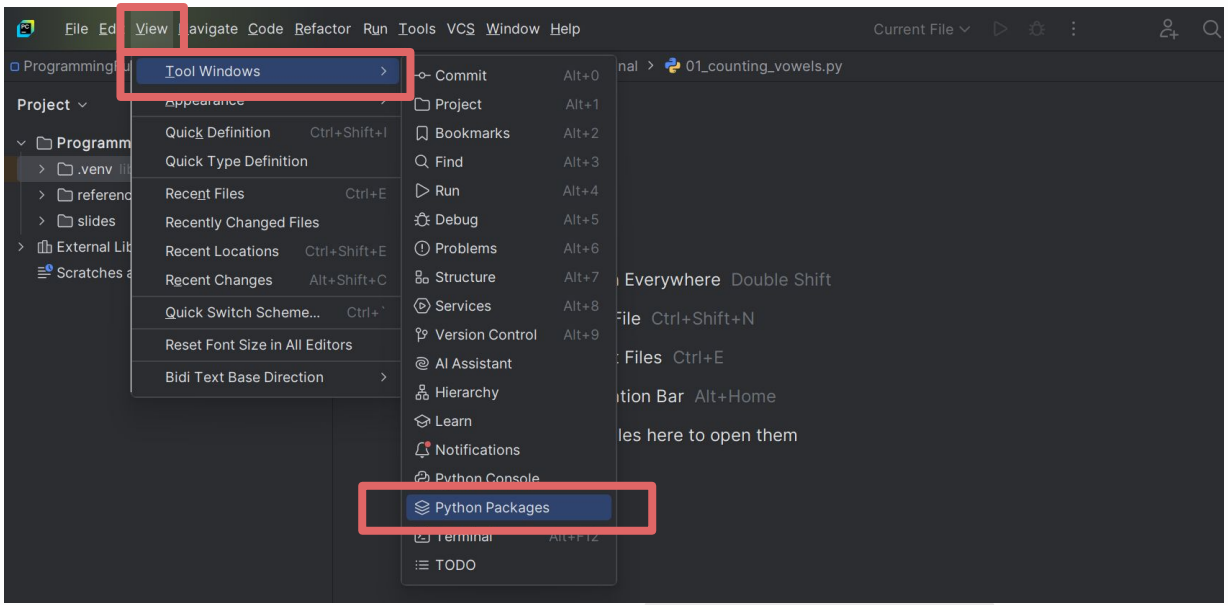
Concurrent Process

Concurrent Process

Concurrent Process

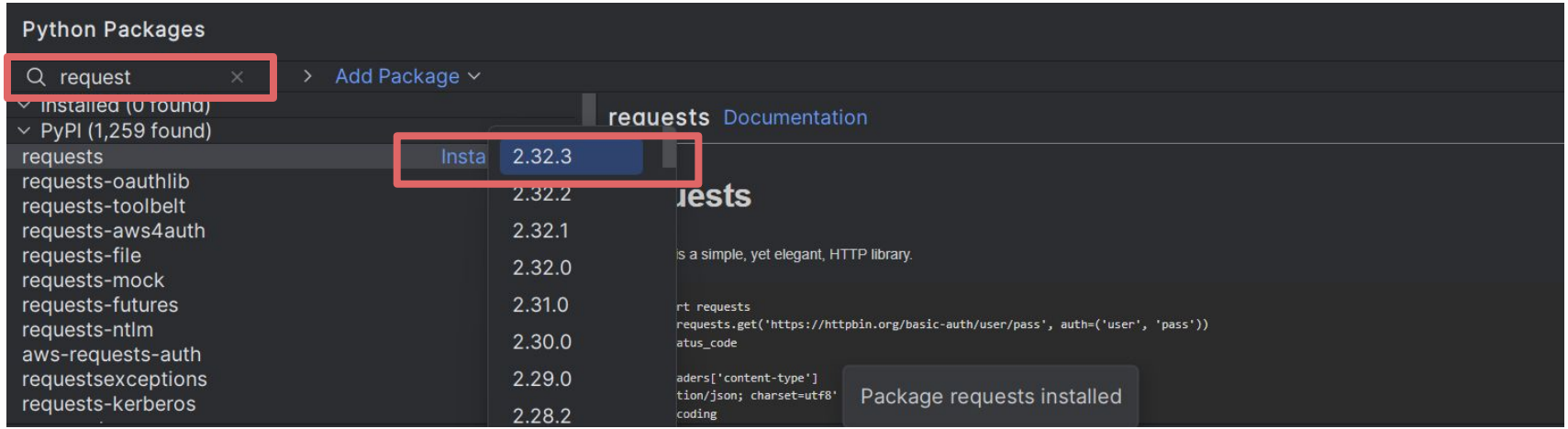
Prerequisite: Python Packages

In the upper left menu navigation bar select **`View > Tool Windows > Python Packages`**



Prerequisite: Download Request Packages

A new menu will open on the lower right. Search for the **`request`** library. Then select **`install`**. Make sure to select the latest version available.



Thread Pool Submission

```
1  import concurrent.futures
2  import time
3
4  def process(number):
5      _ = number * 1_000_000 ** 1_000_000
6      print("Finished computation")
7
8  if __name__=="__main__":
9      start_time = time.time()
10     with concurrent.futures.ThreadPoolExecutor() as executor:
11         x = executor.submit(process, 3)
12         y = executor.submit(input, "Enter number: ")
13
14     end_time = time.time()
15     print(end_time - start_time)
16
```

Thread Pool Mapping

```
1  import concurrent.futures
2  import requests
3  import time
4
5  def fetch_url(url):
6      return requests.get(url).status_code
7
8  urls = [ 'https://httpbin.org/delay/5',
9           'https://httpbin.org/delay/7']
10 if __name__==”__main__”:
11     start_time = time.time()
12     with concurrent.futures.ThreadPoolExecutor() as executor:
13         results = executor.map(fetch_url, urls)
14
15     end_time = time.time()
16     print(end_time - start_time)
```

H2

Website Check

Check multiple websites if they are working

Website Check

```python
import concurrent.futures
import requests
import time

def check_website(url):
    try:
        response = requests.get(url)
        if response.status_code == 200:
            print(f"{url} is up!")
        else:
            print(f"{url} status {response.status_code}")
    except:
        print(f"{url} failed to reach.")

```

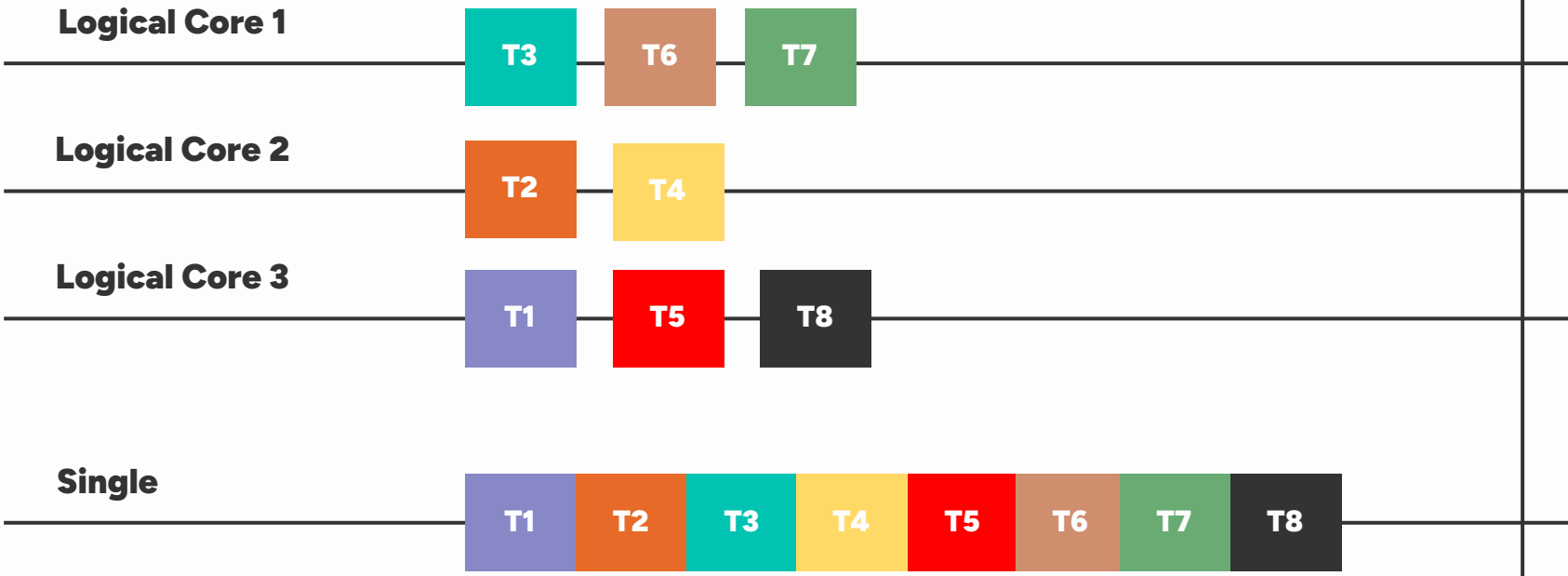
Manual Task

```
15 def read_websites(file_path):
16     with open(file_path, 'r') as file:
17         websites = file.readlines()
18         return [website.strip() for website in websites]
19
20 start_time = time.time()
21
22 websites = read_websites('websites.txt')
23 with concurrent.futures.ThreadPoolExecutor() as executor:
24     executor.map(check_website, websites)
25
26 end_time = time.time()
27 print(end_time - start_time)
```

Multiprocessing

Actually doing multiple tasks at once

Parallelism using Multiprocessing



Sequential Task

```python
import multiprocessing
import time

def process(number):
    return number * 1_000_000 ** 1_000_000

if __name__=="__main__":
    start_time = time.time()

    numbers = [(number + 1) for number in range(3)]
    results = [process(number) for number in numbers]

    end_time = time.time()
    print(end_time - start_time)

```

Multi-Process Task

```python
from multiprocessing import Pool
import time

def process(number):
    return number * 1_000_000 ** 1_000_000

if __name__==”__main__”:
    start_time = time.time()

    numbers = [(number + 1) for number in range(3)]
    with Pool() as pool:
        results = pool.map(process, numbers)

    end_time = time.time()
    print(end_time - start_time)
```

H3

Fibonacci Task

Fancy counting done fast

Sequential Fibonacci Calculation

```python
from multiprocessing import Pool
import time

def fibonacci(n):
    if n <= 1:
        return n
    return fibonacci(n - 1) + fibonacci(n - 2)

if __name__=="__main__":
    start_time = time.time()
    numbers = [35, 36, 37, 38]
    for number in numbers:
        print(f"Fibonacci({number}) = {fibonacci(number)}")

    end_time = time.time()
    print(end_time - start_time)
```

03

Best Practices

Recommended way to write Python code

Example Code No. 1

```
def function(s):
    ws = s.split()

    vc = 0
    vs  = "aeiou"

    for w in ws:
        if any(v in w for v in vs):
            vc += 1

    return vc
```

Example Code No. 1 (Refactor)

```
def count_words_with_vowel(text):
    words = text.split()

    words_with_vowels_count = 0
    vowels= "aeiou"

    for word in words:
        if any(vowel in words for vowel in vowels):
            words_with_vowels_count += 1

    return words_with_vowels_count
```

Example Code No. 2

```
def function(is):
    ic = {}

    for i in is:

        if i in ic:
            ic[i] += 1
        else:
            ic[i] = 1

    return ic
```

Example Code 2 (Refactor)

```
def count_per_item(items):
    item_count = {}

    for item in items:

        if item in item_count:
            item_count[item] += 1
        else:
            item_count[item] = 1

    return item_count
```

“Code is read much more often than it is written.”

— Guido van Rossum

Python ≈ English

Programming language

import this

If the implementation is **hard to explain**, it's a **bad idea**

Programming Principles

Don’t Repeat Yourself

Code duplication is a sign to use variables, functions, classes, and loops

Keep it Simple, Silly

Always aim for the simplest approach to the code

Loose Coupling

Minimize dependency of functions and classes with each other

Abstraction

Hide details in classes and functions to make things simpler at a quick glance

Python Enhancement Proposal (PEP) 8

Consistency

Makes it easier to read code quickly out of experience

Maintenance

PEP 8 is built for the purpose of making code easier to debug

Community

PEP 8 reflects the format and conventions that communities use

PEP 8 Quick Notes

Use 4 Spaces

Don’t use tabs and especially don’t mix spaces and tab

Limit to 79 Chars

Limit lines (72 characters for comments) to make code more readable or digestible

Start Private

If you’re not sure, start private as it’s harder to go from public to private

Naming Convention

Use snake_case for variables, functions, and files. Use PascalCase for classes.

PEP 8 Long Statements

For long operations, place the operator at the front

```
income = (gross_wages
          + taxable_interest
          + (dividends - qualified_dividends)
          - ira_deduction
          - student_loan_interest)
```

```
income = (gross_wages +
          taxable_interest +
          (dividends - qualified_dividends) -
          ira_deduction -
          student_loan_interest)
```

PEP 8 Extra Whitespaces

Avoid extra spaces as it is unnecessary

```
spam(ham[1], {eggs: 2})
```

```
spam( ham[ 1 ], { eggs: 2 } )
```

```
dct['key'] = lst[index]
```

```
dct ['key'] = lst [index]
```

```
x             = 1
y             = 2
long_variable = 3
```

PEP 8 Implicit Boolean Checks

If your variable is a Boolean, don’t use an equality check (remember, it auto-uses `bool()`)

```
if greeting == True:
```

```
if greeting is True:
```

```
if greeting:
```

Documentation

Provide Some Context

Note all of the prerequisites or key insights needed to understand a process. Mainly, explain why you are doing it

Enhance Readability

If a process is really hard to understand, explain it in alternative ways of phrasing

Summarize Immediately

One line can summarize paragraphs or entire documents depending on the use case

Hallmarks of a Good Comment

Clear

Very specific and relevant

Updated

Outdated code is a severe liability

Not Redundant

Provide information not yet revealed

Proper Grammar

Keep it professional

Simple

A New Developer should follow it

References

Provide links to related or source of truth

Inline Comments

Inline comments can be used to make quick notes or one-off explanations on why

```
# Convert temperature from Celsius to Fahrenheit
temperature_f = (temperature_c * 9/5) + 32
```

```
# This is a variable
x = 10

# This prints x
print(x)
```

Docstrings

Docstrings are commonly used to document functions (summary, args, return, errors).

```
def calculate_circle_area(radius):
    """
    Return the area of a circle with the given radius.

    Args:
        radius (float): Circle's radius. Must be non-negative.

    Returns:
        float: Area of the circle.

    Raises:
        ValueError: If radius is negative.
    """
    if radius < 0:
        raise ValueError("Radius cannot be negative")
    return math.pi * radius ** 2
```

Docstrings

Docstrings can still be used for simple functions. In this case, they span for a single line

```
def greet():
    """Print a simple greeting message."""
    print("Hello, welcome!")
```

Docstrings

Besides the documentation on-hover, you can use docstrings to provide support for `help`

```
help(calculate_circle_area)
```

Docstrings

Docstrings can also be used for classes.

```
class VideoPlayer:
    """Provides convenient functions for playing and processing video files”""


    def __init__(self, video):
        """Provides convenient functions for playing and processing video files

        Args:
            video (str): Filename of video

        """
        self.video = video
```

Type Hinting (Input)

You can provide a hint on what data type you're expecting for function parameters

```
def add(number1: int, number2: int):
    """Returns the mathematical summation of the two numbers.

    Args:
        number1 (int): First addend in summation
        number2 (int): Second addend in summation

    Returns:
        int: Addition of the two numbers
    """
    return number1 + number2
```

Type Hinting (Output)

You can provide a hint on what data type you're expecting for function outputs

```
def add(number1: int, number2: int) -> int:
    """Returns the mathematical summation of the two numbers.

    Args:
        number1 (int): First addend in summation
        number2 (int): Second addend in summation

    Returns:
        int: Addition of the two numbers
    """
    return number1 + number2
```

Type Hinting Examples

There are a lot of built-in type hints for the standard data types and for nested data types

```
variable1: int = 1
variable2: list[int] = [1, 2, 3]
variable3: dict[str, int] = {“a”: 123, “b”: 456, “c”: 890}
variable4: dict[str, list[int]] = {“num1”: [1, 2, 3], “num2”: [4]}
variable5: tuple[int, int] = (0, 1)
variable6: list[tuple[int, int]] = [(9, 1), (2, 3), (5, 2)]
```

Type Hinting Store

For repeated type hints or to simplify processes, you can store type hints in variables

```
Pair = tuple[int, int]
list_of_pairs: list[Pair] = [(9, 1), (2, 3), (5, 2)]
```

```
UserData = dict[str, str]
users: list[UserData] = [
    {"name": "Alice", "email": "alice@example.com"},
    {"name": "Bob", "email": "bob@example.com"},
]
```

Consistent Variable Names

Do not suddenly shift your themes or word choice in-between cod

```
customer_name = "John Doe"
client_age = 30 customer
shopper_order = ["apple", "banana", "orange"]
```

```
customer_name = "John Doe"
customer_age = 30 customer
customer_order = ["apple", "banana", "orange"]
```

Avoid Abbreviations

It seems to make sense when you made it. But will we remember after a few weeks?

```
hrb = 5000
```

Avoid Abbreviations

Make it very clear from the get-go

```
hrb = 5000
```

```
human_resources_budget = 5000
```

Descriptive Variables

The variable name should be enough

```
x = 10
y = [1, 2]
data = “yes”
```

```
total_items = 10
list_of_attendees_per_day = [1, 2]
question01_response = “yes”
```

H4

Code Review

Let’s assess how to improve code

Improve this code:

```
def u(p):
    v = 1
    for w in range(1, p + 1):
    v *= w
    return v

x = 5
y = u(x)
print(y)
```

Improve this code:

```
def m(n):
    p = True
    for q in range(2, n):
    if n % q == 0:
        p = False
        break
    return p

r = 29
s = m(r)
print(s)
```

Improve this code:

```
def m(n):
    p = []
    for q in n:
    if q not in p:
        p.append(q)
    return p

r = [1, 2, 3, 3, 4, 5, 5]
s = m(r)
print(s)
```

Improve this code:

```
def n(a, b):
    t = []
    for i in range(len(a)):
    t.append(a[i] + b[i])
    return t

x = [1, 2, 3]
y = [4, 5, 6]
z = n(x, y)
print(z)
```

Common Types of Testing

Unit

Testing individual parts or functions in isolation

Integration

Testing if different components work together correctly

Regression

Testing if changes in the code doesn’t accidentally break anything

Unit Test

Testing individual components or functions in isolation from other parts

```
def square(x):
    return x * x

def test_square():
    assert square(2) == 4
    assert square(-3) == 9
    assert square(0) == 0
    print("All unit tests passed!")

test_square()
```

Integration Test

Testing if different components work as intended when combined together

```
def add(a, b):
    return a + b

def square(x):
    return x * x

def multiply(a, b):
    return a * b

```

Integration Test

Testing if different components work as intended when combined together

```
def calculate_expression(x, y):
    return add(square(x), multiply(y, 2))

def test_calculate_expression():
    assert calculate_expression(2, 3) == 10
    assert calculate_expression(0, 5) == 10

    print("All integration tests passed!")

test_calculate_expression()
```

Regression Test

Check if changes in the code have not affected existing functionality

```
10 def calculate_expression(x, y, z=0):
11     return add(square(x), multiply(y, 2)) - z
12
13 def test_calculate_expression():
14     assert calculate_expression(2, 3) == 10
15     assert calculate_expression(0, 5) == 10
16     assert calculate_expression(2, 3, 2) == 10
17     print("All integration tests passed!")
18
19 test_calculate_expression()
```

Test Driven Development

Red: Write tests first

Create a series of test that anticipates the creation of functions or classes and their expected behavior. This will automatically Fail

Stop writing tests on fail

When a test fails, stop creating the test and examine what functions or classes must be made or changed to pass the test.

Green: Write the minimum code

Write the needed function or classes to pass the tests. Do not make more than necessary as we will go back to step 1 until feature fully expressed and tested.

H5

Intentional Bug

A surprising amount of time is invested here

Fix the possible bug

```
def find_even_numbers(numbers):
    evens = []
    for num in numbers:
        if num % 2 == 1:
            evens.append(num)
        return evens

numbers = [1, 2, 3, 4, 5, 6]
print(f"Even numbers: {find_even_numbers(numbers)}")
```

Fix the possible bug

```
def remove_duplicates(numbers):
    for num in numbers:
        if numbers.count(num) > 1:
            numbers.remove(num)
        return numbers

numbers = [1, 2, 2, 3, 3, 4]
print(f"Unique numbers: {remove_duplicates(numbers)}")
```

Fix the possible bug

```
def average(numbers):
    total = 0
    for num in numbers:
        total += num
    return total / len(nums)

numbers = [10, 20, 30, 40]
print(average(numbers))
```

Fix the possible bug

```
def count_positive_numbers(numbers):
    count = 0
    for num in numbers:
    if num > 0:
        count += 1
    else:
        count -= 1
    return count

numbers = [1, -2, 3, 4, -5, 6]
print(count_positive_numbers(numbers))
```

04

Web Dev

Providing online access to your business logic

Web Frameworks

Flask

- Minimalist and lightweight
- Freedom to choose tools for each part
- **Small and Fast Web Applications**

Django

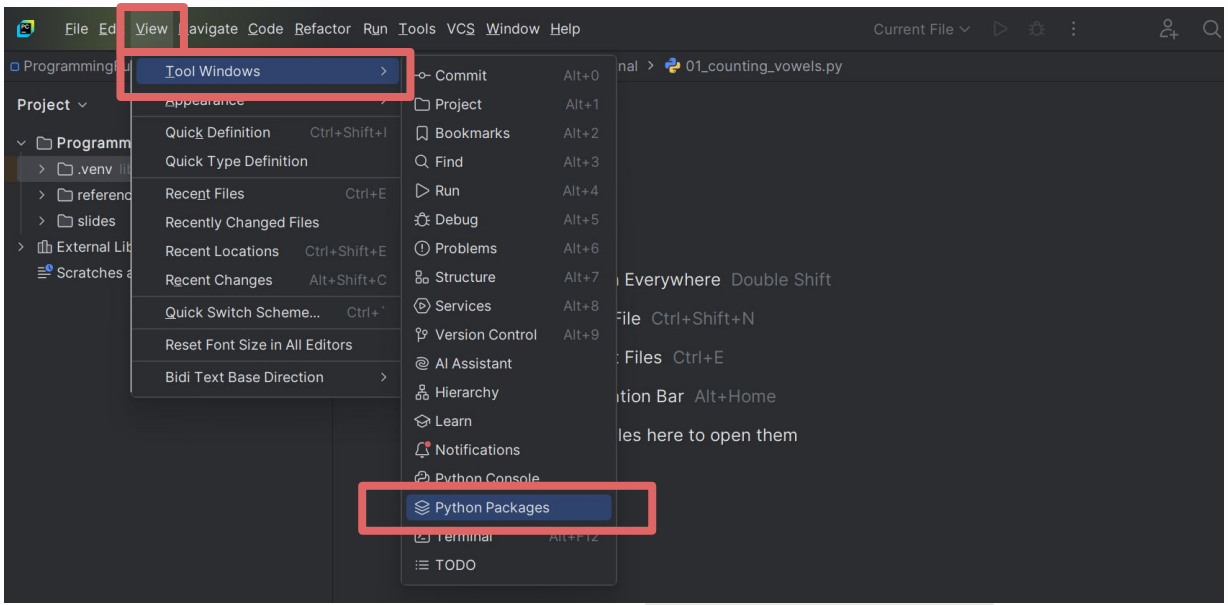
- Multiple out-of-the-box features
  - Object Relational Mapping
  - Fully functional Admin Panel
  - Security Measures and Authentication
- **Medium to Large Web applications**

Initial Setup

Package download and Initial Page

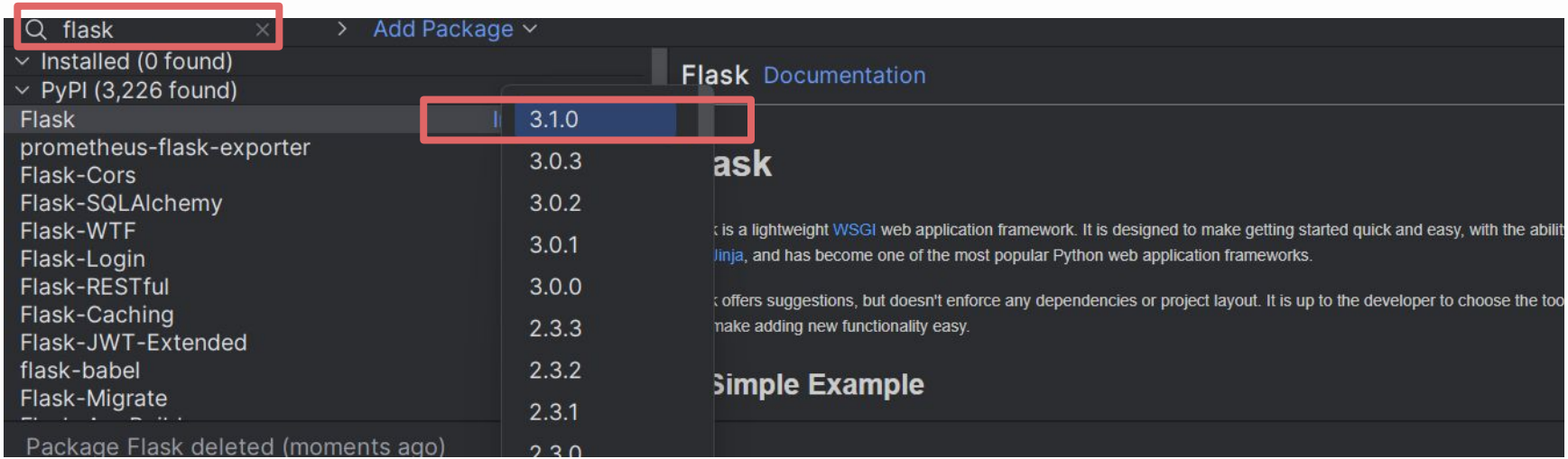
Prerequisite: Python Packages

In the upper left menu navigation bar select **`View > Tool Windows > Python Packages`**



Prerequisite: Download Request Packages

A new menu will open on the lower right. Search for the **`flask`** library.
Then select **`install`**. Make sure to select the latest version available.



Minimum Setup

```python
from flask import Flask

app = Flask(__name__)
app.run()
```

Routing

Setting up the subpages of the site

Index Route

```
1  from flask import Flask
2
3  app = Flask(__name__)
4
5  @app.route("/")
6  def index():
7      return "Index Page"
8
9  app.run()
10
11
12
13
14
15
```

Additional Route

```
1  from flask import Flask
2
3  app = Flask(__name__)
4
5  @app.route("/")
6  def index():
7      return "Index Page"
8
9  @app.route("/profile/")
10 def profile():
11     return "Profile Page"
12
13 app.run()
14
15
```

Route Aliasing

```
1  from flask import Flask
2
3  app = Flask(__name__)
4
5  @app.route("/")
6  def index():
7      return "Index Page"
8
9  @app.route("/profile/")
10 @app.route("/profiles/")
11 def profile():
12     return "Profile Page"
13
14 app.run()
15
```

Dynamic Route

```python
from flask import Flask

app = Flask(__name__)

@app.route("/")
def index():
     return "Index Page"

@app.route("/profiles/")
def profile():
     return "Profile Page"

@app.route("/profile/<username>")
def dynamic_profile(username):
     return f"Profile {username} Page"

app.run()

```

Full Dynamic Route

```
1  from flask import Flask
2
3  app = Flask(__name__)
4
5  @app.route("/")
6  def index():
7       return "Index Page"
8
9  @app.route("/profile/")
10 @app.route("/profiles/")
11 @app.route("/profile/<username>")
12 def profile(username=None):
13      if username:
14           return f"Profile {username} Page"
15      else:
16           return "Profile Page"
17
18 app.run()
```

Quick Exercise: Provide these routes

/ /home/ /index/	“Welcome to my page”
/home/<name> /index/<name>	“Welcome to my page <name>”
/page/<topic>	Viewing page for<topic>

HTML

A crash course on styling text in web pages

HTML: Hypertext Markup Language

HTML is used to structure and organize content on web pages. It relies on tags, which define elements like headings, paragraphs, and links, to create a webpage's layout and content.

<tag> Text </tag>

<tag>

Headers

Heading tags (**<h1>** to **<h6>**) define the importance and hierarchy of text, with **<h1>** being the highest and **<h6>** the lowest.

<h1>Header </h1>

<h2>Header </h2>

<h3>Header </h3>

<h4>Header </h4>

<h5>Header </h5>

<h6>Header </h6>

Headers

Heading tags (**<h1>** to **<h6>**) define the importance and hierarchy of text, with **<h1>** being the highest and **<h6>** the lowest.

<h1> **Header** </h1>

<h2> **Header** </h2>

<h3> **Header** </h3>

<h4> **Header** </h4>

<h5> **Header** </h5>

<h6> **Header** </h6>

Paragraphs

The tag is used to define paragraphs, separating blocks of text for better readability.

<h1>Header </h1>

The p tag is used to define paragraphs

Paragraphs

The tag is used to define paragraphs, separating blocks of text for better readability.

<h1> **Header** </h1>

 The p tag is used to define paragraphs

Anchor

The <a> tag is used to create hyperlinks that redirect the user to a different URL.

```
<a href ="https://www.example.com  ">Example </a>
```

Anchor

The **<a>** tag is used to create hyperlinks that redirect the user to a different URL.

```
<a href="https://www.example.com">
```

Example

```
</a>
```

https://www.example.com

Unordered List

The **<ul>** tag with **<li>** tags enumerate items in bullet point style

```html
<ul>
    <li>First Item</li>
    <li>Second Item</li>
    <li>Third Item</li>
</ul>
```

- First Item
- Second Item
- Third Item

Ordered List

The **<ol>** tag with **<li>** tags enumerate items by number

```
<ol>
    <li>First Item</li>
    <li>Second Item</li>
    <li>Third Item</li>
</ol>
```

```
1.  First Item
2.  Second Item
3.  Third Item
```

Nested List

Subitems require an additional tag

```html
<ul>
    <li>First Item</li>
    <ul>
        <li>Sub Item</li>
    </ul>
    <li>Second Item</li>
    <li>Third Item</li>
</ul>
```

- First Item
  - Sub Item
- Second Item
- Third Item

HTML Example

```python
from flask import Flask
app = Flask(__name__)

@app.route("/")
def home():
    return """
        <h1>Welcome to Flask</h1>
        <p>This is a simple example of HTML in Flask</p>
        <ol>
            <li>Learn Flask</li>
            <li>Build a project</li>
        </ol>
        <a href="https://flask.palletsprojects.com/">Guide</a>
    """
app.run()
```

Refactor: Add styling and content

/ /home/ /index/	“Welcome to my page”
/home/<name> /index/<name>	“Welcome to my page <name>”
/page/<topic>	Viewing page for<topic>

URL Handling

Special cases for handling subpages

Dynamic URL

```
1  from flask import Flask, url_for
2  app = Flask(__name__)
3
4  @app.route("/")
5  def index():
6      return f'''
7      <a href="{url_for('login')}">Login Page</a>
8      <a href="{url_for('profile', username='Ace')}">Ace</a>
9      '''
10
```

Dynamic URL

```
11  @app.route("/login/")
12  def login():
13      return “Login Page”
14
15  @app.route("/user/<username>")
16  def profile(username):
17      return f”{username}'s Profile Page”
18
19  app.run()
20
```

Redirect URL

```python
from flask import Flask, url_for, redirect
app = Flask(__name__)

@app.route("/user/<username>")
def profile(username):
    if username != "admin":
        return redirect(url_for('login'))
    else:
        return "Welcome Admin"

@app.route('/login')
def login():
    return "Please login"

app.run()
```

Abort Error

```
1  from flask import Flask, abort
2
3  app = Flask(__name__)
4
5  @app.route('/')
6  def index():
7      return "Index Page"
8
9  @app.route('/login')
10 def login():
11     abort(501)
12
13 app.run()
```

Error Handler

```
1  from flask import Flask, url_for, abort, redirect
2
3  app = Flask(__name__)
4
5  @app.route("/user/<username>")
6  def profile(username):
7      if username in ['Alex', 'Steve']:
8          return f"{username}'s Profile Page"
9      elif username == 'Guest':
10         return "Guest Profile"
11     else:
12         abort(401)
```

Error Handler

```
14 @app.errorhandler(401)
15 def handle_401_error(error):
16     print(“Undetected visitor”)
17     return redirect(url_for('profile', username='Guest'))
18
19 app.run()
```

Quick Exercise: Provide these routes

/lion, /animal/lion	→	"Welcome to the lion page"
/giraffe, /animal/giraffe	→	"Welcome to the giraffe page"
/eagle, /animal/eagle	→	"Welcome to the eagle page"
/?, /animal/?	→	"Coming Soon!"
/	→	"Welcome to the zoo!"

Requests

Asking users for information

Login Get

```python
from flask import Flask, request

app = Flask(__name__)

@app.get('/login')
def login_get():
    return """
    <form method="post">
        <label for="username">Username:</label>
        <input type="text" name="username">

        <input type="submit">
    </form>
    """
```

Login Post

```
15 @app.post('/login')
16 def login_post():
17     username = request.form['username']
18     return f”Form Submitted by {username}”
19
20 app.run()
```

Login Form Get

```
1  from flask import Flask, request
2  app = Flask(__name__)
3
4  @app.get('/login')
5  def login_get():
6      return """
7      <form method="post">
8          <label for="username">Username:</label>
9          <input type="text" name="username"><br>
10         <label for="password">Password:</label>
11         <input type="password" name="password”><br>
12         <label for="email">Email:</label>
13         <input type="email" name="email"><br>
14         <input type="submit" value="Login">
15     </form>
16     """
```

Login Form Post

```
17 def valid(username, email , password ):
18     return not (
19         username == "admin"
20         and password == "pass"
21         and email == "admin@gmail.com"
22     )
23 @app.post('/login')
24 def login_post():
25     username = request.form['username']
26     password = request.form['password']
27     email = request.form['email']
28     If not valid(username, email , password ):
29         return 'Invalid credentials!'
30     else:
31         return 'Login successful!'
```

Sessions

Server-side data storage

Session Setup

```python
from flask import Flask, request, redirect, url_for, session

app = Flask(__name__)
app.secret_key = 'your_secret_key'

users = {
    "admin": "password123",
    "user": "pass456"
}

```

Session Home

```
11  @app.route('/')
12  def home():
13      if 'username' in session:
14          return f"""
15              Welcome, {session['username']}!
16              <a href='/logout'>Logout</a>
17          """
18      else:
19          return f"""
20              Welcome!
21              <a href='/login'>Login</a>
22          """
```

Session Login Get

```
23 @app.get('/login')
24 def login_get():
25     return f"""
26     <form method="post">
27         <label for="username">Username:</label>
28         <input type="text" name="username"><br>
29         <label for="password">Password:</label>
30         <input type="password" name="password"><br>
31         <input type="submit" value="Login">
32     </form>
33     """
34
```

Session Validation

```
35 @app.post('/login')
36 def login_post():
37     username = request.form['username']
38     password = request.form['password']
39     if username in users and users[username] == password :
40         session['username'] = username
41         return redirect(url_for('home'))
42     else:
43         return redirect(url_for('login_get'))
44
45 @app.route('/logout')
46 def logout():
47     session.pop('username', None)
48     return redirect(url_for('home'))
49
50 app.run()
```

Templates

Adding placeholders and logic to HTML

Render Template

```python
from flask import Flask, render_template

app = Flask(__name__)

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

app.run()
```

Render Template - HTML

```html
<!DOCTYPE html>
<html lang="en">
    <head>
        <title>Demo App</title>
    </head>

    <body>
        <h1>Demo Page</h1>
        <p>Simple demo application</p>
    </body>
</html>

```

Render Template - Parameter

```python
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/')
def index():
    return render_template(
        ”index_variable.html”,
        title="Template App",
        message="Template Demo Page",
        additional_message="Template used",
    )

app.run()
```

Render Template - HTML Parameter

```html
<!DOCTYPE html>
<html lang="en">
    <head>
        <title>{{ title }}</title>
    </head>

    <body>
        <h1>{{ message }}</h1>
        <p>This is a simple Flask demo application</p>
        {{ additional_message }}
    </body>
</html>

```

Render Template - Conditional

```
1  from flask import Flask, render_template
2
3  app = Flask(__name__)
4
5  @app.route('/')
6  def index():
7      return render_template(conditional.html', logged_in=True)
8
9  app.run()
```

Render Template - HTML Conditional

```html
<!DOCTYPE html>
<html lang="en">
    <head>
        <title>Login</title>
    </head>
    <body>
        {% if logged_in %}
            <p>Welcome back, user!</p>
        {% else %}
            <p>Please log in to continue.</p>
        {% endif %}
    </body>
</html>
```

Render Template - Items

```python
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/')
def index():
    items = ['Apple', 'Banana', 'Cherry']
    return render_template('items.html', items=items)

app.run()
```

Render Template - HTML Loop

```html
<!DOCTYPE html>
<html lang="en">
    <head>
        <title>Items</title>
    </head>
    <body>
        <h2>Available Items:</h2>
        <ul>
        {% for item in items %}
            <li>{{ item }}</li>
        {% endfor %}
        </ul>
    </body>
</html>
```

Render Template - Dictionary

```
1  from flask import Flask, render_template
2
3  app = Flask(__name__)
4
5  @app.route('/')
6  def index():
7      user_info = {
8          'name': 'Eren',
9          'location': 'Manila'
10     }
11     return render_template('profiles.html', user=user_info)
12
13 app.run()
```

Render Template - HTML Dictionary

```html
<!DOCTYPE html>
<html lang="en">
    <head>
        <title>User Profile</title>
    </head>
    <body>
        <h2>User Profile</h2>
        <p>Name: {{ user['name'] }}</p>
        <p>Age: {{ user['age'] }}</p>
        <p>Location: {{ user['location'] }}</p>
    </body>
</html>
```

Components

Templating the HTML files themselves

Parent HTML

```html
<!DOCTYPE html>
<html lang="en">
    <head>
        <title>
            {% block title %} My App {% endblock %}
        </title>
    </head>
    <body>
        <header>
            <h1>Welcome to My Flask App</h1>
        </header>
        {% block content %} {% endblock %}
        <footer>
            <p>Flask 2025</p>
        </footer>
    </body>
</html>
```

Child HTML

```
{% extends 'parent.html' %}

{% block title %}
    Home
{% endblock %}

{% block content %}
    <h1>Subclass Page</h1>
    <p>Welcome to the subclass page!</p>
{% endblock %}
```

05

Lab Session

Additional References

Additional references you can look into:

Books

- Automate the Boring Stuff with Python
- Python Distilled
- Fluent Python

YouTube

- CS50 - CS50P Python
- Bro Code - Python Full Course
- Corey Schafer - Python Playlist

Python: Day 04

Advanced Programming

Happy Coding!

stephen.singer.098@gmail.com