Python: Day 04

Advanced Programming

Agenda

01

Packaging

Internal and external files

02

Multiple Tasks

Handling bottlenecks

03

Best Practices

Professional Development

04

Web Dev

Introduction to Flask

05

Lab Session

Culminating Exercise

01

Packaging

How to organize Python files

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!")
3
4  def say_goodbye():
5      print("Goodbye")
6
7  message = "Hello World"
8  var1 = "Hello"
9  var2 = "Hi"
10
11 print("Module hello")
12
13
```

./example.py

```
1  import hello
2
3  hello.say_hello()
4
5
6
7
8
9
10
11
12
13
```

Basic Import

./hello.py

```
1  def say_hello():
2      print("Hello!")
3
4  def say_goodbye():
5      print("Goodbye")
6
7  message = "Hello World"
8  var1 = "Hello"
9  var2 = "Hi"
10
11 if __name__=='__main__':
12     print("Module hello")
13
```

./example.py

```
1  import hello
2
3  hello.say_hello()
4
5
6
7
8
9
10
11
12
13
```

Specific Import

./hello.py

```
1  def say_hello():
2      print("Hello!")
3
4  def say_goodbye():
5      print("Goodbye")
6
7  message = "Hello World"
8  var1 = "Hello"
9  var2 = "Hi"
10
11 if __name__=='__main__':
12     print("Module hello")
13
```

./example.py

```
1  import hello
2
3  from hello import say_goodbye
4
5  hello.say_hello()
6
7  say_goodbye()
8
9
10
11
12
13
```

Basic Import with Alias

./hello.py

```
1  def say_hello():
2      print("Hello!")
3
4  def say_goodbye():
5      print("Goodbye")
6
7  message = "Hello World"
8  var1 = "Hello"
9  var2 = "Hi"
10
11 if __name__=='__main__':
12     print("Module hello")
13
```

./example.py

```
1  import hello
2  import hello as ho
3
4  from hello import say_goodbye
5
6  hello.say_hello()
7
8  say_goodbye()
9
10 ho.say_hello()
11
12
13
```

Multiple Specific Imports

./hello.py

```
1  def say_hello():
2      print("Hello!")
3
4  def say_goodbye():
5      print("Goodbye")
6
7  message = "Hello World"
8  var1 = "Hello"
9  var2 = "Hi"
10
11 if __name__=='__main__':
12     print("Module hello")
13
```

./example.py

```
1  import hello
2  import hello as ho
3
4  from hello import say_goodbye
5  from hello import var1, var2
6
7  hello.say_hello()
8
9  say_goodbye()
10
11 ho.say_hello()
12 print(var1, var2)
13
```

Basic Nested Import

./package/module_01.py

```
1  def say_hello():
2      print("Hello!")
3
4  def say_goodbye():
5      print("Goodbye")
6
7  message = "Hello World"
8  var1 = "Hello"
9  var2 = "Hi"
10
11
12
13
```

./nested_example.py

```
1  import package.module_01
2
3  package.module_01.say_hello()
4
5
6
7
8
9
10
11
12
13
```

Specific Nested Import

./package/module_01.py

```
1  def say_hello():
2       print("Hello!")
3
4  def say_goodbye():
5       print("Goodbye")
6
7  message = "Hello World"
8  var1 = "Hello"
9  var2 = "Hi"
10
11
12
13
```

./nested_example.py

```
1  import package.module_01
2
3  from package.module_01 import say_goodbye
4
5
6  package.module_01.say_hello()
7  say_goodbye()
8
9
10
11
12
13
```

Specific Nested Import

./package/module_01.py

```
1  def say_hello():
2       print("Hello!")
3
4  def say_goodbye():
5       print("Goodbye")
6
7  message = "Hello World"
8  var1 = "Hello"
9  var2 = "Hi"
10
11
12
13
```

./nested_example.py

```
1  import package.module_01
2  import package.module_01 as pm1
3
4  from package.module_01 import say_goodbye
5
6
7  package.module_01.say_hello()
8  say_goodbye()
9  print(pm1.message)
10
11
12
13
```

Standard Packaging Format 01

```
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/
```

Standard Packaging Format 02

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

Quick Exercise: Organize RPG

```
rpg/
   ├── character/
   │    ├── character.py
   │    ├── mage.py
   │    ├── knight.py
   │    ├── warrior.py
   │    └── __init__.py
   └── main.py
```

Relative Imports

./character.py

```
1
2
3
4
```

```
class Character:
    pass
```

./knight.py

```
1
2
3
4
```

```
from .character import Character

class Knight:
    pass
```

./main.py

```
1
2
3
4
```

```
from character.knight import Knight
```

```
rpg/
├── character/
│   ├── character.py
│   ├── knight.py
│   └── __init__.py
└── main.py
```

Libraries

Please don’t reinvent the wheel

Try these Libraries!

Math

Common math constants and operations

Functools

Module for higher-order functions

Collections

Additional data structures

Time

Access to system time, delays, and conversions

Request

Quick setup for a light database system

Itertools

Efficient looping and combinatorials

Time Demo

```
1  import time
2
3  print("Measuring Execution Time:")
4  print("Current Time:", time.ctime())
5  time.sleep(10)
6  print("Current Time:", time.ctime())
7  print()
8
9  print("Measuring Execution Time:")
10 start_time = time.time()
11 for _ in range(1_000_000):
12     x = 10 ** 1000
13 end_time = time.time()
14 print(f"Spent {end_time - start_time:.5f} seconds")
```

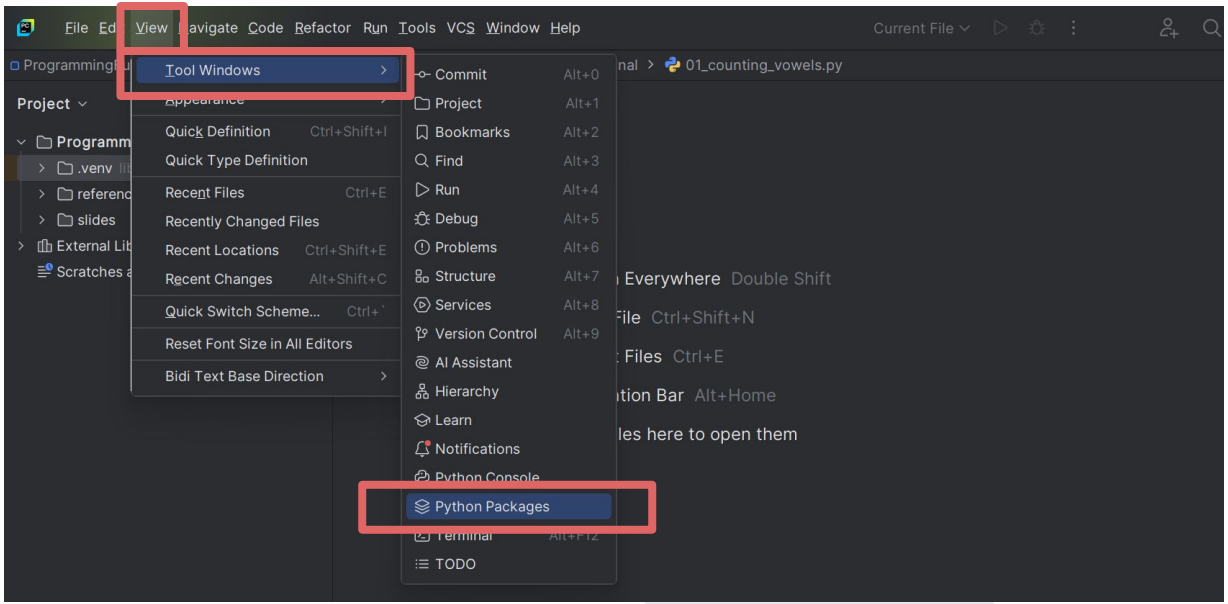
Functools Demo

```
1  def fib(n):
2      if n <= 1:
3          return n
4      return fib(n-1) + fib(n-2)
5
6  print(fib(38))
```

```
1  from functools import cache
2
3  @cache
4  def fib(n):
5      if n <= 1:
6          return n
7      return fib(n-1) + fib(n-2)
8
9  print(fib(300))
```

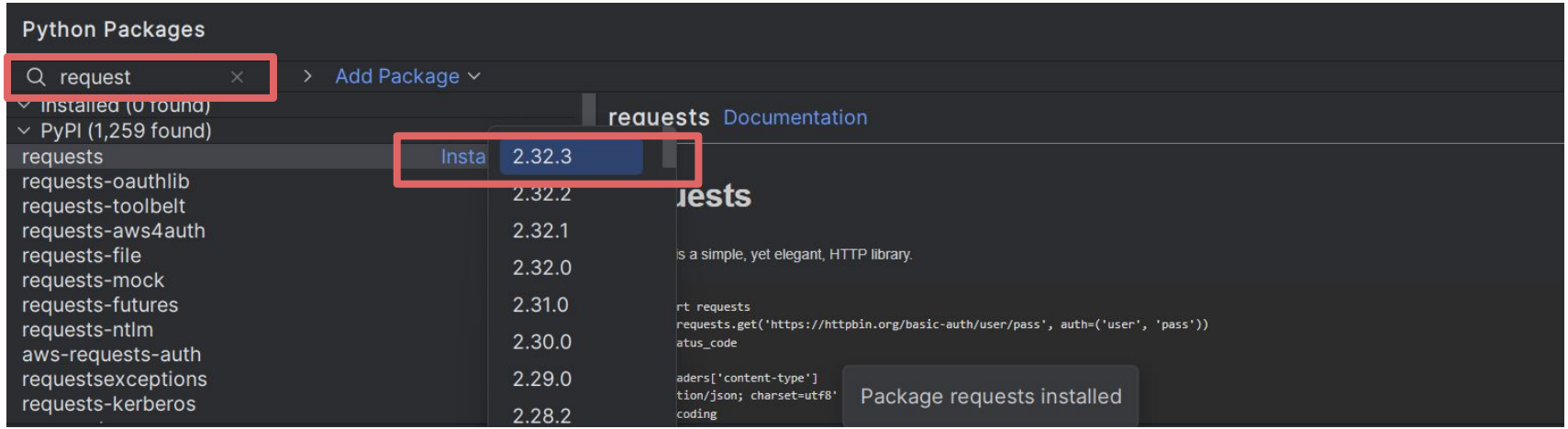
Prerequisite: Python Packages

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.



Requests Demo

The requests library allows Python to simplify HTTP requests

```python
import requests

# Send a GET request to a free joke API
site = "https://official-joke-api.appspot.com/random_joke"
response = requests.get(site)

# Check if the request was successful
if response.status_code == 200:
    joke = response.json()
    print(joke['setup'])
    print(joke['punchline'])
else:
    print("Failed. Server said:", response.status_code)
```

H1

USD Conversion

Real-time data with Python

USD Conversion

01_usd_conversion.py

```
1  import requests
2
3  response = requests.get("https://open.er-api.com/v6/latest/USD")
4
5  # Get the latest conversion rate from USD to PHP
6  print()
```

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

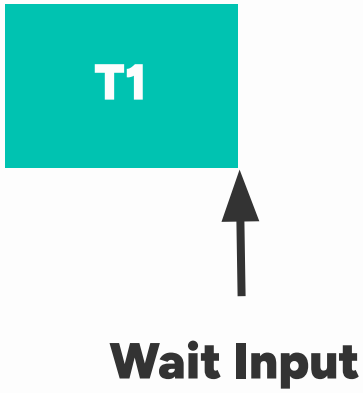
Concurrency

Working while waiting for other tasks

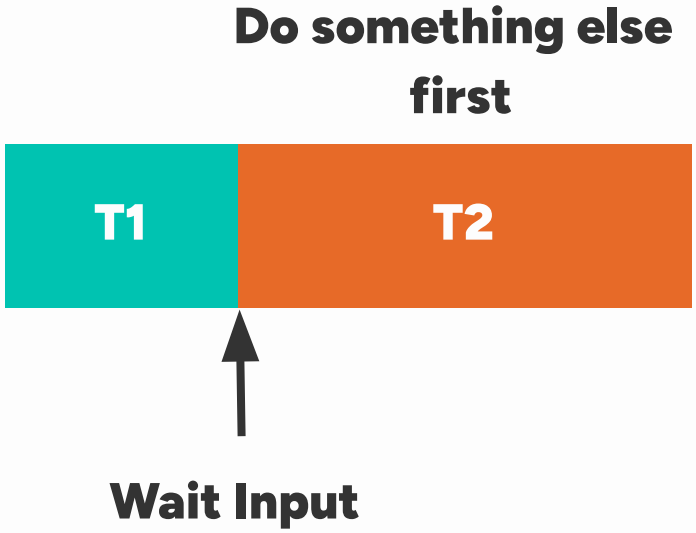
Concurrent Process

Current Task

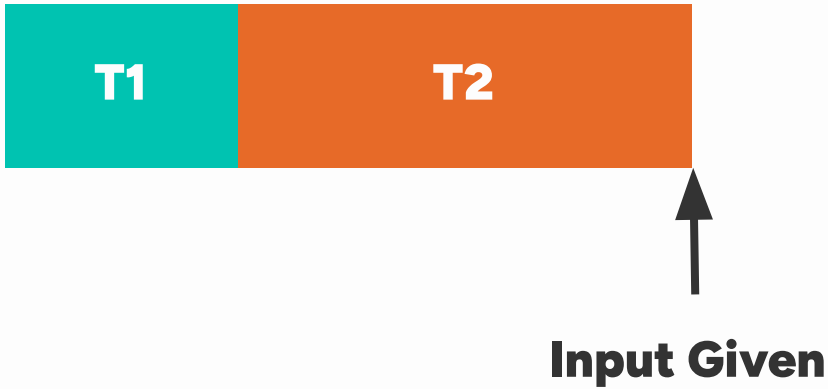
Concurrent Process



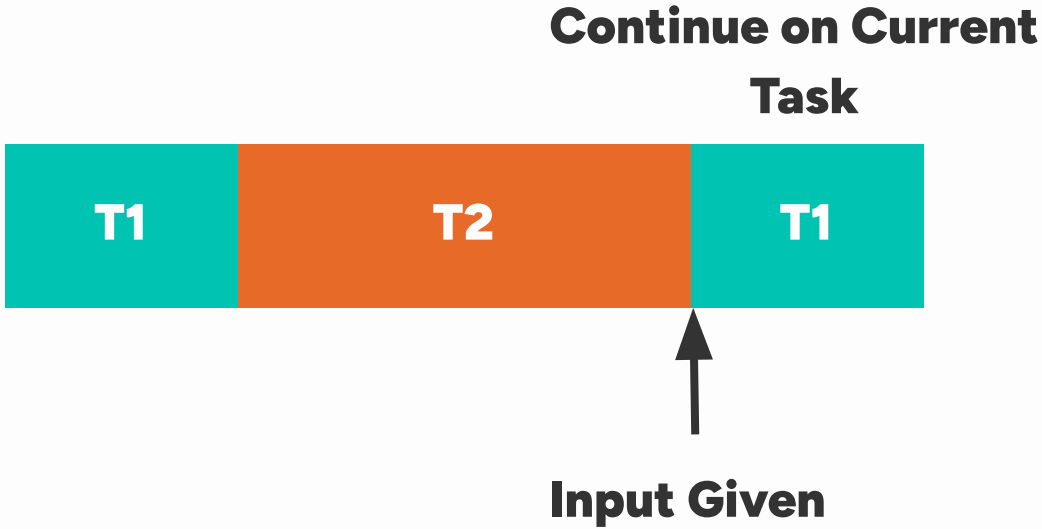
Concurrent Process



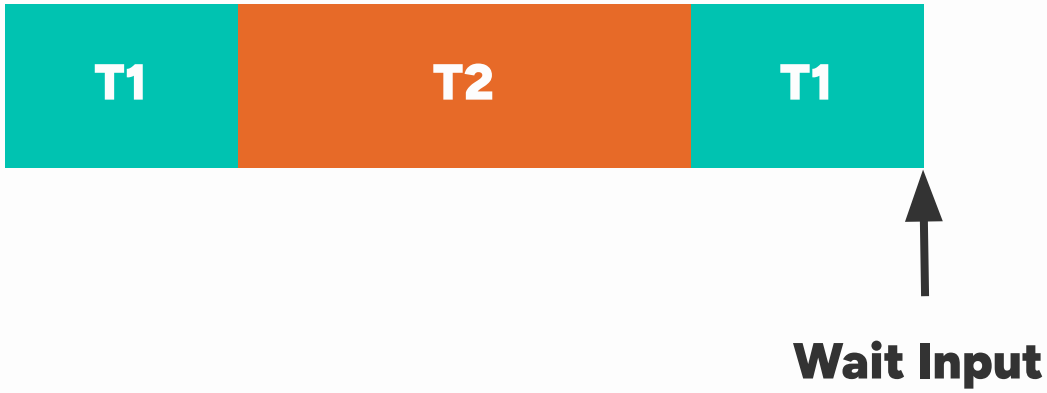
Concurrent Process



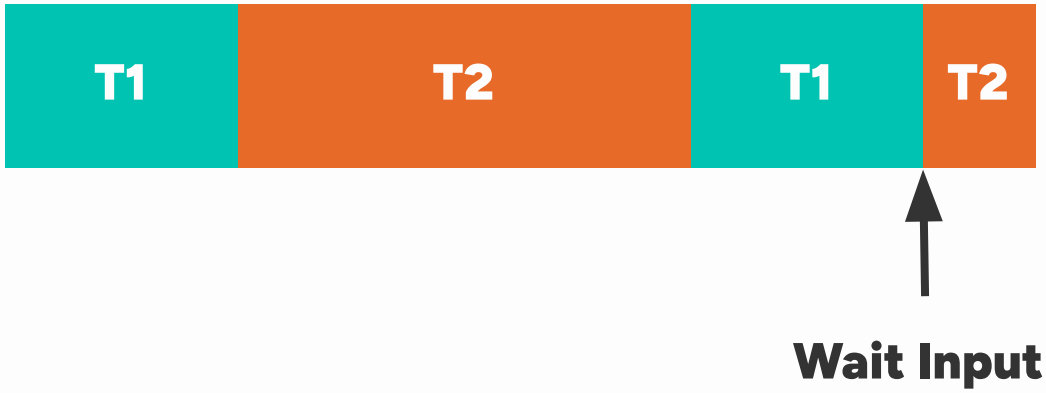
Concurrent Process



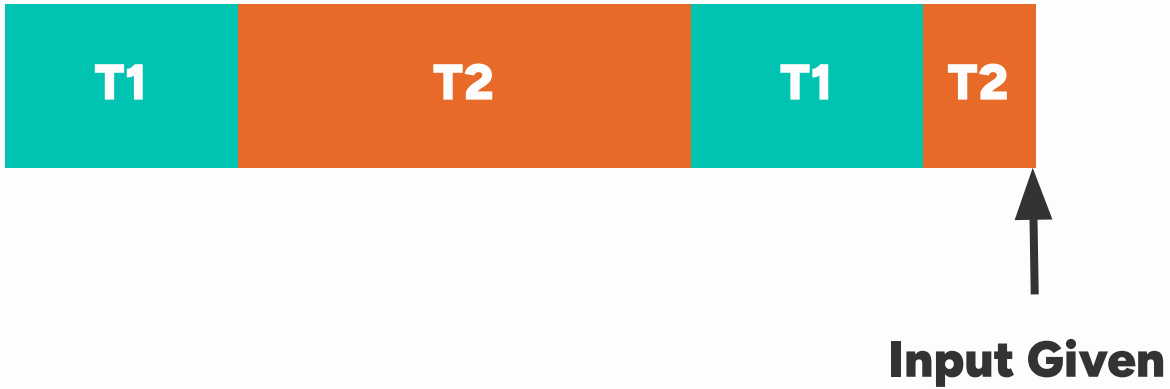
Concurrent Process



Concurrent Process

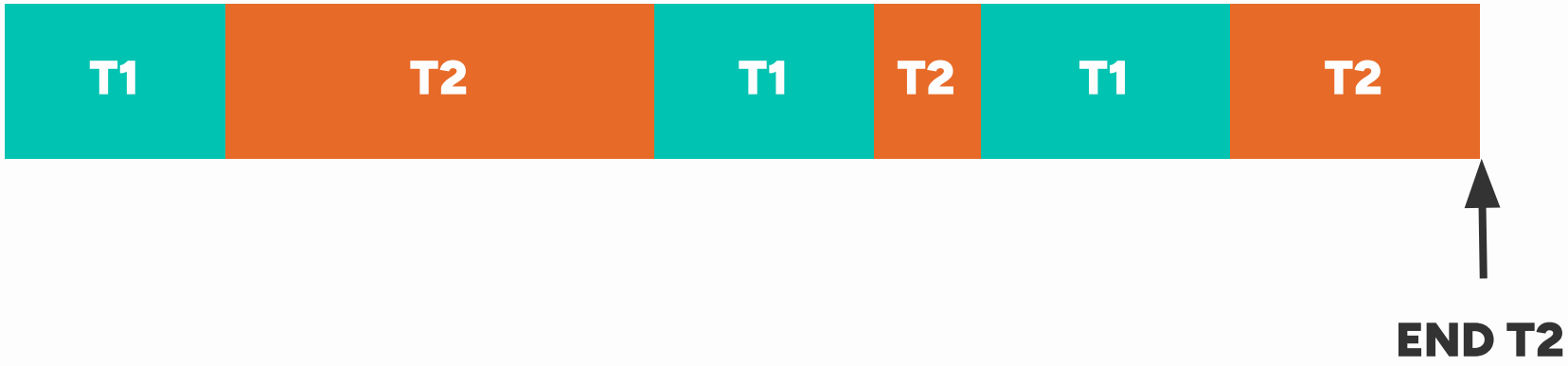


Concurrent Process

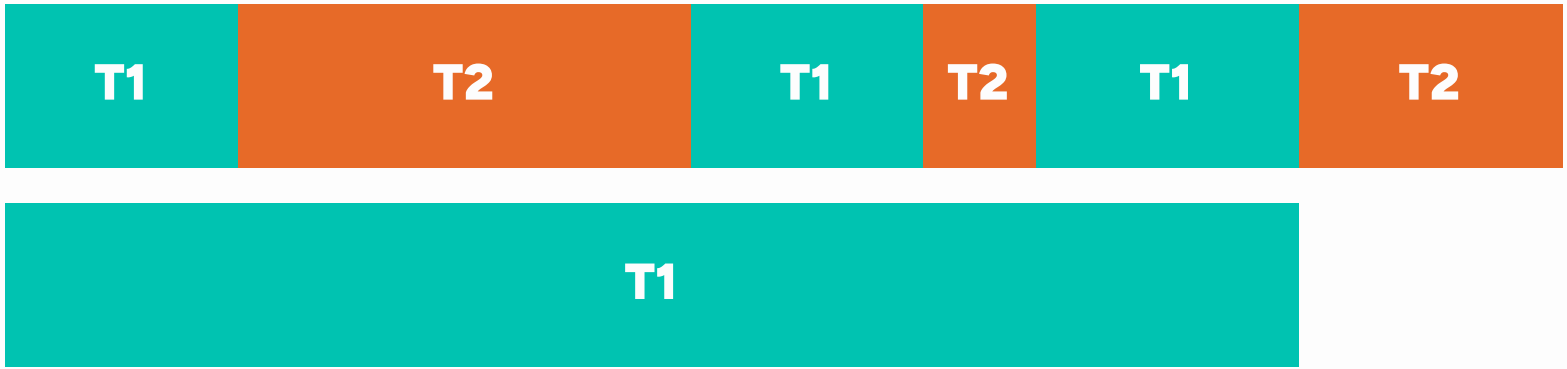


Concurrent Process

Concurrent Process



Concurrent Process



Thread Pool Mapping

```
1  import requests
2  import time
3
4  def fetch_url(url):
5       return requests.get(url).status_code
6
7  inputs = ['https://httpbin.org/delay/5', 'https://httpbin.org/delay/10']
8
9  if __name__=='__main__':
10      start_time = time.time()
11
12      for input in inputs:
13           outputs = [fetch_url(url) for url in inputs]
14
15      end_time = time.time()
16      print(end_time - start_time)
```

Thread Pool Mapping

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

H2

Website Check

Check multiple websites if they are working

Website Check - Main Function

```
from concurrent.futures import ThreadPoolExecutor
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.")
```

Website Check - Get Text Data

```
15 base_url = "https://raw.githubusercontent.com/"
16 file_name = "bensooter/URLchecker/master/top-1000-websites.txt"
17 response = requests.get(base_url + file_name)
18
19 websites = response.text.splitlines()
20 websites = ["https://" + for site in websites if site.strip()]
21 websites = [site.strip() for site in websites if site.strip()]
22
23 websites = websites[:100]
```

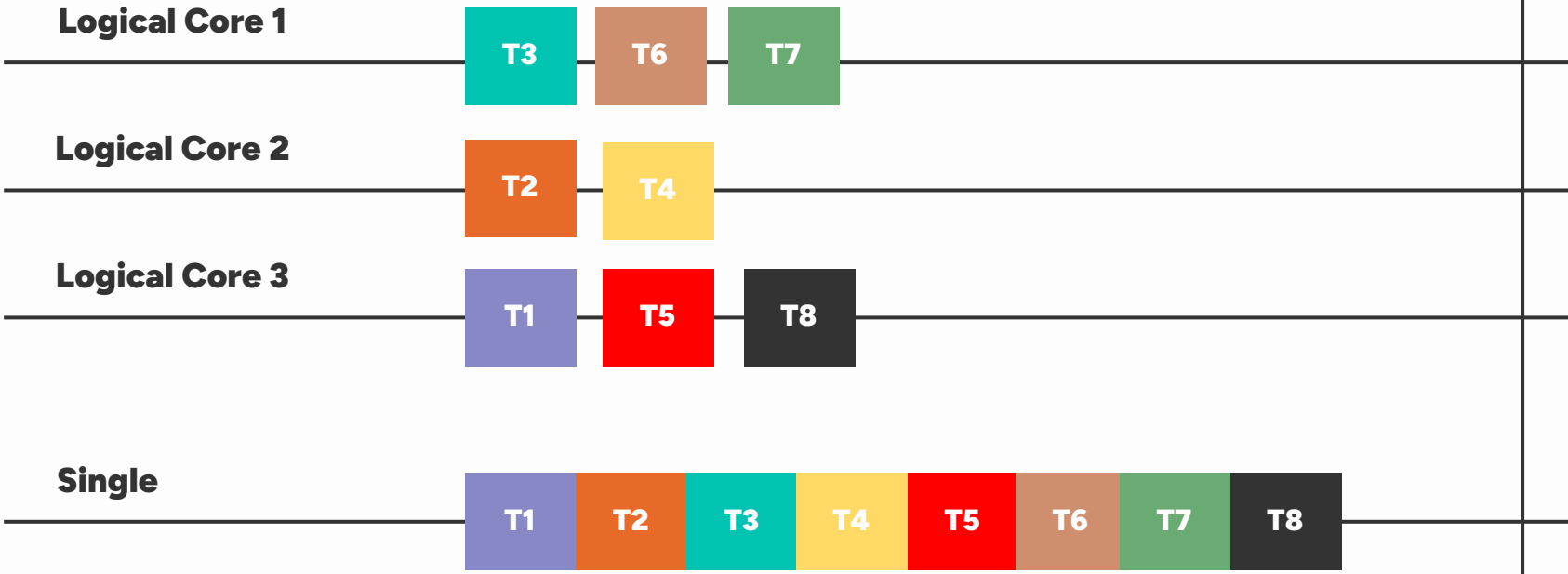
Website Check - Get Text Data

```
24 if __name__=='__main__':
25     start_time = time.time()
26
27     for website in websites:
28         check_website(website)
29
30     end_time = time.time()
31     print(end_time - start_time)
```

Multiprocessing

Actually doing multiple tasks at once

Parallelism using Multiprocessing



Sequential Task

```
1  import time
2
3  def process(number):
4      time.sleep(number)
5      print("Finished")
6
7  if __name__=="__main__":
8      start_time = time.time()
9
10     inputs = [1, 2, 3]
11     outputs = [process(number) for number in inputs]
12
13     end_time = time.time()
14     print(end_time - start_time)
```

Multi-Process Task

```
1  from multiprocessing import Pool
2  import time
3
4  def process(number):
5      time.sleep(number)
6      print("Finished")
7
8  if __name__=="__main__":
9      start_time = time.time()
10
11     inputs = [1, 2, 3]
12     with Pool() as pool:
13         outputs = pool.map(process, inputs)
14
15     end_time = time.time()
16     print(end_time - start_time)
```

H3

Fibonacci Task

Fancy counting done fast

Sequential Fibonacci Calculation

```
1  from multiprocessing import Pool
2  import time
3
4  def fib(n):
5      if n <= 1:
6          return n
7      return fib(n - 1) + fib(n - 2)
8
9  if __name__=="__main__":
10     start_time = time.time()
11
12     inputs = [35, 36, 37, 38]
13     outputs = [fib(number) for number in inputs]
14
15     end_time = time.time()
16     print(end_time - start_time)
```

03

Best Practices

Recommended way to write Python code

Readability

Writing code for people

Example Code No. 1

```
1  def function(ix):
2      ic = {}
3
4      for i in ix:
5
6          if i in ic:
7              ic[i] += 1
8          else:
9              ic[i] = 1
10
11     return ic
```

Example Code 1 (Refactor)

```
1  def count_per_item(items):
2      item_count = {}
3
4      for item in items:
5
6          if item in item_count:
7              item_count[item] += 1
8          else:
9              item_count[item] = 1
10
11     return item_count
```

Example Code No. 2

```
1  class P:
2      def __init__(x,n): x.nm=n
3      def g(x): return"hi "+x.nm
4  class G:
5      def __init__(s,p): s.p=p
6      def sG(s): print(s.p.g())
```

Example Code No. 2 (Refactor)

```
1  class Person:
2      """This class represents a person with a name"""
3      def __init__(self, name):
4          self.name = name
5
6      def greet(self):
7          return "Hi " + self.name
8
9  class ConsoleGreeter:
10     """This wrapper class can print greetings in a terminal"""
11     def __init__(self, person):
12         self.person = person
13
14     def show_greeting(self):
15         print(self.person.greet())
```

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

— Guido van Rossum

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

You aren't gonna need it

Don’t fall into the trap of over engineering for simple features and processes

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

Adding notes for future self and developers

Hallmarks of a Good Comment

Specific

No alternative meaning

Updated

Outdated code is a severe liability

Not Redundant

Remember, DRY

Simple

A new developer should understand it

Context

Provide references and acknowledgement

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

Function Docstrings

```
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
```

Function Docstrings

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

```
help(calculate_circle_area)
```

Class Docstring

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

        Args:
            video (str): Filename of video
        """
        self.video = video
```

Module and __init__ Docstring

```
"""Module for processing common media files"""

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

        Args:
            video (str): Filename of video
        """
        self.video = video
```

Type Hinting

Saving yourself future debugging headaches

Type Hinting (Input)

```
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)

```
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 (Unions)

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

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

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

Variable Type Hinting

```
counter: int = 1
```

```
numbers: list[int] = [1, 2, 3]
```

```
months: dict[str, int] = {"Jan": 1, "Feb": 2, "Mar": 3}
```

```
tasks: dict[str, list[int]] = {"dev": [1, 2, 3], "test": [4]}
```

```
point: tuple[int, int] = (0, 1)
```

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

Type Hinting Examples

```
total_tasks: int = 81

points: list[int] = [1, 2, 3]
priority: tuple[str, str, str] = ("low", "medium", "urgent")

employees: dict[int, str] = dict()
employees.update({9823: "Jay", 1821: "Caroline"})

downtime_logs: list[ dict[str, str] ] = [
    {"Engineering": "Lunch", "Finance": "Team Building"},
    {"Security": "Maintenance"},
    {"Hiring": "Tax Filing", "Engineering": "System Update"},
]
```

Complex Type Hinting

```
UserData = dict[str, str|int|float]

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

Typing Module

The typing module has additional typing and syntax for convenience

```
from typing import Literal, Iterable

priority = Literal["low", "medium", "urgent"]
priorities: list[priority] = ["medium", "urgent", "urgent", "low"]

def urgent_points(items: Iterable) -> int:
    urgent_point: int = 10
    return sum(urgent_point for item in items if item == "urgent")
```

Class Typing: Pen and Paper

```
1  class Paper:
2      def __init__(self):
3          self.content = ""
4  class Pen:
5      def __init__(self, ink_level: int):
6          self.ink_level = ink_level
7
8      def write(self, paper: Paper, text: str):
9          if self.ink_level > 0:
10             paper.content += text
11
12 pen = Pen(100)
13 paper_piece = Paper()
14 pen.write(paper_piece,"Example")
15 print(paper_piece.content)
```

SOLID Principle

Conceptual Discussion on Design Principles

Single Responsibility Rule

A class should have only one reason to change. It should only have one job or responsibility.

```
class User:
    def __init__(self, name):
        self.name = name

    def save(self):
        print(f"Saving {self.name} to database")

    def send_email(self):
        print(f"Sending email to {self.name}")
```

Single Responsibility Rule

A class should have only one reason to change. It should only have one job or responsibility.

```
class User:
    def __init__(self, name):
        self.name = name

class UserRepository:
    def save(self, user):
        print(f"Saving {user.name} to database")

class EmailService:
    def send_email(self, user):
        print(f"Sending email to {user.name}")
```

Open/Closed Principle

Classes (even functions and modules) should be open for extension but closed for modification

```
class AreaCalculator:
    def calculate_area(self, shape):
        if isinstance(shape, Rectangle):
            return shape.width * shape.height
        elif isinstance(shape, Circle):
            return 3.14 * shape.radius ** 2
```

Open/Closed Principle

Classes (even functions and modules) should be open for extension but closed for modification

```
class Rectangle(Shape):
    def __init__(self, width, height):
        self.width = width
        self.height = height
    def area(self):
        return self.width * self.height

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius
    def area(self):
        return 3.14 * self.radius ** 2

class AreaCalculator:
    def calculate_area(self, shape):
        return shape.area()
```

```
class Shape:
    def area(self):
        pass
```

Liskov Substitution Principle

Subclasses must be able to substitute their superclass without issues

```
class Rectangle:
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def set_width(self, width):
        self.width = width

    def set_height(self, height):
        self.height = height

    def get_area(self):
        return self.width * self.height
```

```
class Square(Rectangle):
    def __init__(self, side):
        super().__init__(side, side)

    def set_width(self, width):
        self.width = width
        self.height = width

    def set_height(self, height):
        self.height = height
        self.width = height
```

Liskov Substitution Principle

Subclasses must be able to substitute their superclass without issues

```
class Shape:
    def get_area(self):
        pass
```

```
class Rectangle(Shape):
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def get_area(self):
        return self.width * self.height
```

```
class Square(Shape):
    def __init__(self, side):
        self.side = side

    def get_area(self):
        return self.side * self.side
```

Interface Segregation Principle

Subclasses should not be forced to implement methods it doesn’t need

```
class CoffeeMachine:
    def make_espresso(self): pass
    def make_latte(self): pass
    def make_hot_chocolate(self): pass

class EspressoMachine(CoffeeMachine):
    def make_espresso(self):
        print("Espresso ready!")
    def make_latte(self):
        raise Exception("This machine can't make latte")
    def make_hot_chocolate(self):
        raise Exception("This machine can't make hot chocolate")
```

Interface Segregation Principle

Subclasses should not be forced to implement methods it doesn't need

```
class FancyMachine(
    EspressoMaker,
    LatteMaker,
    HotChocoMaker
):
    def make_espresso(self):
        print("Espresso ready!")
    def make_latte(self):
        print("Latte ready!")
    def make_hot_chocolate(self):
        print("Hot choco ready!")
```

```
class EspressoMaker:
    def make_espresso(self):
        Pass

class LatteMaker:
    def make_latte(self):
        pass

class TeaMaker:
    def make_tea(self):
        pass
```

Dependency Inversion Principle

High-level modules should not depend on low-level modules. Rely on abstractions

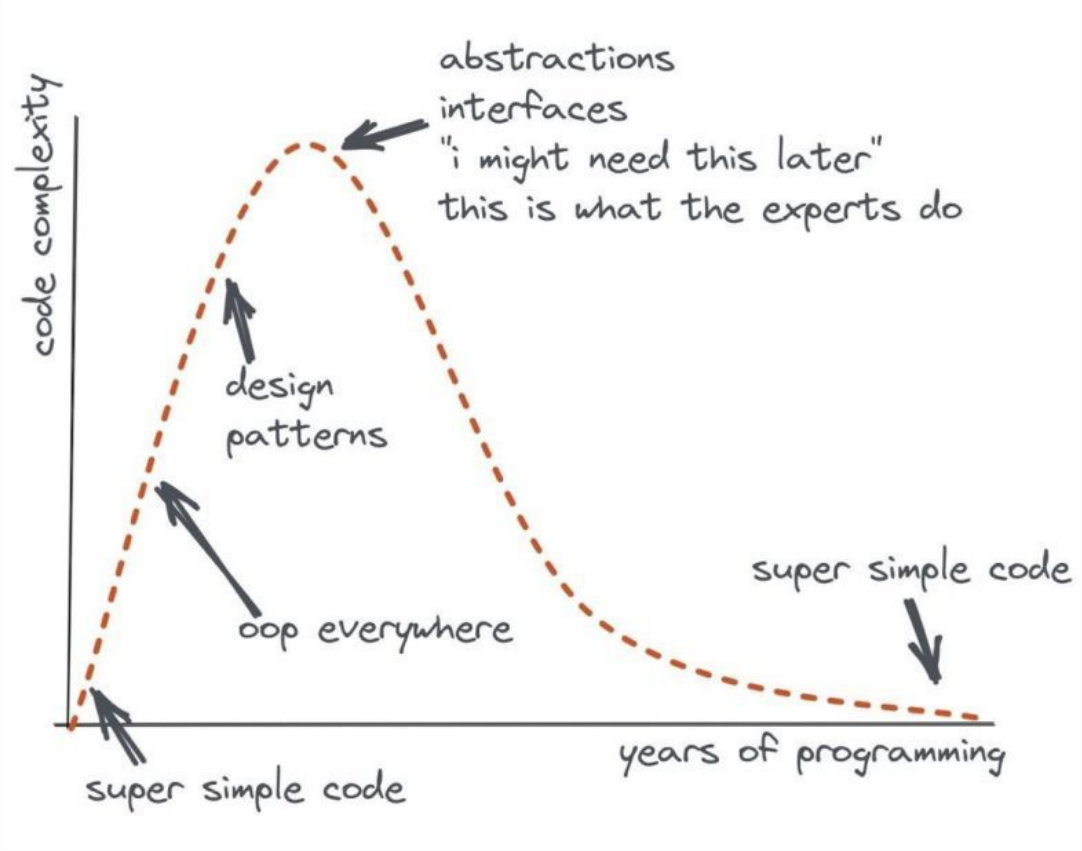
```
class LightBulb:
    def turn_on(self):
        print("Light on")

    def turn_off(self):
        print("Light off")

class LightSwitch:
    def __init__(self, bulb):
        self.bulb = bulb

    def operate(self):
        self.bulb.turn_on()
```

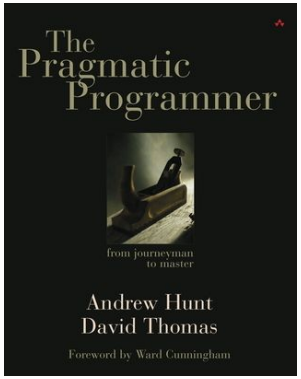
Dependency Inversion Principle

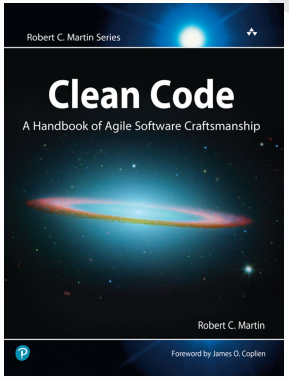
High-level modules should not depend on low-level modules. Rely on abstractions

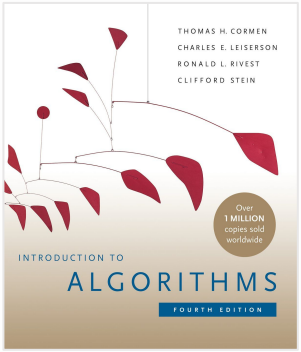
```
class LightSwitch:
    def __init__(self, device):
        self.device = device

    def operate(self):
        self.device.turn_on()
```

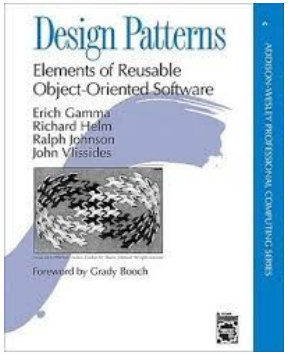
```
class Switchable:
    def turn_on(self):
        pass

    def turn_off(self):
        pass

class LightBulb(Switchable):
    def turn_on(self):
        print("Light on")

    def turn_off(self):
        print("Light off")
```


code complexity
abstractions
interfaces
"i might need this later"
this is what the experts do
design
patterns
oop everywhere
super simple code
super simple code
years of programming


The Pragmatic Programmer
from journeyman to master
Andrew Hunt
David Thomas
Foreword by Ward Cunningham


Robert C. Martin Series
Clean Code
A Handbook of Agile Software Craftsmanship
Robert C. Martin
Foreword by James O. Coplien


THOMAS H. CORMEN
CHARLES E. LEISERSON
RONALD L. RIVEST
CLIFFORD STEIN
Over 1 MILLION copies sold worldwide
INTRODUCTION TO
ALGORITHMS
FOURTH EDITION


Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Testing

Security for your colleagues and future self

Common Types of Testing

Unit

Testing individual parts or functions in isolation

Integration

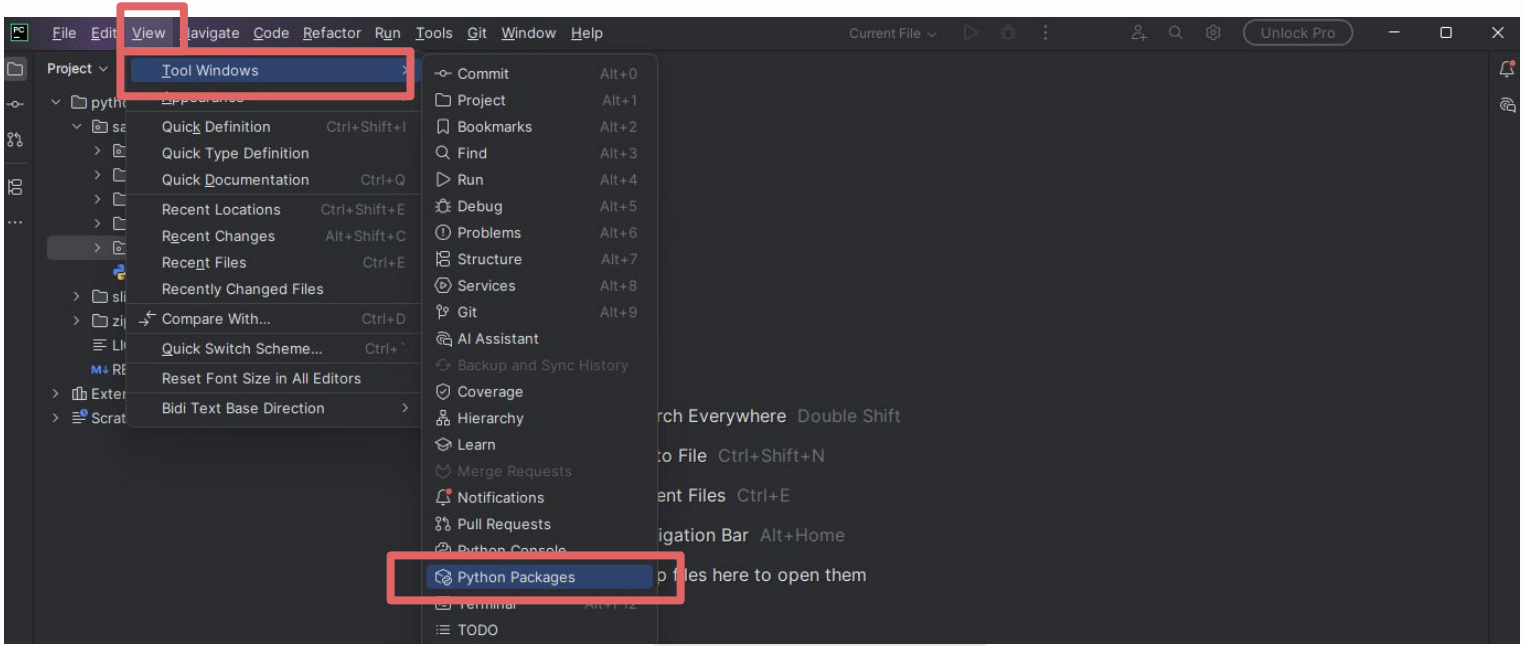
Testing if different components work together correctly

Regression

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

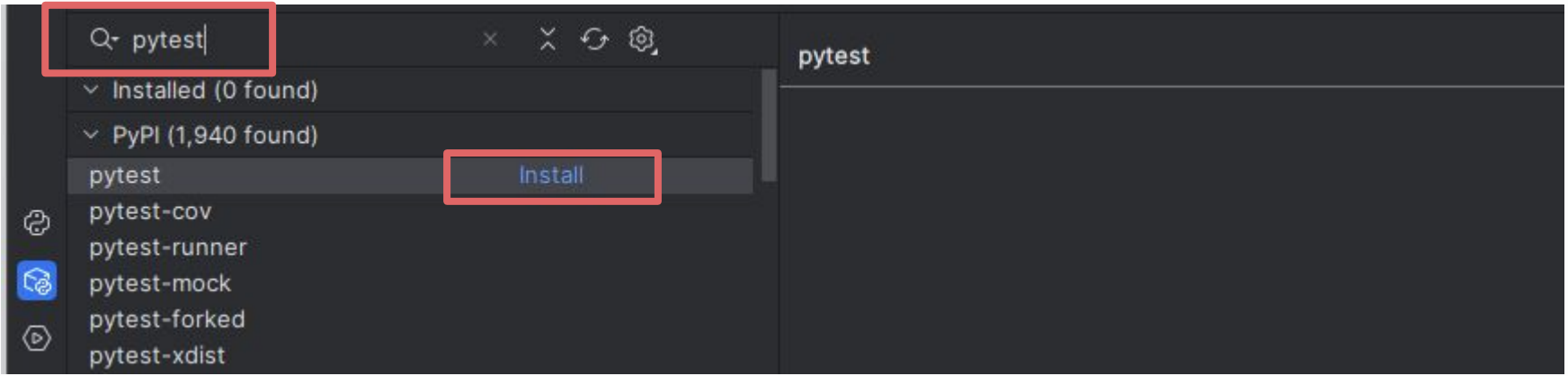
Prerequisite: Python Packages

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



Prerequisite: Download Pytest Packages

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



Unit Test

Testing individual components or functions in isolation from other parts

```
1  def square(x):
2      return x * x
3
4  def test_square():
5      assert square(2) == 4
6      assert square(-3) == 9
7      assert square(0) == 0
8      print("All unit tests passed!")
9
10 test_square()
```

Pytest Classes

Tests can be grouped into classes for further organization

```
1  class TestClass:
2      def test_one(self):
3          word = "this"
4          assert "h" in word
5
6      def test_two(self):
7          word = "hello"
8          assert not hasattr(word, "check")
```

Integration Test

Testing if different components work as intended when combined together

```
1  def add(a, b):
2      return a + b
3
4  def square(x):
5      return x * x
6
7  def multiply(a, b):
8      return a * b
9
```

Integration Test

Testing if different components work as intended when combined together

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

H4

Code Refactor

Improving existing code

inventory_tracker.py

```
def create(inventory, item):
    """Add a new item (dict) to the inventory (list[dict])"""
def read(inventory, index):
    """Return the item (dict) in the given index (int) of inventory"""
def update(inventory, index, detail_key, detail_value):
    """Change/add the key and value to the given index in inventory"""
def delete(inventory, index):
    """Remove item (dict) in the given index (int) of inventory"""
    return 0
def show(inventory):
    """Print the items and their details line-by-line"""
def save(inventory):
    """Print the items and their details line-by-line"""
def load(inventory, filepath):
    """Return a list[dict] from a filepath"""
```

inventory_tracker.py

```
def main():
    current_inventory = []
    command = input("Command: ")

    while command:
        # Handle command here
        # Ask the inputs here, not in the function

        # Ask for more
        command = input("Command: ")

main()
```

04

Web Dev

Interacting with the typical user

Web Frameworks

Flask

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

Django

- Great Object Relational Mapping
- Fully functional Admin Panel
- Built-in Security and Authentication
- **Medium to Large Full-Stack**

Streamlit

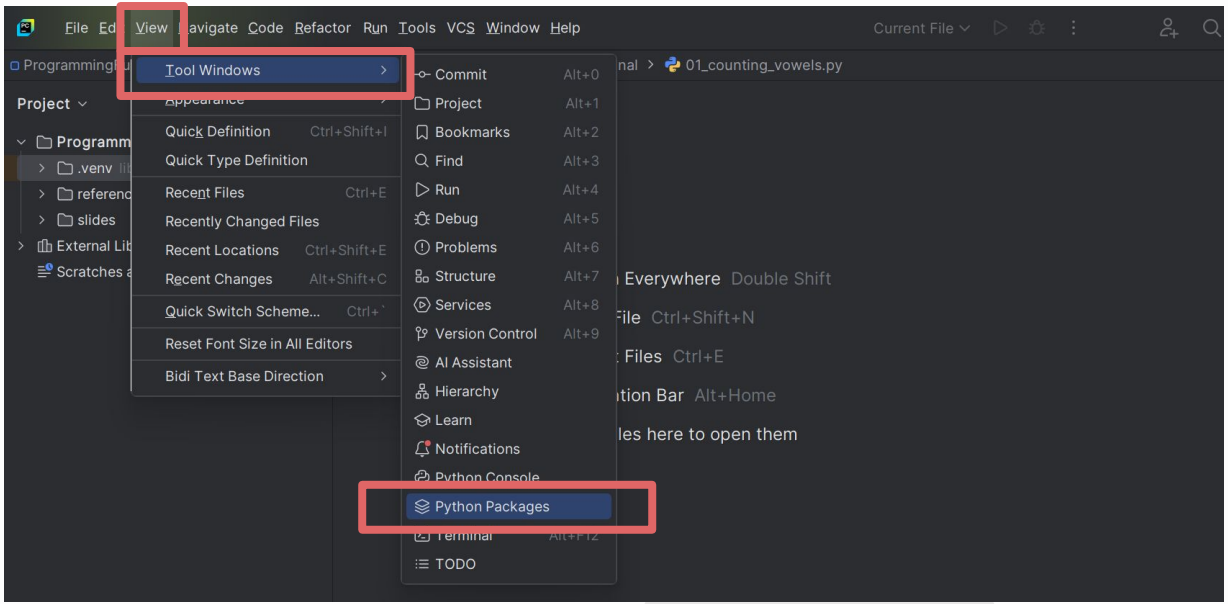
- Very easy syntax
- Built-in Pandas and Plotting Support
- **Small Pages or Data Dashboards**

Fast API

- Minimalist and lightweight
- Automatic documentation
- Built-in Asynchronous Features
- **Very Fast Backend**

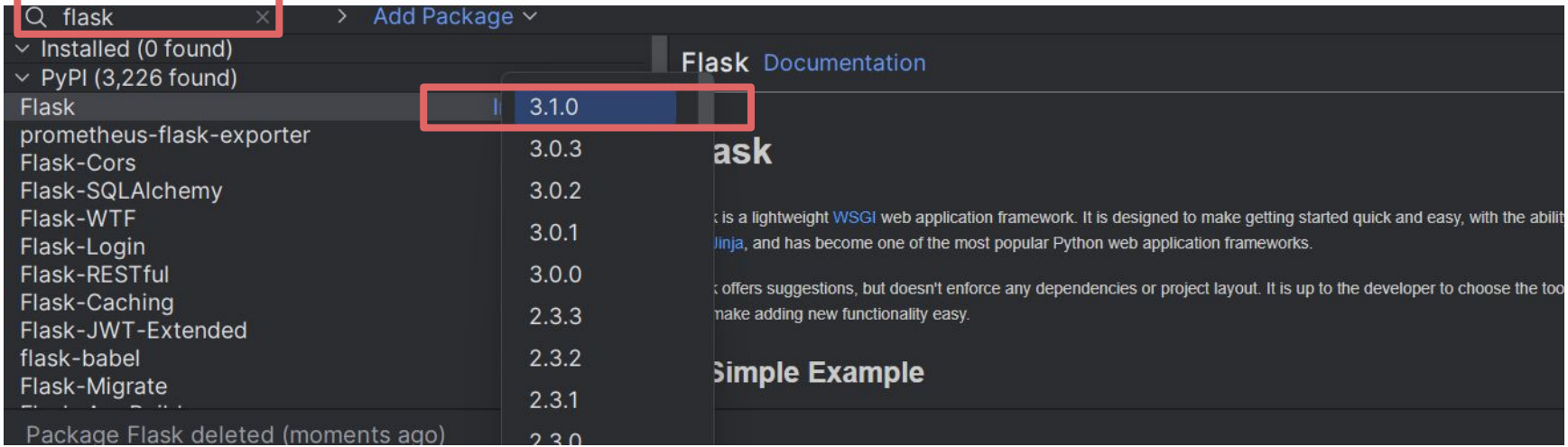
Prerequisite: Python Packages

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



Prerequisite: Download Flask Package

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

```
1  from flask import Flask
2
3  app = Flask(__name__)
4  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
```

```python
from flask import Flask

app = Flask(__name__)

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

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

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

app.run()
```

```python
from flask import Flask

app = Flask(__name__)

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

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

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

app.run()
```

Quick Exercise: Personal Site

Route	Page	Description
/	Landing Page	Introduce yourself
/hobby/	Hobby Page	Enumerate three things you do outside work
/hobbies/		
/opinion/<topic>	Opinion Page	Mention a different statement for specific topics
/opinions/<topic>		
/opinion/food	Food Page	Enumerate your top five favorite food (in order)

HTML

A crash course on organizing 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

```
<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

```
<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 Structure

```
1  <!DOCTYPE html>
2  <html lang="en">
3
4  <head>
5      <meta charset="UTF-8">
6      <title>Website Title Here</title>
7  </head>
8
9  <body>
10     Your content goes here
11 </body>
12
13 </html>
```

CSS

A crash course on organizing text in web pages

CSS: Cascading Style Sheets

It controls how HTML elements look (colors, fonts, spacing, and layout) by applying rules that target tags, classes, or IDs.

```
tag {
    prop: value;
}
```

CSS: Cascading Style Sheets

styles.css

```css
body {
  font-family: sans-serif;
  color: white;
  background: black;
  padding: 2rem;
}
h1, h2 {
  text-decoration: underline;
}
a {
    background: white;
    color: black;
}
```

HTML with CSS

```
1  <!DOCTYPE html>
2  <html lang="en">
3
4  <head>
5      <meta charset="UTF-8">
6      <link rel="stylesheet" href="styles.css">
7      <title>Website Title Here</title>
8  </head>
9
10 <body>
11     Your content goes here
12 </body>
13
14 </html>
```

Templates

Adding placeholders and logic to HTML

Project Structure

```
personal/
    ├── static/
    │   ├── base.css
    │   └── base.js
    ├── templates/
    │   ├── introduction.html
    │   ├── hobby.html
    │   ├── food.html
    │   ├── opinion.html
    │   └── skills.html
    └── main.py
```

Static HTML

./templates/introduction.html

```
1  <h1>Introduction Page</h1>
2  <p>Hello! My name is Jeff Jeff!</p>
3  <ul>
4      <li><a href="/hobby/">Favorite Activities</a></li>
5      <li><a href="/opinion/food">Favorite Food</a></li>
6  </ul>
```

Template Render

main.py

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

Quick Exercise: Personal Site (Update)

Route	Page	Description
/	Landing Page	Introduce yourself *Add links to the hobby and opinion/food page
/hobby/	Hobby Page	Enumerate three things you do outside work
/hobbies/		
/opinion/<topic>	Opinion Page	Mention a different statement for specific topics
/opinions/<topic>		
/opinion/food	Food Page	Enumerate your top five favorite food (in order)

HTML with Loops

./templates/hobbies.html

```
1  <h1>Hobby Page</h1>
2  <p>Here are my favorite activities outside work</p>
3
4  <ul>
5    {% for hobby in hobbies %}
6      <li>Hobby {{loop.index}}: {{ hobby }}</li>
7    {% end for %}
8  </ul>
9
10 <a href="/">Go Back</a>
```

```python
from flask import Flask, render_template

app = Flask(__name__)

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

@app.route("/hobby/")
@app.route("/hobbies/")
def hobby():
    hobbies = ['Play Stardew','Write Essay','Casual Walk']
    return render_template('hobbies.html', hobbies=hobbies)

app.run()
```

Quick Exercise: Personal Site (Update)

Route	Page	Description
/	Landing Page	Introduce yourself *Add links to the hobby and opinion/food page
/hobby/	Hobby Page	Enumerate three things you do outside work
/hobbies/		
/opinion/<topic>	Opinion Page	Mention a different statement for specific topics
/opinions/<topic>		
/opinion/food	Food Page	Enumerate your top five favorite food (in order)

Conditional

./templates/introduction.html

```
1  <h1>Introduction Page</h1>
2  <h2>
3    {% if hour < 12 %}
4      Good morning!
5    {% elif hour < 18 %}
6      Good afternoon!
7    {% else %}
8      Good evening!
9    {% endif %}
10 </h2>
11 <p>My name is Jeff Jeff!</p>
12 <ul>
13     <li><a href="/hobby/">Favorite Activities</a></li>
14     <li><a href="/opinion/food">Favorite Food</a></li>
15 </ul>
```

```python
from flask import Flask, render_template
from datetime import datetime

app = Flask(__name__)

@app.route('/')
def index():
    now = datetime.now()
    return render_template('introduction.html', hour=now.hour)

@app.route("/hobby/")
@app.route("/hobbies/")
def hobby():
    hobbies = ['Play Stardew','Write Essay','Casual Walk']
    return render_template('hobbies.html', hobbies=hobbies)

app.run()
```

Quick Exercise: Personal Site (Update)

Route	Page	Description
/	Landing Page	Introduce yourself *Add links to the hobby and opinion/food page
/hobby/	Hobby Page	Enumerate three things you do outside work
/hobbies/		
/opinion/<topic>	Opinion Page	Mention a different statement for specific topics
/opinions/<topic>		
/opinion/food	Food Page	Enumerate your top five favorite food (in order)

Skills Page

./templates/skills.html

```
...

@app.route("/skills")
def skills():
    skill_levels = {
        "Painting": "Intermediate",
        "Dancing": "Abysmal",
        "Singing": "Poor",
        "Translation": "Proficient",
        "Eating": "Professional"
    }
    return render_template("skills.html", skills=skill_levels)

...
```

Dictionary

./templates/skills.html

```
1  <h1>Skills Page</h1>
2  <ul>
3    {% for skill, level in skills.items() %}
4      <li>
5        {{ skill }} - {{ level }}
6      </li>
7    {% endfor %}
8  </ul>
9
10 <a href="/">Go Back</a>
```

Quick Exercise: Personal Site (Formatting)

Route	Page	Description
/	Landing Page	Introduce yourself *Add links to the hobby and opinion/food page
/hobby/	Hobby Page	Enumerate three things you do outside work
/hobbies/		
/opinion/<topic>	Opinion Page	Mention a different statement for specific topics
/opinions/<topic>		
/opinion/food	Food Page	Enumerate your top five favorite food (in order)
/skills/	Skill Page	Enumerate your skills with years of experience

Templating

Reducing redundancy in html

./templates/base.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <link rel="stylesheet" href="../static/navbar.css">
    <title>{% block title %} My App {% endblock %}</title>
</head>
<body>
    <nav>
        <a href="/">Home</a>
        <a href="/hobbies/">About</a>
        <a href="/opinion/food">Food</a>
    </nav>
    {% block content %} {% endblock %}
</body>
</html>
```

./templates/introduction.html

```
{% extends 'base.html' %}
{% block title %}Introduction{% endblock %}

{% block content %}
<h1>Introduction Page</h1>
<h2>
  {% if hour < 12 %}
    Good morning!
  {% elif hour < 18 %}
    Good afternoon!
  {% else %}
    Good evening!
  {% endif %}
</h2>
<p>My name is Jeff Jeff!</p>
{% endblock %}
```

Quick Exercise: Personal Site

Route	Page	Description
/	Landing Page	Introduce yourself *Add links to the hobby and opinion/food page
/hobby/	Hobby Page	Enumerate three things you do outside work
/hobbies/		
/opinion/<topic>	Opinion Page	Mention a different statement for specific topics
/opinions/<topic>		
/opinion/food	Food Page	Enumerate your top five favorite food (in order)
/skills/	Skill Page	Enumerate your skills with years of experience

Request

Getting data from the user

To-Do List Page

./templates/todo.html

```
<h1>To-Do List</h1>

<form method="POST">
  <input type="text" name="todo" placeholder="New task">
  <button type="submit">Add</button>
</form>

<ul>
  {% for item in todos %}
    <li>{{ item }}</li>
  {% endfor %}
</ul>
```

Request Form

```
from flask import Flask, render_template, request, redirect

app = Flask(__name__)
session = {"todos": []}

@app.get("/todo/")
def show_todo():
    return render_template("index.html", todos=session["todos"])

@app.post("/todo/")
def add_todo():
    if request.form["todo"]:
        session["todos"].append(request.form["todo"])
    return redirect("/todo/")

...
```

Session

```
from flask import Flask, render_template, request, redirect, session

app = Flask(__name__)
app.secret_key = "secret"
...
@app.get("/todo/")
def show_todo():
    if "todos" not in session:
        session["todos"] = []
    return render_template("todo.html", todos=session["todos"])

@app.post("/todo/")
def add_todo():
    if request.form["todo"]:
        session["todos"].append(request.form["todo"])
        session.modified = True
    return redirect("/todo/")
...
```

05

Lab Session

Recommended Next Steps

For more intermediate development, read on the following topics

External Libraries

- **Web Scraping:** Beautiful Soup, Requests, Scrapy
- **Web Development:** Django, FastAPI
- **Data Science:** Sklearn, Pandas, Seaborn

Internal Libraries

- **Refactoring:** functools, Itertools, contextlib
- **File Management:** pathlib, shutil, os, tempfile

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