Python: Day 04

Advanced Programming

Agenda

01

Packaging

Organizing 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 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

```python
def say_hello():
    print("Hello!")

def say_goodbye():
    print("Goodbye")

message = "Hello World"
var1 = "Hello"
var2 = "Hi"

print("Module hello")
```

./example.py

```python
import hello

hello.say_hello()
```

Basic Import

./hello.py

```python
def say_hello():
    print("Hello!")

def say_goodbye():
    print("Goodbye")

message = "Hello World"
var1 = "Hello"
var2 = "Hi"

if __name__=='__main__':
    print("Module hello")

```

./example.py

```python
import hello

hello.say_hello()
```

Specific Import

./hello.py

```python
def say_hello():
    print("Hello!")

def say_goodbye():
    print("Goodbye")

message = "Hello World"
var1 = "Hello"
var2 = "Hi"

if __name__=='__main__':
    print("Module hello")

```

./example.py

```python
import hello

from hello import say_goodbye

hello.say_hello()

say_goodbye()
```

Basic Import with Alias

./hello.py

```python
def say_hello():
    print("Hello!")

def say_goodbye():
    print("Goodbye")

message = "Hello World"
var1 = "Hello"
var2 = "Hi"

if __name__=='__main__':
    print("Module hello")

```

./example.py

```python
import hello
import hello as ho

from hello import say_goodbye

hello.say_hello()

say_goodbye()

ho.say_hello()
```

Multiple Specific Imports

./hello.py

```python
def say_hello():
    print("Hello!")

def say_goodbye():
    print("Goodbye")

message = "Hello World"
var1 = "Hello"
var2 = "Hi"

if __name__=='__main__':
    print("Module hello")

```

./example.py

```python
import hello
import hello as ho

from hello import say_goodbye
from hello import var1, var2

hello.say_hello()

say_goodbye()

ho.say_hello()
print(var1, var2)

```

Basic Nested Import

./package/module_01.py

```python
def say_hello():
     print("Hello!")

def say_goodbye():
     print("Goodbye")

message = "Hello World"
var1 = "Hello"
var2 = "Hi"
```

./nested_example.py

```python
import package.module_01

package.module_01.say_hello()
```

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

```python
def say_hello():
     print("Hello!")

def say_goodbye():
     print("Goodbye")

message = "Hello World"
var1 = "Hello"
var2 = "Hi"
```

./nested_example.py

```python
import package.module_01
import package.module_01 as pm1

from package.module_01 import say_goodbye


package.module_01.say_hello()
say_goodbye()
print(pm1.message)
```

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
```

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

```python
import time

print("Measuring Execution Time:")
print("Current Time:", time.ctime())
time.sleep(10)
print("Current Time:", time.ctime())
print()

print("Measuring Execution Time:")
start_time = time.time()
for _ in range(1_000_000):
    x = 10 ** 1000
end_time = time.time()
print(f"Spent {end_time - start_time:.5f} seconds")
```

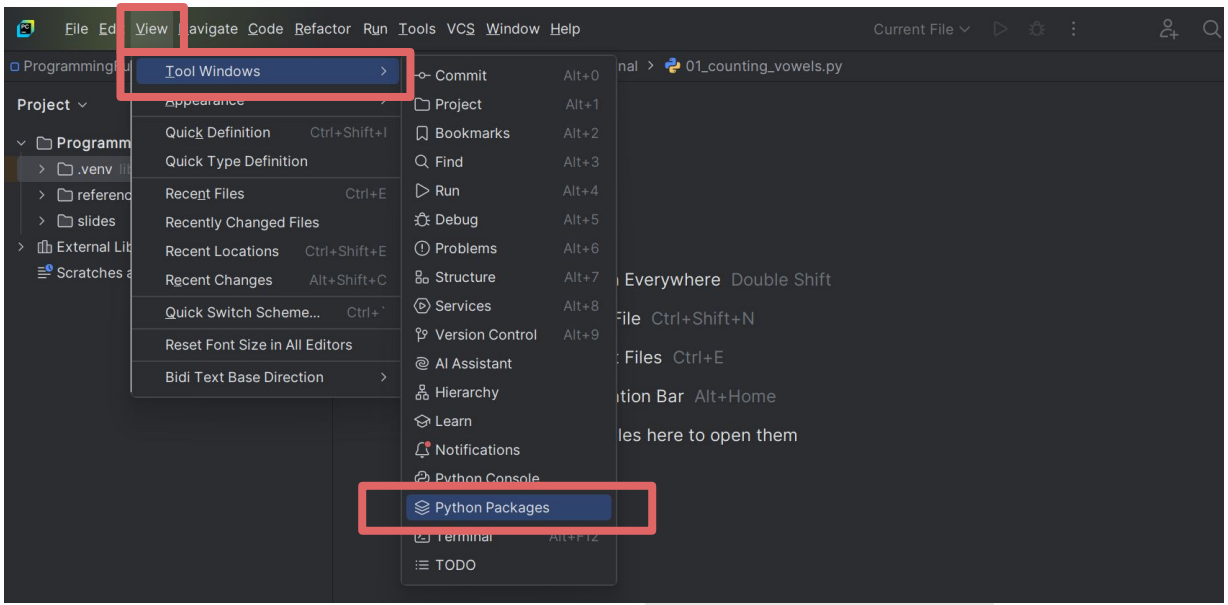
Functools Demo

```python
def fib(n):
    if n <= 1:
        return n
    return fib(n-1) + fib(n-2)

print(fib(38))
```

```python
from functools import cache

@cache
def fib(n):
    if n <= 1:
        return n
    return fib(n-1) + fib(n-2)

print(fib(300))
```

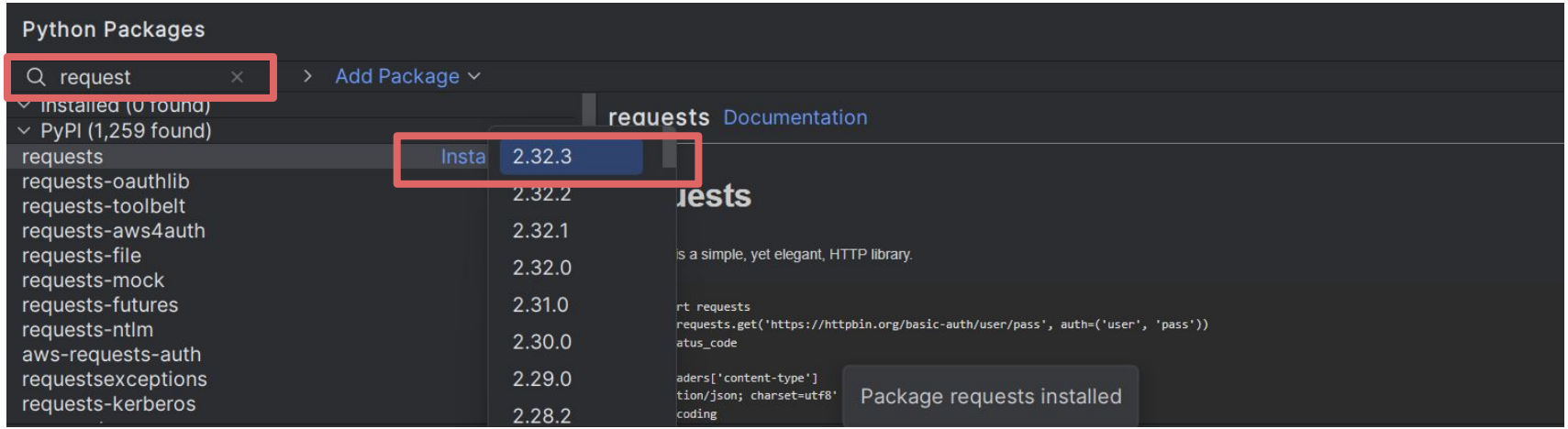
Prerequisite: Python Packages

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

```python
import requests

response = requests.get("https://open.er-api.com/v6/latest/USD")

# Get the latest conversion rate from USD to PHP
print()
```

02

Multiple Tasks

A preview of Multiprocessing and Multithreading

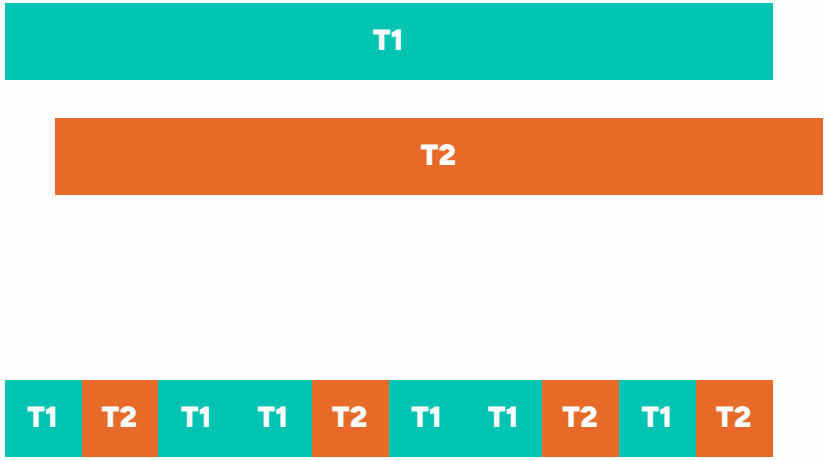
Parallelism versus Concurrency

Parallel Process

Tasks running simultaneously
or at the same time

Concurrent Process

Switching between tasks
when waiting for results



Concurrency

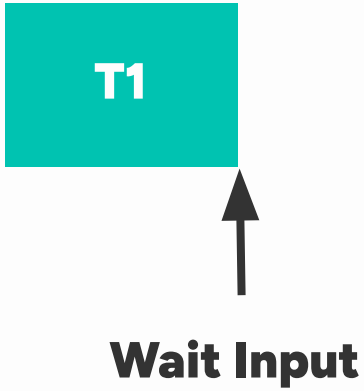
Working while waiting for other tasks

Concurrent Process

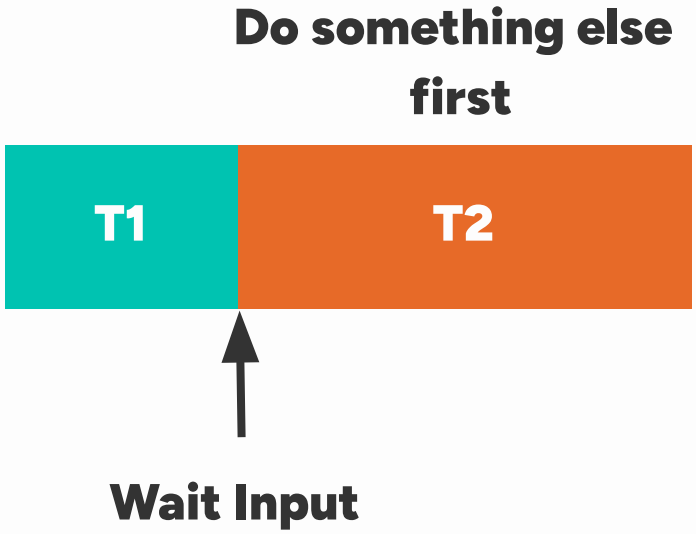
Current Task

T1

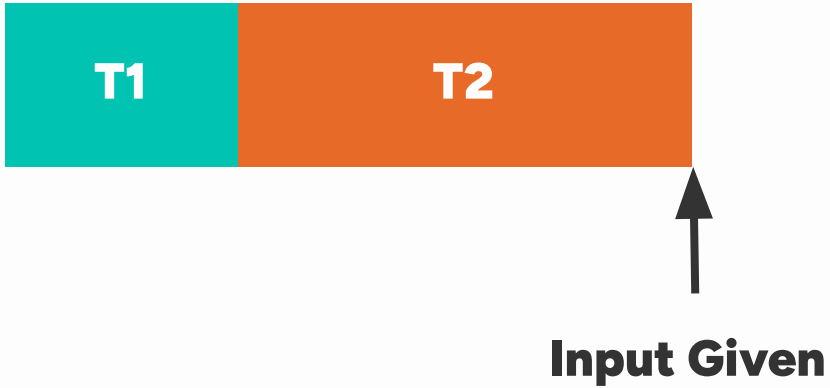
Concurrent Process



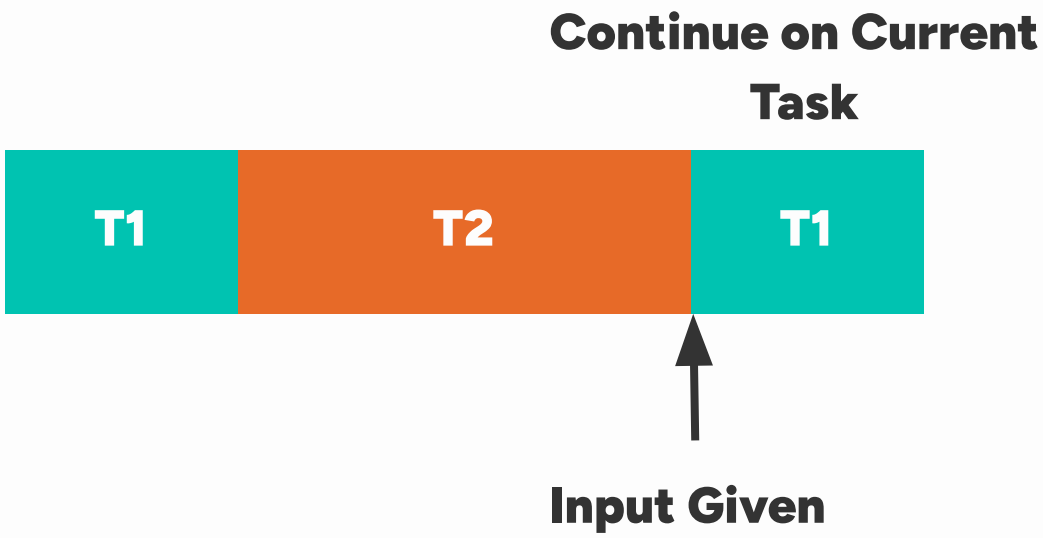
Concurrent Process



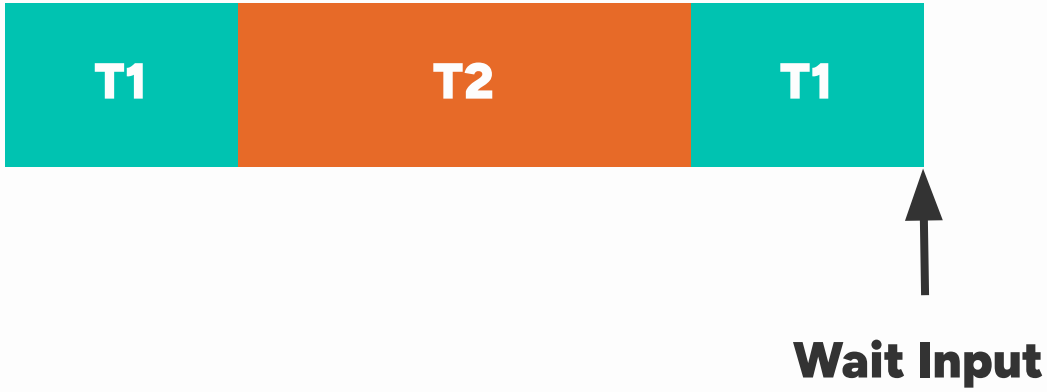
Concurrent Process



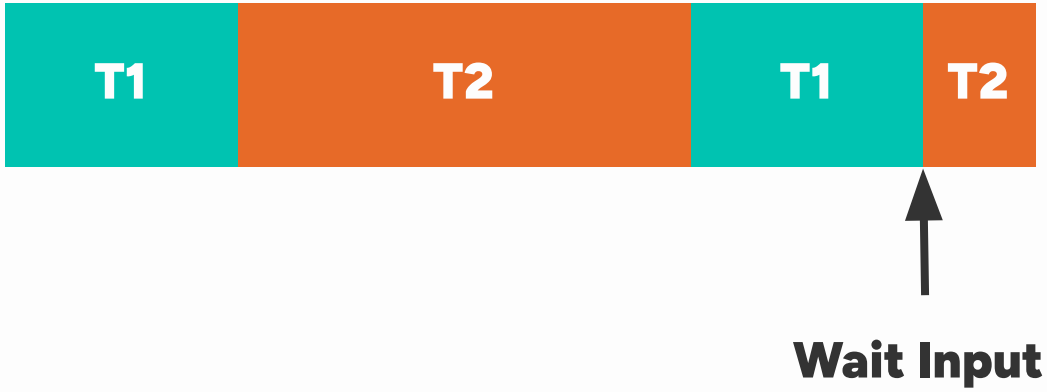
Concurrent Process



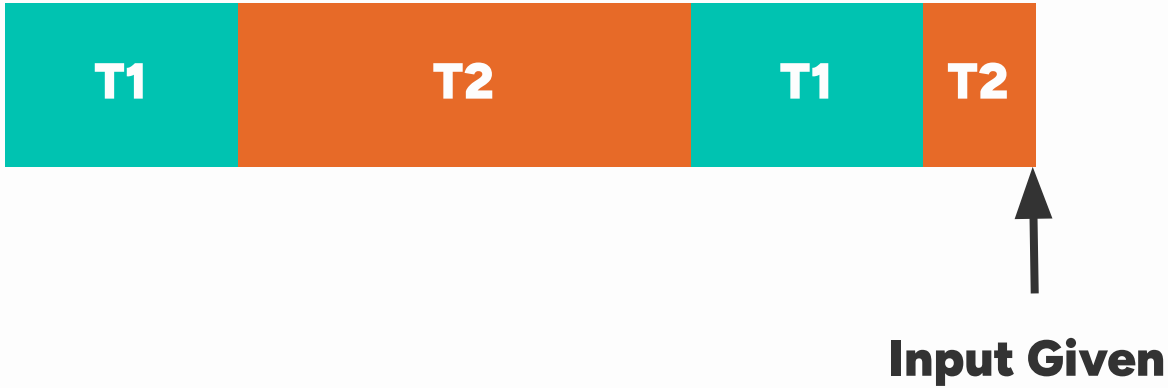
Concurrent Process



Concurrent Process



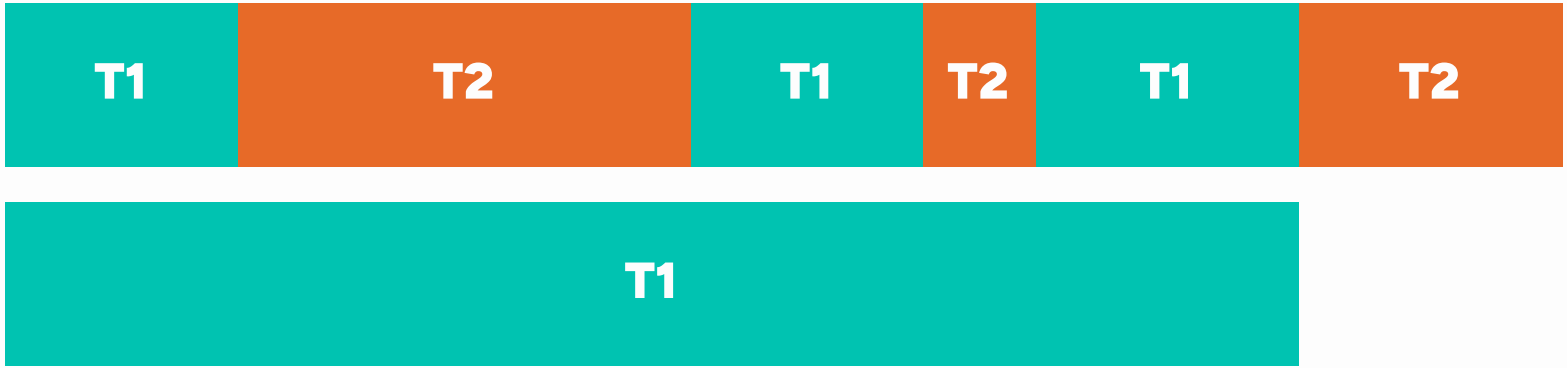
Concurrent Process



Concurrent Process

Concurrent Process

Concurrent Process



Thread Pool Submission

```
1  from concurrent.futures import ThreadPoolExecutor
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
11     with ThreadPoolExecutor() as executor:
12         x = executor.submit(process, 3)
13         y = executor.submit(input, 'Enter number: ')
14
15     end_time = time.time()
16     print(end_time - start_time)
```

Thread Pool Mapping

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

def fetch_url(url):
    return requests.get(url).status_code

inputs = ['https://httpbin.org/delay/5', 'https://httpbin.org/delay/7']

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

    with ThreadPoolExecutor() as executor:
        outputs = executor.map(fetch_url, inputs)

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

H2

Website Check

Check multiple websites if they are working

Website Check - Main Function

```python
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.")
```

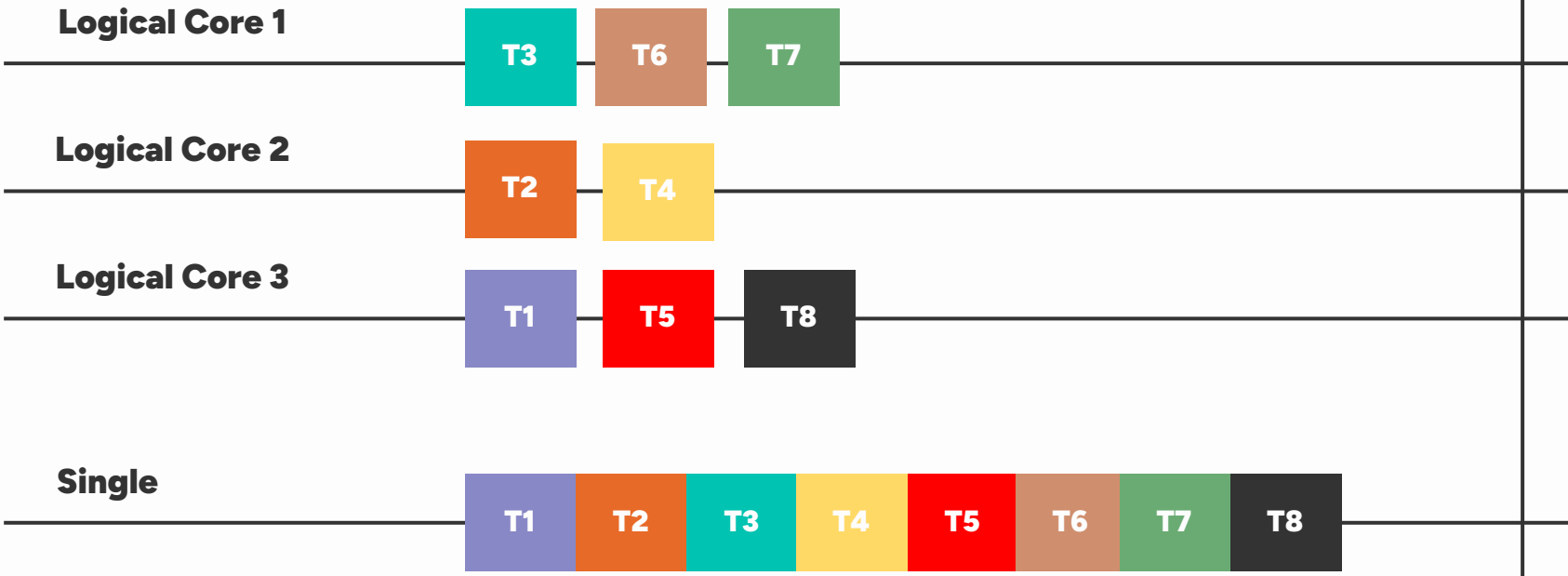
Website Check - Get Text Data

```
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 = [site.strip() for site in websites if site.strip()]
21
22 if __name__=='__main__':
23     start_time = time.time()
24
25     for website in websites:
26         check_website(website)
27
28     end_time = time.time()
29     print(end_time - start_time)
```

Multiprocessing

Actually doing multiple tasks at once

Parallelism using Multiprocessing



Sequential Task

```python
import time

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

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

    inputs = [1, 2, 3]
    outputs = [process(number) for number in inputs]

    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()

    inputs = [1, 2, 3]
    with Pool() as pool:
        outputs = pool.map(process, inputs)

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

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

    inputs = [35, 36, 37, 38]
    outputs = [fib(number) for number in inputs]

    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(ix):
    ic = {}

    for i in ix:

        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
```

Example Code No. 3

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

Example Code No. 3 (Refactor)

```
class Person:
    """This class represents a person with a name"""
    def __init__(self, name):
        self.name = name

    def greet(self):
        return "Hi " + self.name

class ConsoleGreeter:
    """This wrapper class can print greetings in a terminal"""
    def __init__(self, person):
        self.person = person

    def show_greeting(self):
        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

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

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

Specific Comment

```
# Process data
process(data)
```

```
# Filter out inactive users before processing
process(active_users_only(data))
```

Updated Comment

```
# Multiply score by 2
score = score + 10
```

```
# Add 10 to the score as a bonus
score = score + 10
```

Non-redundant Comment

```
# This is a for loop
for i in range(10):
    print(i)
```

```
# Check if stack is compromised
for i in range(10):
    print(i)
```

```
# Get user name
name = user.get_name()
# Get user age
age = user.get_age()
# Get user email
email = user.get_email()
```

```
# Extract basic user details
name = user.get_name()
age = user.get_age()
email = user.get_email()
```

Simple Comment

```
# Initialize a numeric accumulator
# that begins with the additive identity element
count = 0
```

```
# Positive counter
count = 0
```

Comment with Context

```
# 86400 is the number of seconds in a day
expiration_time = 86400
```

```
# Set expiration to 1 day (86400 seconds)
# Based on OAuth2 token policy:
# See https://oauth.net/2/access-tokens/
expiration_time = 86400
```

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)

```python
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)

```python
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)

```python
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

```python
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"},
]
```

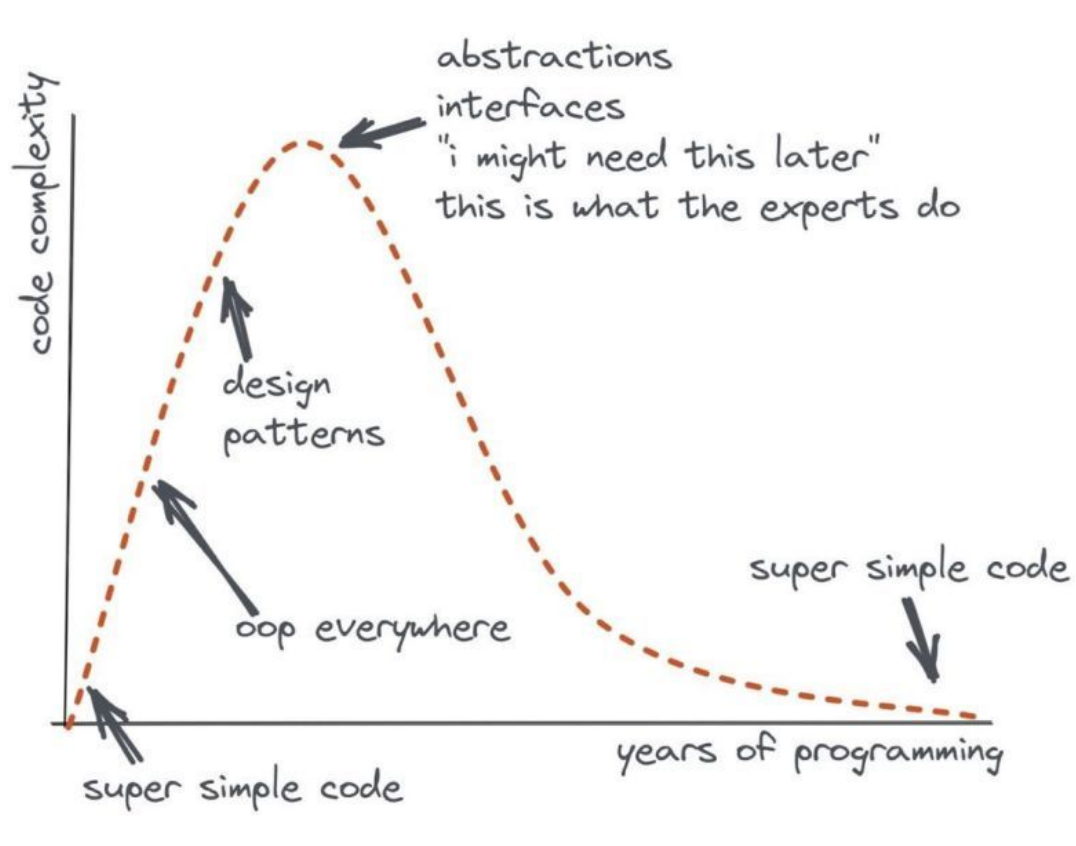
Typing Module

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

```python
class Paper:
    def __init__(self):
        self.content = ""
class Pen:
    def __init__(self, ink_level: int):
        self.ink_level = ink_level

    def write(self, paper: Paper, text: str):
        if self.ink_level > 0:
            paper.content += text

pen = Pen(100)
paper_piece = Paper()
pen.write(paper_piece,"Example")
print(paper_piece.content)
```



Discussion On Tech Debt

H4

Code Review

Let’s see how you have been coding

Testing

Security for your colleagues and future self

Common Types of Testing

Unit

Testing individual parts or functions in isolation

Integration

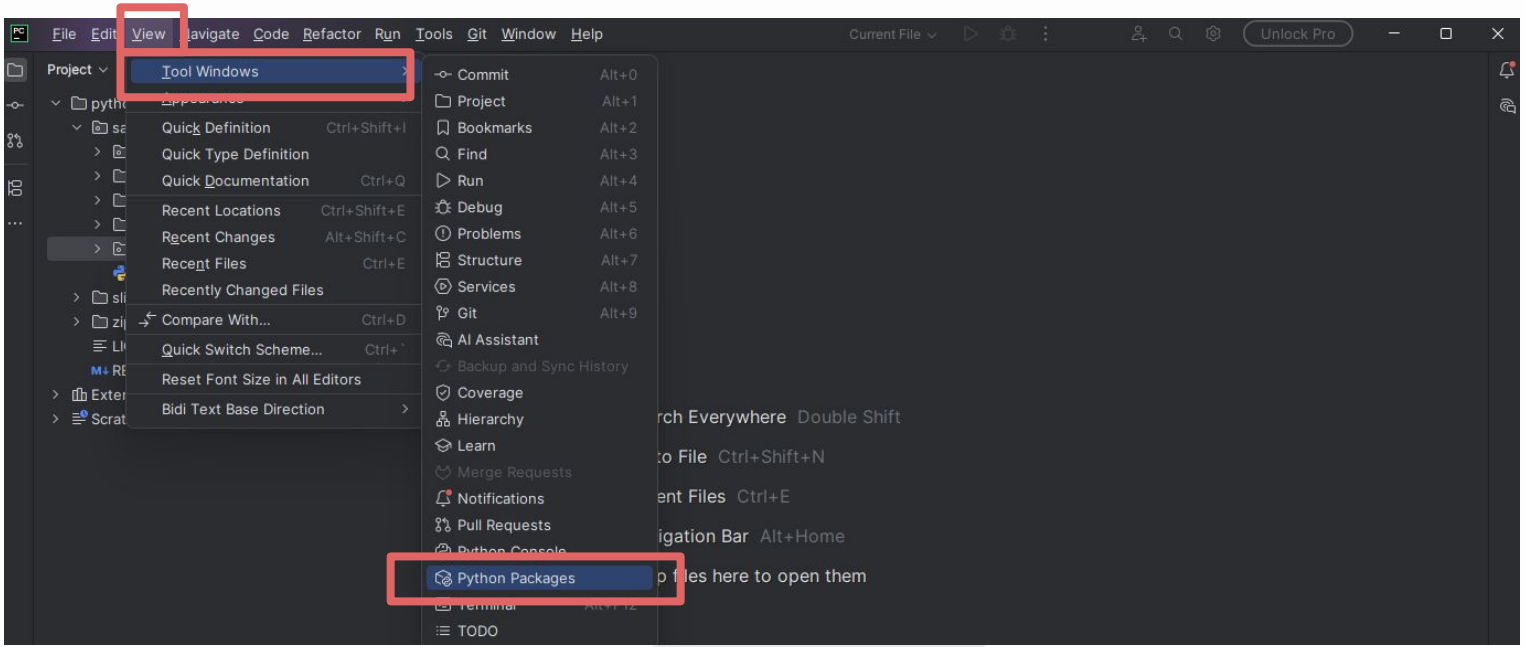
Testing if different components work together correctly

Regression

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

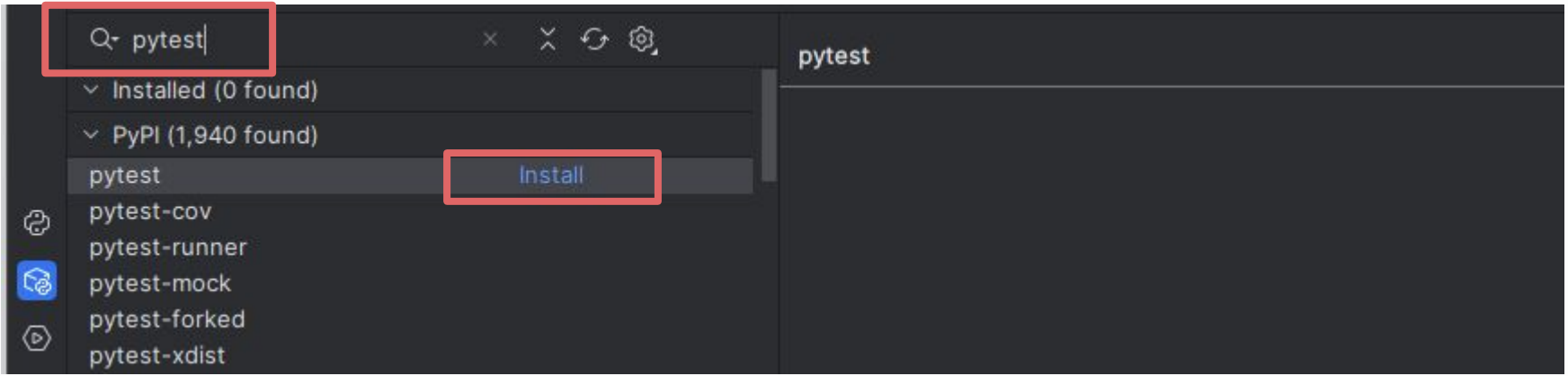
Prerequisite: Python Packages

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

```
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

```
def calculate_expression(x, y, z=0):
    return add(square(x), multiply(y, 2)) - z

def test_calculate_expression():
    assert calculate_expression(2, 3) == 10
    assert calculate_expression(0, 5) == 10
    assert calculate_expression(2, 3, 2) == 10
    print("All integration tests passed!")

test_calculate_expression()
```

Pytest Classes

Tests can be grouped into classes for further organization

```
class TestClass:
    def test_one(self):
        word = "this"
        assert "h" in word

    def test_two(self):
        word = "hello"
        assert not hasattr(word, "check")
```

H5

Test-Driven

A surprising amount of time is invested here

Reverse String

Student A

Write tests for a function reverse_string(s) that returns the reversed version of a string. Consider special cases such as the following:

- Normal string (e.g. "hello" → "olleh")
- Empty string
- One-character string
- Palindrome string

Student B

Implement the function `reverse_string(s)` so it passes all the tests.

Grocery List

Student A

Write tests for a class `GroceryList` that should:

- Add tasks with `add_task(task: str)`
- Save tasks to a file with `save(filepath: str)`

Test if:

- The file is created
- It contains all added tasks

Student B

Implement the class `GroceryList` so it passes all the tests.

Extract Date

Student A

Write tests for a function `extract_date(date_str)` that:

- Parses valid YYYY-MM-DD strings and returns a tuple containing the year, month, and date (in that order)
- Returns None for invalid dates or formats (e.g. "2025-13-99" or "hello")

Student B

Implement the function `extract_date(date_str)` so it passes all the tests.

04

Web Dev

Providing online access to your business logic

Web Frameworks

Flask

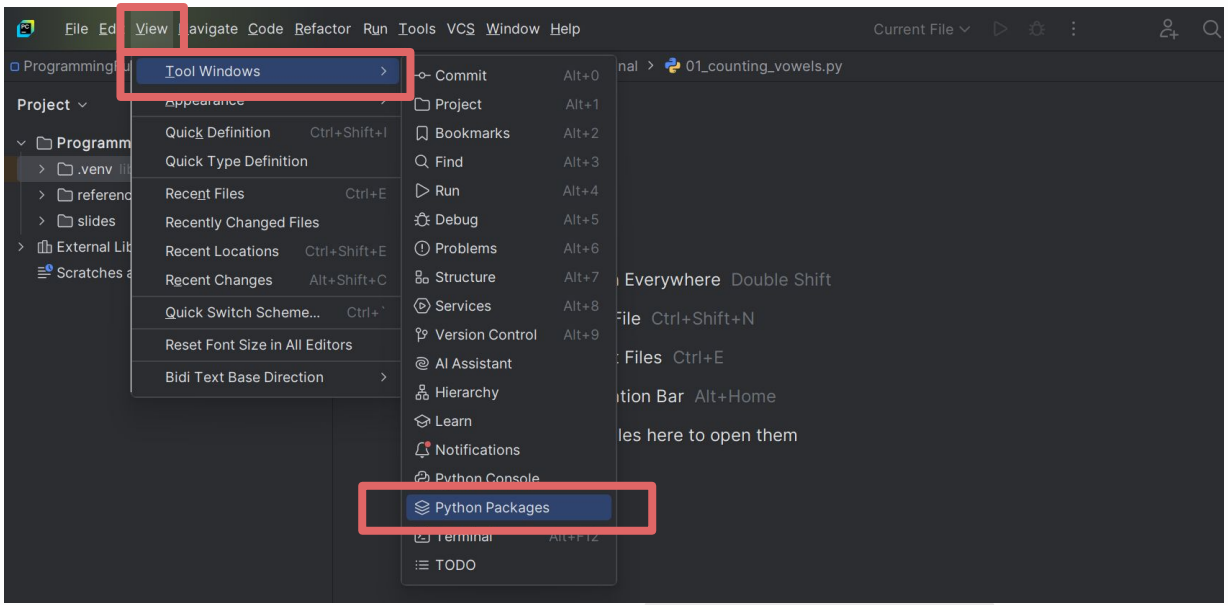
- Minimalist and lightweight
- Freedom to choose tools for each part
- **Small and Fast Web Applications**

Django

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

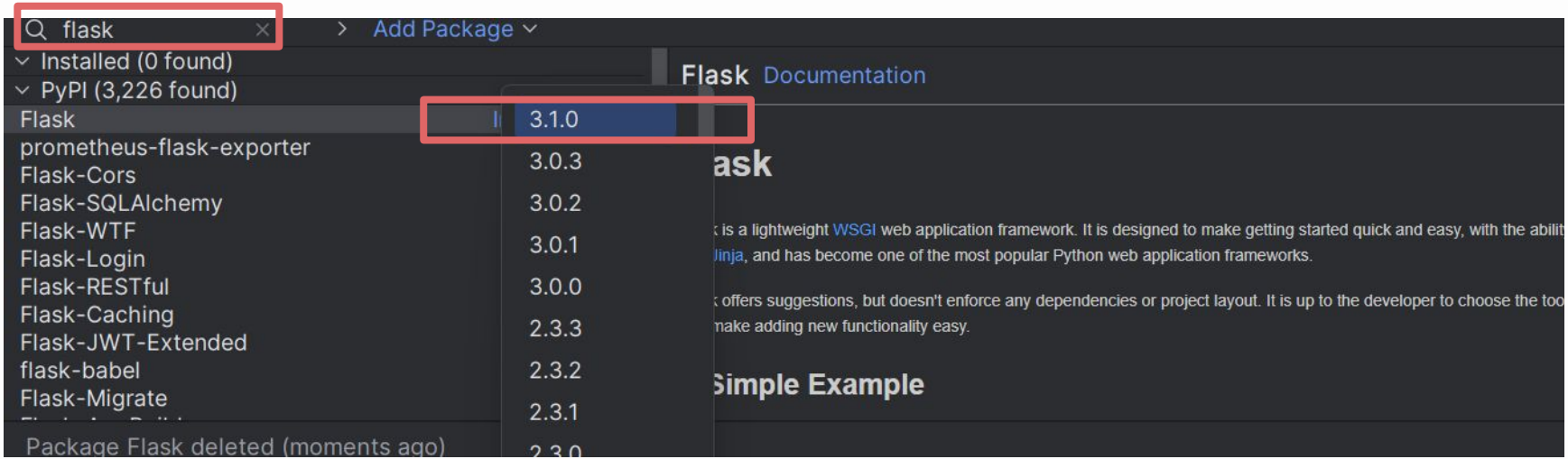
Prerequisite: Python Packages

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

```python
from flask import Flask

app = Flask(__name__)

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

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

app.run()
```

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

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

app.run()
```

Dynamic Route

```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()
```

Full Dynamic Route

```python
from flask import Flask

app = Flask(__name__)

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

@app.route("/profile/")
@app.route("/profiles/")
@app.route("/profile/<username>")
@app.route("/profiles/<username>")
def profile_dynamic(username=None):
    if username:
        return f"Profile {username}"
    else:
        return "Profile Page"

app.run()
```

Quick Exercise: Personal Page

/	Introduction to yourself
/hobby/ /hobbies/ /interest/ /interests	Hobbies and interests
/opinion/ <topic> /opinions/ <topic>	Your opinions on certain topics

Challenge: Random Jokes

/joke/
/jokes/

→

Random Joke from
https://official-joke-api.appspot.com/random_joke

/joke/<topic>
/jokes/<topic>

→

Random Joke Base
https://official-joke-api.appspot.com/jokes/

General: general/random
Programming: programming/random
Knock-Knock: knock-knock/random

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

```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

Div

The ** ** tag is commonly used to group related parts together

```html
<div>
    <h1>Welcome to Flask</h1>
    <p>This is a simple example of HTML in Flask</p>
    <a href="https://flask.palletsprojects.com/">Guide</a>
</div>
<div>
    <ol>
        <li>Learn Flask</li>
        <li>Build a project</li>
    </ol>
</div>
```

Quick Exercise: Coffee Lover

Coffee Lover's Guide

Welcome to your simple guide to enjoying coffee like a true enthusiast.

Top Coffee Drinks

- Espresso
- Cappuccino
- Latte
- Americano

Steps to Brew the Perfect Cup

1. Choose quality beans
2. Grind just before brewing
3. Use clean, filtered water
4. Brew at the right temperature
5. Enjoy immediately

More Coffee Resources

Coffee on Wikipedia

Templates

Adding placeholders and logic to HTML

Project Structure

```
flask_app/
    ├── static/
    │    ├── base.css
    │    └── base.js
    ├── templates/
    │    ├── base.html
    │    ├── dashboard.html
    │    ├── menu.html
    │    ├── order_summary.html
    │    └── orders.html
    └── main.py
```

Static Template

./templates/dashboard.html

```
<h1>Welcome, Admin</h1>
<p>Use the links below to manage the platform:</p>
<ul>
  <li><a href="/orders/metro">View Metro Orders</a></li>
  <li><a href="/menu">View Menu</a></li>
  <li><a href="/order_summary">Sample Order Summary</a></li>
</ul>
```

Render Template

```python
from flask import Flask, render_template

app = Flask(__name__)

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

@app.route("/orders/<area>")
def orders(area):
    return render_template("orders.html", area=area)

app.run()
```

Variable + Conditional

./templates/orders.html

```
1  <h1>Orders for: {{ area }}</h1>
2
3  {% if area == "metro" %}
4      <p>These are high-priority city orders.</p>
5  {% elif area == "rural" %}
6      <p>Allow extra time for rural delivery.</p>
7  {% else %}
8      <p>Unknown delivery zone.</p>
9  {% endif %}
```

Render Template with Conditionals

```
1  from flask import Flask, render_template
2
3  app = Flask(__name__)
4
5  @app.route('/')
6  def index():
7      return render_template('dashboard.html')
8
9  @app.route("/orders/<area>")
10 def orders(area):
11     return render_template("orders.html", area=area)
12
13 app.run()
```

Loops

./templates/menu.html

```
1 <h1>Current Menu</h1>
2 <ul>
3   {% for item in items %}
4     <li>{{ item }}</li>
5   {% end for %}
6 </ul>
```

Render Template with Lists

```python
from flask import Flask, render_template

app = Flask(__name__)

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

@app.route("/orders/<area>")
def orders(area):
    return render_template("orders.html", area=area)

@app.route("/menu")
def menu():
    items = ["Adobo", "Tapsilog", "Spaghetti", "Burger", "Chicken"]
    return render_template("menu.html", items=items)

app.run()
```

Dictionary

./templates/order_summary.html

```
<h1>Order Summary</h1>

<p><strong>Customer:</strong> {{ order.customer }}</p>
<p><strong>Address:</strong> {{ order.address }}</p>

<h3>Items Ordered:</h3>
<ul>
  {% for item in order.choices %}
    <li>{{ item }}</li>
  {% endfor %}
</ul>

{% if order.special_notes %}
  <p><strong>Notes:</strong> {{ order.special_notes }}</p>
{% else %}
  <p>No special instructions.</p>
{% endif %}
```

Render Template with Dictionary

```
@app.route("/order_summary")
def order_summary():
    order = {
        "customer": "Juan Dela Cruz",
        "address": "Manila, Metro",
        "choices": ["Tapsilog", "Fried Chicken", "Extra Rice"],
        "special_notes": "Allergic to peanuts"
    }
    return render_template("order_summary.html", order=order)

app.run()
```

Quick Exercise: Menu Details

/menu/adobo	→	Show name, picture, and description
/menu/tapsilog	→	Show name, picture, and description
/menu/spaghetti	→	Show name, picture, and description
/menu/burger	→	Show name, picture, and description
/menu/chicken	→	Show name, picture, and description

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 %}
```

Project Structure

```
flask_app/
    ├── static/
    │    ├── base.css
    │    └── base.js
    ├── templates/
    │    ├── base.html
    │    ├── dashboard.html
    │    ├── menu.html
    │    ├── order_summary.html
    │    └── orders.html
    └── main.py
```

Quick Exercise: Menu Details (v2)

/menu/adobo	→	Show name, picture, and description
/menu/tapsilog	→	Show name, picture, and description
/menu/spaghetti	→	Show name, picture, and description
/menu/burger	→	Show name, picture, and description
/menu/chicken	→	Show name, picture, and description

URL Handling

Special cases for handling subpages

Redirect URL

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

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

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

app.run()
```

Abort Error

```
1  from flask import Flask, redirect, abort
2  app = Flask(__name__)
3
4  @app.route("/user/<username>")
5  def profile(username):
6      if username != "admin":
7          return redirect('/login')
8      return "Welcome Admin"
9
10 @app.route('/login')
11 def login():
12     abort(501)
13
14 app.run()
```

Error Handler

```
1  from flask import Flask, redirect, abort
2  app = Flask(__name__)
3
4  @app.route("/user/<username>")
5  def profile(username):
6      if username != "admin":
7          return redirect('/login')
8      return "Welcome Admin"
9
10 @app.route('/login')
11 def login():
12     abort(501)
13
14 @app.errorhandler(501)
15 def handle_501_error(error):
16     return "Undetected visitor"
17
18 app.run()
```

Quick Exercise: Menu Details (v3)

/menu/adobo	→	Show name, picture, and description
/menu/tapsilog	→	Show name, picture, and description
/menu/spaghetti	→	Show name, picture, and description
/menu/burger	→	Show name, picture, and description
/menu/chicken	→	Show name, picture, and description
/menu/?	→	"Coming Soon"

Sessions

Server-side data storage

To-Do List Page

./templates/index.html

```
1  <h1>To-Do List</h1>
2
3  <form method="POST">
4    <input type="text" name="todo" placeholder="New task">
5    <button type="submit">Add</button>
6  </form>
7
8  <ul>
9    {% for item in todos %}
10     <li>{{ item }}</li>
11   {% endfor %}
12 </ul>
```

Session Setup

```
1  from flask import Flask
2
3  app = Flask(__name__)
4  app.secret_key = "secret"
```

Get Data

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

app = Flask(__name__)
app.secret_key = "secret"

@app.get("/")
def show_todos():
    if "todos" not in session:
        session["todos"] = []
    return render_template("index.html", todos=session["todos"])

app.run()
```

Post Data

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

app = Flask(__name__)
app.secret_key = "secret"

@app.get("/")
def show_todo():
    if "todos" not in session:
        session["todos"] = []
    return render_template("index.html", todos=session["todos"])
@app.post("/")
def add_todo():
    if request.form["todo"]:
        session["todos"].append(request.form["todo"])
        session.modified = True
    return redirect("/")

app.run()
```

To-Do List Page

./templates/index_complete.html

```
1  <h1>To-Do List</h1>
2
3  <form method="POST" action="/">
4    <input type="text" name="todo" placeholder="New task">
5    <button type="submit">Add</button>
6  </form>
7
8  <ul>
9    {% for item in todos %}
10     <li>
11       <form method="POST" action="/delete/item/" style="display:inline;">
12         <input type="hidden" name="todo" value="{{ item }}">
13         <input type="checkbox" onChange="this.form.submit()"> {{ item }}
14       </form>
15     </li>
16   {% endfor %}
17 </ul>
```

Delete Data

```
18  @app.post("/delete/item/")
19  def delete_item():
20      todo = request.form["todo"]
21      if todo in session.get("todos", []):
22          session["todos"].remove(todo)
23          session.modified = True
24      return redirect("/")
25
26  app.run()
```

Quick Exercise: Menu Details (final)

/cook/	Ask user for name, picture URL, and description
/menu/tapsilog	Show name, picture, and description
/menu/spaghetti	Show name, picture, and description
/menu/burger	Show name, picture, and description
/menu/chicken	Show name, picture, and description
/menu/?	“Coming Soon” and link to /cook/

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