Python: Day 04

Advanced Programming

Previous Agenda

01

Definition

Data-Centric Approach

02

Relationship

Code Reuse

03

Structure

Code Architecture

04

GUI

Introduction to Tkinter

05

Lab Session

Culminating Exercise

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

```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

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

def say_goodbye():
      print("Goodbye")

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

./nested_example.py

```
import package.module_01

from package.module_01 import say_goodbye


package.module_01.say_hello()
say_goodbye()
```

Specific Nested Import

./package/module_01.py

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

def say_goodbye():
      print("Goodbye")

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

./nested_example.py

```
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/
```

Relative Imports

./character.py

```python
class Character:
    pass
```

./knight.py

```python
from .character import Character

class Knight:
    pass
```

./main.py

```python
from rpg_character.knight import Knight
```

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

Quick Exercise: Organize RPG

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

Python STL

Python Standard Library

Try these Libraries!

Math

Common math constants and operations

Functools

Module for higher-order functions

Collections

Additional data structures

CProfile

Useful for optimizing execution time

Request

Access data from online servers and sites

Itertools

Efficient looping and combinatorials

CProfile Demo

```python
import cProfile

def main():
    for _ in range(1_000_000):
        x = 10 ** 1000

if __name__=='__main__':
    cProfile.run("main()")
```

Slow Function

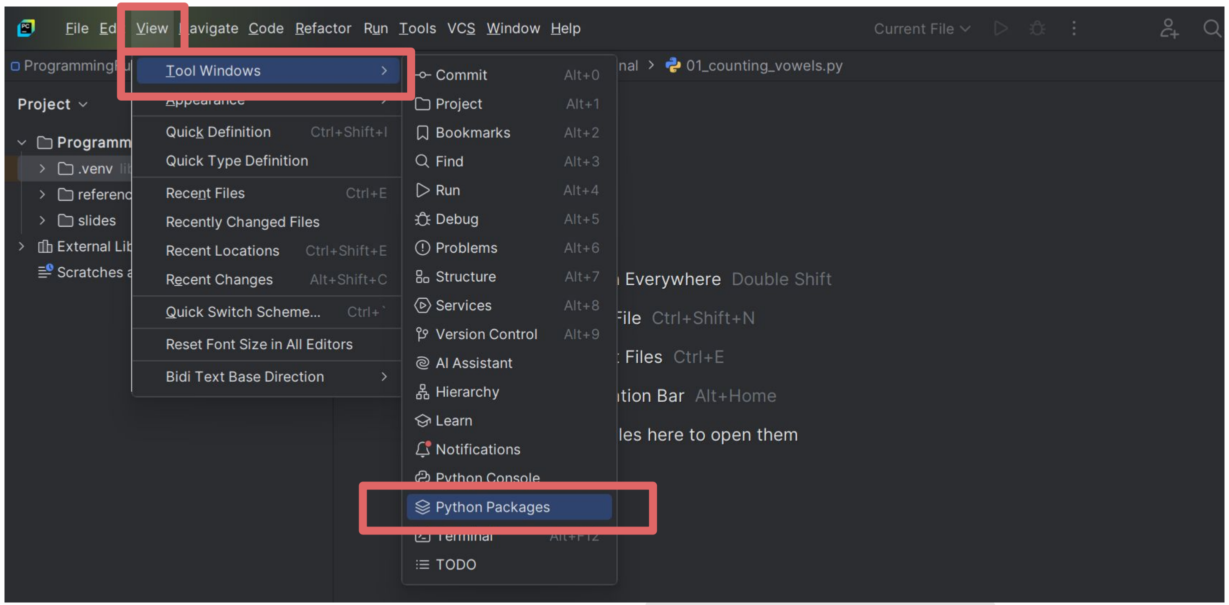
```python
import cProfile

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

def main():
    print(fib(38))

if __name__=='__main__':
    cProfile.run("main()")
```

Functools Demo

```python
import cProfile
from functools import cache

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

def main():
    print(fib(38))

if __name__=='__main__':
    cProfile.run("main()")
```

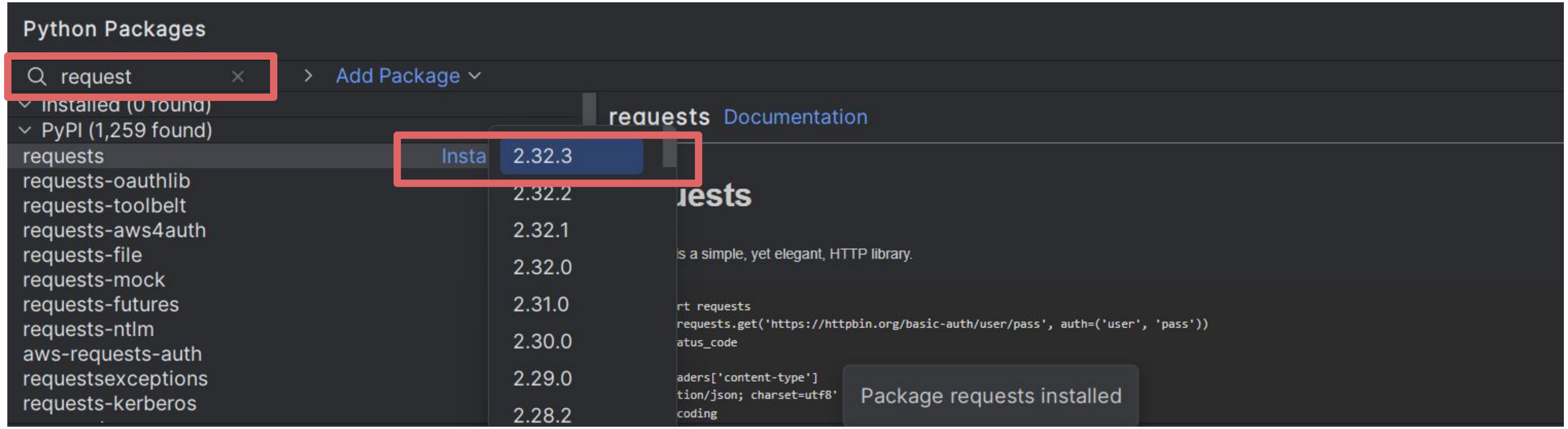
Prerequisite: Python Packages

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.



is a simple, yet elegant, HTTP library.

```
rt requests
requests.get('https://httpbin.org/basic-auth/user/pass', auth=('user', 'pass'))
atus_code

aders['content-type']
tion/json; charset=utf8'
coding
```

Package requests installed

Requests Demo

The requests library allows Python to simplify HTTP requests

```python
import requests

site = "https://official-joke-api.appspot.com/random_joke"
response = requests.get(site)

joke = response.json()
print(joke['setup'])
print(joke['punchline'])
```

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

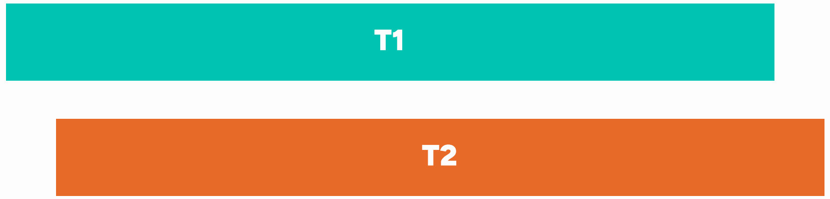
Multiple Tasks

A preview of Multiprocessing and Multithreading

Parallelism versus Concurrency

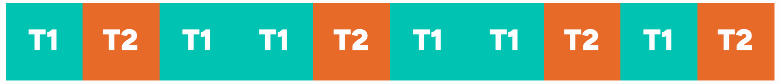
Parallel Process

Tasks running simultaneously or at the same time



Concurrent Process

Switching between tasks when waiting for results



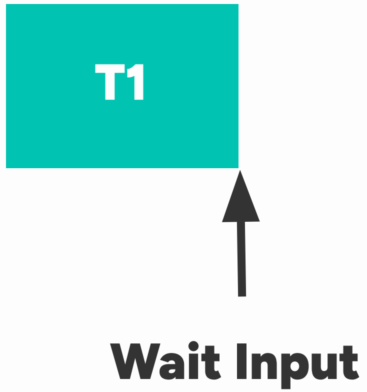
Concurrency

Working while waiting for other tasks

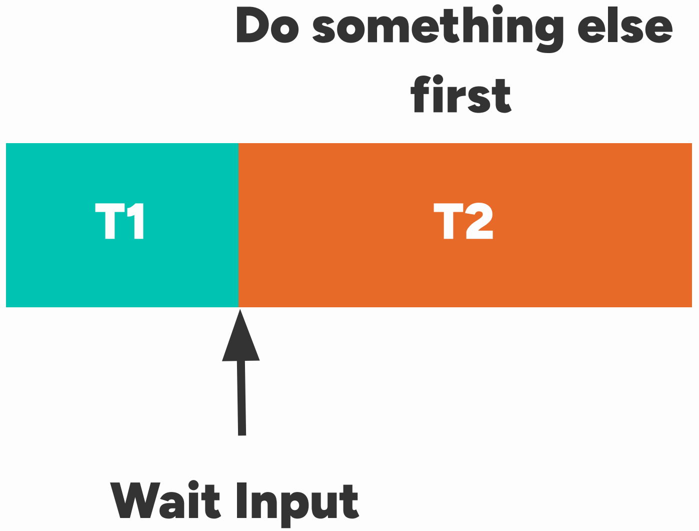
Concurrent Process

Current Task

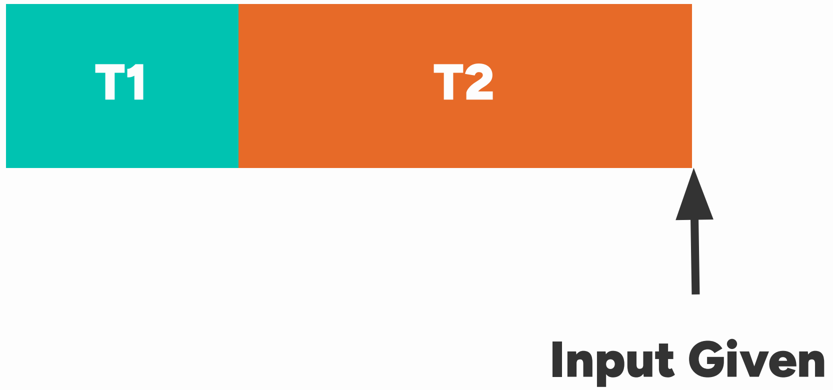
Concurrent Process



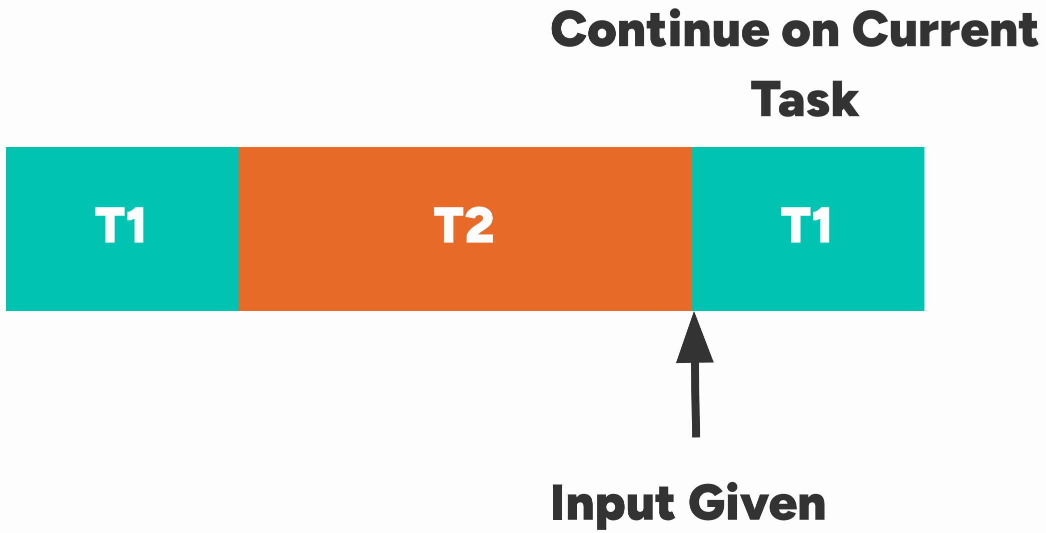
Concurrent Process



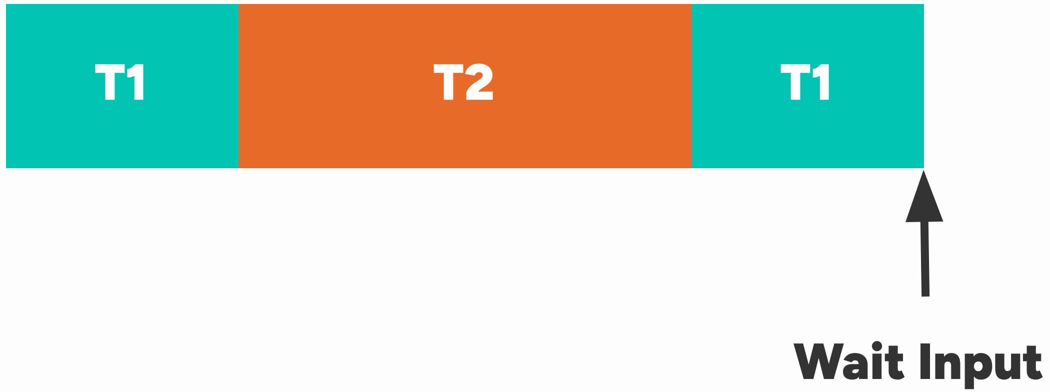
Concurrent Process



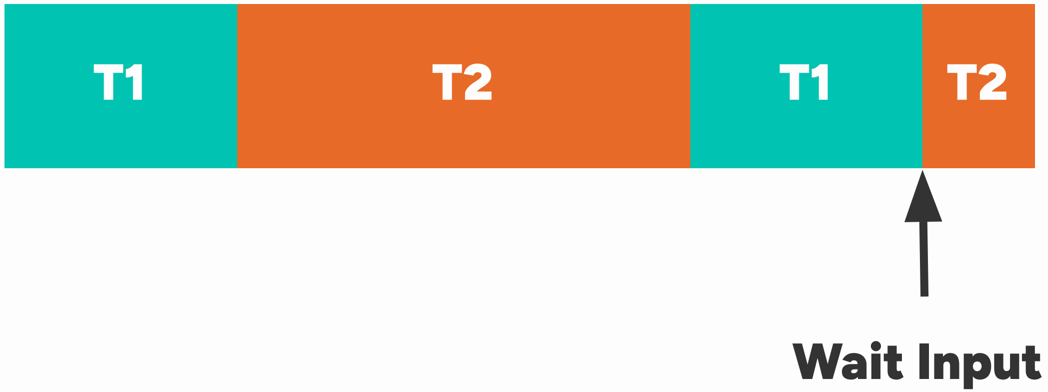
Concurrent Process



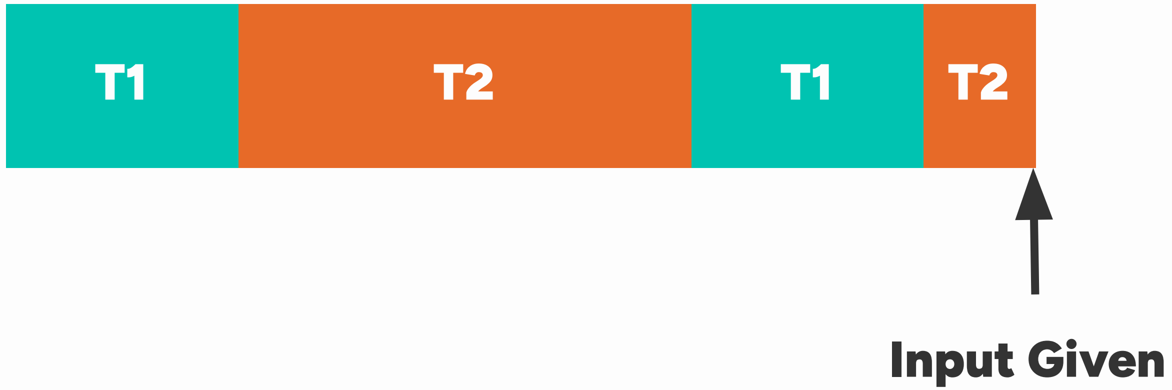
Concurrent Process



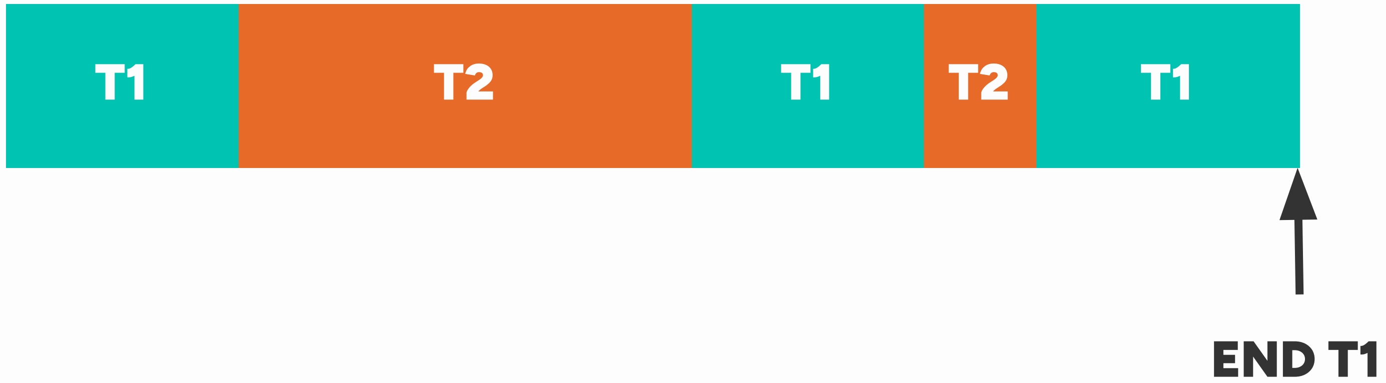
Concurrent Process



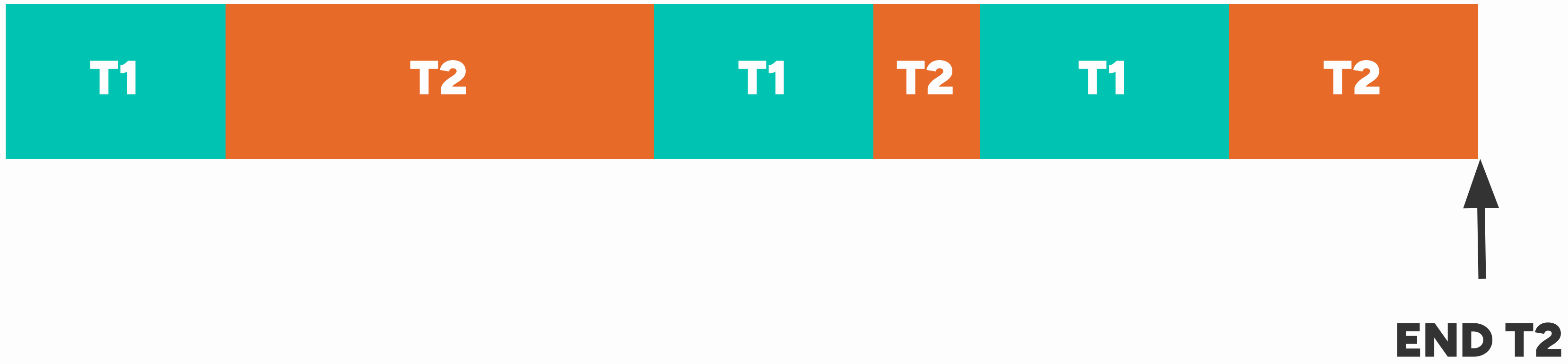
Concurrent Process



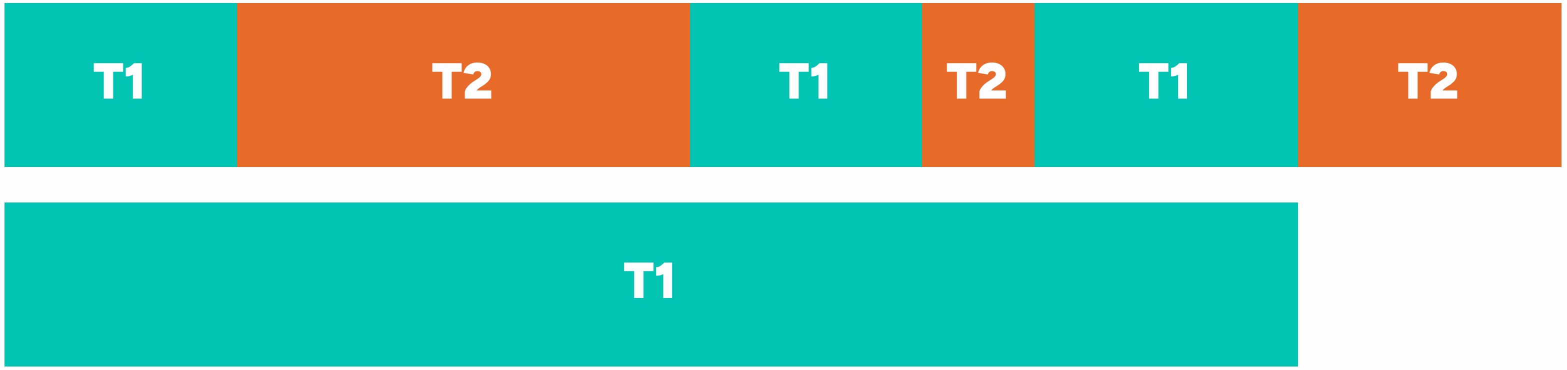
Concurrent Process



Concurrent Process



Concurrent Process



Thread Pool Mapping

```python
import requests
import cProfile

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

def main():
    inputs = [
        f'https://httpbin.org/delay/{wait}'
        for wait in range(1, 5)
    ]
    outputs = [fetch_url(url) for url in inputs]

if __name__=='__main__':
    cProfile.run("main()", sort="cumtime")
```

Thread Pool Mapping

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

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

def main():
    inputs = [
        f'https://httpbin.org/delay/{wait}'
        for wait in range(1, 5)
     ]
     with ThreadPoolExecutor() as pool:
        outputs = pool.map(fetch_url, inputs)

if __name__=='__main__':
    cProfile.run("main()", sort="cumtime")
```

H2

Website Check

Check multiple websites if they are working

Website Check - Main Function

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

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

```python
base_url = "https://raw.githubusercontent.com/"
file_name = "bensooter/URLchecker/master/top-1000-websites.txt"
response = requests.get(base_url + file_name)

websites = response.text.splitlines()
websites = ["https://" + s.strip() for s in websites if site.strip()]

websites = websites[:100]
```

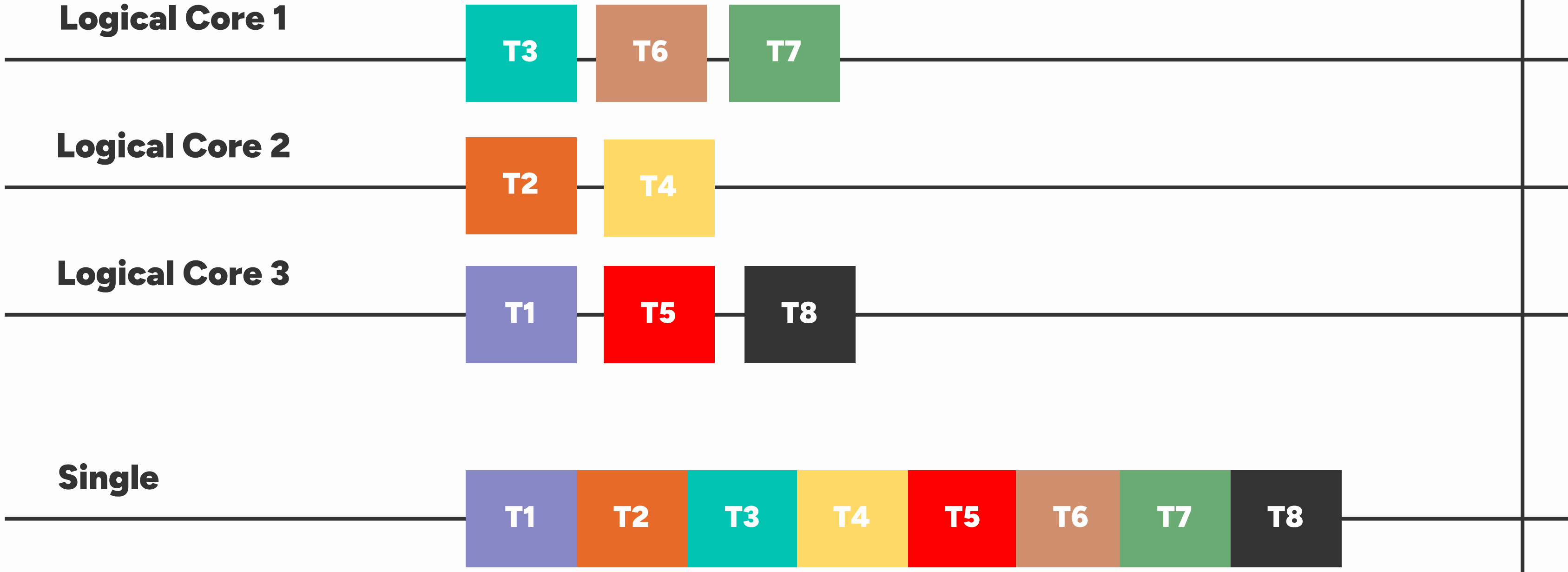
Website Check - Get Text Data

```python
def main():
    for website in websites:
        check_website(website)

if __name__=='__main__':
    cProfile.run("main()", sort="cumtime")
```

Multiprocessing

Actually doing multiple tasks at once

Parallelism using Multiprocessing



Sequential Task

```python
import cProfile

def process(number):
    for _ in range(1_000_000):
        x = 10 ** 1000

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

if __name__=='__main__':
    cProfile.run("main()", sort="cumtime")
```

Multi-Process Task

```python
from multiprocessing import Pool
import cProfile

def process(number):
    for _ in range(1_000_000):
        x = 10 ** 1000

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

if __name__=='__main__':
    cProfile.run("main()", sort="cumtime")
```

H3

Fibonacci Task

Fancy counting done fast

Sequential Fibonacci Calculation

```python
from multiprocessing import Pool
import cProfile

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

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

if __name__=='__main__':
    cProfile.run("main()", sort="cumtime")
```

03

Best Practices

Recommended way to write Python code

Readability

Writing code for people

Example Code No. 1

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

    for i in ix:

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

    return ic
```

Example Code 1 (Refactor)

```python
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. 2

```python
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. 2 (Refactor)

```python
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

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

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

```python
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

```python
"""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)

```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

```python
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

```python
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)
```

Quick Exercise: Document RPG

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

Testing

Security for your colleagues and future self

Common Types of Testing

Unit

Testing individual parts or functions in isolation

Integration

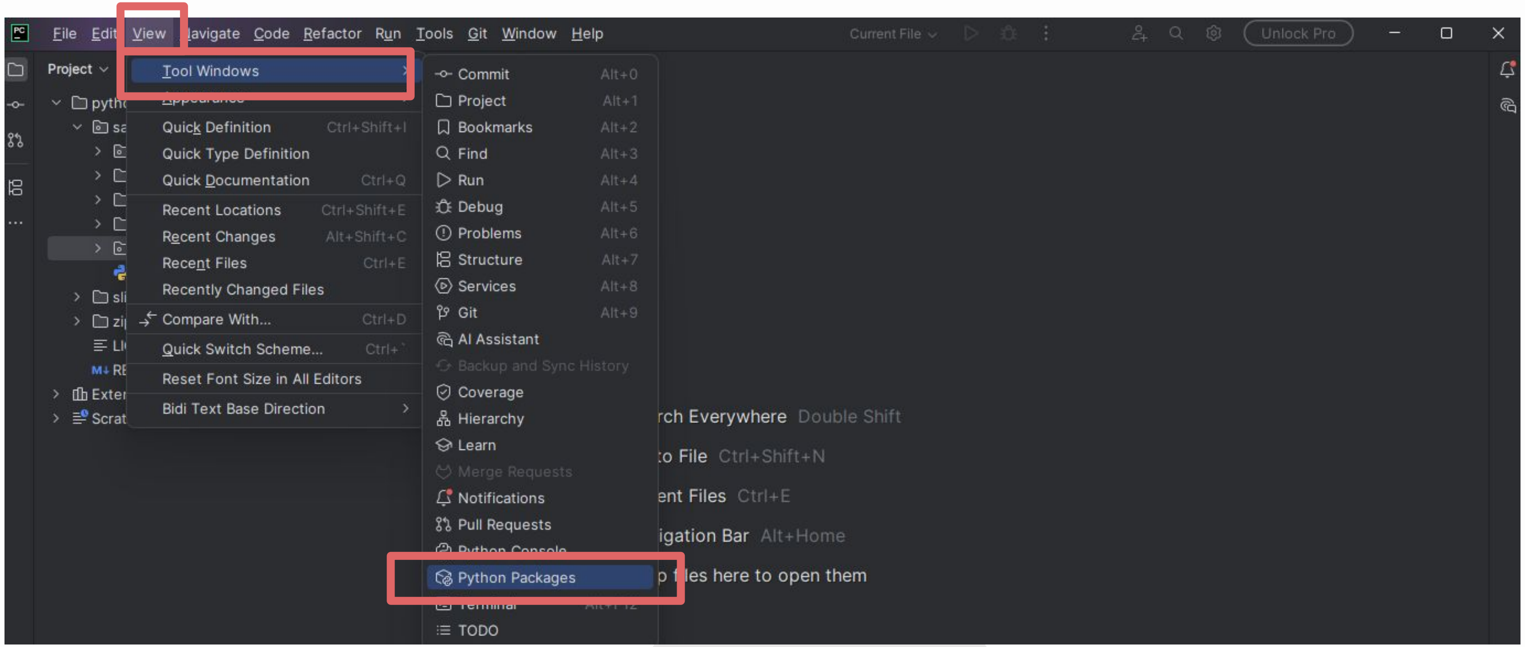
Testing if different components work together correctly

Regression

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

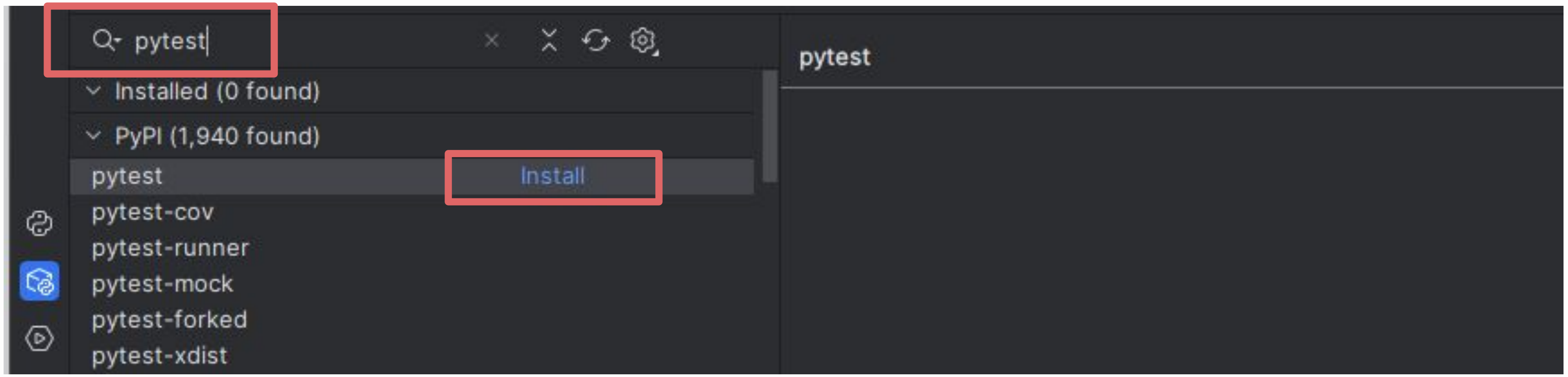
Prerequisite: Python Packages

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

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

def test_square_positive():
    assert square(2) == 4

def test_square_negative():
    assert square(-3) == 9

def test_square_zero():
    assert square(0) == 0
```

Grouped Unit Tests

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

class TestSquareNumber:
    def test_square_positive(self):
        assert square(2) == 4
    def test_square_negative(self):
        assert square(-3) == 9

class TestSquareSpecial:
    def test_square_zero(self):
        assert square(0) == 0
    def test_square_increase(self):
        assert square(10) > 10
```

Integration Test

Testing if different components work as intended when combined together

```python
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
```

Regression Test

Check if changes in the code have not affected existing functionality

```python
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

def test_calculate_expression_three_inputs():
    assert calculate_expression(2, 3, 2) == 8
```

H4

Code Check

Prove the code works, one test at a time

```python
def is_anagram(word1, word2):
    pass
```

```python
def is_palindrome(word1, word2):
    pass
```

```python
def is_pangram(word):
    pass
```

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

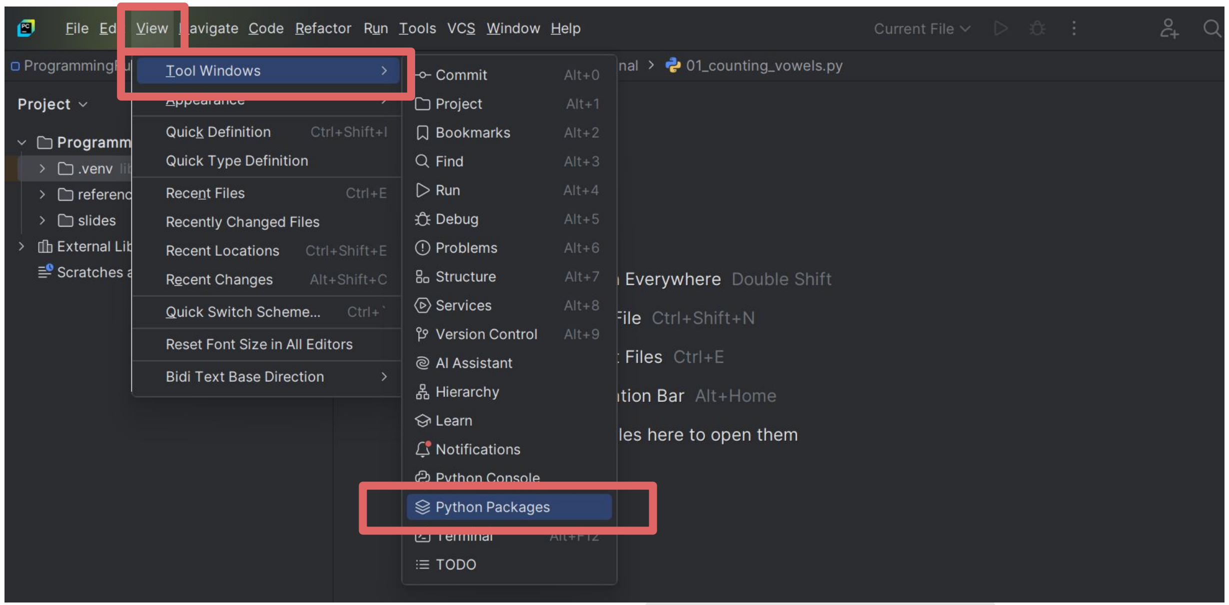
- Very easy syntax
- Built-in Pandas and Plotting Support
- **Small Pages or Data Dashboards**

Fast API

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

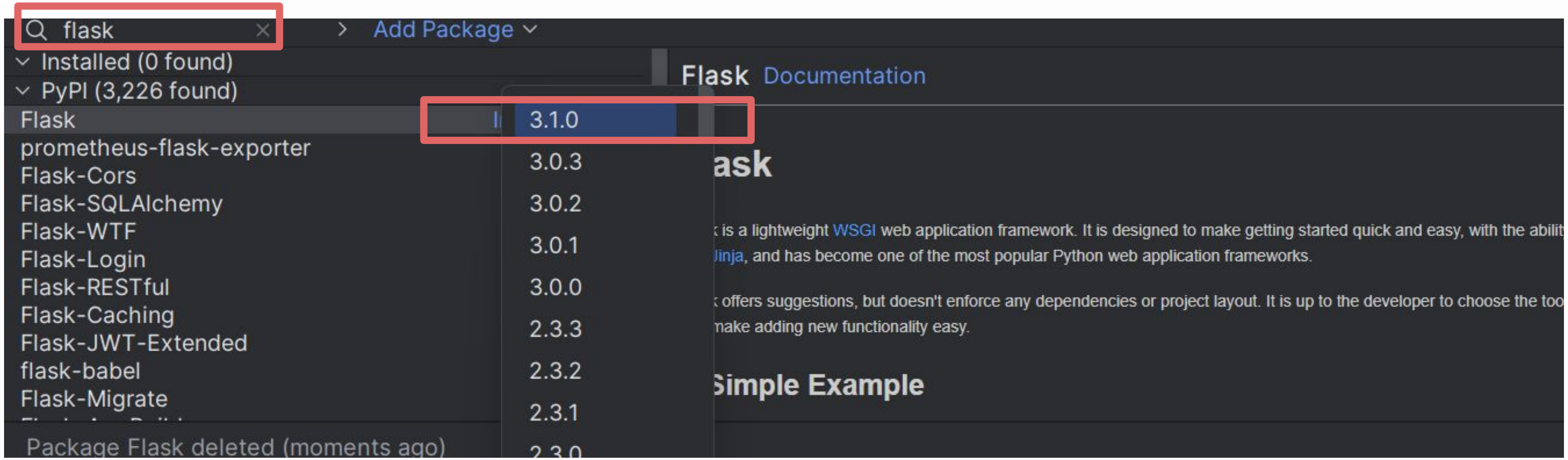
Prerequisite: Python Packages

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.



flask

Add Package

Installed (0 found)

PyPI (3,226 found)

Flask
prometheus-flask-exporter
Flask-Cors
Flask-SQLAlchemy
Flask-WTF
Flask-Login
Flask-RESTful
Flask-Caching
Flask-JWT-Extended
flask-babel
Flask-Migrate

3.1.0
3.0.3
3.0.2
3.0.1
3.0.0
2.3.3
2.3.2
2.3.1
2.3.0

Flask Documentation

ask

is a lightweight WSGI web application framework. It is designed to make getting started quick and easy, with the abilit

Jinja, and has become one of the most popular Python web application frameworks.

offers suggestions, but doesn't enforce any dependencies or project layout. It is up to the developer to choose the too

make adding new functionality easy.

Simple Example

Package Flask deleted (moments ago)

Minimum Setup

```python
from flask import Flask

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

Routing

Setting up the subpages of the site

Index Route

```python
from flask import Flask

app = Flask(__name__)

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

app.run()
```

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

```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.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>")
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>
```

Paragraphs

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

```
<h1>Header </h1>
<p>The p tag is used to define paragraphs   </p>
```

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

```html
<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

```html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <title>Website Title Here</title>
</head>

<body>
    Your content goes here
</body>

</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

```html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <link rel="stylesheet" href="styles.css">
    <title>Website Title Here</title>
</head>

<body>
    Your content goes here
</body>

</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

```html
<h1>Introduction Page</h1>
<p>Hello! My name is Jeff Jeff!</p>
<ul>
    <li><a href="/hobby/">Favorite Activities</a></li>
    <li><a href="/opinion/food">Favorite Food</a></li>
</ul>
```

Template Render

main.py

```python
from flask import Flask, render_template

app = Flask(__name__)

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

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/food	Food Page	Enumerate your top five favorite food (in order)

HTML with Loops

./templates/hobbies.html

```html
<h1>Hobby Page</h1>
<p>Here are my favorite activities outside work</p>

<ul>
  {% for hobby in hobbies %}
    <li>{{ hobby }}</li>
  {% endfor %}
</ul>

<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

```
<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>
<ul>
    <li><a href="/hobby/">Favorite Activities</a></li>
    <li><a href="/opinion/food">Favorite Food</a></li>
</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/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",
        "Translation": "Proficient",
        "Eating": "Professional"
    }
    return render_template("skills.html", skills=skill_levels)

...
```

Dictionary

./templates/skills.html

```
<h1>Skills Page</h1>
<ul>
  {% for skill, level in skills.items() %}
    <li>
      {{ skill }} - {{ level }}
    </li>
  {% endfor %}
</ul>

<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/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

```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/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

Request Form

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

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

```
from flask import Flask, render_template, request

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

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

Request Form

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

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

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

To-Do List Page

./templates/todo.html

```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

Books

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

YouTube

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

Email me at `stephen.singer.098@gmail.com`

Python: Day 04

Advanced Programming