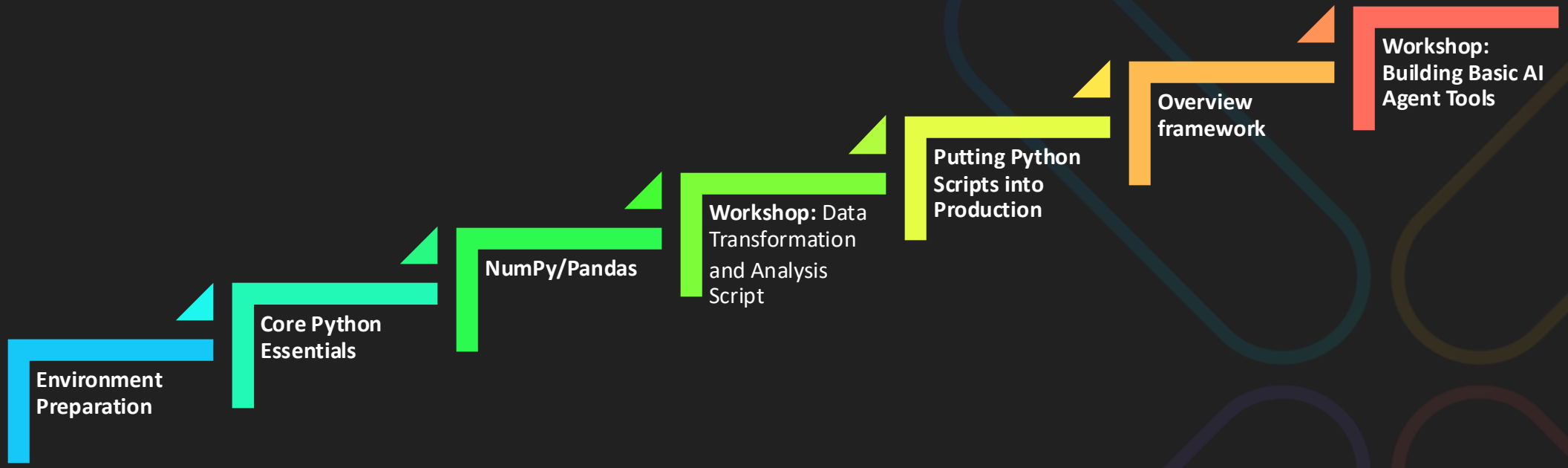
Basic Python for Microservices, Batch, and Data Science

rattanarit.prasomsab@scbtechx.io

Course Overview

Course Overview

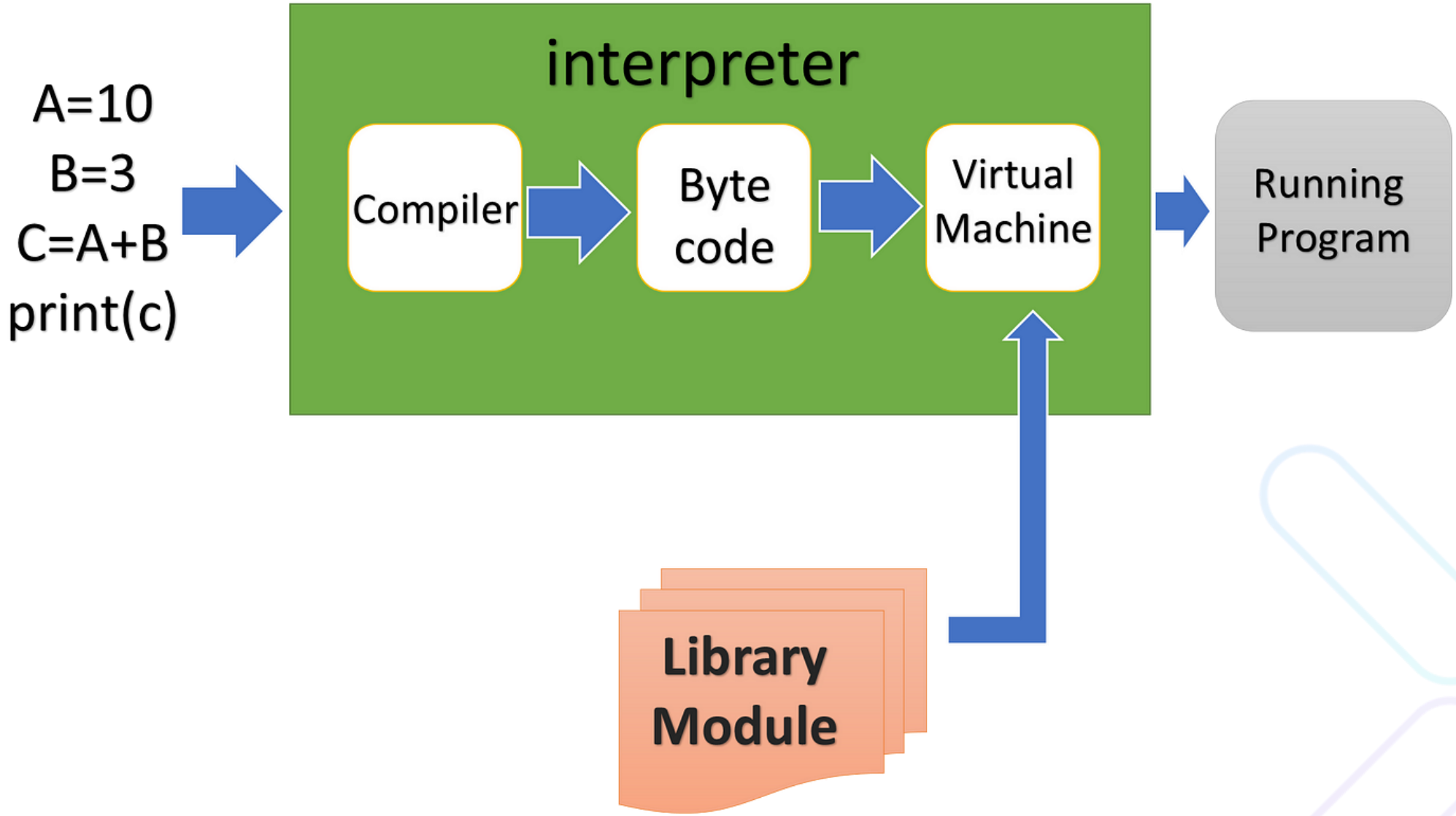


Python is a high-level programming language known for its simplicity, readability, and flexibility. Created by Guido van Rossum and first released in 1991

Environment Preparation

- Code Editor (VS Code)
- Python Interpreter
- Package Manager (pip)
- Virtual Environment (venv, pipenv)

How Python Interpreter Works?

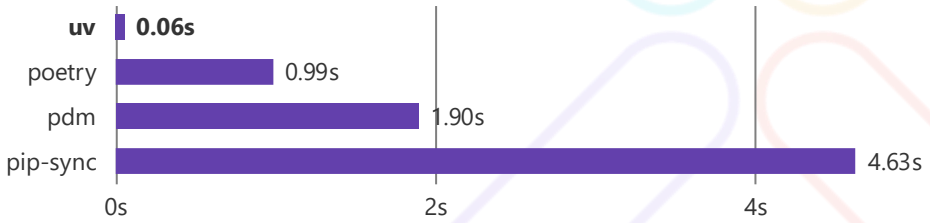


uv

An extremely fast Python package and project manager, written in Rust.

Highlights

- 🚀 A single tool to replace `pip`, `pip-tools`, `pipx`, `poetry`, `pyenv`, `twine`, `virtualenv`, and more.
- ⚡ 10-100x faster than `pip`.
- 🗂️ Provides comprehensive project management, with a universal lockfile.
- ✨ Runs scripts, with support for inline dependency metadata.
- 🐍 Installs and manages Python versions.
- 🛠️ Runs and installs tools published as Python packages.
- 🔩 Includes a pip-compatible interface for a performance boost with a familiar CLI.
- 🏢 Supports Cargo-style workspaces for scalable projects.
- 💾 Disk-space efficient, with a global cache for dependency deduplication.
- ⬇️ Installable without Rust or Python via `curl` or `pip`.
- 🖥️ Supports macOS, Linux, and Windows.



Environment Preparation

macOS and Linux

```
#Use curl to download the script and execute it with sh:
curl -LsSf https://astral.sh/uv/install.sh | sh
#If your system doesn't have curl, you can use wget:
wget -qO- https://astral.sh/uv/install.sh | sh
```

Homebrew

```
brew install uv
```

Windows

```
powershell -ExecutionPolicy ByPass -c "irm https://astral.sh/uv/install.ps1 | iex"
```

https://docs.astral.sh/uv/getting-started/installation/

Live demo

Core Python Essentials

Python Indentation

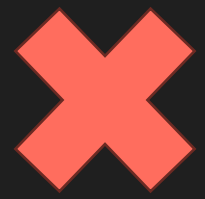
Python uses indentation to indicate a block of code.

Example

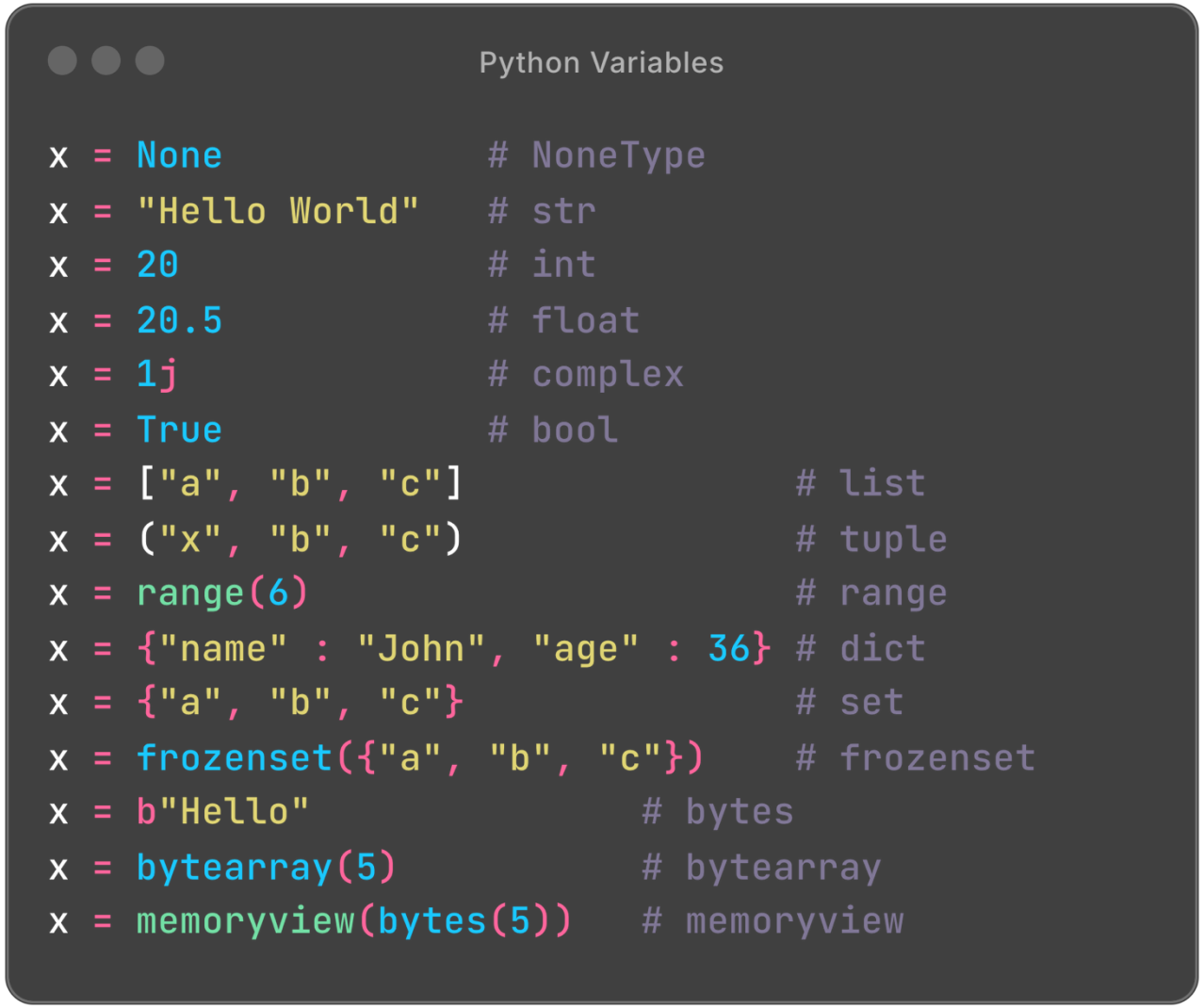
```python
if 5 > 2:
  print("Five is greater than two!")
```

Syntax Error

```python
if 5 > 2:
print("Five is greater than two!")
```



Basic Data Types



None Type	NoneType
Text Type	str
Numeric Types	int, float, complex
Boolean Type	bool
Sequence Types	list, tuple, range
Mapping Type	dict
Set Types	set, frozenset
Binary Type	bytes, bytearray, memoryview

Python Operators

Operator	Description	Example
Python Logical Operators		
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)
Python Identity Operators		
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y
Python Membership Operators		
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

https://www.w3schools.com/python/python_operators.asp

Functions & Modules

Python Function

```python
def add(x, y):
    """This function adds two numbers."""
    return x + y

result = add(5, 3)  # Calling the function
print(result)  # Output: 8
```

A function is a reusable block of code that performs a specific task. It takes inputs (arguments), does something with them, and may return an output. Think of it as a mini-program within your program.

*In Python a function is defined using the def keyword

A module is a single file containing Python code (functions, classes, variables). It's a way to organize your code into logical units. Think of it as a chapter in a book.

Python Module

```python
# my_module.py
def greet(name):
    """This function greets the person passed in as a parameter."""
    print(f"Hello, {name}!")

# main.py
import my_module

my_module.greet("Alice")  # Output: Hello, Alice!
```

Package

Python Package

```
my_package/
    __init__.py
    module1.py
    module2.py

# main.py
import my_package.module1

my_package.module1.my_function()
```

A package is a collection of modules organized in a directory hierarchy. It's a way to group related modules together. Think of it as a book with multiple chapters.

The __init__.py file is executed when the package is first imported and can be used to initialize the package or define package-level attributes.
(which can be empty in newer versions of Python #3.3+).

Python Namespace Package (PEP 420)

```
dir1/
    my_package/
        module1.py  # No __init__.py

dir2/
    my_package/
        module2.py  # No __init__.py

# main.py
import sys
sys.path.append("dir1")
sys.path.append("dir2")

import my_package.module1
import my_package.module2

my_package.module1.my_function()
my_package.module2.my_other_function()
```

Scripts(.py) vs Notebooks(.ipynb)

Feature	.py Files	.ipynb Notebooks
Purpose	Production, automation, modules	Exploration, analysis, prototyping
Execution	Top-to-bottom	Cell-based
Interactivity	Limited	High
Structure	Code only	Code, text, visualizations
Version Control	Well-supported with Git	Can be challenging with complex notebooks
Deployment	Easy to deploy to production environments	Requires conversion to .py or use of specialized tools like Voilà for creating interactive web applications

NumPy

What is NumPy

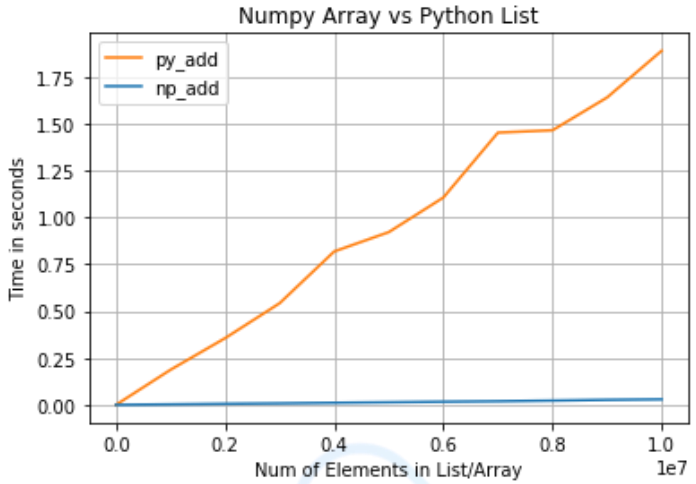
NumPy is a Python library used for working with arrays.

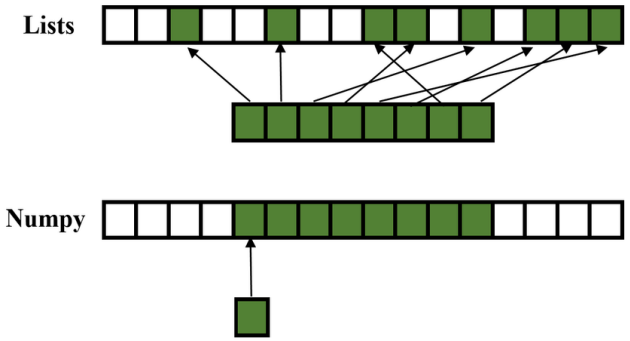
NumPy stands for Numerical Python.

Why Use NumPy?

In Python we have lists that serve the purpose of arrays, but they are slow to process.

NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.





Why is NumPy Faster Than Lists?

NumPy arrays are stored at one continuous place in memory.

Also it is optimized to work with latest CPU architectures.

Pandas

Workshop:
Data Transformation and Analysis Script

Workshop: Data Transformation and Analysis Script

Problem: Write a Python script (.py file) that uses pandas to read data from a provided CSV file, performs data cleaning and transformation using both pandas and numpy, and saves the result to a new CSV file. Then filter the new CSV to answer the question.

Detail Tasks:

1. **Data Loading:** Use pandas to read a CSV file named data.csv into a DataFrame.
2. **Missing Data Handling and Data Transformation :**
    1. **Year:** If year >= 2023, delete it.
    2. **Gender:** if gender is null or empty -> replace it with '**Male**'
    3. **Age:** if age is null/empty/na -> replaced with the '**mean**' value of age. (Round up 0.5 -> 1)
3. **Save Results:** Save the transformed and filtered DataFrame to a new CSV file named transformed_data.csv.
4. **Quiz**: Write a program and run it to answer questions using the transformed_data.csv file.

File: data/data.csv [https://github.com/hiiamtin/python-workshop01]

https://forms.office.com/r/f9fNrBnnzn

SCB TECH[X]

Production Concepts

Logging / Why is Logging Important?

- **Tracking:** Know what the program is doing at any given time.
- **Debugging:** When issues arise, you can trace back what happened previously.
- **Auditing:** Record important access or actions for review.
- **Analysis:** Keep data records for usage or performance analysis.
- **Most importantly:** On a Production Environment, we can't always use print() to see the output.

Basic Usage of the logging module

Logging

```python
import logging

logging.basicConfig(
    level=logging.INFO, # Set the minimum level to display
    format='%(asctime)s - %(levelname)s - %(message)s',
    filename='app.log', # (Optional) Save to a file named app.log
    filemode='w'        # (Optional) 'w' overwrites, 'a' append
)

logging.info("starting...")
logging.warning("Slow connection to external server")
try:
    result = 10 / 0
except ZeroDivisionError:
    logging.error("Error: Cannot divide by zero.")
```

Logging Levels

DEBUG: The most detailed information for debugging a program.

INFO: General status messages (e.g. "Program started", "Database connection successful")

WARNING: Alerts when something unexpected has happened but the program can continue to run

ERROR: Errors that cause some functions to fail

CRITICAL: The most serious error that can cause the entire program to crash

Multithreading

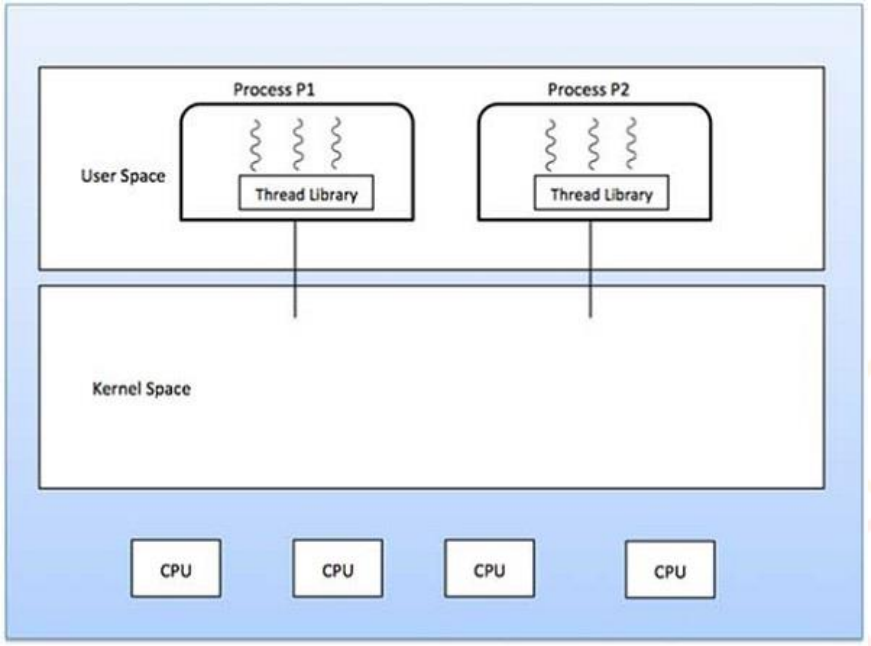
SCB TECH×

What is Multithreading?

- Multithreading is the concept of allowing a program to execute multiple sub-parts (called "threads") concurrently within a single process.
- The main goal is to improve performance, especially with tasks that involve waiting (I/O-Bound Tasks).

I/O-Bound Tasks

- Calling external APIs
- Reading or writing files to disk
- Connecting and retrieving data from a database
- Downloading data from the Internet



https://realpython.com/python-gil/

Multithreading

Caution

Race Condition: A problem that occurs when multiple threads attempt to access and modify shared resources at the same time, resulting in erroneous or unexpected results.

Example: Writing **logs** to the **same file** from **multiple threads** at the **same time** can cause corruption or commingling of the data in the file.

Thread-safe: This means writing code that is guaranteed to work properly even if multiple threads are running at the same time.

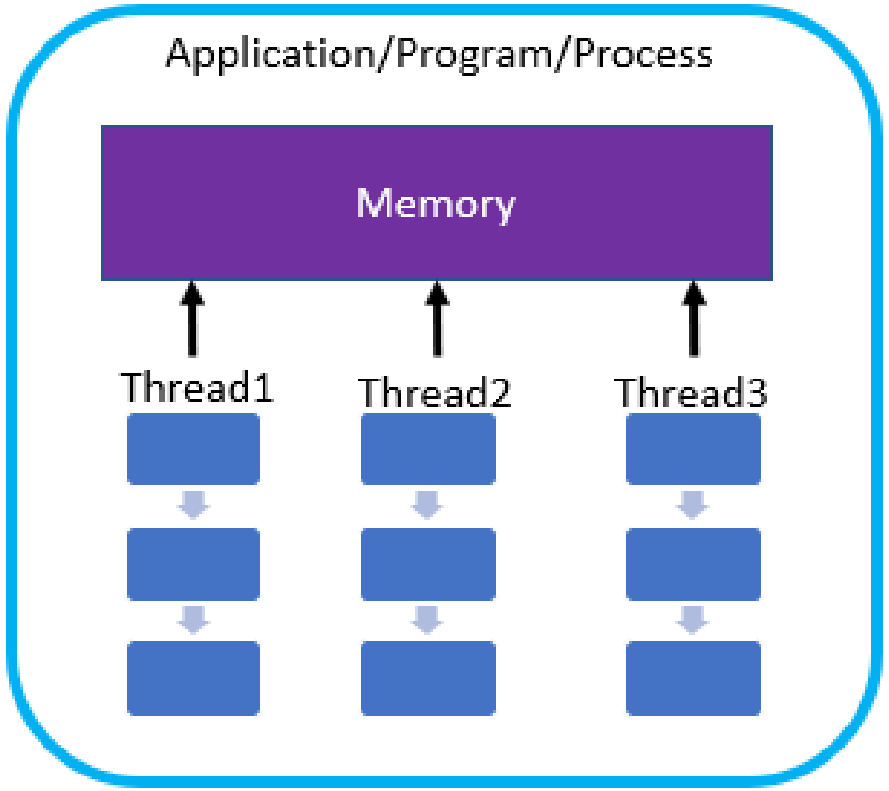
Solution: Use the right tools for the situation, such as Locks for simple protection or Queues for securely passing data between threads.

Multithreading-01

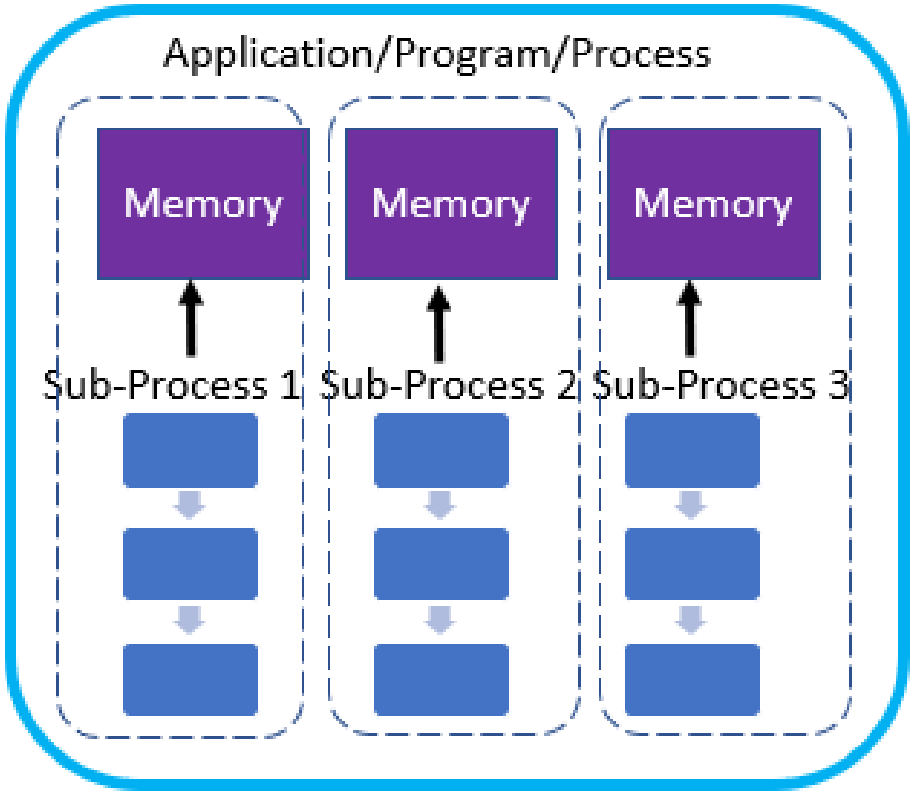
SCB TECH[X]

```python
import threading
import time

def fetch_data(source):
        print(f"Starting to fetch data from {source}...")
        time.sleep(2) # จำลองการรอนาน 2 วินาที
        print(f"Finished fetching from {source}.")

# สร้าง Thread
thread1 = threading.Thread(target=fetch_data, args=("API",))
thread2 = threading.Thread(target=fetch_data, args=("Database",))

# เริ่มการทำงานของ Thread
thread1.start()
thread2.start()

print("Main thread continues to run...")

# รอให้ทั้งสอง Thread ทำงานจบ
thread1.join()
thread2.join()

print("All tasks finished.")
```

Multiprocessing



Multi-Threading



Multi-processing

OOP

- stands for **Object-Oriented Programming**.
- Python is an object-oriented language, allowing you to structure your code using classes and objects for better organization and reusability.

OOP-1

```python
# อาจจะกระจัดกระจายในไฟล์
def process_data(data):
    # process logic...
    return processed

def save_data(data, path):
    # save logic...
    pass

# ตอนใช้งาน
raw_data = [...]
processed = process_data(raw_data)
save_data(processed, 'file.csv')
```

OOP-2

```python
class DataProcessor:
    def __init__(self, data):
        self.data = data # Attribute

    def process(self): # Method
        # process logic...
        self.data = processed_data

    def save(self, path): # Method
        # save logic...
        pass

# ตอนใช้งาน
processor = DataProcessor(raw_data)
processor.process()
processor.save('file.csv')
```

https://www.w3schools.com/python/python_oop.asp

OOP - Inheritance

OOP-3

```python
class Person:
    def __init__(self, fname, lname):
        self.firstname = fname
        self.lastname = lname

    def printname(self):
        print(self.firstname, self.lastname)
```

```python
y = Student("John", "Doe", 2023)
y.printname()
print(y.graduationyear)
y.welcome()
```

OOP-4

```python
class Student(Person):

    def __init__(self, fname, lname, year):
        super().__init__(fname, lname)
        self.graduationyear = year

    def welcome(self):
        print("Welcome", self.firstname, self.lastname, "to the class of",
self.graduationyear)
```

Polymorphism

The word "polymorphism" means "many forms", and in programming it refers to methods/functions/operators with the same name that can be executed on many objects or classes.

Class Polymorphism

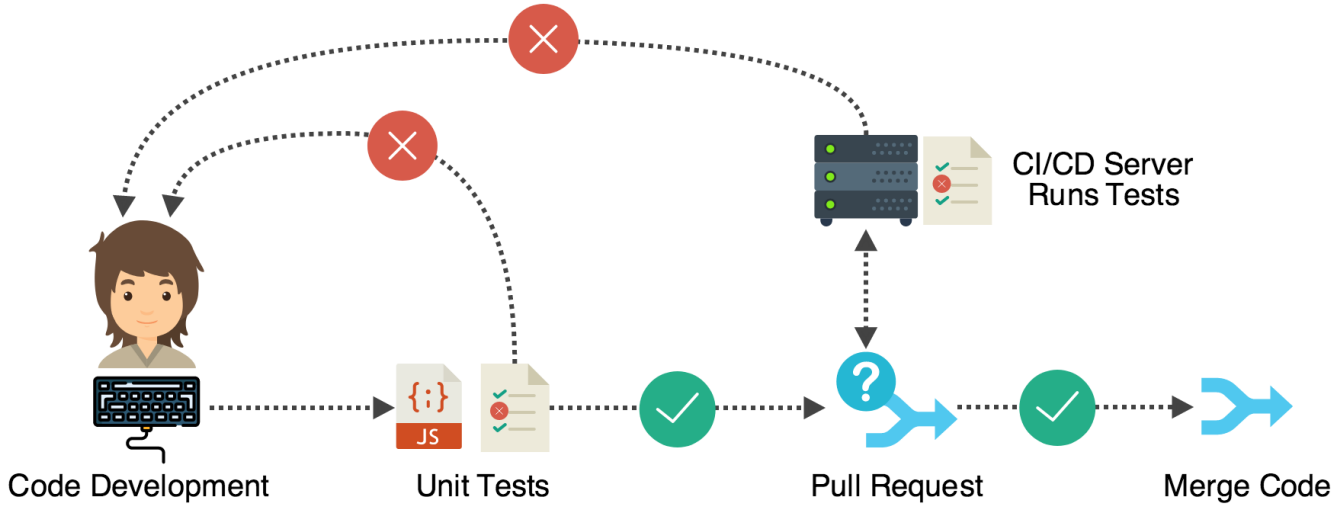
```
class Car:
  def __init__(self, brand, model):
    self.brand = brand
    self.model = model

  def move(self):
    print("Drive!")

class Boat:
  def __init__(self, brand, model):
    self.brand = brand
    self.model = model

  def move(self):
    print("Sail!")

class Plane:
  def __init__(self, brand, model):
    self.brand = brand
    self.model = model

  def move(self):
    print("Fly!")

car1 = Car("Ford", "Mustang")       #Create a Car object
boat1 = Boat("Ibiza", "Touring 20") #Create a Boat object
plane1 = Plane("Boeing", "747")     #Create a Plane object

for x in (car1, boat1, plane1):
  x.move()
```

SCB TECH[X]

Testing (Unit test)

Why Test?

- Reduce Bugs
- Improve Code Quality
- Increase Confidence in Refactoring
- Documentation



unittest vs pytest

unittest: Built-in Testing Framework
pytest: A Popular Third-Party Framework

Why use Pytest?

- **Simple Syntax**: Makes writing tests easy and quick to understand.
- **Fixtures**: Powerful fixture system that helps prepare data or environment before testing.
- **Plugins**: A variety of plugins that add various capabilities.

Pytest [https://docs.pytest.org/en/stable]

my_functions.py

```python
# my_functions.py
def add(x, y):
    """This function adds two numbers."""
    return x + y
```

install pytest

```
uv add pytest
# or
pip install pytest
```

```
project/
├── my_functions.py
└── tests/
    └── test_my_functions.py
```

tests/test_my_functions.py

```python
# tests/test_my_functions.py
from my_functions import add

def test_add_positive_numbers():
    """Tests adding two positive numbers."""
    assert add(2, 3) == 5

def test_add_negative_numbers():
    """Tests adding two negative numbers."""
    assert add(-1, -1) == -2
```

run pytest

```
pytest
# or uv run pytest
# if ModuleNotFoundError add this to pyproject.toml
[tool.pytest.ini_options]
pythonpath = ["."]
```

Pytest - Fixtures

tests/test_data_processing.py

```python
import pytest
import pandas as pd

@pytest.fixture
def sample_dataframe():
    """Provides a sample DataFrame for testing."""
    data = {'col1': [1, 2], 'col2': [3, 4]}
    return pd.DataFrame(data)

def test_dataframe_shape(sample_dataframe):
    """Uses the fixture to test DataFrame shape."""
    # `sample_dataframe` ถูกส่งมาจาก fixture ด้านบน
    assert sample_dataframe.shape == (2, 2)

def test_column_names(sample_dataframe):
    """Uses the same fixture for another test."""
    assert 'col1' in sample_dataframe.columns
```

SCB TECH×

Basic Python Packaging

- Packaging: The process of organizing our Python code into a standard format, so it's easy to install, distribute, and reuse in other projects.

Key Components of a Package
Code: A collection of .py files that form our module or library.
Metadata Files: Files that describe our package, such as:

- Name
- Version
- Author
- Dependencies (e.g., pandas, numpy)

Currently Used Tools:

- **pyproject.toml:** A modern standard file used for configuring various project settings and the package building process. It is recommended to use this file as the primary configuration.
- **setup.py:** A traditional file still commonly seen, but newer practices are increasingly shifting towards pyproject.toml.

Basic Python Packaging

Concept Overview

Organize: Arrange our code files into an organized folder structure.

Configure: Create a pyproject.toml file to provide important information about our package (name, version, dependencies).

Build: Use a tool (e.g., build) to create "Distribution Files" (.whl and .tar.gz files) from our code and configuration files.

Distribute: (Optional) Upload the created files to a package repository like PyPI (Python Package Index) so others can pip install it.

Overview framework

Backend Frameworks

SCB TECH[X]

Django

Highlights: "The web framework for perfectionists with deadlines." A full-stack framework that comes with everything needed for rapid web application development (ORM, Admin Panel, Authentication, Templating Engine). Ideal for large and complex projects.

Best for: Feature-rich websites, CMS, E-commerce, social networks.

Current Status: Active Development, large community, abundant resources.

Links:

- Main Website: https://www.djangoproject.com
- Document: https://docs.djangoproject.com/en/stable/intro/tutorial01

Backend Frameworks

Flask

Highlights: "A microframework for Python." A minimalist framework offering high flexibility. It doesn't come with built-in features like Django but allows you to choose extensions as needed, giving you more control.

Best for: Small APIs, Microservices, medium-sized projects, or when you want full control.

Current Status: Active Development, large community, abundant resources.

Links:

- Main Website: https://flask.palletsprojects.com
- Document: https://flask.palletsprojects.com/en/stable/quickstart

Backend Frameworks

FastAPI

Highlights: Focuses on high performance (Fast), supports Asynchronous programming (async/await), built-in Data Validation (Pydantic), automatic API Documentation (OpenAPI/Swagger UI), and has gained popularity rapidly.

Best for: High-performance APIs, Microservices, tasks requiring speed and scalability.

Current Status: Active Development, very popular.

Links:

- Main Website: https://fastapi.tiangolo.com
- Document: https://fastapi.tiangolo.com/tutorial

Frontend Frameworks

SCB TECH[X]

Streamlit

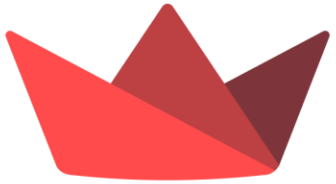
Highlights: Quickly builds interactive web apps for Data Science and Machine Learning using only Python, without needing HTML/CSS/JavaScript.

Best for: Creating dashboards, prototypes for Data Apps, Machine Learning demos.

Current Status: Active Development, popular in the Data Science community.

Links:

- Main Website: https://streamlit.io
- Document: https://docs.streamlit.io



Streamlit

Frontend Frameworks

Kivy

Highlights: A **cross-platform GUI framework** for Python. It's designed for **multi-touch interfaces** and runs on **Windows, macOS, Linux, Android, and iOS**. Uses **KV Language** for UI design and is **GPU accelerated** for high performance.

Best for: Developing **mobile apps (Android, iOS)** and **desktop apps** with custom, interactive UIs. Great for **rapid prototyping** and **interactive kiosks/educational tools**.

Current Status: Actively developed, **mature and stable**, with a **community-driven** approach.

Links:

- Main Website: https://kivy.org
- Document: https://kivy.org/doc/stable

Batch Processing / Data Processing Frameworks

SCB TECH[X]

Apache Spark (PySpark)

Highlights: A distributed processing framework for Big Data. It can process large datasets quickly in batch or streaming modes and has a Python API (PySpark).

Best for: Big Data Analytics, ETL pipelines, Machine Learning on large datasets.

Current Status: Active Development, Industry Standard for Big Data.

Links:

- Main Website: https://spark.apache.org
- Document: https://spark.apache.org/docs/latest/api/python/index.html

Batch Processing / Data Processing Frameworks

SCB TECH×

Apache Airflow

Highlights: A platform for programmatically authoring, scheduling, and monitoring workflows (DAGs). Ideal for managing large and complex data pipelines.

Best for: Data Engineering, ETL Orchestration, Batch Job Scheduling.

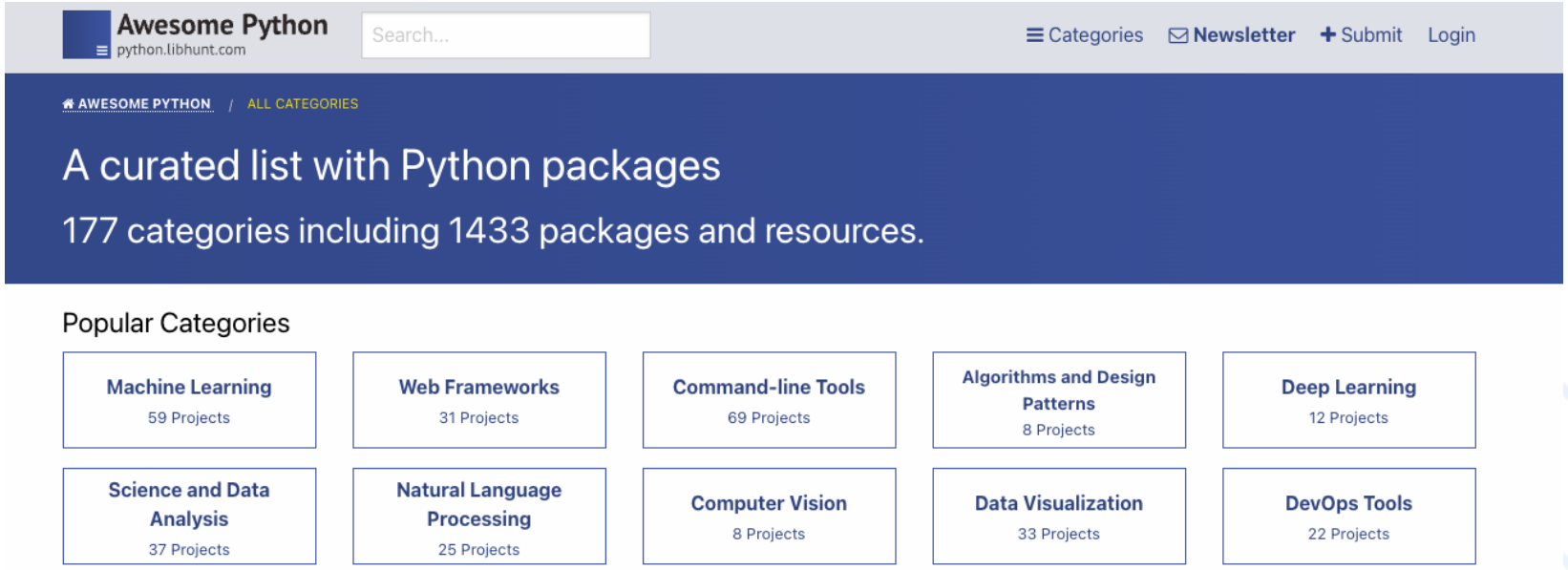
Current Status: Active Development, Industry Standard for Workflow Orchestration.

Links:

- Main Website: https://airflow.apache.org
- Document: https://airflow.apache.org/docs/apache-airflow/stable

More information: Awesome Python

Links: http://python.libhunt.com



LangChain & AI Agents

LangChain

SCB TECH[X]

- **LangChain** is a framework for building applications with LLMs.
- **AI Agent**: Systems that decide on the fly what action to take next, often by calling **Tools** to interact with the external world or access information.
- **Tools** are utilities designed to be called by a model: their inputs are designed to be generated by models, and their outputs are designed to be passed back to models. A toolkit is a collection of tools meant to be used together.

Workshop: Building AI Agent Tools

1: Don't forget your provider's API key, you need it!

2: Fork Repository: https://github.com/hiiamtin/python-workshop02
(or clone it if you don’t have Github account)

3: Follow the Readme.md

4: Submit: https://forms.office.com/r/sDUhbYhPFG

SCB TECH[X]

Q&A and Resources

- **Python:**
  - **https://www.w3schools.com/python/default.asp**
  - **https://pythongeeks.org/**
  - **https://docs.astral.sh/uv/**

- **Frameworks & Libraries:**
  - **Backend:** Django, Flask, และ FastAPI
  - **Data & Batch Processing:** Apache Airflow และ Apache Spark (PySpark)
  - **AI & LLMs:** LangChain Documentation, langchain-mcp-adapters
- **GUI & Data Apps:**
  - **Data Science Apps:** Streamlit
  - **Cross-platform Apps:** Kivy

- **Python Library Resources :**
  - Awesome Python

- Git repo:
  - https://github.com/hiiamtin/python-workshop01
  - https://github.com/hiiamtin/python-workshop02
  - https://github.com/hiiamtin/python-example

SCB TECH[x]

Thank you
for your kind participation