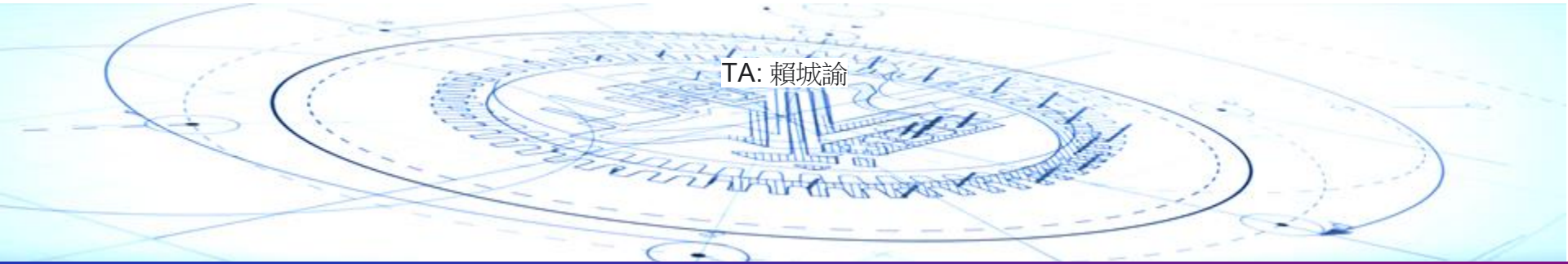
2025 Fall
AOOP Lab6
Pybind11

TA: 賴城諭



Outline

- Introduction
- Basic Setup
  - Task1: Basic Function Binding
- Core Concepts
- Exercise
  - Task2: Binding C++ class
  - Task3: STL containers & Advance features

Introduction

What is pybind11?

Real World use cases

Comparison to other binding tools

Python/C API

CPython

SWIG

Basic Setup

國立陽明交通大學
NATIONAL YANG MING CHIAO TUNG UNIVERSITY

- In your terminal

  ~/$ git clone git@github.com:wilbur1240/aoop-lab6-pybind.git

- Follow the README.md to setup the environment

- Task1 demo

  - Implement three basic math operation: add, multiply, factorial and bind to python.

```
PYBIND11_MODULE(math_ops, m) {
    m.doc() = "Basic math operations module";

    // TODO: Bind your functions here
    // Example: m.def("add", &add, "A function that adds two integers");
    m.def("add", &add, "Add two integers",
        py::arg("a"), py::arg("b"));
```

Core Concepts

How to use pybind11?

```
#include <pybind11/pybind11.h>
```

- Binding functions with different argument types (C++ vs. Python)

```
PYBIND11_MODULE(math_ops, m) {
    m.doc() = "Basic math operations module";

    // TODO: Bind your functions here
    // Example: m.def("add", &add, "A function that adds two integers");
    m.def("add", &add, "Add two integers",
        py::arg("a"), py::arg("b"));
```

- Binding classes (ctors, methods, properties)

```
PYBIND11_MODULE(vector2d, m) {
    py::class_<Vector2D>(m, "Vector2D")
        .def(py::init<double, double>(), py::arg("x") = 0, py::arg("y") = 0)
        .def_readwrite("x", &Vector2D::x)
        .def_readwrite("y", &Vector2D::y)
        .def("length", &Vector2D::length)
        .def("to_string", &Vector2D::to_string)
        .def(py::self + py::self)
        .def("__repr__", &Vector2D::to_string);
```

- STL containers

```
#include <pybind11/stl.h> // STL container support
```

Core Concepts

- Use in python

Note: place the compiled binary file under the same dir as testing python files.

```python
import math_ops

print(math_ops.add(5, 3))
print(math_ops.multiply(2.5, 4.0))
print(math_ops.factorial(5))
```

```cpp
PYBIND11_MODULE(math_ops, m) {
    m.doc() = "Basic math operations module";

    // TODO: Bind your functions here
    // Example: m.def("add", &add, "A function that adds two integers");
    m.def("add", &add, "Add two integers",
          py::arg("a"), py::arg("b"));
```

- Performance considerations (TA demo)

```
======================================================================
  PERFORMANCE SUMMARY
======================================================================

Image Size          Blur            Edge Detection  Histogram
----------------------------------------------------------------------
Small (64x64)       1177.9x faster  2570.8x faster    53.0x faster
Medium (256x256)    1574.1x faster  2048.6x faster    78.7x faster
Large (512x512)     1484.3x faster  2369.2x faster    79.5x faster
```

Summary: Why C++ is Faster

Factor	Python	C++	Speedup
Compilation	Interpreted	Machine code	10-20x
Type checking	Runtime	Compile-time	5-10x
Memory layout	Pointer-heavy	Compact	2-5x
Loop overhead	High	Minimal	10-50x
Optimizations	Limited	Extensive	2-10x
Cache usage	Poor	Excellent	2-5x

Combined effect: 50-1000x faster!

Exercise

- Task 1: Binding C++ functions
- Task 2: Binding C++ class
- Task 3: STL containers (Also a C++ class)
- Expected output

```
(pybind11_env) (base) wilbur@arg07-ROG-Zephyrus-G16-GU605MZ-GU605MZ:~/aoop-lab6-pybind$ python3 Task1_TA/test_math_ops.py
8
10.0
120
(pybind11_env) (base) wilbur@arg07-ROG-Zephyrus-G16-GU605MZ-GU605MZ:~/aoop-lab6-pybind$ python3 Task2_TA/test_geometry.py
v1: (3.000000, 4.000000)
Length of v1: 5.0
v1 + v2: (4.000000, 6.000000)
Dot product: 11.0
(pybind11_env) (base) wilbur@arg07-ROG-Zephyrus-G16-GU605MZ-GU605MZ:~/aoop-lab6-pybind$ python3 Task3_TA/test_statistics.py
Count: 5
Mean: 30.0
Min: 10.0
Max: 50.0
Values: [10.0, 20.0, 30.0, 40.0, 50.0]
```

Thank you for listening