title: Frome Python To Modern C++ class: animation-fade layout: true

.bottom-bar[{{title}}: Lecture 1: Hello {fmt}]

class: center, middle

# Lecture 1: Hello {fmt}

## Hello Python

# file: hello.py  
# Run with:  
# $ python hello.py  
  
from math import sin  
  
a = 12 + 4  
print('The ans is {}.'.format(a))  
  
b = sin(0.4)  
print('The ans is {}.'.format(b))

How about in C++?

## Hello printf

// file: hello2.cpp  
// Compile with:  
// $ g++ hello2.cpp -o hello2  
  
#include <stdio.h> // for printf  
#include <cmath> // for std::sin()  
  
int main() {  
 auto a = 12 + 4; // C++11  
 printf("The ans is %d.\n", a); // %d for int  
  
 auto b = std::sin(0.4);  
 printf("The ans is %f.\n", b); // %f for double  
}

## Hello iostream

// file: hello1.cpp  
#include <iostream> // for std::cout  
#include <cmath> // for std::sin()  
  
int main() {  
 auto a = 12 + 4; // C++11  
 std::cout << "The ans is " << a << ".\n";  
  
 auto b = std::sin(0.4);  
 std::cout << "The ans is " << b << ".\n";  
}

## iostream, good or bad?

Good 👍: - More generic code.

Bad 👎: - The header file is too big. - The run time is usually slower than *printf()*. - The syntax is not readable.

## Hello to {fmt}

// file: hello.cpp  
// Compile with:  
// $ g++ hello.cpp -o hello -lfmt  
  
#include <fmt/core.h> // for fmt::print  
#include <cmath> // for std::sin()  
  
int main() {  
 auto a = 12 + 4;   
 fmt::print("The ans is {}.\n", a);  
  
 auto b = std::sin(0.4);  
 fmt::print("The ans is {}.\n", b);  
}

## Hello container (Python)

# file: hello.py  
a = 12 + 4  
print('The answer is {}'.format(a))  
  
L = [2, 3, 5, 7, 11, 'hello']  
print('The answer is {}'.format(L))

## Hello {fmt} (C++)

// file: hello.cpp  
// Compile with:  
// $ g++ -std=c++17 hello.cpp -lfmt  
  
#include <vector>  
\*#include <fmt/ranges.h>  
  
int main() {  
 auto a = 12 + 4;  
 fmt::print("The ans is {}.\n", a);  
  
 auto L = std::vector{2, 3, 5}; // c++17  
 fmt::print("The ans is {}.\n", L);  
}

## Tuple in Python

def main():  
 T = ("a", "u", "e", 7, 3.4)  
 print(T)  
   
 for i, e in enumerate(T):  
 print("{}: {}, ".format(i, e))  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

## Tuple in C++ 😱😱😱😱

#include <fmt/ranges.h>  
#include <tuple> // tuple is not container  
  
template <std::size\_t I = 0, typename... Tp>  
void print\_test(const std::tuple<Tp...>& t) {  
 if constexpr (I < sizeof...(Tp)) {  
 fmt::print("{}: {}, \n", I, std::get<I>(t));  
 print\_test<I + 1, Tp...>(t);  
 }  
}  
int main() {  
 auto t = std::tuple{"a", "u", "e", 3, 4.5};  
 fmt::print("{}\n", t);  
 print\_test(t);  
}

## Minimum Environment Setup 🔧

* Lubuntu 20.04 LTS:
  + python3 built-in
  + sudo apt install g++
  + sudo apt install libfmt-dev
* Android termux:
  + pkg install python
  + pkg install clang
  + pkg install fmt

## Conclusion

* Almost Always Auto (AAA)
* Modern C++ is faster, safer, and easier to use.
* Modern C++ is more python-like

## Coming up 🔜

* Type checking in python
* None vs. std::optional
* Essential containers
  + list vs. std::vector
  + set vs. std::unordered\_set
  + dict vs. std::unordered\_map
* range base for-loop vs. range-v3
  + zip, enumerate
* numpy vs. xtensor
* pytest vs. doctest
* and more…

class: center, middle

# Q & A 🗣️