Introduction to C++

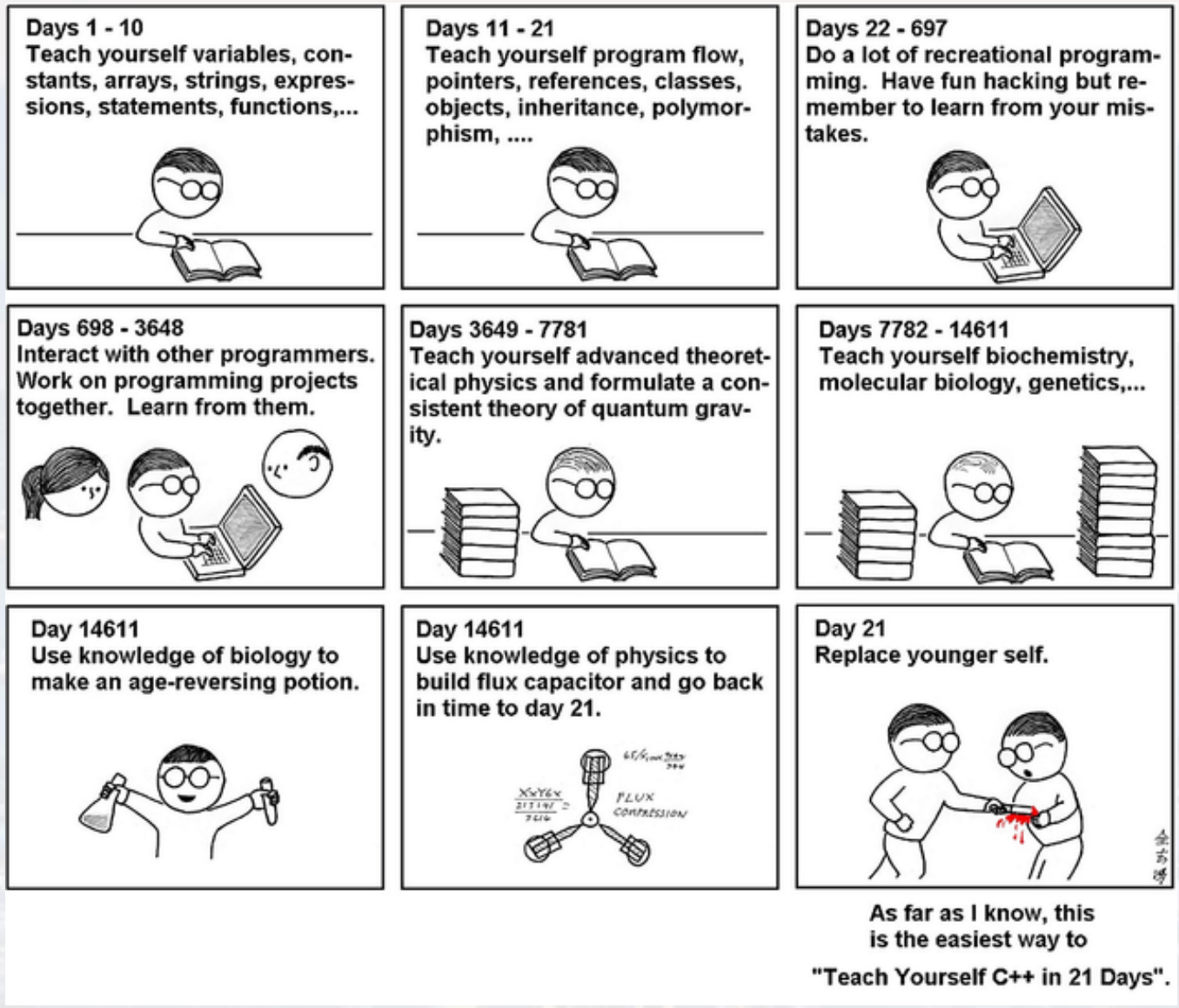
Jessica Nash

University California, Berkeley

Python for Molecular Sciences

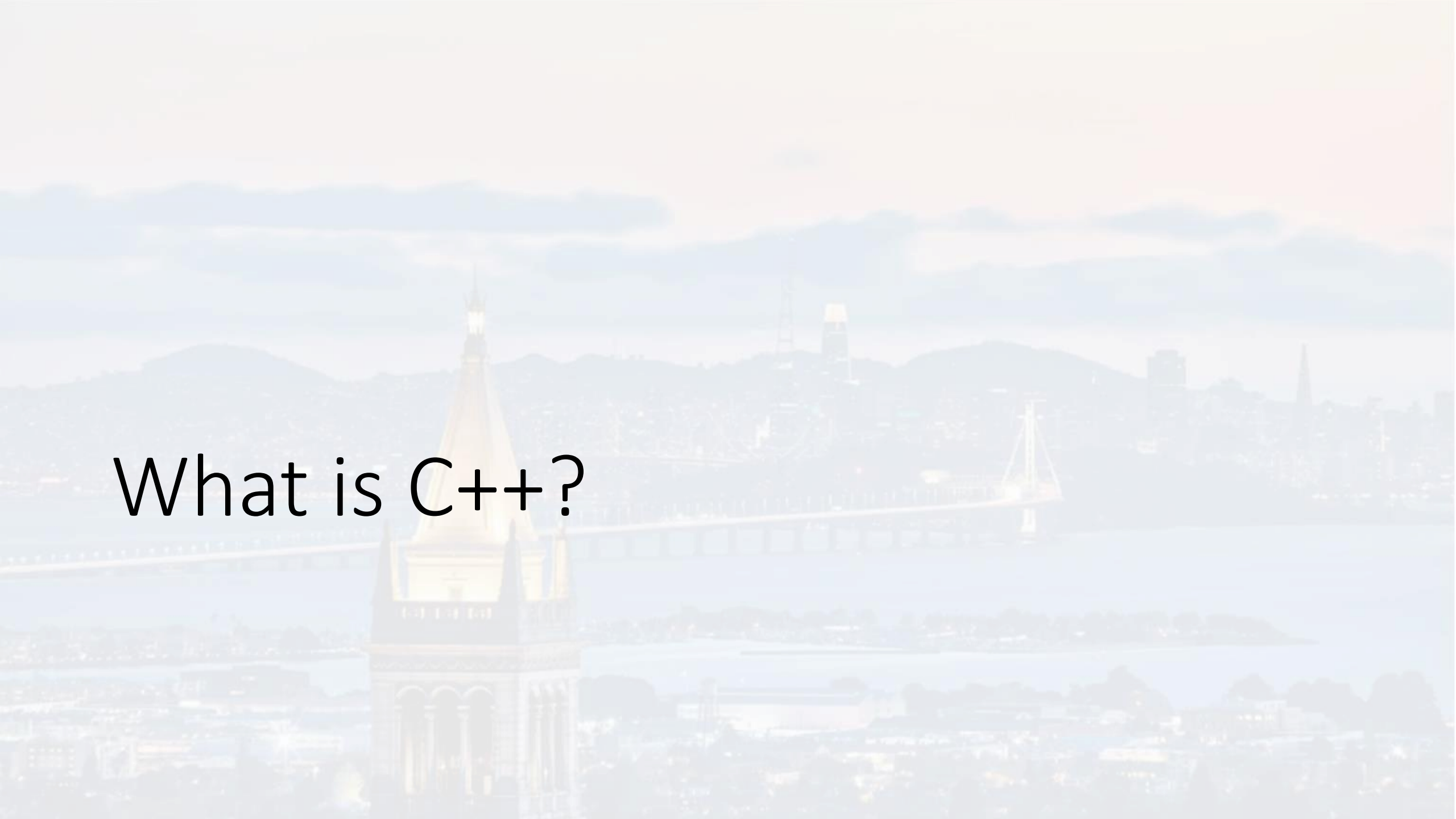
MSSE 272, 3 Units

Berkeley MSSE
MOLECULAR SCIENCE & SOFTWARE ENGINEERING



Outline

- What is C++?
- C++ Popularity
- Why C++?
- Why not C++?
- Python Program Execution
- C++ Program Execution
- Anatomy of a C++ Program
- Compiling and Executing
- Static and Dynamic Typing
- Comparison of Python and C++



What is C++?

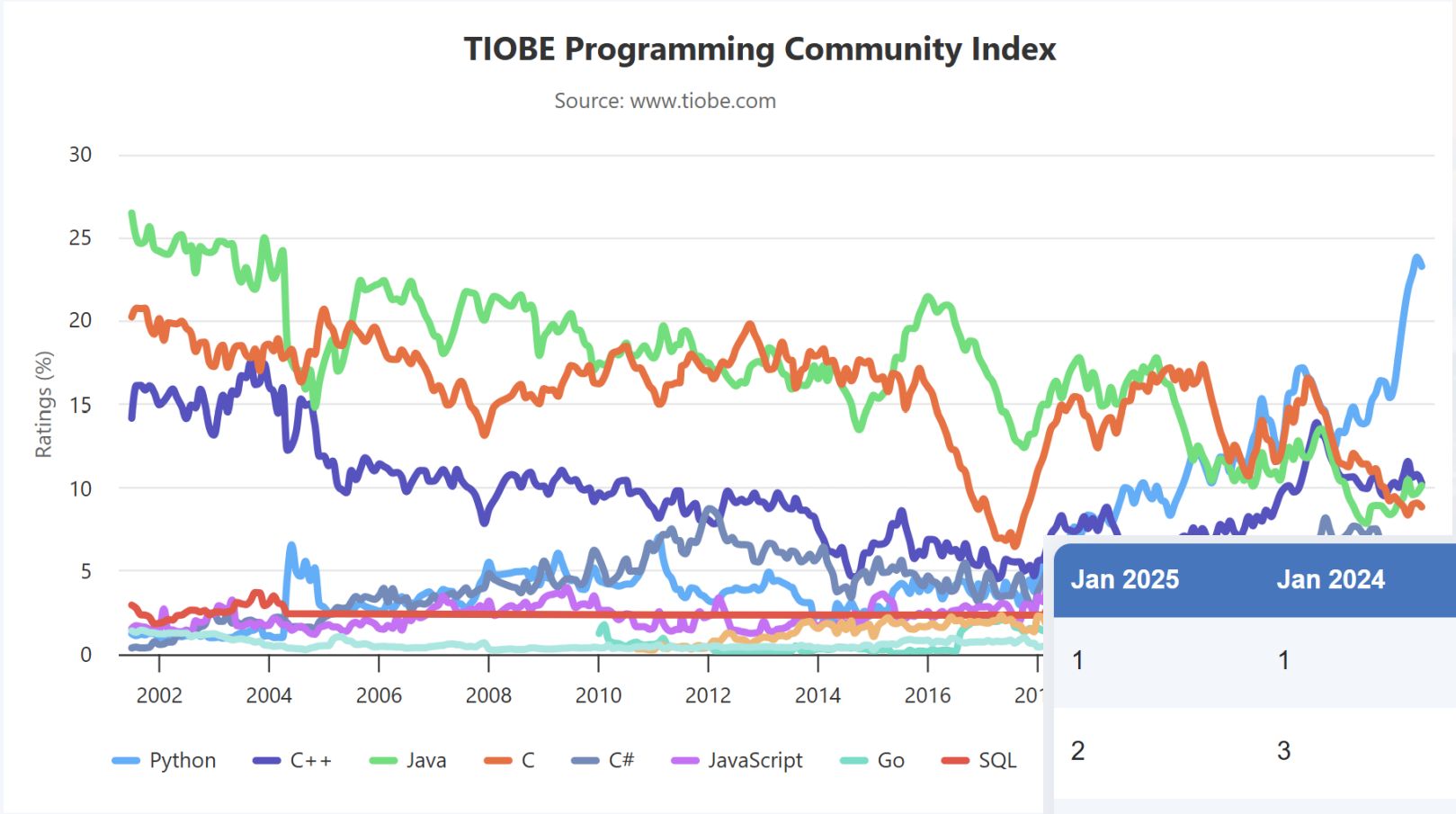
- C++ is a general purpose, **high-performance** programming language.
- C++ was developed as an extension of the C programming language in 1979 by Bjarne Stroustrup. First released to the public in 1983.
- In contrast to Python, C++ is **compiled** and **statically typed**.
  - Python is **interpreted** and **dynamically typed**.
- In contrast to Python, C++ often involves **manual memory management**.

- C++ is a general purpose, **high-performance** programming language.

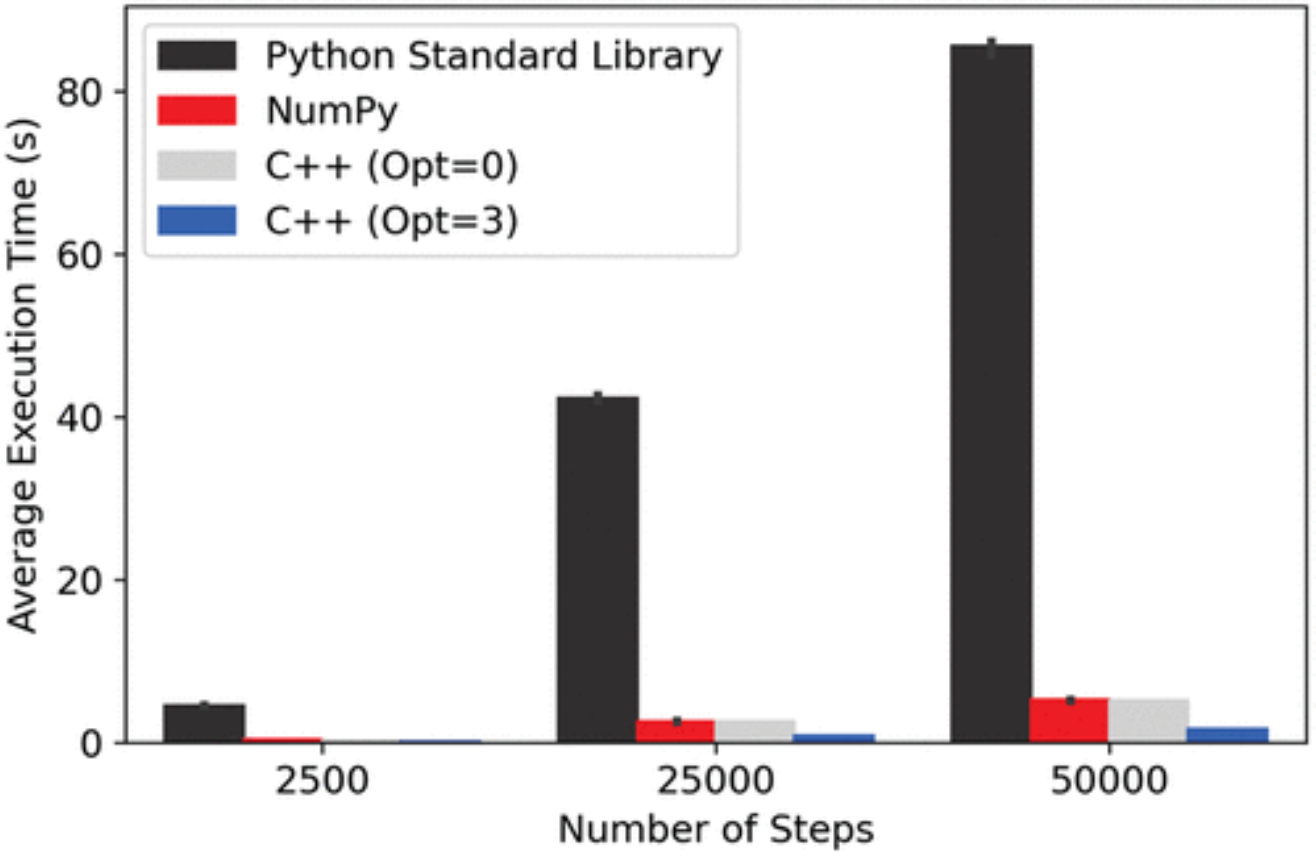
- C++ was developed as an extens
  programming language in 1979 b
  Stroustrup. First released to the
- In contrast to Python, C++ is **com**
  statically typed.
  - Python is **interpreted** and **dy**
    typed.
- In contrast to Python, C++ often involves **manual memory management**.

- C++ is a general purpose, **high-performance** programming language.
- C++ was developed as an extension of the C programming language in 1979 by Bjarne Stroustrup. First released to the public in 1983.
- In contrast to Python, C++ is **compiled** and **statically typed**.
  - Python is **interpreted** and **dynamically typed**.
- In contrast to Python, C++ often involves **manual memory management**.

Python for Molecular Sciences: C++ Popularity



Jan 2025	Jan 2024	Change	Programming Language	Ratings
1	1		Python	23.28%
2	3	▲	C++	10.29%
3	4	▲	Java	10.15%
4	2	▼	C	8.86%
5	5		C#	4.45%



Plot shows run time for Monte Carlo simulation written using the Python Standard Library (black), Python with NumPy (NumPy runs C on the backend), and with C++ with different optimization levels.

- C++ is commonly used in high-performance applications.
- It is dramatically faster than standard Python.

What's the catch?

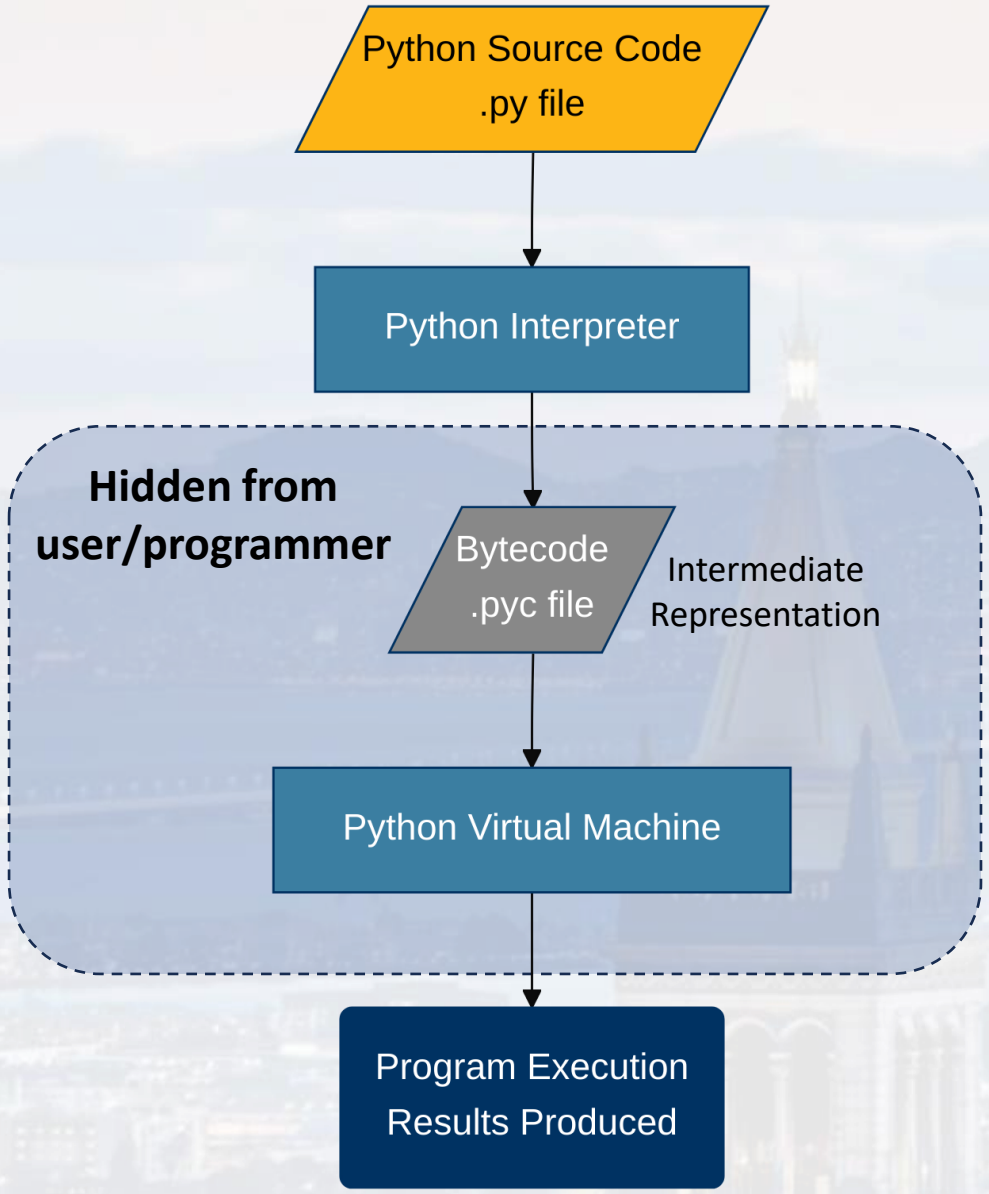
- C++ is harder to write because it requires careful attention to types and memory and it also has a longer development cycle because programs are compiled.

Python for Molecular Sciences: Why not C++?

```python
numbers = [1, 2, 3, 4, 5]
total = 0
for num in numbers:
    total += num
mean = total / len(numbers)
print(mean)
```

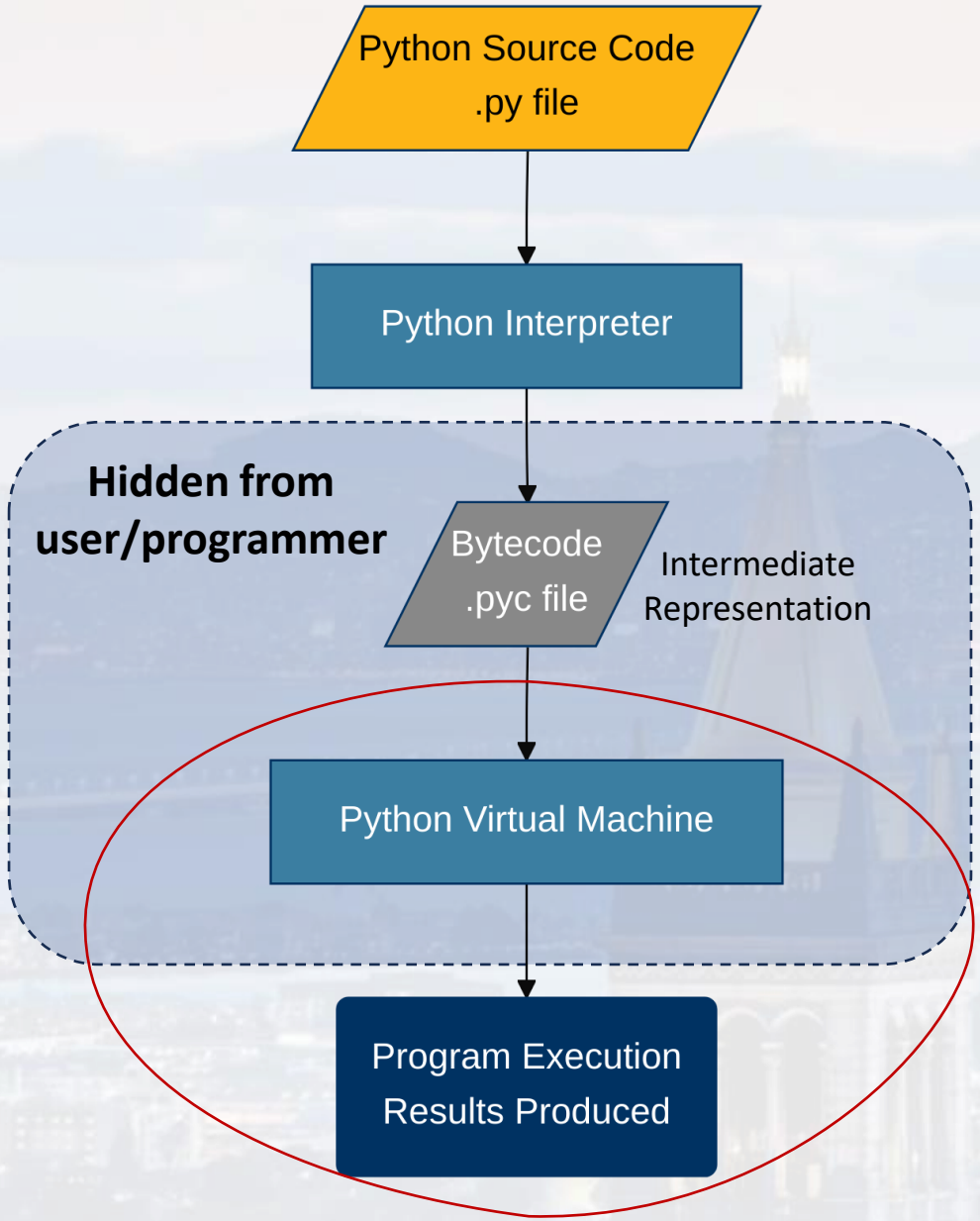
```cpp
#include <iostream>
#include <vector>

int main() {
    std::vector<double> numbers = {1, 2, 3, 4, 5};
    double total = 0;
    for(int i = 0; i < numbers.size(); i++) {
        total += numbers[i];
    }
    double mean = total / numbers.size();
    std::cout << mean << std::endl;
    return 0;
}
```

We'll learn more about C++ program structure later!



How Python code runs:

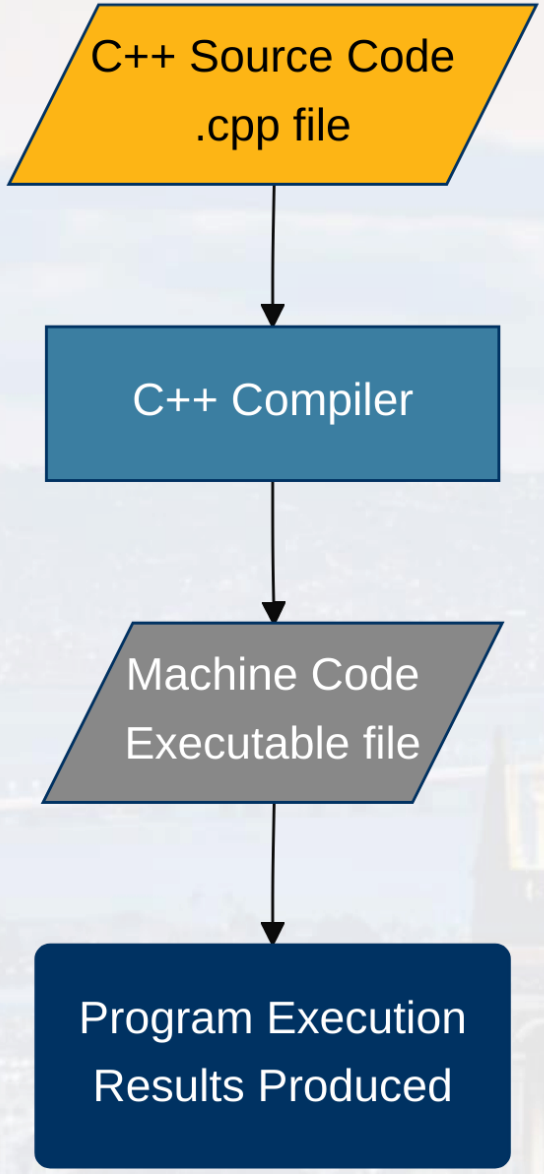
1. A programmer writes Python source code in a .py file.
2. When the programmer executes the code (python script.py), the Python interpreter translates the source code to bytecode.
   a) Bytecode
   b) This is saved in a hidden folder called __pycache__ and has the extension .pyc
3. The bytecode is executed by the Python Virtual Machine.



How Python code runs:

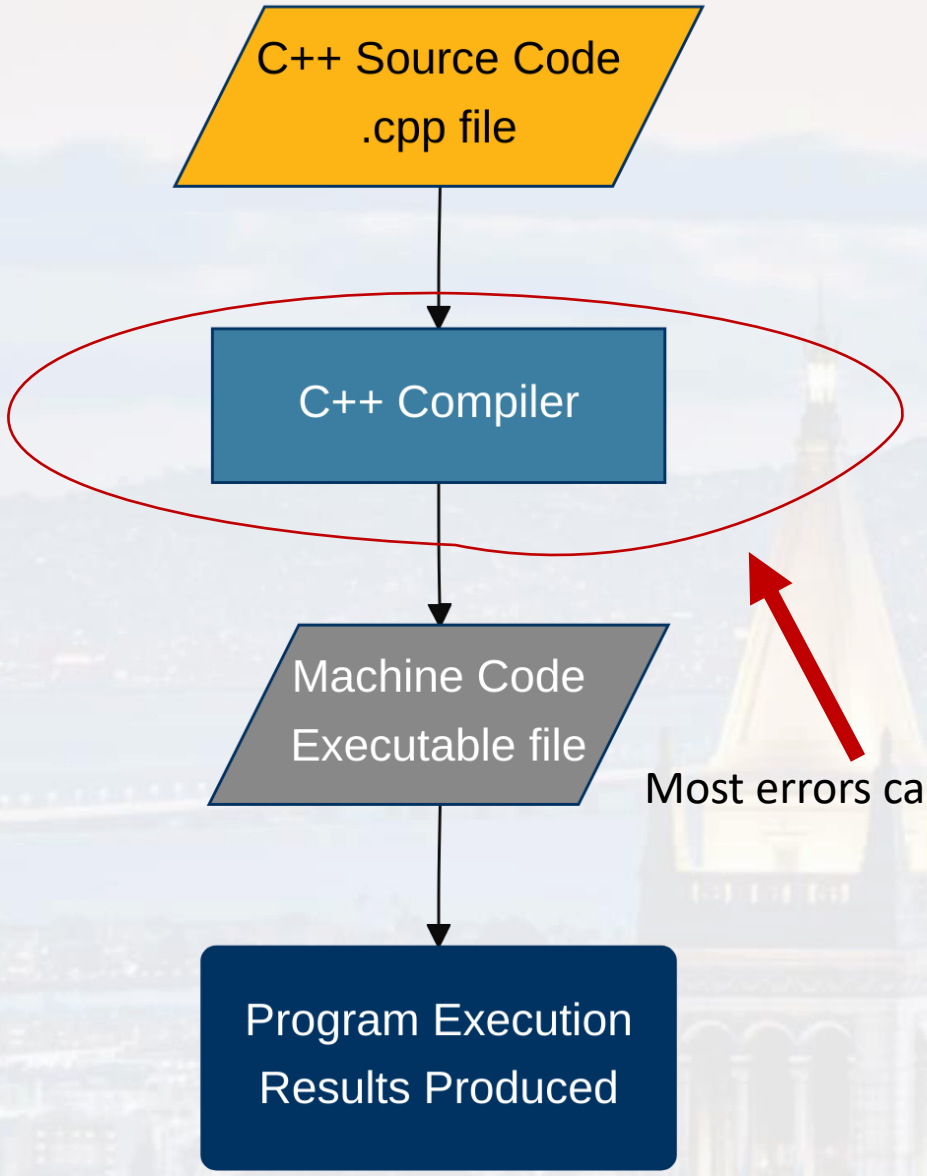
1. A programmer writes Python source code in a .py file.
2. When the programmer executes the code (python script.py), the Python interpreter translates the source code to bytecode.
   a) Bytecode
   b) This is saved in a hidden folder called __pycache__ and has the extension .pyc
3. The bytecode is executed by the Python Virtual Machine.

Most errors caught here!



How C++ code runs:

1. A programmer writes C++ source code in a .cpp file.
2. When the programmer is ready to run the code, they compile the program (using a compiler).
   a) The compiler prepares a file with machine code. This is called an **executable.**
3. The programmer runs (or "executes") the executable to get results.



How C++ code runs:

1. A programmer writes C++ source code in a .cpp file.
2. When the programmer is ready to run the code, they compile the program (using a compiler).
   a) The compiler prepares a file with machine code. This is called an **executable.**
3. The programmer runs (or "executes") the executable to get results.

First C++ Program

The syntax of C++ can look similar to that of Python, with a few differences.

- Single-line comments start with `//`.
- Multi-line comments start with `/*` and end with `*/`
- Whitespace is not significant
- Statements end with a semicolon (`;`)
- Rather than whitespace, statements are grouped together with curly braces
- A type is specified before the first use of a variable (including in a function argument)
- The return type of a function is specified before the function name; the function must return an object of that type and only that type.

The syntax of C++ can look similar to that of Python, with a few differences.

- Single-line comments start with `//`.
- Multi-line comments start with `/*` and end with `*/`
- Whitespace is not significant
- Statements end with a semicolon (`;`)
- Rather than whitespace, statements are grouped together with curly braces
- A type is specified before the first use of a variable (including in a function argument)
- The return type of a function is specified before the function name; the function must return an object of that type and only that type.

Let's see this in action!

```cpp
#include <iostream>

int main()
{
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```

What would you guess this program does?

```cpp
#include <iostream>

int main()
{
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```

What would you guess this program does?

It prints “Hello, World!”

```cpp
#include <iostream>

int main()
{
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```

Similar to Python import statement. In this cause, this lets us use std::cout (used for printing). iostream has functions related to input and output.

It is part of the C++ Standard Library.

```cpp
#include <iostream>

int main()
{
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```

Every C++ program **must** have a defined main function. The main function is where the program starts to run.

“int main” means that main will return an integer.

```cpp
#include <iostream>

int main()
{
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```

In C++ function beginnings and ends (as well as if statements and for loops) are marked with { }.

Contrast this with Python where this is indicated with indentation.

```cpp
#include <iostream>

int main()
{
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```

This is the function body.

```cpp
#include <iostream>

int main()
{
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```

This is the equivalent of print("Hello, World!") in Python.

Cout (pronounced "c-out") is for printing to the screen. "std::cout" indicates that cout is in the "standard namespace". cout comes from iostream that we imported.

std::endl represents a newline character when printing. endl also comes from iostream

```cpp
#include <iostream>

int main()
{
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```

Statements in C++ **must** end with a semicolon.

If not, an error will occur at compilation.

```cpp
#include <iostream>

int main()
{
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```

To end the program the main function has a return statement.

```cpp
#include <iostream>

int main()
{
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```

To end the program the main function has a return statement.

The return type matches the function signature. By convention, returning 0 from main means that the program executed successfully.

How do we run it?

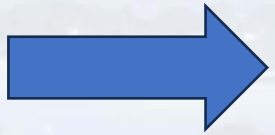
```cpp
#include <iostream>

int main()
{
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```

How do we run it?

```cpp
#include <iostream>

int main()
{
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```

Save text file.

hello_world.cpp



Compile and execute

Open the terminal

Compile and execute in your terminal

```
user@computer:~$ g++ hello_world.cpp -o hello_world
user@computer:~$ ./hello_world
Hello, World!
user@computer:~$
```

Compile and execute in your terminal

```
user@computer:~$ g++ hello_world.cpp -o hello_world
user@computer:~$ ./hello_world
Hello, World!
user@computer:~$
```

Compile the program

Compile and execute in your terminal

```
user@computer:~$ g++ hello_world.cpp -o hello_world
user@computer:~$ ./hello_world
Hello, World!
user@computer:~$
```

Compile the program. This produces an executable file called "hello_world".

A Closer Look at the Compile Command

```
g++ hello_world.cpp -o hello_world
```

Command Part	Purpose
g++	The C++ compiler program. g++ is the GNU compiler.
hello_world.cpp	Input source code file (has your C++ code)
-o	Option flag that means "output to a file"
hello_world	Name of the executable file to create

Compile and execute in your terminal

```
user@computer:~$ g++ hello_world.cpp -o hello_world
user@computer:~$ ./hello_world
Hello, World!
user@computer:~$
```

Run the executable. The syntax is "./executable_name"

Compile and execute in your terminal

```
user@computer:~$ g++ hello_world.cpp -o hello_world
user@computer:~$ ./hello_world
Hello, World!
user@computer:~$
```

Program output.

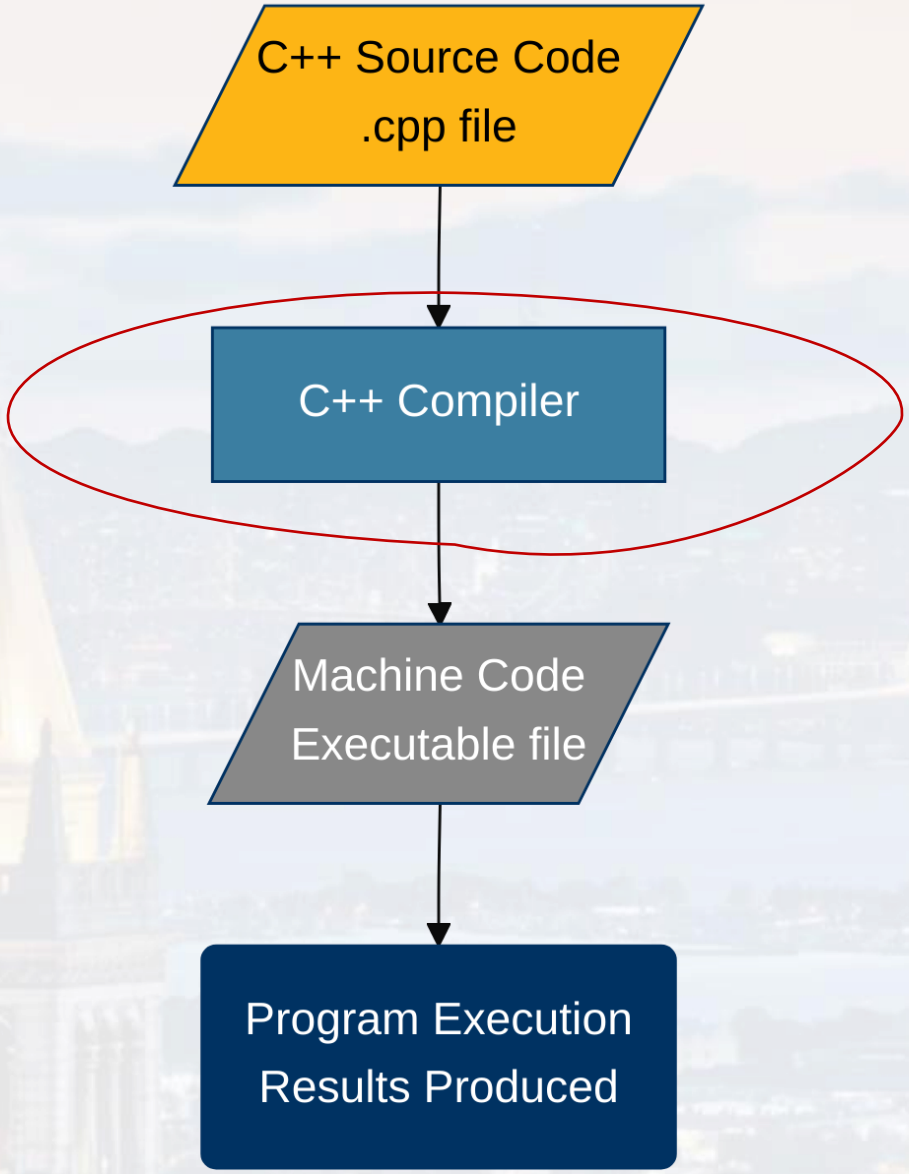
What about that “return 0”?

```
user@computer:~$ g++ hello_world.cpp -o hello_world
user@computer:~$ ./hello_world
Hello, World!
user@computer:~$ echo $?
0
```

Query the exit code. We will see the 0. This comes from the return 0 in our program. We see this because it exited successfully.

Reading Compiler Errors

Reminder: C++ Program Execution



Compiler Errors

- In C++ you will often encounter errors at the compilation step.

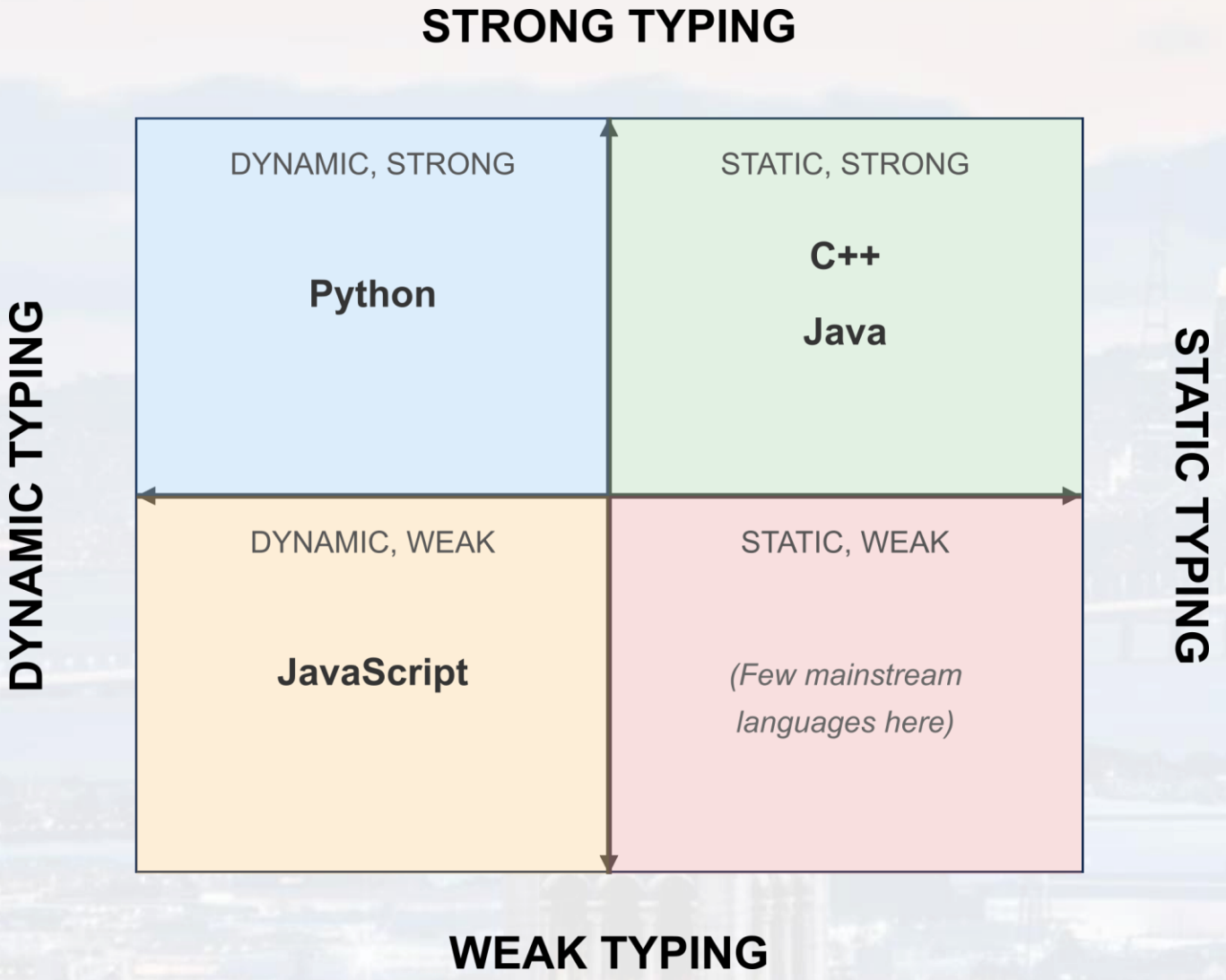
- Consider the error message to the right.

- "hello.cpp:5" → Error is in file "hello.cpp", line 5.
- The arrow ^ indicates where the error occurs.
- The first line tells us we are missing a semicolon

```
hello.cpp: In function ‘int main()’:
hello.cpp:5:45: error: expected ‘;’ before ‘return’
   5 |     std::cout << "Hello, World" << std::endl
     |                                             ^
     |                                             ;
   6 |
   7 | return 0;
     | ~~~~~~
```

- Always read the first line first.
- Look for missing semicolons and curly braces

C++ Data Types



Two independent axes define how programming languages handle variables:

- **Static vs. Dynamic Typing** – When is a variable's type known? (runtime or compile time?) Can the variable's data type change after a variable's initial declaration (in Python yes, in C++, no)
- **Strong vs. Weak Typing** – How strictly are types enforced?

```python
x = 5           # x is an integer
x = "hello"     # Now x is a string
x = [1, 2, 3]   # Now x is a list
```

- C++ is statically typed. This means you must specify the type of a variable when it is declared.
- In C++ the type of a variable cannot change.
- Python infers the type of a variable based on the variable value.
- Python uses dynamic typing - variables can change type during program execution.

```cpp
// Type must be declared and cannot change
int x = 5;          // x must always be an integer
x = "hello";        // Error! Can't change type
vector<int> y;      // y must always be a vector of integers
```

```python
x = 5           # x is an integer
x = "hello"     # Now x is a string
x = [1, 2, 3]   # Now x is a list
```

- C++ is statically typed. This means you must specify the type of a variable when it is declared.
- In C++ the type of a variable cannot change.
- Python infers the type of a variable based on the variable value.
- Python uses dynamic typing - variables can change type during program execution.

```cpp
// Type must be declared and cannot change
int x = 5;          // x must always be an integer
x = "hello";        // Error! Can't change type
vector<int> y;      // y must always be a vector of integers
```

In C++ we have to think about the data type of a variable AND its possible value. Each variable takes up a certain amount of computer memory and that dictates what variable values can be.

Basic Types

- **Character types:** They can represent a single character, such as 'A' or '$'. The most basic type is char, which is a one-byte character. Other types are also provided for wider characters.
- **Numerical integer types:** They can store a whole number value, such as 7 or 1024. They exist in a variety of sizes, and can either be signed or unsigned, depending on whether they support negative values or not.
- **Floating-point types:** They can represent real values, such as 3.14 or 0.01, with different levels of precision, depending on which of the three floating-point types is used.
- **Boolean type:** The boolean type, known in C++ as bool, can only represent one of two states, true or false.

https://cplusplus.com/doc/tutorial/variables/#fundamental

Python for Molecular Sciences: Fundamental C++ Data Types

Group	Type names*	Notes on size / precision
Character types	**char**	Exactly one byte in size. At least 8 bits.
	char16_t	Not smaller than `char`. At least 16 bits.
	char32_t	Not smaller than `char16_t`. At least 32 bits.
	wchar_t	Can represent the largest supported character set.
Integer types (signed)	**signed char**	Same size as `char`. At least 8 bits.
	signed **short** *int*	Not smaller than `char`. At least 16 bits.
	signed **int**	Not smaller than `short`. At least 16 bits.
	signed **long** *int*	Not smaller than `int`. At least 32 bits.
	signed **long long** *int*	Not smaller than `long`. At least 64 bits.
Integer types (unsigned)	**unsigned char**	(same size as their signed counterparts)
	unsigned short *int*	
	unsigned *int*	
	unsigned long *int*	
	unsigned long long *int*	
Floating-point types	**float**	
	double	Precision not less than `float`
	long double	Precision not less than `double`
Boolean type	**bool**	
Void type	**void**	no storage
Null pointer	**decltype(nullptr)**	

https://cplusplus.com/doc/tutorial/variables/#fundamental

Python for Molecular Sciences: Fundamental C++ Data Types

Group	Type names*	Notes on size / precision
Character types	**char**	Exactly one byte in size. At least 8 bits.
	char16_t	Not smaller than `char`. At least 16 bits.
	char32_t	Not smaller than `char16_t`. At least 32 bits.
	wchar_t	Can represent the largest supported character set.
Integer types (signed)	**signed char**	Same size as `char`. At least 8 bits.
	signed **short** *int*	Not smaller than `char`. At least 16 bits.
	signed **int**	Not smaller than `short`. At least 16 bits.
	signed **long** *int*	Not smaller than `int`. At least 32 bits.
	signed **long long** *int*	Not smaller than `long`. At least 64 bits.
Integer types (unsigned)	**unsigned char**	(same size as their signed counterparts)
	unsigned short *int*	
	unsigned *int*	
	unsigned long *int*	
	unsigned long long *int*	
Floating-point types	**float**	
	double	Precision not less than `float`
	long double	Precision not less than `double`
Boolean type	**bool**	
Void type	**void**	no storage
Null pointer	**decltype(nullptr)**	

These are guaranteed sizes and can be taken as “minimums”.

Most modern 64-bit systems will be a bit larger – for example your computer likely uses 4 bytes (32 bits) for int and 8 bytes (64 bits) for long int.

https://cplusplus.com/doc/tutorial/variables/

Python for Molecular Sciences: Fundamental C++ Data Types

Group	Type names*	Notes on size / precision
Character types	**char**	Exactly one byte in size. At least 8 bits.
	char16_t	Not smaller than `char`. At least 16 bits.
	char32_t	Not smaller than `char16_t`. At least 32 bits.
	wchar_t	Can represent the largest supported character set.
Integer types (signed)	**signed char**	Same size as `char`. At least 8 bits.
	signed **short** *int*	Not smaller than `char`. At least 16 bits.
	signed **int**	Not smaller than `short`. At least 16 bits.
	signed **long** *int*	Not smaller than `int`. At least 32 bits.
	signed **long long** *int*	Not smaller than `long`. At least 64 bits.
Integer types (unsigned)	**unsigned char**	(same size as their signed counterparts)
	unsigned short *int*	
	unsigned *int*	
	unsigned long *int*	
	unsigned long long *int*	
Floating-point types	**float**	
	double	Precision not less than `float`
	long double	Precision not less than `double`
Boolean type	**bool**	
Void type	**void**	no storage
Null pointer	**decltype(nullptr)**	

Consider 16 bit integer. This means a string of 16 0's and 1's represents the number in binary.

The range of possible values that can represented are $-2^{\wedge 15}$ to $2^{15} - 1$. The first bit (of the 16 total bits) is used to store the sign.

https://cplusplus.com/doc/tutorial/variables/

C++ Type	Python Type	Description
bool	bool	True/False
int	int	Integer number (not floating point)
double	float	Floating point number (64-bit)
std::string	str	String of characters
std::vector	list	List/array of data
std::map	dict	Key/Value association

C++ Integer Data Types

- Stores whole numbers (e.g., -10, 0, 42)
- The range of possible values for an int data type is determined by the number of bits.
- int, long int, short int all store integers, but vary in the amount of memory and range of values they can store.
- Typically int is 4 bytes
  - 4 bytes (8 bits each) = 32 bits
  - Possible values range from -2^{31} to 2^{31}
  - Range: -2,147,483,648 to 2,147,483,647

```cpp
#include <iostream>

int main() {
    int x = 42;
    int y = 19;
    int z = 4.5;

    // What do you think this will output?
    std::cout << "x is " << x << std::endl;
    std::cout << "y is " << y << std::endl;
    std::cout << "z is " << z << std::endl;
}
```

C++ Integer Data Types

- Stores whole numbers (e.g., -10, 0, 42)
- The range of possible values for an int data type is determined by the number of bits.
- int, long int, short int all store integers, but vary in the amount of memory and range of values they can store.
- Typically int is 4 bytes
  - 4 bytes (8 bits each) = 32 bits
  - Possible values range from -2^{31} to 2^{31}
  - Range: -2,147,483,648 to 2,147,483,647

```cpp
#include <iostream>

int main() {
    int x = 42;
    int y = 19;
    int z = 4.5;

    // What do you think this will output?
    std::cout << "x is " << x << std::endl;
    std::cout << "y is " << y << std::endl;
    std::cout << "z is " << z << std::endl;
}
```

It outputs 42, 19, and 4 – the decimal is discarded when z is declared as an int. The decimal is truncated, not rounded.

Decimal Data Types: Float and Double

- Stores numbers with decimal points (e.g., 3.14, -0.001, etc)

- `float` (also called single precision)
  - Typically 4 bytes (32 bits)

- `double` (also called double precision)
  - Typically 8 bytes

- `double` is more precise than `float` and preferred for most scientific applications.

- `double` is equivalent to Python's `float` type.

```c
int main() {
    // 'f' at the end makes it a float
    float a = 3.141592653589793f;

    double b = 3.141592653589793;

    return 0;
}
```

- Stores true/false values
- Uses 1 byte of memory
- Used for: Making decisions in programs
- Storing results of comparisons
  - Is something equal? Greater than?
- Tracking states
  - is game over? is logged in?

```
int main() {

    double temperature = 31;
    double score = 95;

    bool is_freezing = temperature < 32;
    bool has_passed = score >= 70;

return 0;
}
```

Python/C++ Comparison

Python for Molecular Sciences: Overview Comparison of Python and C++

Characteristic	Python	C++
When errors are found	During execution	During compilation
Development speed	Quick to write and test	More setup/compilation time
Code preparation	Runs immediately	Must compile first
Error checking	Finds errors when code runs	Finds errors at compilation (before running)
Variable typing	Dynamic - type determined at runtime	Static - must declare types
Type checking	Types checked during execution	Types checked during compilation
Performance	Slower - interpreted line by line	Faster - runs directly as machine code

Python for Molecular Sciences:

Thank you very much for your attention