Python → C++

What’s the difference?

- Syntax
- Types
- Compiled
- Scope
- Objects

- Raw memory access
- Garbage collection

Syntax

Python

```python
def hello():
  if True:
    print("Hello world!")
```

C++

```cpp
void hello() {
  if (true) {
    cout << "Hello world!" << endl;
  }
}
```

Types

Python

```
num = 42
```

C++

```
int num = 42;
```

Types

Python

```python
num = 42

percent = 0.42
```

C++

```cpp
int num = 42;

double percent = 0.42;
```

Types

Python

```python
num = 42

percent = 0.42

isValid = True
```

C++

```cpp
int num = 42;

double percent = 0.42;

bool isValid = true;
```

Types

Python

```python
num = 42

percent = 0.42

isValid = True

ch = "A" or ch = 'A'
```

C++

```cpp
int num = 42;

double percent = 0.42;

bool isValid = true;

char ch = 'A';
```

Types

Python

```python
num = 42

percent = 0.42

isValid = True

ch = "A" or ch = 'A'
```

C++

```cpp
int num = 42;

double percent = 0.42;

bool isValid = true;

char ch = 'A';
```

There's a difference!

Types

Python

```
num = 42

percent = 0.42

isValid = True

ch = “A” or ch = ‘A’

greeting = "hello there!"
```

C++

```
int num = 42;

double percent = 0.42;

bool isValid = true;

char ch = 'A';

string greeting = "hello there!";
```

Types

Python

No static types!

C++

Static types

`int`

`double`

`bool`

`char`

`string`

Compiled

- Checks syntax
- Checks static types
- Performs optimization
- Creates an executable

Compiled

- Checks syntax
- Checks static types
- Performs optimization
- Creates an executable

Type Errors

Python

- Type errors happen at **run time**

C++

- Type errors happen at **compile time**

More Type Examples -- Lists/Vectors

Python

```python
nums = []
for i in range(42):
  nums.append(i)
```

or

```python
nums = [i for i in range(42)]
```

C++

```cpp
Vector<int> nums;
for (int i = 0; i < 42; i++) {
  nums.add(i);
}
```

More Type Examples -- Lists/Vectors

Python

```python
nums = []
for i in range(42):
  nums.append(i)
```

or

```python
nums = [i for i in range(42)]
```

C++

```cpp
Vector<int> nums;
for (int i = 0; i < 42; i++) {
  nums.add(i);
}
```

More documentation:
https://stanford.edu/~stepp/cppdoc/Vector-class.html

Writing methods

Python

```python
def isEven(num):
  return num % 2 == 0


def printGreeting():
  print("Hello CS106B!")
```

C++

```cpp
bool isEven(int num) {
  return num % 2 == 0;
}

void printGreeting() {
  cout << "Hello CS106B!" << endl;
}
```

Writing methods - multiple return values

Python

```
def modTwoAndThree(num):
  return (num % 2, num % 3)
```

C++

Can’t do this in the same way*

*but there is a way

Passing by Reference

Python

```
def appendOne(lst):
  lst.append(1)

lst = []
appendOne(lst)
print(lst)

Output:
[1]
```

C++

```
void appendOne(Vector<int> v) {
  v.add(1);
}
Vector<int> v;
appendOne(v);
cout << v << endl;

Output:
{}
```

Passing by Reference

Python

```
def appendOne(lst):
  lst.append(1)

lst = []
appendOne(lst)
print(lst)

Output:
[1]
```

C++

```
void appendOne(Vector<int> v) {
  v.add(1);
}
Vector<int> v;
appendOne(v);
cout << v << endl;

Output:
{}
```

Passes a copy of v!

Passing by Reference

Python

```
def appendOne(lst):
  lst.append(1)

lst = []
appendOne(lst)
print(lst)

Output:
[1]
```

C++

```
void appendOne(Vector<int> &v) {
  v.add(1);
}
Vector<int> v;
appendOne(v);
cout << v << endl;

Output:
{1}
```

Passes a reference to v (changes persist)

Cleanest way to do multiple return values

Python

```python
def modTwoAndThree(num):
  return (num % 2, num % 3)


a, b = modTwoAndThree(10)
```

C++

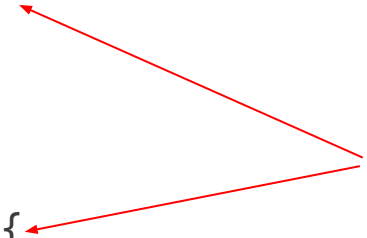
```cpp
void modTwoAndThree(int num,
      int &modTwo, int &modThree) {
  modTwo = num % 2;
  modThree = num % 3;
}
int a, b;
modTwoAndThree(10, a, b);
```

Note: C++ requires method prototypes

```
bool isEven(int num);
void printGreeting();
...
bool isEven(int num) {
  return num % 2 == 0;
}
void printGreeting() {
  cout << "Hello CS106B!" << endl;
}
```

Note: C++ uses method prototypes

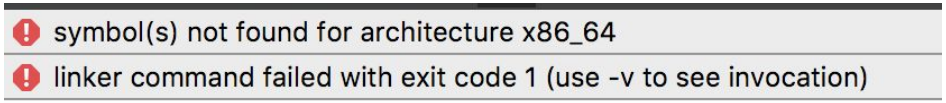
```
bool isEven(int num);
void printGreeting();
...
bool isEven(int num) {
  return num % 2 == 0;
}
void printGreeting() {
  cout << "Hello CS106B!" << endl;
}
```



Need to match exactly!!

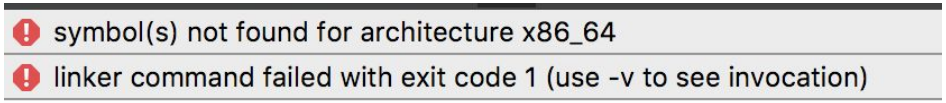
Note: C++ uses method prototypes

```
void isEven(int num);
void printGreeting();

...

bool isEven(int num) {
  return num % 2 == 0;
}
void printGreeting() {
  cout << "Hello CS106B!" << endl;
}
```



Note: C++ uses method prototypes

```
bool isEven(int &num);
void printGreeting();
...
bool isEven(int num) {
  return num % 2 == 0;
}
void printGreeting() {
  cout << "Hello CS106B!" << endl;
}
```



Scope

Python

```python
def myFunction():
  if True:
    greeting = "hello"

  print(greeting)
```

C++

```cpp
void myFunction() {
  if (true) {
    string greeting = "hello";
  }
  cout << greeting << endl;
}
```

Scope

Python

```python
def myFunction():
  if True:
    greeting = "hello"

  print(greeting)
```

C++

```cpp
void myFunction() {
  if (true) {
    string greeting = "hello";
  }
  cout << greeting << endl; // error
}
```

File Reading

Python

```python
def longestLineLength(filename):
  with open(filename) as f:
    max = 0
    for line in f:
      if len(line) > max:
        max = len(line)
    return max
```

C++

```cpp
int longestLineLength(filename) {
  ifstream f;
  openFile(f, filename); //from filelib.h
  int max = 0;
  string line;
  while (getline(f, line)) {
    if (line.length() > max) {
      max = line.length();
    }
  }
  return max;
}
```

Getting user input

```
#include "simpio.h"
string filename = promptUserForFile("Give me a filename! ") //from filelib.h
int n = getInteger("Give me an integer! ");
double x = getReal("Give me a double! ");
string line = getLine("Give me a string! ");
if (getYesOrNo("Continue? ")) {...
```

More documentation:
https://stanford.edu/~stepp/cppdoc/filelib.html
https://stanford.edu/~stepp/cppdoc/simpio.html

Garbage collection and raw memory access

- Clean up your own memory!
- C++ requires that you explicitly destroy the memory you create
- But not the main focus of CS 106B

Why learn C++??

- Simplicity comes at a cost!
  - C++ is much more memory efficient and faster
- Being able to directly access and modify memory is powerful, and avoids a lot of overhead!

Good luck this quarter!