**Self-Paced Beginner Coding Course (Python & C++)**

**Instructor Information**

This course is led by instructors with backgrounds in computer science, software engineering, and real-world military applications of technology.

* **Instructor 1: Carson Denison**
  + B.S. in Computer Science
  + 117 ARW Innovator
  + Specializes in AI, data transformation, and distributed systems
* **Instructor 2: Desmond Harris**
  + CCNA
  + 10 yr Software Engineer
  + Specializes in Networks, Back-end development and image processing

**Lesson 1: Hello World & Setup**

**Objective**

Learn how to install and run basic Python and C++ programs. Get familiar with your coding environment.

**Key Concepts**

* **Syntax**: Refers to the correct structure of code. In Python, indentation is mandatory; in C++, semicolons and braces define blocks.
* **Logic**: The way instructions are ordered and executed. Printing 'Hello, world!' confirms your environment is working.
* **Application**: This basic output demonstrates that your system is set up and code can run successfully.
* **Comparison between Python and C++**: Python uses a simple function call. C++ uses stream-based output and requires headers and main function.

**Example Code**

**Python:**

print('Hello, world!')

**C++:**

std::cout << "Hello, world!";

**Practice Tasks**

- Write your own example based on this lesson.  
- Try modifying the example code.

**Lesson 2: Variables & Data Types**

**Objective**

Understand how to store data using variables. Learn about integers, floats, strings, and booleans.

**Key Concepts**

* **Syntax**: Python assigns variables with =, no types declared. C++ requires explicit types like int or string.
* **Logic**: Storing values lets you reuse data. Booleans help control program flow.
* **Application**: Variables represent information like numbers, names, or status flags.
* **Comparison between Python and C++**: Python is dynamically typed, C++ is statically typed and stricter.

**Example Code**

**Python:**

x = 5,

pi = 3.14,

name = "Alice",

is\_valid = True

**C++:**

int x = 5;

double pi = 3.14;

std::string name = "Alice";

bool isValid = true;

**Practice Tasks**

- Write your own example based on this lesson.  
- Try modifying the example code.  
- Document what each line does.

**Lesson 3: Conditionals**

**Objective**

Learn to use if-else logic to make decisions in your code.

**Key Concepts**

* **Syntax:** In Python, conditionals use colons and indentation. In C++, braces define blocks and conditions are inside parentheses.
* **Logic:** Conditionals allow code to react based on values (e.g., take different actions if x > 10).
* **Application:** Used to control flow: e.g., checking a user's age, comparing values.
* **Comparison:** Python uses indentation, while C++ uses curly braces. Python is more concise; C++ more explicit.

**Example Code**

**Python:**

if x > 10:

print('High')

**C++:**

if (x > 10) {

std::cout << 'High';

}

**Practice Tasks**

- Write your own example based on this lesson.  
- Try modifying the example code.  
- Document what each line does.

**Lesson 4: Loops**

**Objective**

Use loops to repeat actions in your code with for and while.

**Key Concepts**

* **Syntax:** Python uses for item in range and while loops with indentation. C++ uses for(init; cond; incr) and braces.
* **Logic:** Loops automate repetition, reducing code duplication.
* **Application:** Used for iterating over data, repeating tasks a set number of times.
* **Comparison:** Python is simpler; C++ offers more control over initialization and incrementing.

**Example Code**

**Python:**

for i in range(5):

print(i)

**C++:**

for (int i = 0; i < 5; ++i) {

std::cout << i;

}

**Practice Tasks**

- Write your own example based on this lesson.  
- Try modifying the example code.  
- Document what each line does.

**Lesson 5: Functions**

**Objective**

Learn to organize code using functions for reusability and clarity.

**Key Concepts**

* **Syntax:** Functions in Python use def. C++ uses return type, function name, and parameter types.
* **Logic:** Functions help reuse code and improve structure.
* **Application:** Useful for breaking code into small, testable parts (like greeting users).
* **Comparison:** C++ requires more detail (types), while Python is more flexible.

**Example Code**

**Python:**

def greet(name):

return 'Hi ' + name

**C++:**

std::string greet(std::string name) {

return 'Hi ' + name;

}

**Practice Tasks**

- Write your own example based on this lesson.  
- Try modifying the example code.  
- Document what each line does.

**Lesson 6: Lists & Dictionaries**

**Objective**

Work with collections of data. Lists and dictionaries (maps) are essential for handling multiple items.

**Key Concepts**

* **Syntax:** Lists and dictionaries in Python use [ ] and { }. C++ uses vectors and maps from STL.
* **Logic:** Useful for storing multiple items like records, inventories, etc.
* **Application:** Helps manage collections of data efficiently.
* **Comparison:** Python’s structures are built-in; C++ needs include directives and type specification.

**Example Code**

**Python:**

fruits = ["apple"];

info = {"name": "Alice"}

**C++:**

vector<string> fruits = {"apple"};

map<string, string> info = {{"name", "Alice"}}

**Practice Tasks**

- Write your own example based on this lesson.  
- Try modifying the example code.  
- Document what each line does.

**Lesson 7: File I/O & Error Handling**

**Objective**

Read and write to files and handle unexpected issues using try/except or if checks.

**Key Concepts**

* **Syntax:** Python uses try/except; C++ uses if (file.is\_open()) and includes headers.
* **Logic:** Handles potential problems like missing files or user errors.
* **Application:** Essential for building reliable programs.
* **Comparison:** Python provides easier error catching; C++ provides granular control with file stream objects.

**Example Code**

**Python:**

open('file.txt') with try/except

**C++:**

ifstream file('file.txt');

if (file.is\_open()) {...}

**Practice Tasks**

- Write your own example based on this lesson.  
- Try modifying the example code.  
- Document what each line does.

**Lesson 8: Final Project**

**Objective**

Combine what you’ve learned to build a useful program, like a to-do list that saves data.

**Key Concepts**

* **Syntax:** Combines many earlier concepts: functions, lists, conditionals.
* **Logic:** Builds toward a full working program—practice of integration.
* **Application:** This project cements learning and shows practical use.
* **Comparison:** Python supports rapid prototyping; C++ is more verbose but efficient for compiled systems.

**Example Code**

**Python:**

tasks = [];

def add\_task(t):

tasks.append(t)

**C++:**

vector<string> tasks;

void addTask(string t) {

tasks.push\_back(t);

}

**Practice Tasks**

- Write your own example based on this lesson.  
- Try modifying the example code.  
- Document what each line does.