

OFFICIAL COURSE STUDY GUIDE

---

# The Complete Harvard CS50: Computer Science & Python

---

*Two Harvard courses combined - from computational thinking to Python mastery.*

---

COMPUTER SCIENCE · BEGINNER · 4 MODULES · 40 HR 50 MIN

Course by CS50 · Curated and published by MegaCourse

**MEGACOURSE.APP**

## **The Complete Harvard CS50: Computer Science & Python: The Complete Study Guide**

2026 Edition

Published by MegaCourse · [megacourse.app](https://megacourse.app)

Learn Anything. Achieve Everything.

MegaCourse curates the world's best free courses into structured learning paths with study guides, practice material, and certificates. This guide accompanies the MegaCourse course "The Complete Harvard CS50: Computer Science & Python", taught by CS50. It is designed to be read alongside the video lessons and to serve as a standalone reference for review.

Study guide text copyright © 2026 MegaCourse. Course video content remains the property of its creators. You may print this guide for personal study. Redistribution or resale is not permitted.

Enroll, track progress, and earn your certificate at

**[megacourse.app/courses/the-complete-harvard-cs50-computer-science-and-python](https://megacourse.app/courses/the-complete-harvard-cs50-computer-science-and-python)**

# Contents

---

|                                                 |    |
|-------------------------------------------------|----|
| <i>About This Course</i>                        | 4  |
| CHAPTER 1      Computer Science Foundations     | 6  |
| CHAPTER 2      Computer Science - Part 2        | 16 |
| CHAPTER 3      Programming with Python - Part 1 | 21 |
| CHAPTER 4      Advanced Python Programming      | 27 |
| <i>Glossary</i>                                 | 35 |
| <i>Answer Key</i>                               | 38 |
| <i>Where to Go Next</i>                         | 42 |

## INTRODUCTION

# About This Course

---

This study guide accompanies "The Complete Harvard CS50: Computer Science & Python." This course combines two foundational Harvard programs. It moves students from computational thinking to Python mastery.

Computer science is essential in the modern world. It provides the skills to understand and build technology. This course equips you with those fundamental abilities.

MegaCourse has structured these renowned courses into a clear learning path. You will find 4 modules and 23 lessons, totaling about 40 hours and 50 minutes of instruction. The program includes learning outcomes, progress tracking, and a certificate of completion. This guide provides additional support for your study.

## Course Facts

|                          |                                                                                                                                                                                 |
|--------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>FORMAT</b>            | 4 modules · 23 lessons                                                                                                                                                          |
| <b>VIDEO INSTRUCTION</b> | About 40 hr 50 min                                                                                                                                                              |
| <b>LEVEL</b>             | Beginner                                                                                                                                                                        |
| <b>CATEGORY</b>          | Computer Science                                                                                                                                                                |
| <b>INSTRUCTOR</b>        | CS50                                                                                                                                                                            |
| <b>CERTIFICATE</b>       | Included with course completion on MegaCourse                                                                                                                                   |
| <b>COURSE HOME</b>       | <a href="https://megacourse.app/courses/the-complete-harvard-cs50-computer-science-and-python">megacourse.app/courses/the-complete-harvard-cs50-computer-science-and-python</a> |

## What You Will Learn

- Build strong computer science foundations, from C to Python.
- Program fluently in Python while completing real projects.
- Understand fundamental algorithms, data structures, and abstraction principles.
- Apply computational thinking to solve problems effectively.
- Prepare for specialized studies in web development, data science, or artificial intelligence.

## Who This Course Is For

This course is designed for beginners. It suits anyone seeking a strong foundation in computer science and programming. No prior experience in coding or computer science is necessary to begin.

Students who are curious about how computers work will find this course valuable. It provides a comprehensive introduction. You will gain practical skills in Python programming.

## Before You Start

- No prior experience is required.

## How to Use This Guide

This study guide is a companion to the video lessons. Watch each video lesson first. Then, read the corresponding section in this guide. This approach reinforces key concepts.

Pace yourself consistently. The course has 4 modules and 23 lessons. Break down your study into manageable segments. Review the material regularly. Use the end-of-lesson summaries and practice problems to test your understanding.

To get the most from this guide:

- Watch the video lessons before reading.
- Review key terms in the glossary.
- Complete all practice exercises. Active engagement with both the videos and this guide will maximize your learning.

## CHAPTER 1

# Computer Science Foundations

---

This chapter introduces the fundamental concepts of computer science, laying the groundwork for your programming path. We begin by exploring computational thinking, a problem-solving approach applicable far beyond coding, using the visual programming language Scratch. This helps build intuition for how computers 'think' and execute instructions.

We then transition to C, a powerful low-level language that reveals how computers manage data and memory. You will learn about basic data types, control flow, and how to compile programs. The chapter progresses to cover efficient data organization using arrays and the crucial topic of algorithms, including how to analyze their efficiency. Finally, we introduce Python, a high-level language widely used in modern applications, and provide an overview of Artificial Intelligence concepts.

## Computational Thinking and Visual Programming with Scratch

---

Covers: CS50x 2024 - Lecture 0 - Scratch

This section introduces computational thinking, a fundamental approach to problem-solving that applies principles from computer science. It is not just about coding. Computational thinking involves four key pillars: decomposition, pattern recognition, abstraction, and algorithms. Decomposition means breaking down a complex problem into smaller, more manageable parts. Pattern recognition involves identifying similarities or trends within or across problems. Abstraction focuses on the essential information while ignoring irrelevant details. Finally, an algorithm is a step-by-step set of instructions to solve a problem. These skills are valuable in many fields, not only in computer science.

To begin practicing these concepts, we use Scratch, a visual programming language developed by MIT. Scratch allows you to create interactive stories, games, and animations by dragging and dropping graphical blocks of code. This eliminates syntax errors that often frustrate beginners in text-based languages. In Scratch, projects are built around *sprites*, which are characters or objects, and a *stage*, which is the background. You program sprites by attaching blocks to form scripts.

Consider a simple task: making a cat sprite move across the screen and say "Hello!". First, you need an *event* block, such as "when green flag clicked", to start the script. Next, add *motion* blocks. A "move 10 steps" block makes the sprite move. To make it move across the screen, you might place this inside a *control* block like "repeat 10 times". This creates a loop, executing the "move" block multiple times. Then, add a *looks* block, such as "say Hello! for 2 seconds". This makes the sprite display a speech bubble.

### Basic Programming Constructs in Scratch

Scratch uses several fundamental programming constructs:

- **Sequences:** Blocks run from top to bottom, in order.

- **Loops:** Blocks like "repeat" or "forever" execute a set of instructions multiple times. For example, a "repeat 10" block will run its contained code ten times. A "forever" block runs its code indefinitely until the program stops.
- **Conditionals:** "If...then" and "if...then...else" blocks allow your program to make decisions. For instance, "if touching color [red] then change color effect by 25" executes an action only when a specific condition is met. The "else" part provides an alternative action if the condition is false.
- **Variables:** These store data that can change during the program's execution. You can create variables to keep track of scores, time, or other values.

A common mistake for beginners is to create overly complex scripts for simple actions. Try to decompose your problem into the smallest possible steps first. For example, if you want a sprite to move to a specific position, then bounce, and then change costume, plan each step individually before combining them. This structured approach helps prevent errors and makes debugging easier.

Practice: Open Scratch and create a project. Make a sprite move from the left side of the stage to the right. When it reaches the right edge, make it say "I'm here!" and then return to the left side. Use a loop and a conditional statement to detect the edge. This exercise reinforces sequencing, loops, and conditionals.

## Introduction to C Programming: Syntax and Control Flow

---

Covers: CS50x 2024 - Lecture 1 - C

C is a foundational programming language, often described as a "middle-level" language. It offers a balance between high-level abstraction and low-level control over computer hardware. Many modern operating systems, databases, and other programming languages are built using C. Understanding C provides a strong grasp of how computers manage memory and execute instructions, which is invaluable for any computer scientist.

A basic C program starts with the `main` function, which is the entry point for execution. You will also typically see `#include` statements at the top. These directives tell the C preprocessor to include the contents of another file, usually a header file, into your program. For example, `#include <stdio.h>` provides access to standard input/output functions like `printf` and `scanf`.

### Compiling and Running C Programs

Unlike interpreted languages, C code must be *compiled* into machine code before it can be executed. A compiler translates your human-readable C source code into an executable file. On systems like Linux or macOS, you might use `clang` or `gcc`. For example, `clang hello.c -o hello` compiles `hello.c` into an executable named `hello`. You then run it with `./hello`. This compilation step catches many errors before your program even runs.

### Data Types and Variables

C is a *statically typed* language. This means you must declare the type of a variable before using it, and that type cannot change. Common data types include:

- `int`: for integers (whole numbers), like 10, -5.

- `char`: for single characters, like 'A', 'z'.
- `float`: for single-precision floating-point numbers (decimals), like 3.14.
- `double`: for double-precision floating-point numbers, offering more precision than `float`.
- `bool`: for boolean values (true or false), available by including `<stdbool.h>`.

To declare a variable, specify its type followed by its name: `int age;`. You can also initialize it immediately: `int age = 30;`.

## Input, Output, and Operators

The `printf` function is used to print output to the console. It supports format specifiers like `%i` for integers, `%f` for floats, and `%s` for strings. For example, `printf("Your age is %i.\n", age);`. The `scanf` function reads input from the user. It requires a format specifier and the memory address of the variable where the input should be stored, using the `&` operator: `scanf("%i", &age);`.

C supports various operators:

- **Arithmetic:** `+` (addition), `-` (subtraction), `*` (multiplication), `/` (division), `%` (modulo, remainder).
- **Relational:** `==` (equal to), `!=` (not equal to), `<` (less than), `>` (greater than), `<=` (less than or equal to), `>=` (greater than or equal to). These return boolean results.
- **Logical:** `&&` (logical AND), `||` (logical OR), `!` (logical NOT).

## Control Flow: Conditionals and Loops

Programs need to make decisions and repeat actions.

- **Conditionals:** The `if`, `else if`, and `else` statements allow your program to execute different blocks of code based on conditions.

```
if (age < 18)
{
    printf("You are a minor.\n");
}
else if (age >= 18 && age < 65)
{
    printf("You are an adult.\n");
}
else
{
    printf("You are a senior.\n");
}
```

- **Loops:** `while` and `for` loops repeat a block of code. A `while` loop continues as long as its condition is true. A `for` loop is often used when you know how many times you need to iterate.

```
// while loop
int i = 0;
while (i < 5)
{
```

```
printf("%i\n", i);
i++;
}

// for loop
for (int j = 0; j < 5; j++)
{
    printf("%i\n", j);
}
```

Common mistakes include forgetting semicolons at the end of statements, mismatching curly braces, or using `=` (assignment) instead of `==` (comparison) in conditional statements. Always compile your code frequently to catch these syntax errors early.

Practice: Write a C program that asks the user for two integers, calculates their sum, difference, product, and quotient, and then prints the results. Use `printf` and `scanf`.

## Organizing Data with Arrays and Analyzing Algorithm Efficiency

Covers: CS50x 2024 - Lecture 2 - Arrays · CS50x 2024 - Lecture 3 - Algorithms

As programs grow, managing multiple pieces of related data becomes crucial. *Arrays* provide a way to store a fixed-size sequential collection of elements of the same data type. Think of an array as a row of numbered boxes, where each box can hold one item. In C, arrays are declared by specifying the data type, the array name, and its size in square brackets. For example, `int scores[5];` declares an array named `scores` that can hold five integers.

### Array Indexing and Strings

Array elements are accessed using an *index*, which starts at 0 for the first element. So, `scores[0]` refers to the first score, `scores[1]` to the second, and so on, up to `scores[4]` for the fifth element. You can assign values to elements: `scores[0] = 90;` or retrieve them: `printf("First score: %i\n", scores[0]);`. Loops are commonly used to process all elements in an array.

```
int numbers[3] = {10, 20, 30}; // Declaration and initialization
for (int i = 0; i < 3; i++)
{
    printf("Element %i: %i\n", i, numbers[i]);
}
```

In C, *strings* are essentially arrays of characters. A special character, the null terminator (`\0`), marks the end of a string. This allows functions to know where the string ends, regardless of the array's declared size. For example, `char name[10] = "Alice";` creates an array of 10 characters, but "Alice" only uses 5 plus the null terminator, leaving 4 unused spaces. Common mistakes include forgetting the null terminator when manually constructing strings or not allocating enough space for it.

## Introduction to Algorithms

An *algorithm* is a precise, step-by-step set of instructions designed to solve a specific problem or perform a computation. Algorithms are the core of computer science. They are not tied to any particular programming language. Instead, they represent the logical procedure. Examples include finding an item in a list, sorting a list of numbers, or calculating the shortest path between two points.

Consider the problem of finding a specific value in an array.

- **Linear Search:** This is the simplest approach. You start at the beginning of the array and check each element one by one until you find the target value or reach the end of the array. If the array has  $n$  elements, in the worst case, you might have to check all  $n$  elements.

```
// Example of a linear search
int search_array[] = {12, 45, 67, 89, 23};
int target = 67;
bool found = false;
for (int i = 0; i < 5; i++)
{
    if (search_array[i] == target)
    {
        printf("Target %i found at index %i.\n", target, i);
        found = true;
        break; // Exit loop once found
    }
}
if (!found)
{
    printf("Target %i not found.\n", target);
}
```

- **Binary Search:** If the array is sorted, you can use a more efficient *binary search*. This algorithm repeatedly divides the search interval in half. You compare the target value with the middle element of the array. If they are not equal, the half in which the target cannot lie is eliminated, and the search continues on the remaining half. This significantly reduces the number of comparisons.

## Algorithm Efficiency and Big O Notation

The efficiency of an algorithm is measured by how its performance scales with the size of the input. We typically consider two aspects: *time complexity* (how many operations) and *space complexity* (how much memory). *Big O notation* is a mathematical notation that describes the limiting behavior of a function when the argument tends towards a particular value or infinity. It provides an upper bound on the growth rate of an algorithm's running time or space requirements.

- **$O(n)$  - Linear Time:** The time taken grows linearly with the input size  $n$ . Linear search is  $O(n)$ .
- **$O(\log n)$  - Logarithmic Time:** The time taken grows logarithmically with  $n$ . Binary search is  $O(\log n)$ , which is much faster for large inputs.
- **$O(n^2)$  - Quadratic Time:** The time taken grows proportionally to the square of the input size. Simple sorting algorithms like bubble sort or selection sort are  $O(n^2)$ .

Understanding Big O notation helps you choose the most appropriate algorithm for a given problem and dataset size. An  $O(n)$  algorithm is generally preferred over an  $O(n^2)$  algorithm when dealing with large inputs because it will complete much faster.

Practice: Write a C program that declares an array of 10 integers, fills it with random numbers, and then implements a linear search to find if a user-specified number is present in the array. For an extra challenge, try to implement a simple sorting algorithm like selection sort on the array before searching.

## Understanding Computer Memory and Fundamental Data Structures

Covers: CS50x 2024 - Lecture 4 - Memory · CS50x 2024 - Lecture 5 - Data Structures

To write efficient and robust programs, it is essential to understand how computers manage memory. At its most basic level, computer memory is a vast collection of *bits*, each storing a 0 or a 1. Bits are grouped into *bytes*, typically 8 bits, which can represent a single character or a small number. Each byte in memory has a unique numerical *address*. When your program declares a variable, it reserves a specific location in memory to store that variable's value.

### Pointers and Memory Management

A *pointer* is a special type of variable that stores a memory address, rather than a direct value. In C, you declare a pointer by placing an asterisk (\*) before the variable name, like `int *p;`. The & operator (address-of operator) gives you the memory address of a variable: `p = &my_variable;`. The \* operator (dereference operator) accesses the value stored at the address a pointer holds: `printf("%i\n", *p);` would print the value of `my_variable`. Pointers are powerful because they allow direct manipulation of memory, which is crucial for building complex data structures and optimizing performance.

Consider this C code:

```
int x = 10;
int *ptr_to_x = &x; // ptr_to_x now stores the memory address of x
printf("Value of x: %i\n", x); // Output: Value of x: 10
printf("Address of x: %p\n", &x); // Output: Address of x: (some hex address)
printf("Value of ptr_to_x: %p\n", ptr_to_x); // Output: Value of ptr_to_x: (same hex address)
printf("Value at address stored in ptr_to_x: %i\n", *ptr_to_x); // Output: Value at address stored in ptr_to_x: 10
*ptr_to_x = 20; // Change the value at the address ptr_to_x points to
printf("New value of x: %i\n", x); // Output: New value of x: 20
```

### Dynamic Memory Allocation

Sometimes, you do not know the exact amount of memory your program will need until it runs. *Dynamic memory allocation* allows you to request memory from the operating system during runtime. In C, the `malloc` function (memory allocate) is used to request a block of memory of a specified size. It returns a pointer to the beginning of the allocated block. When you are finished with the memory, you must explicitly release it using the `free` function. Failure to `free` allocated memory leads to *memory leaks*, where your program consumes more and more memory over time, potentially crashing the system.

Memory allocated with `malloc` resides in an area called the *heap*, distinct from the *call stack* where local variables and function calls are managed.

## Fundamental Data Structures

A *data structure* is a particular way of organizing and storing data in a computer so that it can be accessed and modified efficiently. Choosing the right data structure is vital for an algorithm's performance.

- **Linked Lists:** Unlike arrays, which store elements contiguously in memory, a *linked list* stores elements as individual *nodes*. Each node contains two parts: the data itself and a pointer to the next node in the sequence. The last node's pointer is typically `NULL`. This structure allows for efficient insertion and deletion of elements anywhere in the list without shifting other elements, but accessing an element by its index requires traversing the list from the beginning.

```
// Conceptual C structure for a linked list node
typedef struct Node
{
    int data;
    struct Node *next; // Pointer to the next node
} Node;
```

- **Trees:** A *tree* is a hierarchical data structure composed of nodes. It has a single *root* node, and each node can have zero or more child nodes. Unlike linked lists, nodes in a tree can branch. A common type is the *binary tree*, where each node has at most two children: a left child and a right child. *Binary search trees* are a special kind of binary tree where the value in each left child node is less than its parent, and the value in each right child node is greater. This property allows for very efficient searching, insertion, and deletion operations.
- **Hash Tables:** A *hash table* stores data in key-value pairs. It uses a *hash function* to compute an index into an array of buckets or slots, from which the desired value can be found. Hash tables offer very fast average-case time complexity for insertion, deletion, and lookup operations (often  $O(1)$ ). However, *collisions* can occur when two different keys hash to the same index. Various strategies exist to handle collisions, such as separate chaining or open addressing.

Each data structure has its strengths and weaknesses, involving trade-offs between time complexity for operations and space complexity for storage. For example, arrays offer fast random access but slow insertions/deletions in the middle. Linked lists offer fast insertions/deletions but slow random access. Understanding these trade-offs helps in selecting the most suitable structure for a given application.

Practice: Draw a diagram of a linked list with three nodes, showing the data in each node and how the pointers connect them. Then, draw how the list would change if you inserted a new node in the middle.

## Transition to Python and Introduction to Artificial Intelligence

---

Covers: CS50x 2024 - Lecture 6 - Python · CS50x 2024 - Artificial Intelligence

After exploring the foundational concepts with C, we now transition to Python, a high-level, interpreted programming language. Python is celebrated for its readability, clear syntax, and extensive standard

library, making it an excellent choice for rapid development, web applications, data science, and artificial intelligence. Its design philosophy emphasizes code readability with its use of significant indentation.

## Python Fundamentals

Python differs from C in several key ways. It is an *interpreted* language, meaning code is executed line by line by an interpreter, without a separate compilation step. This speeds up the development cycle. Python is also *dynamically typed*, so you do not need to declare the data type of a variable explicitly. The interpreter infers the type at runtime.

Basic Python syntax is straightforward:

- **Variables:** Assign values directly. `name = "Alice"` or `age = 30`.
- **Data Types:** Python has built-in types like `int`, `float`, `str` (strings). It also offers powerful collection types:
  - `list`: an ordered, mutable collection of items (like `[1, 2, 3]`).
  - `dict` (dictionary): an unordered, mutable collection of key-value pairs (like `{"name": "Alice", "age": 30}`).
- **Output:** The `print()` function is used for output: `print("Hello, world!")`.

## Control Flow in Python

Python uses indentation to define code blocks, replacing the curly braces used in C.

- **Conditionals:** `if`, `elif` (else if), `else` statements.

```
age = 25
if age < 18:
    print("You are a minor.")
elif age >= 18 and age < 65:
    print("You are an adult.")
else:
    print("You are a senior.")
```

- **Loops:** `for` and `while` loops. The `for` loop is often used to iterate over sequences (lists, strings, ranges).

```
# For loop iterating over a list
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(fruit)

# While loop
count = 0
while count < 3:
    print("Count is:", count)
    count += 1
```

- **Functions:** Define reusable blocks of code using the `def` keyword.

```
def greet(name):  
    return f"Hello, {name}!"  
  
message = greet("Bob")  
print(message)
```

## Introduction to Artificial Intelligence

*Artificial Intelligence (AI)* is a broad field of computer science dedicated to creating machines that can perform tasks that typically require human intelligence. This includes learning, problem-solving, perception, and understanding language. AI is not a single technology but a collection of techniques and algorithms.

One prominent subfield of AI is *Machine Learning (ML)*. ML algorithms enable systems to learn from data, identify patterns, and make decisions with minimal human intervention. There are three main types of machine learning:

- **Supervised Learning:** The model is trained on labeled data, meaning the input data is paired with the correct output. Examples include image classification (identifying objects in pictures) and spam detection.
- **Unsupervised Learning:** The model is given unlabeled data and must find patterns or structures within it on its own. Clustering algorithms, which group similar data points, are an example.
- **Reinforcement Learning:** An agent learns to make decisions by performing actions in an environment and receiving rewards or penalties. This is often used in robotics and game playing.

*Neural Networks* are a type of machine learning model inspired by the structure and function of the human brain. They consist of interconnected "neurons" organized in layers (input, hidden, output). Each connection has a weight, and the network learns by adjusting these weights during training to minimize errors. Deep learning, a subset of machine learning, uses neural networks with many hidden layers.

AI applications are widespread, from natural language processing (like virtual assistants and translation services) to computer vision (facial recognition, self-driving cars) and recommendation systems (streaming services, online shopping). As AI becomes more integrated into daily life, ethical considerations such as bias in algorithms, privacy, and the responsible use of AI become increasingly important.

**Practice:** Write a Python program that asks the user for their favorite animal. Based on their input, print a different fun fact about that animal. If the animal is not recognized, print a generic message. Use `input()` for user input, `if/elif/else` for conditions, and `print()` for output.

**KEY TAKEAWAYS**

- Computational thinking involves decomposition, pattern recognition, abstraction, and algorithms, essential skills for problem-solving.
- Scratch provides a visual, block-based introduction to programming concepts like sequences, loops, and conditionals.
- C is a foundational, statically typed language that offers direct control over memory and requires compilation before execution.
- Arrays store collections of same-type data, accessed by 0-based indices, and strings are character arrays terminated by a null character.
- Algorithms are step-by-step procedures to solve problems, and their efficiency is measured using Big O notation, such as  $O(n)$  or  $O(\log n)$ .
- Pointers store memory addresses, enabling dynamic memory allocation with `malloc` and `free`, which must be managed to prevent leaks.
- Data structures like linked lists, trees, and hash tables organize data for efficient access and modification, each with distinct trade-offs.
- Python is a high-level, dynamically typed, interpreted language known for its readability and extensive use in AI.
- Artificial Intelligence, particularly Machine Learning (supervised, unsupervised, reinforcement), aims to enable machines to learn from data and perform intelligent tasks.

**CHECK YOUR UNDERSTANDING**

*Answer these before moving on. Model answers are in the Answer Key at the back of this guide.*

1. What are the four pillars of computational thinking, and how does Scratch help illustrate them?
2. Explain the key differences between C and Python in terms of typing and execution.
3. Describe the concept of an array in C, including how elements are indexed and what a null terminator signifies for strings.
4. What is Big O notation, and why is it important when discussing algorithms? Provide an example of an  $O(n)$  and an  $O(\log n)$  algorithm.
5. How do pointers relate to memory addresses, and what is the purpose of `malloc` and `free` in C?
6. Briefly describe two different data structures discussed in this chapter and one scenario where each might be preferred.

## CHAPTER 2

# Computer Science - Part 2

---

This chapter advances your computer science understanding. It introduces Structured Query Language (SQL) for effective database management. You will then explore the core technologies behind web development: HTML for structure, CSS for styling, and JavaScript for interactivity.

The chapter also covers server-side web application development using the Python Flask framework. Finally, it addresses essential cybersecurity principles, helping you understand how to protect digital systems and data.

## Managing Data with SQL

---

Covers: CS50x 2024 - Lecture 7 - SQL

Structured Query Language (SQL) is the standard language for managing and manipulating relational databases. A relational database organizes data into one or more tables, each with a unique name. Each table consists of rows (records) and columns (fields). SQL allows you to create these tables, insert data, retrieve specific information, update existing records, and delete data.

To begin, you create a table. The `CREATE TABLE` statement defines the table's structure, including column names and their data types. Common data types include `INTEGER` for whole numbers, `TEXT` for strings, `REAL` for floating-point numbers, and `BLOB` for binary data. Every table should have a `PRIMARY KEY`, a column or set of columns that uniquely identifies each row. A `FOREIGN KEY` establishes a link between data in two tables, ensuring referential integrity.

Consider creating a table for students:

```
CREATE TABLE students (  
  id INTEGER PRIMARY KEY AUTOINCREMENT,  
  name TEXT NOT NULL,  
  major TEXT,  
  gpa REAL  
);
```

After creating the table, you can add data using the `INSERT` statement:

```
INSERT INTO students (name, major, gpa) VALUES ('Alice', 'Computer Science', 3.8);  
INSERT INTO students (name, major, gpa) VALUES ('Bob', 'Mathematics', 3.5);
```

The `SELECT` statement is used to retrieve data. You can specify which columns to retrieve and add conditions using the `WHERE` clause. The `ORDER BY` clause sorts the results.

```
SELECT name, gpa FROM students WHERE major = 'Computer Science' ORDER BY gpa DESC;
```

This query fetches the names and GPAs of Computer Science majors, sorted from highest GPA to lowest. SQL also provides aggregate functions like `COUNT`, `SUM`, `AVG`, `MIN`, and `MAX` to perform calculations on sets of rows.

```
SELECT AVG(gpa) FROM students WHERE major = 'Mathematics';
```

When working with multiple tables, `JOIN` clauses combine rows based on related columns. An `INNER JOIN` returns rows when there is a match in both tables. For example, if you had a `courses` table and wanted to see which students are enrolled in which courses, you would use a join.

Common mistakes include forgetting semicolons at the end of statements, misspelling column or table names, and using incorrect data types. A significant security concern is SQL injection, where malicious SQL code is inserted into input fields, potentially compromising the database. Always validate user input to prevent this.

## Building Web Applications: Front-End to Back-End

---

Covers: CS50x 2024 - Lecture 8 - HTML, CSS, JavaScript · CS50x 2024 - Lecture 9 - Flask

Web applications consist of two main parts: the front-end (what users see and interact with) and the back-end (the server-side logic and data storage). The front-end is built using HTML, CSS, and JavaScript.

### Front-End Fundamentals

**HTML (HyperText Markup Language)** provides the structure of a web page. It uses tags to define elements like headings (`<h1>`), paragraphs (`<p>`), lists (`<ul>`, `<ol>`, `<li>`), links (`<a>`), and images (`<img>`). Every HTML document starts with `<!DOCTYPE html>` and an `<html>` tag, containing a `<head>` section for metadata and a `<body>` section for visible content.

**CSS (Cascading Style Sheets)** controls the presentation and styling of HTML elements. You can apply styles using element selectors, class selectors (`.my-class`), or ID selectors (`#my-id`). Properties like `color`, `font-size`, `margin`, and `padding` define appearance. Styles can be inline (within an HTML tag), internal (in a `<style>` tag in the head), or external (linked from a separate `.css` file). External stylesheets are preferred for maintainability.

**JavaScript** adds interactivity and dynamic behavior to web pages. It is a programming language that runs in the user's browser. JavaScript can manipulate the Document Object Model (DOM), allowing it to change HTML content, CSS styles, and respond to user events like clicks or form submissions. Variables, functions, and conditional statements are core components. For example, a JavaScript function might validate a form before submission or dynamically update content without reloading the page.

### Back-End with Flask

**Flask** is a micro web framework for Python. It simplifies the process of building server-side web applications. Flask handles incoming HTTP requests, processes data, interacts with databases, and sends back HTTP responses, often in the form of HTML pages.

A basic Flask application involves defining routes, which are URL paths that trigger specific Python functions called view functions. These functions typically render HTML templates to send dynamic content to the user's browser.

Here is a simple Flask application:

```
from flask import Flask, render_template, request

app = Flask(__name__)

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/greet', methods=['POST'])
def greet():
    name = request.form.get('name', 'Guest')
    return render_template('greeting.html', user_name=name)

if __name__ == '__main__':
    app.run(debug=True)
```

In this example, `/` renders `index.html`. The `/greet` route handles a `POST` request, retrieves a `name` from a form submission, and passes it to `greeting.html`. Flask uses Jinja2 for templating, allowing Python variables to be embedded directly into HTML files. Common mistakes include incorrect routing, not handling different HTTP methods (GET, POST), and neglecting to include necessary imports. Security considerations for web applications include protecting against Cross-Site Scripting (XSS) and Cross-Site Request Forgery (CSRF) attacks, often mitigated with proper input sanitization and framework-provided tokens.

## Understanding Cybersecurity Fundamentals

Covers: CS50x 2024 - Cybersecurity

Cybersecurity is the practice of protecting computer systems, networks, and data from digital attacks, damage, or unauthorized access. It ensures the confidentiality, integrity, and availability of information, often referred to as the CIA triad.

- **Confidentiality** means preventing unauthorized disclosure of information. This includes protecting personal data, trade secrets, and other sensitive information from falling into the wrong hands. Encryption is a primary tool for ensuring confidentiality.
- **Integrity** ensures that data has not been altered or tampered with by unauthorized individuals. It maintains the accuracy and trustworthiness of information. Hashing and digital signatures help verify data integrity.
- **Availability** means that authorized users can access information and systems when needed. This involves protecting against denial-of-service (DoS) attacks and ensuring robust system infrastructure and backups.

Common cybersecurity threats include:

- **Malware:** Malicious software like viruses, worms, and ransomware designed to disrupt, damage, or gain unauthorized access to computer systems.
- **Phishing:** Deceptive attempts to trick individuals into revealing sensitive information, often through fake emails or websites.
- **Denial-of-Service (DoS) Attacks:** Attempts to make a machine or network resource unavailable to its intended users by overwhelming it with traffic.
- **SQL Injection:** As discussed earlier, inserting malicious SQL code into input fields to manipulate a database.
- **Cross-Site Scripting (XSS):** Injecting malicious scripts into web pages viewed by other users, often to steal session cookies or deface websites.

Effective defense mechanisms are crucial. Strong, unique passwords combined with multi-factor authentication (MFA) significantly enhance account security. Regularly updating software and applying security patches fixes known vulnerabilities that attackers might exploit. Firewalls act as barriers, monitoring and controlling incoming and outgoing network traffic based on predefined security rules.

Encryption protects data both at rest (stored on a device) and in transit (moving across networks). Secure coding practices, such as rigorous input validation, are essential to prevent vulnerabilities like SQL injection and XSS. Regular data backups ensure that information can be restored in case of data loss or system compromise. User awareness training is also vital, as many attacks rely on social engineering, manipulating individuals into performing actions or divulging confidential information.

#### KEY TAKEAWAYS

- SQL is a standard language for managing and querying relational databases, using commands like SELECT, INSERT, UPDATE, and DELETE.
- Web pages are structured with HTML, styled with CSS, and made interactive with JavaScript on the client side.
- Flask is a Python microframework used for building server-side web applications, handling routes, requests, and rendering templates.
- Cybersecurity involves protecting computer systems and data from digital threats, focusing on confidentiality, integrity, and availability.
- Understanding web development requires knowledge of both client-side (HTML, CSS, JavaScript) and server-side (Flask) technologies.
- Input validation is a critical security practice to prevent common web vulnerabilities such as SQL injection and Cross-Site Scripting (XSS).

---

#### CHECK YOUR UNDERSTANDING

*Answer these before moving on. Model answers are in the Answer Key at the back of this guide.*

1. What are the three core languages used for front-end web development and what is the primary role of each?
2. Explain the purpose of a primary key and a foreign key in a relational database.
3. How does Flask facilitate the creation of dynamic web pages?
4. Describe the CIA triad in cybersecurity and provide a brief example for each component.
5. What is SQL injection and how can developers help prevent it?
6. When building a web application, what is the main distinction between client-side and server-side scripting?

## CHAPTER 3

# Programming with Python - Part 1

---

This chapter introduces you to the world of Python programming. You will transition from foundational computer science concepts to practical application using Python's clear and powerful syntax. This module focuses on building a strong understanding of Python's core features, from basic data handling to structured program control and error management.

You will learn how to write functions, manage variables, and use conditional logic and loops to create dynamic programs. The chapter also covers how to handle unexpected errors gracefully, incorporate external code using libraries, and ensure your code works as intended through unit testing. Finally, you will explore how to interact with the file system to store and retrieve data.

## Introduction to Python, Functions, and Variables

---

Covers: CS50P - Introduction · CS50P - Lecture 0 - Functions, Variables

Python is a high-level, interpreted programming language known for its readability and versatility. It allows developers to write less code to achieve more. When you begin programming in Python, one of the first things you will encounter is the `print()` function, which displays output to the console. For example, `print("Hello, world!")` will show "Hello, world!" on your screen.

Programs often need to interact with users. Python's `input()` function allows you to prompt the user for data. The data received from `input()` is always a string. If you need to perform numerical operations, you must convert it. For instance, `age = int(input("What is your age? "))` converts the user's input into an integer.

**Variables** are names that refer to values stored in memory. In Python, you do not need to declare a variable's type explicitly. The type is inferred when you assign a value. For example, `name = "Alice"` creates a string variable, and `score = 100` creates an integer variable. Python supports several fundamental data types, including integers (`int`), floating-point numbers (`float`), strings (`str`), and Booleans (`bool`). You can convert between types using functions like `int()`, `float()`, and `str()`. For example, `str(123)` converts the integer 123 to the string "123".

**Functions** are reusable blocks of code that perform a specific task. They help organize code, make it more readable, and prevent repetition. You define a function using the `def` keyword, followed by the function name, parentheses for parameters, and a colon. The function's body is indented. For example:

```
def greet(name):  
    print(f"Hello, {name}!")  
  
greet("Bob") # Calls the function, prints "Hello, Bob!"
```

Functions can also return values using the `return` keyword. This allows the result of a function to be used elsewhere in your program. For instance, a function that calculates the square of a number might look like this:

```
def square(number):
    return number * number

result = square(5) # result will be 25
print(result)
```

**Common Mistakes:** A frequent error is forgetting to convert input from a string to a number when arithmetic operations are intended. Forgetting to indent code blocks correctly after `def` or other control structures will lead to an `IndentationError`. Also, be mindful of variable naming conventions; Python variables are typically lowercase with underscores separating words (snake\_case).

## Controlling Program Flow: Conditionals and Loops

Covers: CS50P - Lecture 1 - Conditionals · CS50P - Lecture 2 - Loops

Programs rarely execute instructions strictly from top to bottom. **Conditional statements** allow your program to make decisions and execute different code blocks based on certain conditions. The primary conditional structure in Python uses `if`, `elif` (else if), and `else` keywords. Each block of code associated with a condition must be indented.

```
score = 85

if score >= 90:
    print("Grade A")
elif score >= 80:
    print("Grade B")
else:
    print("Grade C")
```

This example prints "Grade B" because 85 is not greater than or equal to 90, but it is greater than or equal to 80. You use comparison operators like `==` (equal to), `!=` (not equal to), `<` (less than), `>` (greater than), `<=` (less than or equal to), and `>=` (greater than or equal to) within your conditions. Logical operators such as `and`, `or`, and `not` combine or modify conditions. For example, `if age >= 18 and has_id:` checks if both conditions are true.

**Loops** enable your program to repeat a block of code multiple times. Python offers two main types of loops: `while` loops and `for` loops.

A **while loop** continues to execute as long as its condition remains true. It is useful when you do not know in advance how many times you need to loop. You must ensure the condition eventually becomes false to avoid an infinite loop.

```
count = 0
while count < 3:
```

```
print(f"Count is {count}")
count += 1 # Increment count to eventually make the condition false
```

A **for loop** is used for iterating over a sequence (like a string, list, or the output of `range()`). It is ideal when you know the number of iterations or are processing items in a collection.

```
for i in range(3): # Iterates 0, 1, 2
    print(f"Iteration {i}")

name = "Python"
for char in name:
    print(char)
```

The `range(3)` function generates a sequence of numbers from 0 up to (but not including) 3. You can use `break` to exit a loop immediately or `continue` to skip the rest of the current iteration and move to the next one.

**Common Mistakes:** Incorrect indentation is a primary source of errors in conditional statements and loops. For `while` loops, forgetting to update the loop variable can lead to an infinite loop, causing your program to hang. For `for` loops with `range()`, remember that the end value is exclusive, meaning the loop runs up to, but not including, that number.

## Handling Errors and Reusing Code: Exceptions and Libraries

Covers: CS50P - Lecture 3 - Exceptions · CS50P - Lecture 4 - Libraries

Even well-written programs can encounter issues during execution. These runtime errors are called **exceptions**. When an exception occurs, the program normally stops. Python provides a mechanism to handle these errors gracefully, preventing your program from crashing and allowing you to respond to problems.

You use a `try` block to enclose the code that might raise an exception. If an exception occurs within the `try` block, Python immediately jumps to the `except` block. You can specify which type of exception to catch. For example:

```
try:
    x = int(input("Enter a number: "))
    result = 10 / x
    print(f"Result: {result}")
except ValueError:
    print("That's not a valid integer!")
except ZeroDivisionError:
    print("Cannot divide by zero!")
except:
    print("An unexpected error occurred.")
```

In this example, if the user enters text instead of a number, a `ValueError` is caught. If they enter `0`, a `ZeroDivisionError` is caught. A bare `except` catches any other unhandled exception. Handling exceptions

makes your programs more robust and user-friendly.

**Libraries**, also known as modules, are collections of pre-written functions and code that you can include in your own programs. They allow you to reuse code written by others, saving time and effort. Python has a vast standard library, and many third-party libraries are also available. To use a module, you use the `import` statement.

```
import random

# Generate a random integer between 1 and 10, inclusive
number = random.randint(1, 10)
print(f"Your lucky number is: {number}")

import math

# Calculate the square root of 25
sqrt_value = math.sqrt(25)
print(f"The square root of 25 is: {sqrt_value}")
```

You can also import specific functions from a module using `from module_name import function_name`. For example, `from sys import exit` allows you to call `exit()` directly without `sys.exit()`.

**Common Mistakes:** Not catching specific exceptions can lead to unexpected behavior if multiple types of errors can occur. Forgetting to `import` a module before trying to use its functions will result in a `NameError`. Also, be careful with naming conflicts if you import functions directly using `from module import function` and you have a function with the same name in your own code.

## Ensuring Correctness and Persistence: Unit Tests and File I/O

Covers: CS50P - Lecture 5 - Unit Tests · CS50P - Lecture 6 - File I/O

Writing correct code is crucial. **Unit tests** are small, isolated tests that verify individual units of code, typically functions, work as expected. They help catch bugs early, confirm that changes do not break existing functionality (regressions), and serve as documentation for how functions should behave. In Python, you can write simple unit tests using `assert` statements.

An `assert` statement checks if a condition is true. If the condition is false, it raises an `AssertionError`, indicating a test failure. For example, if you have a function that adds two numbers:

```
def add(a, b):
    return a + b

# Unit tests for the add function
assert add(2, 3) == 5, "Test Case 1 Failed: 2 + 3 should be 5"
assert add(-1, 1) == 0, "Test Case 2 Failed: -1 + 1 should be 0"
assert add(0, 0) == 0, "Test Case 3 Failed: 0 + 0 should be 0"

print("All add tests passed!")
```

If any `assert` condition is false, the program stops and prints the provided error message. This simple approach is effective for verifying function correctness.

**File Input/Output (I/O)** allows your programs to interact with files on a computer's storage. This is essential for saving data permanently, loading configurations, or processing large datasets. To work with files, you first need to open them using the `open()` function. You must specify the file path and the mode.

Common file modes include:

- `'r'` (read): Opens a file for reading. The file must exist.
- `'w'` (write): Opens a file for writing. Creates the file if it does not exist, or truncates (empties) it if it does.
- `'a'` (append): Opens a file for appending. Creates the file if it does not exist, or adds new content to the end if it does.

After performing operations, it is vital to close the file to release system resources and ensure data is saved. The `close()` method handles this. A safer and more Pythonic way to handle files is using a `with` statement, which acts as a context manager. It ensures the file is automatically closed, even if errors occur.

```
# Writing to a file
with open("names.txt", "w") as file:
    file.write("Alice\n")
    file.write("Bob\n")

# Reading from a file
with open("names.txt", "r") as file:
    content = file.read() # Reads entire file content as a single string
    print("File content:")
    print(content)

# Reading line by line
with open("names.txt", "r") as file:
    for line in file:
        print(f"Line: {line.strip()}") # .strip() removes newline characters
```

**Common Mistakes:** Forgetting to close files can lead to data corruption or resource leaks. Using `'w'` mode when you intend to add to an existing file will overwrite its contents. Attempting to read a file that does not exist will raise a `FileNotFoundError`.

**KEY TAKEAWAYS**

- Python simplifies programming with its clear syntax and powerful features, making it an excellent language for beginners.
- Functions and variables are fundamental building blocks for organizing code and storing data effectively.
- Conditional statements and loops control the flow of execution, allowing programs to make decisions and repeat actions.
- Exceptions provide a structured way to handle errors gracefully, preventing program crashes and improving user experience.
- Libraries offer pre-written code that extends program functionality, enabling code reuse and faster development.
- Unit tests help verify code correctness, catch bugs early, and ensure that changes do not introduce new issues.
- File I/O enables programs to store and retrieve data persistently, making information available across program runs.

**CHECK YOUR UNDERSTANDING**

*Answer these before moving on. Model answers are in the Answer Key at the back of this guide.*

1. How do you define a function in Python and what are its main components?
2. Explain the difference between a `while` loop and a `for` loop, and provide a use case for each.
3. Why is it good practice to use `try` and `except` blocks in your code?
4. What is the purpose of importing a module, and how do you do it?
5. Describe two common file modes used when opening a file and what they do.
6. What is a unit test, and how does an `assert` statement contribute to it?

## CHAPTER 4

# Advanced Python Programming

---

This chapter builds on your foundational Python knowledge, introducing powerful tools and paradigms for writing more sophisticated, efficient, and maintainable programs. You will explore regular expressions for advanced text processing, learn the principles of object-oriented programming to structure complex applications, and discover essential advanced Python features that enhance code clarity and user interaction.

Mastering these concepts will equip you to tackle a wider range of programming challenges, from data validation and parsing to designing scalable software systems. Each topic is presented with practical examples to solidify your understanding and prepare you for real-world application.

## Mastering Regular Expressions

---

Covers: CS50P - Lecture 7 - Regular Expressions (video, 2 hr 5 min)

Regular expressions, often shortened to "regex" or "regexp," are sequences of characters that define a search pattern. They are extremely powerful for pattern matching, text searching, and data validation. Python's built-in `re` module provides full support for regular expressions.

### Basic Regex Syntax

Here are some fundamental elements of regex patterns:

- `.` (dot): Matches any single character except a newline.
- `*`: Matches zero or more occurrences of the preceding character or group.
- `+`: Matches one or more occurrences of the preceding character or group.
- `?`: Matches zero or one occurrence of the preceding character or group.
- `[]`: Matches any one of the characters inside the brackets. For example, `[aeiou]` matches any vowel.
- `[^]`: Matches any character *not* inside the brackets. For example, `[^0-9]` matches any non-digit.
- `{n}`: Matches exactly `n` occurrences of the preceding character or group.
- `{n,m}`: Matches between `n` and `m` occurrences.
- `^`: Matches the start of the string.
- `$`: Matches the end of the string.
- `|`: Acts as an OR operator. For example, `cat|dog` matches "cat" or "dog".
- `\d`: Matches any digit (0-9). Equivalent to `[0-9]`.
- `\w`: Matches any word character (alphanumeric + underscore). Equivalent to `[a-zA-Z0-9_]`.
- `\s`: Matches any whitespace character (space, tab, newline, etc.).
- `\`: Used to escape special characters. For example, `\.` matches a literal dot.

## The re Module Functions

Python's `re` module offers several key functions:

- `re.search(pattern, string)`: Scans through a string looking for the first location where the regex pattern produces a match. Returns a match object if found, `None` otherwise.
- `re.match(pattern, string)`: Checks for a match *only at the beginning* of the string. Returns a match object if found, `None` otherwise.
- `re.findall(pattern, string)`: Returns a list of all non-overlapping matches of the pattern in the string.
- `re.sub(pattern, replacement, string)`: Replaces all occurrences of the pattern in the string with the replacement.

### Worked Example: Validating an Email Address

Consider validating a simple email format. A basic pattern might be `username@domain.com`.

```
import re

def validate_email(email):
    # Pattern: one or more word characters, followed by '@',
    # then one or more word characters, then a dot, then 2-3 word characters.
    # The 'r' prefix makes it a raw string, which is good for regex to avoid backslash
    # issues.
    pattern = r"^\w+@\w+\.\w{2,3}$"
    if re.search(pattern, email):
        return True
    return False

print(f"test@example.com is valid: {validate_email('test@example.com')}") # True
print(f"user123@domain.org is valid: {validate_email('user123@domain.org')}") # True
print(f"invalid-email.com is valid: {validate_email('invalid-email.com')}") # False
print(f"test@example is valid: {validate_email('test@example')}") # False
```

This pattern is a starting point. Real-world email validation is much more complex. The `^` and `$` anchors ensure the entire string matches the pattern, not just a substring.

### Common Mistakes

1. **Forgetting raw strings (`r""`):** Without `r`, backslashes (`\`) are interpreted as escape sequences by Python first, before the regex engine. This can lead to unexpected behavior or errors.
2. **Greedy vs. Non-Greedy Matching:** Quantifiers like `*` and `+` are "greedy" by default, matching as much as possible. Adding a `?` after them (e.g., `*?`, `+`) makes them "non-greedy" or "lazy," matching as little as possible.
3. **Not anchoring patterns:** If you want to match the *entire* string, use `^` at the beginning and `$` at the end of your pattern. Otherwise, `re.search` might find a match within a larger string.

## Practice Cues

- Experiment with `re.findall` to extract all phone numbers or dates from a block of text.
- Write a regex to replace all instances of a specific word (case-insensitive) with another word using `re.sub`.
- Try to build a regex that validates a simple password policy, such as requiring at least one uppercase letter, one lowercase letter, and one digit.

## Object-Oriented Programming in Python

---

Covers: CS50P - Lecture 8 - Object-Oriented Programming (video, 2 hr 48 min)

Object-Oriented Programming (OOP) is a programming paradigm based on the concept of "objects," which can contain data and code. Data is in the form of attributes (properties), and code is in the form of methods (functions). OOP helps structure programs into reusable, modular pieces, making complex applications easier to manage and scale.

### Classes and Objects

A **class** is a blueprint for creating objects. It defines the attributes and methods that all objects of that class will have. An **object** (or instance) is a specific realization of a class.

Let's define a simple `Dog` class:

```
class Dog:
    # Class attribute, shared by all instances of Dog
    species = "Canis familiaris"

    def __init__(self, name, breed):
        # The __init__ method is a special method called a constructor.
        # It initializes the object's attributes when a new object is created.
        # 'self' refers to the instance of the class.
        self.name = name # Instance attribute
        self.breed = breed # Instance attribute

    def bark(self):
        # Instance method: operates on the object's data
        return f"{self.name} says Woof!"

    def describe(self):
        return f"{self.name} is a {self.breed} of species {self.species}."

# Creating objects (instances) of the Dog class
my_dog = Dog("Buddy", "Golden Retriever")
your_dog = Dog("Lucy", "Labrador")

print(my_dog.describe()) # Output: Buddy is a Golden Retriever of species Canis
familiaris.
print(your_dog.bark()) # Output: Lucy says Woof!
```

In this example, `Dog` is the class, and `my_dog` and `your_dog` are objects. `name` and `breed` are instance attributes, unique to each dog object. `species` is a class attribute, shared by all dogs.

## Key OOP Concepts

1. **Encapsulation:** Bundling data (attributes) and methods (functions) that operate on the data into a single unit (the class). It also involves restricting direct access to some components of an object, preventing accidental modification. In Python, this is often achieved by convention (e.g., using a single underscore `_attribute` for protected, double underscore `__attribute` for name mangling).
2. **Inheritance:** A mechanism where a new class (subclass or derived class) inherits attributes and methods from an existing class (superclass or base class). This promotes code reuse and establishes a natural hierarchy.

```
class GoldenRetriever(Dog):
    def __init__(self, name, age):
        super().__init__(name, "Golden Retriever") # Call the parent class's constructor
        self.age = age

    def fetch(self):
        return f"{self.name} loves to fetch!"

golden_buddy = GoldenRetriever("Buddy", 5)
print(golden_buddy.describe()) # Inherited from Dog
print(golden_buddy.fetch()) # Specific to GoldenRetriever
print(golden_buddy.bark()) # Inherited from Dog
```

Here, `GoldenRetriever` is a subclass of `Dog`. It inherits `bark` and `describe` methods and adds its own `fetch` method and `age` attribute.

3. **Polymorphism:** The ability of different objects to respond to the same method call in their own way. For example, if you had a `Cat` class with a `make_sound` method, calling `make_sound` on a `Dog` object would produce "Woof!", while on a `Cat` object it would produce "Meow!". The method name is the same, but the behavior differs based on the object type.

## Common Mistakes

1. **Forgetting `self`:** Methods must always have `self` as their first parameter, even if it's not explicitly used within the method. `self` is how an instance refers to its own attributes and methods.
2. **Confusing class and instance attributes:** Class attributes are shared; instance attributes are unique to each object. Incorrectly using one when the other is needed can lead to bugs.
3. **Not calling `super().__init__()` in subclasses:** When a subclass has its own `__init__` method, it must explicitly call the parent class's `__init__` method using `super().__init__()` to ensure the parent's attributes are properly initialized.

## Practice Cues

- Design a `Vehicle` class with attributes like `make`, `model`, and `year`, and methods like `start_engine` and `stop_engine`. Then, create subclasses like `Car` and `Motorcycle` that inherit from `Vehicle` and add

their own specific attributes or methods.

- Implement a simple `Bank Account` class with methods for `deposit`, `withdraw`, and `get_balance`. Ensure that withdrawal logic prevents negative balances.
- Explore how to use properties (`@property` decorator) to create getter and setter methods for attributes, providing more control over attribute access.

## Advanced Python Features

---

Covers: CS50P - Lecture 9 - Et Cetera (video, 2 hr 29 min)

Beyond the core syntax and object-oriented principles, Python offers several advanced features that significantly improve the robustness, readability, and efficiency of your code. This section covers command-line argument parsing and type hints.

### Command-Line Argument Parsing with `argparse`

Many Python scripts benefit from accepting input directly from the command line. While `sys.argv` provides raw access to command-line arguments, the `argparse` module offers a more structured and user-friendly way to define and parse arguments. It automatically generates help and usage messages.

```
import argparse

def main():
    parser = argparse.ArgumentParser(description="A simple command-line script.")

    # Define arguments
    parser.add_argument("-n", "--name", type=str, default="World",
                        help="The name to greet.")
    parser.add_argument("-v", "--verbose", action="store_true",
                        help="Enable verbose output.")
    parser.add_argument("number", type=int, help="A required integer argument.")

    # Parse the arguments
    args = parser.parse_args()

    if args.verbose:
        print("Verbose mode enabled.")

    print(f"Hello, {args.name}!")
    print(f"You provided the number: {args.number}")

if __name__ == "__main__":
    main()
```

To run this script:

- `python your_script.py 100`
- `python your_script.py --name Alice 200`
- `python your_script.py -v 300`

- `python your_script.py --help` (to see the auto-generated help message)

`argparse` handles type conversion, default values, and automatically generates error messages for invalid arguments. This greatly simplifies creating robust command-line interfaces.

## Type Hints

Python is dynamically typed, meaning you do not declare variable types explicitly. While flexible, this can make large codebases harder to read and maintain. Type hints, introduced in Python 3.5 (PEP 484), allow you to *indicate* the expected types of variables, function parameters, and return values. They are optional and do not change Python's runtime behavior, but they are invaluable for code clarity, static analysis tools (like MyPy), and IDEs.

```
from typing import List, Optional, Dict

def greet(name: str) -> str:
    """Greets a person by name."""
    return f"Hello, {name}!"

def add_numbers(a: int, b: int) -> int:
    """Adds two integers and returns their sum."""
    return a + b

def process_items(items: List[str]) -> None:
    """Processes a list of strings."""
    for item in items:
        print(item.upper())

def get_user_data(user_id: int) -> Optional[Dict[str, str]]:
    """Retrieves user data, returns None if not found."""
    if user_id == 1:
        return {"name": "Alice", "email": "alice@example.com"}
    return None

print(greet("Bob"))
print(add_numbers(5, 3))
process_items(["apple", "banana"])
print(get_user_data(1))
print(get_user_data(2))
```

- `name: str` indicates `name` is expected to be a string.
- `-> str` indicates the function is expected to return a string.
- `List[str]` from the `typing` module specifies a list containing strings.
- `Optional[Dict[str, str]]` indicates the function may return either a dictionary with string keys and string values, or `None`.

Type hints improve code readability, help catch potential errors during development with static analysis tools, and make it easier for others to understand how to use your functions and classes.

## Common Mistakes

1. **Overlooking `argparse` for simple scripts:** While `sys.argv` works, `argparse` provides a much better user experience and robust error handling, even for simple cases.
2. **Not using `typing` module for complex types:** For lists, dictionaries, tuples, and `Optional` types, you must import the specific type constructs from the `typing` module.
3. **Treating type hints as strict enforcement:** Remember that type hints are suggestions for static analysis, not runtime checks. Python will still execute code even if types don't match the hints.

## Practice Cues

- Modify a previous script (e.g., a file processing tool) to accept input and output file paths, and optional flags (like `--force` or `--dry-run`) using `argparse`.
- Add type hints to all functions and methods in a small Python project you have written. Run a static type checker like `MyPy` on your code to identify any inconsistencies.
- Write a function that calculates the area of different shapes (circle, square, rectangle) using a dictionary to map shape names to functions. Use type hints for the input parameters and return values, especially for the dictionary's structure.

### KEY TAKEAWAYS

- Regular expressions provide a powerful and flexible way to search, match, and manipulate text patterns.
- Python's `re` module offers functions like `search`, `match`, `findall`, and `sub` for working with regular expressions.
- Object-Oriented Programming (OOP) uses classes as blueprints to create objects, encapsulating data and behavior.
- Key OOP principles include encapsulation, inheritance for code reuse, and polymorphism for flexible method behavior.
- `argparse` is the standard library module for creating robust and user-friendly command-line interfaces in Python.
- Type hints improve code readability and allow static analysis tools to catch potential type-related errors during development.
- Advanced Python features like `argparse` and type hints contribute to writing more maintainable and reliable software.

---

## CHECK YOUR UNDERSTANDING

Answer these before moving on. Model answers are in the Answer Key at the back of this guide.

1. What is the primary purpose of regular expressions in programming?

2. Explain the difference between `re.search()` and `re.match()` in Python's `re` module.
3. What are classes and objects in the context of Object-Oriented Programming?
4. How does `argparse` improve upon directly using `sys.argv` for command-line arguments?
5. Why are type hints beneficial in Python, even though Python is dynamically typed?

## REFERENCE

# Glossary

---

**Algorithm**

A precise, step-by-step procedure for solving a problem or performing a computation.

**Argparse**

A Python standard library module for parsing command-line arguments, providing a structured way to define and process user input.

**Array**

A data structure that stores a fixed-size, sequential collection of elements of the same data type.

**Artificial intelligence (ai)**

A field of computer science focused on creating machines capable of intelligent behavior, including learning and problem-solving.

**Attribute**

A variable associated with an object or a class, representing a piece of data or a property.

**Big o notation**

A mathematical notation used to describe the limiting behavior of a function, typically used to classify algorithm efficiency.

**C (programming language)**

A foundational, procedural programming language known for its efficiency and low-level memory access.

**Class**

A blueprint or template for creating objects, defining their attributes (data) and methods (functions).

**Computational thinking**

A problem-solving process that includes decomposition, pattern recognition, abstraction, and algorithms.

**Conditional statement**

A programming construct (e.g., `if`, `elif`, `else`) that executes different code blocks based on whether specified conditions are true or false.

**Css**

Cascading Style Sheets, a style sheet language used for describing the presentation of a document written in HTML or XML.

**Cybersecurity**

The practice of protecting computer systems, networks, and data from digital attacks, damage, or unauthorized access.

**Data structure**

A particular way of organizing and storing data in a computer for efficient access and modification.

**Exception**

A runtime error that interrupts the normal flow of a program, which can be caught and handled to prevent crashes.

**File i/o**

The process by which a computer program reads data from or writes data to a file on a storage device.

**Flask**

A lightweight Python web framework that provides tools and libraries for building web applications.

**Function**

A reusable block of code designed to perform a specific task, often taking inputs and returning outputs.

**Html**

HyperText Markup Language, the standard markup language for creating web pages and web applications.

**Inheritance**

An OOP mechanism where a new class (subclass) derives attributes and methods from an existing class (superclass), promoting code reuse.

**Javascript**

A high-level, interpreted programming language that enables interactive web pages and is an essential part of web applications.

**Loop**

A programming construct (e.g., `for`, `while`) that repeatedly executes a block of code until a certain condition is met or a sequence is exhausted.

**Method**

A function that belongs to an object and operates on its data, defined within a class.

**Module (library)**

A file containing Python definitions and statements that can be imported and used in other Python programs to extend functionality.

**Object (instance)**

A specific realization or creation of a class, possessing the attributes and methods defined by its class.

**Pointer**

A variable that stores the memory address of another variable or a memory location.

**Python (programming language)**

A high-level, interpreted, dynamically typed programming language known for its readability and versatility.

**Regular expression (regex)**

A sequence of characters that defines a search pattern, used for pattern matching, text searching, and data validation.

**Relational database**

A type of database that stores and provides access to data points that are related to one another, organized into tables with rows and columns.

**Sql**

Structured Query Language, a domain-specific language used to manage data in relational database management systems.

**Sql injection**

A code injection technique used to attack data-driven applications, in which malicious SQL statements are inserted into an entry field for execution.

**Type hint**

An optional notation in Python code that indicates the expected data type of variables, function parameters, and return values, enhancing readability and enabling static analysis.

**Unit test**

A small, isolated test that verifies a specific piece of code, typically a function, works correctly under various conditions.

**Variable**

A named storage location in a program that holds a value, which can change during execution.

## REFERENCE

# Answer Key

---

## Chapter 1: Computer Science Foundations

### 1. What are the four pillars of computational thinking, and how does Scratch help illustrate them?

The four pillars are decomposition, pattern recognition, abstraction, and algorithms. Scratch helps illustrate these by allowing users to break down problems into smaller scripts (decomposition), reuse code blocks (pattern recognition), focus on block functionality rather than syntax (abstraction), and arrange blocks in a logical sequence (algorithms).

### 2. Explain the key differences between C and Python in terms of typing and execution.

C is a statically typed language, requiring explicit type declarations for variables, and it is compiled into machine code before execution. Python is dynamically typed, inferring variable types at runtime, and it is an interpreted language, executing code line by line without a separate compilation step.

### 3. Describe the concept of an array in C, including how elements are indexed and what a null terminator signifies for strings.

An array in C is a contiguous block of memory storing elements of the same data type. Elements are accessed using a 0-based index. For example, the first element is at index 0. For strings, which are character arrays, a null terminator (`\0`) is a special character that marks the end of the string, allowing functions to determine its length.

### 4. What is Big O notation, and why is it important when discussing algorithms? Provide an example of an $O(n)$ and an $O(\log n)$ algorithm.

Big O notation describes the upper bound on the growth rate of an algorithm's running time or space requirements as the input size increases. It is important for comparing the efficiency of different algorithms. A linear search is an  $O(n)$  algorithm, while a binary search on a sorted array is an  $O(\log n)$  algorithm.

### 5. How do pointers relate to memory addresses, and what is the purpose of `malloc` and `free` in C?

Pointers are variables that store memory addresses. They allow direct manipulation of data at specific memory locations. `malloc` is used for dynamic memory allocation, requesting a block of memory from the heap during program execution. `free` is used to release that dynamically allocated memory back to the system, preventing memory leaks.

### 6. Briefly describe two different data structures discussed in this chapter and one scenario where each might be preferred.

An array is a good choice when you need fast random access to elements by index and the size of the collection is fixed or known in advance, such as storing a list of test scores. A linked list is preferred when frequent insertions or deletions are needed in the middle of the sequence, as it avoids shifting elements, for example, in managing a playlist where songs are often added or removed.

## Chapter 2: Computer Science - Part 2

### 1. What are the three core languages used for front-end web development and what is the primary role of each?

The three core languages are HTML, CSS, and JavaScript. HTML provides the structure and content of a web page. CSS is used for styling the appearance of the HTML elements. JavaScript adds interactivity and dynamic behavior to the web page.

### 2. Explain the purpose of a primary key and a foreign key in a relational database.

A primary key uniquely identifies each record in a table, ensuring that no two rows are identical. A foreign key is a field (or collection of fields) in one table that refers to the primary key in another table. It establishes a link between tables, enforcing referential integrity.

### 3. How does Flask facilitate the creation of dynamic web pages?

Flask facilitates dynamic web pages by allowing developers to define routes that map URLs to Python functions. These functions can process data, interact with databases, and then use templating engines like Jinja2 to embed dynamic Python variables and logic directly into HTML files before sending them to the user's browser.

### 4. Describe the CIA triad in cybersecurity and provide a brief example for each component.

The CIA triad stands for Confidentiality, Integrity, and Availability. Confidentiality means protecting data from unauthorized access, such as encrypting sensitive files. Integrity ensures data is accurate and has not been tampered with, for example, by using checksums to verify file integrity. Availability ensures authorized users can access systems and data when needed, like implementing redundant servers to prevent downtime.

### 5. What is SQL injection and how can developers help prevent it?

SQL injection is a security vulnerability where an attacker inserts malicious SQL code into input fields to trick a database into executing unintended commands. Developers can prevent it by implementing strong input validation, using parameterized queries, or employing Object Relational Mappers (ORMs) which automatically sanitize inputs.

### 6. When building a web application, what is the main distinction between client-side and server-side scripting?

Client-side scripting, typically using JavaScript, runs in the user's web browser and handles user interface interactions and dynamic content updates without requiring a full page reload. Server-side scripting, such as with Flask and Python, runs on the web server, handles data processing, database interactions, user authentication, and generates the HTML content that is then sent to the client's browser.

## Chapter 3: Programming with Python - Part 1

### 1. How do you define a function in Python and what are its main components?

You define a function using the `def` keyword, followed by the function's name and parentheses. The main components include the `def` keyword, the function name, optional parameters within the

parentheses, a colon, and an indented block of code that forms the function's body. Functions can optionally return a value using the `return` keyword.

**2. Explain the difference between a `while` loop and a `for` loop, and provide a use case for each.**

A `while` loop repeatedly executes a block of code as long as a specified condition remains true; it is suitable when the number of iterations is unknown, such as repeatedly asking for valid user input. A `for` loop iterates over a sequence (like a list or range of numbers); it is ideal when you need to process each item in a collection or iterate a known number of times, such as printing numbers from 1 to 10.

**3. Why is it good practice to use `try` and `except` blocks in your code?**

Using `try` and `except` blocks is good practice because it allows your program to handle runtime errors, known as exceptions, gracefully. Instead of crashing, the program can catch specific errors, provide informative messages to the user, or attempt to recover, making the software more robust and user-friendly.

**4. What is the purpose of importing a module, and how do you do it?**

The purpose of importing a module is to reuse code that has already been written by others, extending your program's functionality without having to write everything from scratch. You import a module using the `import` statement, for example, `import random` to use functions from the random module, or `from math import sqrt` to import a specific function.

**5. Describe two common file modes used when opening a file and what they do.**

Two common file modes are `'r'` for read mode and `'w'` for write mode. Read mode (`'r'`) opens a file for reading its content; the file must already exist. Write mode (`'w'`) opens a file for writing; if the file exists, its content is truncated (emptied), and if it does not exist, a new file is created.

**6. What is a unit test, and how does an `assert` statement contribute to it?**

A unit test is a small, isolated test designed to verify that a specific function or a small part of a program works correctly. An `assert` statement contributes to a unit test by checking if a given condition is true. If the condition is false, it raises an `AssertionError`, indicating that the test has failed and the code is not behaving as expected.

## Chapter 4: Advanced Python Programming

**1. What is the primary purpose of regular expressions in programming?**

The primary purpose of regular expressions is to define and match text patterns. They are used for searching for specific sequences of characters, validating input formats, and performing complex text manipulations like finding and replacing substrings based on a pattern. This capability is crucial for tasks involving data parsing and string processing.

**2. Explain the difference between `re.search()` and `re.match()` in Python's `re` module.**

`re.search(pattern, string)` scans the entire `string` to find the first location where the `pattern` produces a match, returning a match object if found anywhere. In contrast, `re.match(pattern, string)` only attempts to match the `pattern` at the *beginning* of the `string`. If the pattern does not start at index 0, `re.match` will return `None`.

### 3. What are classes and objects in the context of Object-Oriented Programming?

In OOP, a class serves as a blueprint or a template that defines the structure and behavior of a particular type of entity. An object, also known as an instance, is a concrete realization of that class. For example, 'Car' could be a class, and 'my\_red\_sedan' would be an object of the 'Car' class, having specific attributes like color and model.

### 4. How does `argparse` improve upon directly using `sys.argv` for command-line arguments?

`argparse` provides a structured and more user-friendly way to handle command-line arguments compared to `sys.argv`. It allows you to define arguments with types, default values, and help messages, automatically parses them, and generates comprehensive help documentation. This reduces boilerplate code and makes scripts more robust and easier for users to interact with.

### 5. Why are type hints beneficial in Python, even though Python is dynamically typed?

Type hints are beneficial because they improve code readability and maintainability, especially in larger projects or when working in teams. They provide clear expectations for function parameters and return values, which aids in understanding code logic. Also, static analysis tools and IDEs can use type hints to identify potential type-related errors before runtime, enhancing code quality and reducing bugs.

## CLOSING

# Where to Go Next

---

Congratulations on completing "The Complete Harvard CS50: Computer Science & Python." To consolidate your skills, continue practicing. Build small projects independently. Apply the concepts you have learned to new problems.

Consider contributing to open-source projects. Participate in coding challenges. These activities will deepen your understanding and build your portfolio. Practical application is key to mastery.

This course prepares you for many advanced topics. You can pursue specializations in web development, data science, or artificial intelligence. Explore frameworks like Django or Flask for web. Study libraries like NumPy and Pandas for data science. This foundation will serve you well in any of these fields.



## MEGACOURSE

*Learn Anything. Achieve Everything.*

Track your progress, take the quizzes, and earn your certificate for this course at  
**[megacourse.app/courses/the-complete-harvard-cs50-computer-science-and-python](https://megacourse.app/courses/the-complete-harvard-cs50-computer-science-and-python)**