# Lecture 1 Handout

## Introduction to Computers and Python

INF 605 - Introduction to Programming - Python

**Prof. Rongyu Lin**
**Quinnipiac University**
School of Computing and Engineering

Fall 2025

## Learning Objectives

**By the end of this lecture, you will be able to:**

1. **Explain** the relationship between hardware, software, and programming
2. **Understand** why Python is an excellent first programming language
3. **Set up** and use Python development environments (Google Colab, Jupyter)
4. **Write** and execute Python programs with proper syntax
5. **Create and use** variables to store different data types
6. **Perform** mathematical calculations including order of operations
7. **Format output** using f-strings and print statements
8. **Use** basic math library functions
9. **Appreciate** Python's role in data science and real-world applications

## 1 Today's Learning Journey

This lecture is structured as a three-part journey designed to build your programming foundation step by step:

### Part I: Getting to Know Each Other (15 min)

- Icebreaker activities and programming background survey
- Exploring programming in daily life
- Setting course goals and expectations

### Part II: Programming Foundations (15 min)

- Understanding what programming really is

- Computer fundamentals: hardware and software
- Evolution of programming languages

**Part III: Python Programming (45 min)**

- Why Python is perfect for beginners
- Setting up development environments
- Writing first programs with variables and data types
- Preview of Python's incredible power

**Goal:** By the end of today, you'll be confident writing Python programs with variables and data types!

# 2 Part I: Getting to Know Each Other

## 2.1 Programming Background Survey

Let's start by understanding where everyone is coming from with programming:

**Reflection Questions:**

- Have you programmed before? What languages have you tried?
- What was your first programming experience like?
- If you're a complete beginner, don't worry - you're in great company!

**Fun Fact:** Every expert programmer started exactly where you are now! The key is persistence and practice.

*"The best time to plant a tree was 20 years ago. The second best time is now."*

## 2.2 Programming is Everywhere!

Let's explore how programming touches every aspect of our daily lives:

**Think-Pair-Share Activity:** Where do you encounter programming in daily life?

**Examples you might not have considered:**

- **Your smartphone:** Every app, every swipe, every notification
- **Social media:** Algorithms deciding what you see
- **Transportation:** GPS navigation, ride-sharing apps, traffic lights
- **Entertainment:** Netflix recommendations, Spotify playlists, video games
- **Shopping:** E-commerce sites, price comparisons, inventory management
- **Education:** Learning management systems, online courses, digital textbooks

**Python Powers:** Instagram, Netflix, Spotify, NASA, Google, and countless financial institutions!

Programming isn't just for "tech people" - it's a valuable skill for everyone in the modern world.

## 2.3 Your Programming Goals

**Partner Activity:** Share with a neighbor what you hope to build or accomplish with programming.

**Possible goals and what Python can help you achieve:**

- **Data Analysis:** Analyze sports statistics, financial data, research findings
- **Web Development:** Create websites and web applications
- **Automation:** Automate repetitive tasks, organize files, send emails
- **Games & Apps:** Build games, mobile apps, desktop applications
- **AI & Machine Learning:** Create intelligent systems, chatbots, image recognition
- **Career Advancement:** Add valuable skills to any field of study

**Fun Python Facts:**

- Named after "Monty Python's Flying Circus" (not the snake!)
- Most popular programming language in the world (2024)
- Used by 8.2 million developers worldwide

## 2.4 Welcome to INF 605!

### Introduction to Programming - Python

- **Practical Focus:** Real-world applications and projects
- **Interactive Learning:** Hands-on coding every class session
- **Modern Approach:** Industry-standard tools and practices
- **Data Science Ready:** Foundation for analytics and AI careers
- **Project-Based:** Build portfolio-worthy applications

**Course Philosophy:** *"Learning programming is like learning to drive - you need practice behind the wheel!"*

**Every class:** Code → Practice → Build

We emphasize:

- **Active Learning:** You'll write and run code in every class session
- **Practical Applications:** Focus on real-world problems and solutions
- **Conceptual Understanding:** Learn not just how to code, but why code works
- **Modern Tools:** Use industry-standard development environments and practices
- **Collaborative Learning:** Work together to solve problems and learn from each other

# 3 Part II: Programming Foundations

## 3.1 What is Programming?

## 3.2 Programming as Problem Solving

Programming is fundamentally about problem solving. When we program, we:

1. **Identify a problem** that needs to be solved
2. **Break it down** into smaller, manageable pieces
3. **Design an algorithm** (step-by-step solution)
4. **Implement the algorithm** in a programming language
5. **Test and refine** the solution

## 3.3 Programming as Communication

Programs are instructions we give to computers, but they're also communication tools. A well-written program communicates:

- **To the computer:** What operations to perform
- **To other programmers:** How the solution works
- **To your future self:** Why certain decisions were made

## 3.4 Real-World Analogy: Cooking Recipe

Programming is like writing a detailed recipe:

- **Recipe = Program:** Step-by-step instructions
- **Ingredients = Data:** The information or materials you work with
- **Cooking steps = Code:** The specific actions to perform
- **Final dish = Output:** The result of following the instructions

Just as a recipe needs to be precise and complete for anyone to follow it successfully, a program needs to be clearly written and logically structured.

## 3.5 Computer Fundamentals: Hardware and Software

## 3.6 Computer Hardware

Computer hardware consists of the physical components that make up a computer system:

**Central Processing Unit (CPU):**

- Often called the "brain" of the computer
- Executes program instructions
- Modern CPUs can perform billions of operations per second
- Examples: Intel Core i7, AMD Ryzen, Apple M1

**Memory (RAM - Random Access Memory):**

- Temporary storage for data and programs currently in use
- Much faster than permanent storage
- Volatile - contents are lost when power is turned off
- Typical capacities: 8GB, 16GB, 32GB or more

**Storage Devices:**

- Permanent storage for data and programs
- Hard Disk Drives (HDDs): Slower but inexpensive
- Solid State Drives (SSDs): Faster, more reliable, more expensive
- Cloud storage: Remote storage accessible via internet

**Input/Output Devices:**

- Input: Keyboard, mouse, touchscreen, microphone, camera
- Output: Monitor, speakers, printer
- Network interfaces: Ethernet, Wi-Fi, Bluetooth

### 3.7 Computer Software

Software consists of the instructions that tell hardware what to do:

**System Software:**

- **Operating Systems:** Windows, macOS, Linux, iOS, Android
- **Device Drivers:** Allow OS to communicate with hardware
- **Utilities:** System maintenance and management tools

**Application Software:**

- **Productivity:** Microsoft Office, Google Workspace
- **Web Browsers:** Chrome, Firefox, Safari, Edge
- **Media:** Photoshop, VLC, Spotify
- **Games:** Everything from simple mobile games to complex AAA titles

**Programming Software:**

- **Text Editors:** VS Code, Sublime Text, Vim
- **IDEs:** PyCharm, Visual Studio, Xcode
- **Compilers and Interpreters:** Python interpreter, Java compiler

### 3.8 The Hardware-Software Relationship

The relationship between hardware and software is symbiotic:

- **Hardware provides the platform** for software to run
- **Software gives hardware its functionality**
- **Neither is useful without the other**
- **Performance depends on both** hardware capabilities and software efficiency

## 3.9 Understanding Data: The Data Hierarchy (Background Information)

Understanding how computers organize and store information is crucial for programmers. Data is organized in a hierarchy from the smallest unit (bits) to complex structures (databases).

## 3.10 Bits and Bytes

**Bits (Binary Digits):**

- The smallest unit of data in computing
- Can hold only one of two values: 0 or 1
- Everything in a computer is ultimately represented in binary
- Example: The letter 'A' is represented as 01000001

**Bytes:**

- A group of 8 bits
- Can represent 256 different values ($2^8$)
- Standard unit for measuring storage and memory
- One byte typically stores one character

## 3.11 Characters and Strings

**Characters:**

- Individual letters, numbers, symbols
- Encoded using standards like ASCII or Unicode
- ASCII: 128 characters (English letters, numbers, basic symbols)
- Unicode: Millions of characters (supports all languages, emojis)

**Strings:**

- Sequences of characters
- Can represent words, sentences, or any text
- Example: "Hello, World!" is a string of 13 characters

## 3.12 Fields and Records

**Fields:**

- Individual data items about an entity
- Examples: first_name, last_name, age, email
- Have specific data types (text, number, date, etc.)

**Records:**

- Collections of related fields
- Represent information about a single entity
- Example: One student's complete information (name, ID, GPA, major)

## 3.13 Files and Databases

**Files:**

- Collections of related records
- Examples: student roster, employee database, customer list
- Can be stored in various formats (CSV, JSON, XML)

**Databases:**

- Collections of related files
- Include relationships between different types of data
- Examples: University system (students, courses, grades, faculty)
- Use database management systems (MySQL, PostgreSQL, MongoDB)

## 3.14 Programming Language Evolution

## 3.15 Machine Language (First Generation)

Machine language is the lowest level of programming language:

- Instructions written in binary (0s and 1s)
- Directly executed by the CPU
- Extremely difficult for humans to read and write
- Different for each type of processor
- Very fast execution, no translation needed

Example of machine language:

```
10110000 01100001   # Load value 97 into register
10110011 01100010   # Load value 98 into register
00000001 11000011   # Add the values
```

## 3.16 Assembly Language (Second Generation)

Assembly language uses symbolic representations:

- Uses mnemonics (memory aids) instead of binary
- One-to-one correspondence with machine language
- Still processor-specific
- Requires an assembler to convert to machine language
- More readable than machine language, but still complex

Example of assembly language:

```
MOV AX, 97    ; Load 97 into register AX
MOV BX, 98    ; Load 98 into register BX
ADD AX, BX    ; Add BX to AX
```

### 3.17 High-Level Languages (Third Generation)

High-level languages are closer to human language:

- Use English-like syntax and mathematical notation
- Portable across different computer systems
- Require compilation or interpretation
- Much easier to learn and use
- Examples: Python, Java, C++, JavaScript

Example in Python:

```
result = 97 + 98
print(f"The result is: {result}")
```

### 3.18 Very High-Level Languages (Fourth Generation)

These languages are even more abstracted:

- Focus on what to do rather than how to do it
- Often domain-specific
- Examples: SQL (database queries), MATLAB (mathematical computing)
- Very productive for specific types of problems

Example in SQL:

```
SELECT name, age FROM students WHERE gpa > 3.5;
```

## 4 Part III: Python Programming

### 4.1 Why Python? The Perfect Learning Language

### 4.2 Python's History and Philosophy

Python was created by Guido van Rossum in 1991 with the goal of creating a programming language that was:

- Easy to read and write
- Powerful enough for serious applications
- Fun to use (named after Monty Python's Flying Circus)

The **Zen of Python** includes principles like:

- "Beautiful is better than ugly"
- "Explicit is better than implicit"
- "Simple is better than complex"
- "Readability counts"

## 4.3 Python's Advantages for Beginners

**Readable Syntax:**

- Uses indentation to define code blocks (no curly braces)
- Keywords are English words (if, else, for, while)
- Minimal punctuation compared to other languages

## 4.4 Python Indentation: The Foundation of Python Syntax

**CRITICAL CONCEPT:** Python uses **indentation** (spaces or tabs) to define code structure, unlike most programming languages that use curly braces {} or keywords like `begin/end`.

**Why Indentation Matters:**

- **Enforces readable code:** Python forces you to write visually organized code
- **Prevents errors:** Proper indentation reduces logic mistakes
- **Universal standard:** All Python programmers follow the same visual structure
- **Natural grouping:** Related code lines are visually grouped together

**The Golden Rules of Python Indentation:**

1. **Consistent Indentation:** Use the same amount of spaces/tabs throughout your program
2. **Standard Practice:** Use 4 spaces per indentation level (recommended by Python)
3. **No Mixing:** Never mix spaces and tabs in the same file
4. **Visual Structure:** Indented code belongs to the line above it
5. **After Colons:** Any line ending with : requires indentation on the next line

**Indentation in Action:**

```
# Example with proper indentation
age = 20
if age >= 18:
    print("You are an adult!")           # 4 spaces indentation
    print("You can vote!")               # 4 spaces indentation
    print("You have responsibilities!")  # 4 spaces indentation
print("This line always runs")          # No indentation

# Multiple indentation levels (nested structure)
temperature = 85
if temperature > 70:
    print("It's warm outside!")                  # 1st level: 4 spaces
    if temperature > 80:
        print("Actually, it's hot!")             # 2nd level: 8 spaces
        print("Stay hydrated!")                  # 2nd level: 8 spaces
    print("Great day for outdoor activities!") # Back to 1st level: 4
        spaces
```

**Common Beginner Mistakes:**

- **Missing indentation:** Forgetting to indent after :
- **Inconsistent spacing:** Mixing 2 spaces and 4 spaces

- **Mixing tabs and spaces:** Using both in the same file
- **Wrong indentation level:** Not matching the expected structure

**Error Examples (Do NOT do this):**

```
# ERROR: Missing indentation after colon
if age >= 18:
print("This will cause IndentationError!")

# ERROR: Inconsistent indentation
if age >= 18:
    print("Four spaces")      # 4 spaces (correct)
  print("Two spaces")         # 2 spaces (inconsistent!)
```

**Key Takeaway:** Indentation in Python isn't just style—it's syntax! Master this concept and you'll avoid 90% of beginner Python errors. We'll practice this extensively in Lecture 2 with conditional statements.

**Interactive Nature:**

- Can run code immediately without compilation
- Interactive shell for experimentation
- Immediate feedback helps learning

**Extensive Library Support:**

- "Batteries included" standard library
- Huge ecosystem of third-party libraries
- Package manager (pip) makes installation easy

**Versatility:**

- Web development (Django, Flask)
- Data science and machine learning (pandas, scikit-learn)
- Desktop applications (tkinter, PyQt)
- System administration and automation
- Scientific computing (NumPy, SciPy)

### 4.5 Python vs Other First Languages

| Aspect | Python | Java | C++ |
|---|---|---|---|
| Hello World | `print("Hello")` | 5+ lines with classes | Complex syntax |
| Memory Management | Automatic | Automatic with GC | Manual |
| Learning Curve | Gentle | Moderate | Steep |
| Industry Use | Very High | Very High | High |
| Beginner Friendly | Excellent | Good | Poor |

### 4.6 Python's Popularity and Industry Adoption

According to various programming language indices:

- **PYPL Index:** Python is #1 globally
- **Stack Overflow Survey:** Among top 3 most popular languages
- **GitHub:** One of the most used languages
- **Job Market:** High demand for Python developers

**Companies Using Python:**

- **Google:** YouTube, Search, Gmail backend
- **Netflix:** Recommendation algorithms, content delivery
- **Instagram:** Web backend serving billions of users
- **Spotify:** Music recommendation and data analysis
- **NASA:** Mission planning and data analysis
- **Financial Firms:** Trading algorithms, risk analysis

### 4.7 Preview: The Power of Python Libraries

Before we dive into basic programming, let's get excited about where this journey will take us! Libraries give Python superpowers by providing pre-written code for complex tasks.

**What are Libraries?** Libraries are collections of pre-written code that provide specific functionality:

- **Reusable code:** Solve common problems without starting from scratch
- **Tested and optimized:** Created and maintained by experts
- **Time-saving:** Focus on your unique problem, not basic functionality
- **Community-driven:** Thousands of contributors worldwide

**Math Library Preview:**

```
import math
radius = 7.5
area = math.pi * radius ** 2
print(f"Circle area: {area:.2f}")
print(f"Square root of 16: {math.sqrt(16)}")
print(f"Sin of 90 degrees: {math.sin(math.pi/2)}")
```

Don't worry about understanding this code yet - we'll build up to it step by step!

## 5 The Complete Power of Libraries

### 5.1 What are Libraries?

Libraries are collections of pre-written code that provide specific functionality:

- **Reusable code:** Solve common problems without starting from scratch
- **Tested and optimized:** Created and maintained by experts
- **Time-saving:** Focus on your unique problem, not basic functionality
- **Standardized:** Common interfaces and conventions

## 5.2 Python Standard Library

Python comes with a comprehensive standard library "batteries included":

**Common Standard Library Modules:**

- **math:** Mathematical functions (sin, cos, sqrt, pi)
- **random:** Generate random numbers and make random choices
- **datetime:** Work with dates and times
- **os:** Interact with the operating system
- **json:** Work with JSON data format
- **urllib:** Download data from web URLs
- **re:** Regular expressions for text pattern matching

**Example using the math library:**

```
import math

# Calculate the area of a circle
radius = 5
area = math.pi * radius ** 2
print(f"Area of circle with radius {radius}: {area:.2f}")

# Use trigonometric functions
angle = math.pi / 4  # 45 degrees in radians
print(f"sin(45$^{\circ}$) = {math.sin(angle):.3f}")
print(f"cos(45$^{\circ}$) = {math.cos(angle):.3f}")

# Other useful math functions
print(f"Square root of 16: {math.sqrt(16)}")
print(f"2 to the power of 8: {math.pow(2, 8)}")
```

## 5.3 Third-Party Libraries

Beyond the standard library, Python has a vast ecosystem of third-party libraries:

**Data Science Libraries:**

- **NumPy:** Numerical computing with arrays
- **pandas:** Data manipulation and analysis
- **matplotlib:** Creating plots and visualizations
- **seaborn:** Statistical data visualization
- **scikit-learn:** Machine learning algorithms

**Web Development:**

- **requests:** HTTP library for API calls
- **Flask:** Lightweight web framework
- **Django:** Full-featured web framework

- **FastAPI:** Modern API framework

**Automation and Scraping:**

- **selenium:** Web browser automation
- **beautifulsoup4:** HTML/XML parsing
- **pillow:** Image processing

## 5.4 Installing and Using Libraries

Python's package manager **pip** makes library installation easy:

```
# Install a single library
pip install pandas

# Install multiple libraries
pip install numpy matplotlib seaborn

# Install from requirements file
pip install -r requirements.txt

# Upgrade a library
pip install --upgrade pandas
```

**Using libraries in your code:**

```
# Import entire library
import pandas

# Import with alias
import pandas as pd

# Import specific functions
from math import sqrt, pi

# Import all functions (generally not recommended)
from math import *
```

## 5.5 Preview: Object-Oriented Programming

Object-Oriented Programming (OOP) is an advanced concept we'll explore later in the course. For now, just know that Python makes it easy to organize code around real-world concepts.

**Quick Preview - Don't worry about understanding this yet!**

```
class Student:
    def __init__(self, name, gpa):
        self.name = name
        self.gpa = gpa

    def is_honor_student(self):
        return self.gpa >= 3.5

# Create a student object
alice = Student("Alice Johnson", 3.75)
print(f"{alice.name} honor student: {alice.is_honor_student()}")
```

This advanced topic will make sense after we master the fundamentals!

# 6 Introduction to Object-Oriented Programming (Advanced Preview)

## 6.1 What is Object-Oriented Programming?

Object-Oriented Programming (OOP) is a programming paradigm that organizes code around objects rather than functions and logic.

**Key Principles:**

- **Encapsulation:** Bundle data and methods that work on that data
- **Inheritance:** Create new classes based on existing classes
- **Polymorphism:** Use the same interface for different types
- **Abstraction:** Hide complex implementation details

## 6.2 OOP Concepts with Real-World Analogies

**Classes and Objects:**

- **Class:** A blueprint or template (like a house blueprint)
- **Object:** An instance created from a class (actual house built from blueprint)
- **Multiple objects:** Many instances can be created from one class

**Example: Car Class**

```
class Car:
    def __init__(self, make, model, year, color):
        # Attributes (properties)
        self.make = make
        self.model = model
        self.year = year
        self.color = color
        self.speed = 0
        self.is_running = False

    # Methods (actions the car can perform)
    def start(self):
        self.is_running = True
        return f"The {self.color} {self.make} {self.model} is now
            running"

    def accelerate(self, amount):
        if self.is_running:
            self.speed += amount
            return f"Speed increased to {self.speed} mph"
        else:
            return "Please start the car first"

    def brake(self, amount):
        self.speed = max(0, self.speed - amount)
        return f"Speed decreased to {self.speed} mph"

    def stop(self):
        self.speed = 0
        self.is_running = False
```

```
        return "Car stopped"

# Creating objects (instances) from the Car class
my_car = Car("Toyota", "Camry", 2022, "blue")
friend_car = Car("Honda", "Civic", 2021, "red")

# Using the objects
print(my_car.start())           # "The blue Toyota Camry is now running
    "
print(my_car.accelerate(30))    # "Speed increased to 30 mph"
print(my_car.brake(10))         # "Speed decreased to 20 mph"

print(friend_car.start())       # "The red Honda Civic is now running"
print(friend_car.accelerate(45)) # "Speed increased to 45 mph"
```

### 6.3 Benefits of Object-Oriented Programming

- **Modularity:** Code is organized into logical units
- **Reusability:** Classes can be reused in different programs
- **Maintainability:** Easier to modify and extend code
- **Real-world modeling:** Natural way to represent real-world entities
- **Collaboration:** Teams can work on different classes independently

### 6.4 Python Development Environments

### 6.5 Text Editors and IDEs

**Simple Text Editors:**

- **Advantages:** Lightweight, fast, available everywhere
- **Examples:** Notepad++, Sublime Text, Atom
- **Best for:** Quick scripts, simple programs

**Integrated Development Environments (IDEs):**

- **Features:** Code completion, debugging, project management
- **Popular Python IDEs:**
  - **PyCharm:** Full-featured, great for large projects
  - **VS Code:** Lightweight, extensible, very popular
  - **Spyder:** Scientific computing focus
  - **IDLE:** Comes with Python, good for beginners

### 6.6 Interactive Python Environments

**Python Interactive Shell:**

- Basic interactive mode
- Start by typing `python` in terminal

- Limited features, basic functionality

**IPython (Interactive Python):**

- Enhanced interactive shell
- Features: Syntax highlighting, tab completion, command history
- Magic commands for special operations
- Better error messages and debugging

**Jupyter Notebooks:**

- Web-based interactive environment
- Mix code, text, equations, and visualizations
- Excellent for learning, data analysis, and sharing
- Used extensively in data science and research

## 6.7 Setting Up Your Development Environment

**Installing Python:**

1. Visit `https://python.org`
2. Download Python 3.x (latest version)
3. Run installer, make sure to check "Add to PATH"
4. Verify installation: `python --version`

**Installing IPython:**

```
pip install ipython
```

**Installing Jupyter:**

```
pip install jupyter
# or install JupyterLab (newer interface)
pip install jupyterlab
```

**Starting Different Environments:**

```
# Start basic Python shell
python

# Start IPython
ipython

# Start Jupyter Notebook
jupyter notebook

# Start JupyterLab
jupyter lab
```

## 6.8 Your First Python Programs

## 6.9 The Traditional "Hello, World!" Program

Every programmer's journey begins with "Hello, World!" - a simple program that displays text:

```
print("Hello, World!")
```

This simple line demonstrates several important concepts:

- **print():** A built-in function that displays output
- **Strings:** Text enclosed in quotes
- **Function calls:** Using parentheses to execute a function

**Variations to try:**

```
print("Hello, Quinnipiac University!")
print("Welcome to INF 605!")
print("My name is", "Alice")  # Multiple arguments
print("Python", "is", "awesome", sep=" - ")  # Custom separator
```

## 6.10 Python as a Powerful Calculator

Python excels as an interactive calculator and can handle all kinds of mathematical operations. Let's explore step by step:

**Basic Arithmetic Operations:**

```
# Addition and subtraction
print(15 + 25)       # 40
print(50 - 18)       # 32

# Multiplication and division
print(6 * 7)         # 42
print(84 / 4)        # 21.0 (regular division always returns float)
print(2 ** 10)       # 1024 (exponentiation: 2 to the power of 10)
```

**More Advanced Operations:**

```
# Floor division and modulus
print(17 // 4)       # 4 (floor division returns integer part)
print(17 % 4)        # 1 (modulus returns remainder)
print(15 // 3)       # 5 (no remainder, clean division)
print(20 % 4)        # 0 (no remainder)
```

**Order of Operations (PEMDAS):** Python follows standard mathematical order of operations:

```
print(2 + 3 * 4)           # 14 (multiplication before addition)
print((2 + 3) * 4)         # 20 (parentheses first)
print(2 ** 3 * 4)          # 32 (exponentiation before multiplication)
print(2 ** (3 * 4))        # 4096 (parentheses change everything!)
print(10 - 3 * 2)          # 4 (multiplication before subtraction)
print((10 - 3) * 2)        # 14 (parentheses change the result)
```

**Working with Variables for Practical Calculations:** Variables make calculations more organized and reusable:

```python
# Shopping calculation example
price = 29.99
quantity = 3
tax_rate = 0.08

# Step-by-step calculation
subtotal = price * quantity
tax_amount = subtotal * tax_rate
total_cost = subtotal + tax_amount

# Professional output formatting (we'll learn f-strings next!)
print(f"Item price: ${price:.2f}")
print(f"Quantity: {quantity}")
print(f"Subtotal: ${subtotal:.2f}")
print(f"Tax ({tax_rate*100}%): ${tax_amount:.2f}")
print(f"Total cost: ${total_cost:.2f}")
```

**Advanced Math with Libraries:**

```python
import math

# Mathematical functions
print(f"Square root of 16: {math.sqrt(16)}")      # 4.0
print(f"Value of pi: {math.pi}")                  # 3.14159...

# Calculate circle area
radius = 7.5
area = math.pi * radius ** 2
print(f"Circle area: {area:.2f}")
```

## 6.11 Understanding Python Data Types

Python automatically recognizes different types of data, which makes programming much easier for beginners!

**Numbers:**

- **Integers (whole numbers):** `age = 20`, `year = 2025`
- **Floats (decimal numbers):** `gpa = 3.75`, `price = 29.99`

**Text (Strings):**

- `name = "Alice Johnson"`
- `university = "Quinnipiac University"`
- Must be enclosed in quotes (single or double)!

**True/False (Booleans):**

- `is_enrolled = True`
- `has_scholarship = False`
- Note: `True` and `False` are capitalized in Python

**Key Point:** Python is smart - you don't need to declare types like in other languages!

**Examples with Different Data Types:**

```python
# Different types of variables
student_name = "Alice Johnson"       # String
student_age = 20                     # Integer
student_gpa = 3.75                   # Float
is_enrolled = True                   # Boolean

# Python automatically knows the types!
print(f"Name: {student_name}")
print(f"Age: {student_age}")
print(f"GPA: {student_gpa}")
print(f"Enrolled: {is_enrolled}")
```

## 6.12 Creating and Using Variables

### Variables are like labeled boxes that store values

Think of variables as containers with labels. You can put different types of data in them and use them in calculations.

**Creating Variables:**

```python
# Simple variable assignment
student_name = "Alice Johnson"
student_age = 20
student_gpa = 3.75
```

**Variable Naming Rules:**

- Use descriptive names: `price`, not `p`
- Use underscores for multiple words: `first_name`, not `firstName`
- Start with letter or underscore, not numbers
- No spaces or special characters (@, #, !, etc.)
- Case-sensitive: `name` and `Name` are different
- Cannot use Python keywords (if, for, while, etc.)

**Good Variable Naming Examples:**

```python
# Good variable names (descriptive and clear)
first_name = "John"
student_count = 25
MAX_ATTEMPTS = 3  # Constants in ALL_CAPS
is_valid = True
total_price = 99.99

# Avoid these (not descriptive or confusing)
a = "John"           # Too short, not descriptive
studentcount = 25    # Hard to read without underscores
max = 3              # Could conflict with built-in function
flag = True          # What does this flag represent?
```

**Using Variables in Calculations:**

```python
# Real-world example: calculating course grade
quiz_grade = 85
exam_grade = 92
homework_grade = 88
```

```
5
6 # Calculate weighted average
7 final_grade = (quiz_grade * 0.3) + (exam_grade * 0.5) + (homework_grade
      * 0.2)
8 print(f"Your final grade is: {final_grade:.1f}")
```

**Checking Variable Types (Advanced):** If you're curious about what type Python thinks your variable is:

```
1 name = "Alice"
2 age = 25
3 height = 5.6
4 is_student = True
5
6 print(f"name is of type: {type(name)}")            # <class 'str'>
7 print(f"age is of type: {type(age)}")              # <class 'int'>
8 print(f"height is of type: {type(height)}")        # <class 'float'>
9 print(f"is_student is of type: {type(is_student)}")# <class 'bool'>
```

## 6.13 Formatting Output with F-Strings

F-strings are Python's modern way to include variables in text output. They make your programs look professional!

**Basic F-String Syntax:**

- Put `f` before the opening quote
- Put variables inside curly braces {}
- `print(f"Hello, {name}!")`

**Practical Examples:**

```
1 name = "Alice"
2 age = 20
3 university = "Quinnipiac"
4
5 # Simple f-string usage
6 print(f"My name is {name} and I am {age} years old.")
7 print(f"I attend {university} University.")
8
9 # Output:
10 # My name is Alice and I am 20 years old.
11 # I attend Quinnipiac University.
```

**Formatting Numbers with F-Strings:**

```
1 price = 29.99567
2 quantity = 3
3 total = price * quantity
4
5 # Format to 2 decimal places
6 print(f"Price per item: ${price:.2f}")
7 print(f"Quantity: {quantity}")
8 print(f"Total cost: ${total:.2f}")
9
10 # Output:
11 # Price per item: $29.99
12 # Quantity: 3
13 # Total cost: $89.99
```

**Advanced F-String Formatting:**

```
# Percentage formatting
gpa = 3.75
percentage = gpa / 4.0 * 100
print(f"GPA: {gpa:.2f} ({percentage:.1f}%)")

# Mathematical expressions inside f-strings
radius = 5
print(f"Area of circle with radius {radius}: {3.14159 * radius**2:.2f}"
    )
```

**Pro Tip:** F-strings make your output look professional and are much easier to read than older string formatting methods!

# 7 Working with IPython

## 7.1 Starting IPython

IPython provides an enhanced interactive Python experience:

```
# Start IPython from terminal
ipython

# You should see something like:
Python 3.9.7 (default, Sep 16 2021, 16:59:28)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.29.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]:
```

## 7.2 IPython Features

**Enhanced Input/Output:**

```
In [1]: 2 + 2
Out[1]: 4

In [2]: "Hello" + " " + "World"
Out[2]: 'Hello World'

In [3]: print("This is a print statement")
This is a print statement

In [4]: # Notice no Out[4] for print statements
```

**Tab Completion:** Type the beginning of a variable name, function name, or method and press Tab:

```
In [5]: import math
In [6]: math.sq<TAB>     # Will complete to math.sqrt
In [7]: math.sqrt(16)
Out[7]: 4.0
```

**Getting Help:**

```
In [8]: len?             # Get help about the len function
In [9]: math.sqrt?       # Get help about sqrt function
In [10]: ?print          # Another way to get help
```

**Magic Commands:** IPython provides special "magic" commands starting with %:

```
In [11]: %pwd           # Show current working directory
In [12]: %ls            # List files in current directory
In [13]: %time 2**100   # Time how long a command takes to run
In [14]: %history       # Show command history
In [15]: %reset         # Clear all variables
```

**Command History:**

- Use Up/Down arrow keys to navigate through previous commands
- `Ctrl+R`: Search through command history
- `%history`: Display all previous commands

### 7.3 Practical IPython Session

Let's work through a practical example using IPython:

```
In [1]: # Let's calculate the compound interest
In [2]: principal = 1000      # Initial amount
In [3]: rate = 0.05           # 5% annual interest rate
In [4]: years = 10            # Investment period

In [5]: # Calculate compound interest
In [6]: amount = principal * (1 + rate) ** years
In [7]: interest_earned = amount - principal

In [8]: print(f"Initial investment: ${principal:.2f}")
Initial investment: $1000.00

In [9]: print(f"After {years} years at {rate*100}% annual rate:")
After 10 years at 5.0% annual rate:

In [10]: print(f"Final amount: ${amount:.2f}")
Final amount: $1628.89

In [11]: print(f"Interest earned: ${interest_earned:.2f}")
Interest earned: $628.89

In [12]: # Let's see what happens with different rates
In [13]: for r in [0.03, 0.05, 0.07]:
    ...:     final = principal * (1 + r) ** years
    ...:     print(f"Rate: {r*100}% -> Final amount: ${final:.2f}")
    ...:
Rate: 3.0% -> Final amount: $1343.92
Rate: 5.0% -> Final amount: $1628.89
Rate: 7.0% -> Final amount: $1967.15
```

## 8 Google Colab: Your Python Playground

### 8.1 What is Google Colab?

Google Colaboratory (Colab) is a free, cloud-based Jupyter notebook environment that runs entirely in your web browser. It's perfect for learning Python because:

**Key Benefits:**

- **No Installation Required:** Everything runs in the cloud
- **Free Access:** No cost to use, just need a Google account
- **Pre-installed Libraries:** NumPy, Pandas, Matplotlib already available
- **Powerful Hardware:** Free access to GPU and TPU resources
- **Easy Sharing:** Share notebooks like Google Docs
- **Persistent Storage:** Save to Google Drive automatically

## 8.2 Getting Started with Google Colab

**Step 1: Access Colab**

1. Open your web browser
2. Go to colab.research.google.com
3. Sign in with your Google account
4. Click "New Notebook" to start coding

**Step 2: Understanding the Interface**

- **Code Cells:** Write and execute Python code
- **Text Cells:** Add explanations using Markdown
- **Menu Bar:** File operations, runtime controls
- **Toolbar:** Quick access to common functions

## 8.3 Why We Use Colab in This Course

**Educational Advantages:**

- **Consistency:** Everyone has the same environment
- **Accessibility:** Works on any device with a browser
- **Immediate Feedback:** Run code instantly and see results
- **Rich Output:** Display graphs, images, and formatted text
- **Collaboration:** Easy to share assignments and get help

**Professional Relevance:**

- Used by Google, Netflix, and other tech companies
- Standard tool for data science and machine learning
- Skills transfer directly to industry work
- Part of the modern Python development ecosystem

### 8.4 Colab vs Traditional Programming

| Aspect | Traditional IDE | Google Colab |
|---|---|---|
| Setup | Install Python, IDE, libraries | Just open a web browser |
| Hardware | Limited by your computer | Access to powerful cloud GPUs |
| Sharing | Email files, complex setup | Share link, instant access |
| Documentation | Separate files | Integrated text and code |
| Learning Curve | Steep | Gentle |

### 8.5 Your First Colab Experience

Let's create your first Google Colab notebook:

**Exercise: Hello Colab**

1. Visit colab.research.google.com
2. Click "New Notebook"
3. In the first cell, type: `print("Hello, Python!")`
4. Press Shift+Enter to run the cell
5. Add a text cell by clicking "+ Text"
6. Write: "This is my first Python program"
7. Save your notebook (File → Save)

**Congratulations!** You've just created your first Python program in the cloud!

## 9 Introduction to Jupyter Notebooks

### 9.1 What are Jupyter Notebooks?

Jupyter Notebooks are web-based interactive computing environments that allow you to:

- Combine code, text, equations, and visualizations in one document
- Execute code in individual cells
- Document your thought process alongside your code
- Share your work with others easily
- Create tutorials, reports, and presentations

### 9.2 Starting Jupyter

```
# Start Jupyter Notebook
jupyter notebook

# Or start JupyterLab (newer interface)
jupyter lab
```

This will:

1. Start the Jupyter server
2. Open your web browser automatically
3. Display the Jupyter file browser

## 9.3 Jupyter Cell Types

**Code Cells:**

- Contain Python code
- Execute when you press Shift+Enter
- Show output below the cell
- Variables persist between cells

**Markdown Cells:**

- Contain formatted text, equations, images
- Use Markdown syntax
- Great for explanations, notes, and documentation
- Render when you press Shift+Enter

**Example Markdown:**

```
# This is a heading
## This is a subheading

This is **bold** text and this is *italic* text.

Here's a bulleted list:
- Item 1
- Item 2
- Item 3

Here's a mathematical equation: $E = mc^2$

Here's some code: `print("Hello World")`
```

## 9.4 Jupyter Keyboard Shortcuts

- **Shift+Enter:** Run cell and move to next
- **Ctrl+Enter:** Run cell but stay in current cell
- **A:** Insert cell above (in command mode)
- **B:** Insert cell below (in command mode)
- **D, D:** Delete cell (press D twice)
- **M:** Change cell to Markdown
- **Y:** Change cell to Code
- **Esc:** Enter command mode
- **Enter:** Enter edit mode

# 10 Programming in the Modern World

## 10.1 The Internet and Programming

Modern programming is inherently connected to the internet and networked systems:

**APIs (Application Programming Interfaces):**

- Allow programs to communicate with web services
- Access data from Twitter, weather services, stock markets
- Examples: REST APIs, GraphQL APIs

**Cloud Computing:**

- Run programs on remote servers
- Scale applications automatically
- Examples: AWS, Google Cloud, Microsoft Azure

**Web Applications:**

- Programs that run in web browsers
- Accessible from anywhere with internet connection
- Examples: Gmail, Facebook, online banking

## 10.2 Big Data and Programming

The scale of data in modern applications is unprecedented:

**Data Scale Examples:**

- **Google:** Processes 20+ petabytes of data per day
- **Facebook:** Stores 300+ petabytes of user data
- **YouTube:** 500+ hours of video uploaded every minute
- **Twitter:** 500+ million tweets sent daily

**Big Data Characteristics (The 5 V's):**

- **Volume:** Massive amounts of data
- **Velocity:** Data generated rapidly
- **Variety:** Different types and formats
- **Veracity:** Data quality and accuracy
- **Value:** Extracting meaningful insights

**Python's Role in Big Data:**

- **Data Collection:** Web scraping, API access
- **Data Processing:** pandas, NumPy for manipulation
- **Data Analysis:** Statistical analysis, pattern recognition
- **Visualization:** matplotlib, plotly for charts and graphs
- **Machine Learning:** scikit-learn, TensorFlow for AI

# 11 Looking Ahead: Course Structure

## 11.1 Weekly Structure

**Monday Sessions:**

- Introduction to new concepts
- Theoretical foundations
- Demonstration of key ideas
- Interactive discussions

**Wednesday Sessions:**

- Hands-on programming practice
- Working through examples together
- Individual and group coding exercises
- Assignment help and Q&A

## 11.2 Topics We'll Cover

**Weeks 1-4: Python Fundamentals**

- Variables, data types, and operations
- Control structures (if statements, loops)
- Functions and scope
- Error handling

**Weeks 5-8: Data Structures**

- Lists, tuples, and dictionaries
- Sets and advanced data structures
- File input/output
- String processing

**Weeks 9-12: Advanced Topics**

- Object-oriented programming
- Modules and packages
- NumPy for numerical computing
- pandas for data analysis

**Weeks 13-16: Applied Programming**

- Data visualization with matplotlib
- Web APIs and data collection
- Final project development
- Professional programming practices

### 11.3 Assignment Structure

**Assignment 1:** Python Fundamentals

- Variables and basic operations
- Control flow and functions
- Problem-solving with Python

**Assignment 2:** Data Structures & File Processing

- Lists, dictionaries, and tuples
- File reading and writing
- Data processing and analysis

**Assignment 3:** NumPy & pandas Analysis

- Numerical computing with NumPy
- Data manipulation with pandas
- Statistical analysis

**Assignment 4:** Data Visualization Project

- Creating charts and graphs
- Interactive visualizations
- Presenting data insights

**Final Project (Weeks 14-16):** Complete Data Analysis Portfolio

- Choose your own dataset and research question
- Complete analysis workflow
- Professional presentation
- Portfolio-worthy deliverable

## 12 Hands-On Exercises

These exercises will help you practice everything we covered in today's three-part journey!

### 12.1 Exercise 1: Your Programming Goals Reflection

Before diving into code, let's connect with your motivation:

**Write a brief reflection (in a Jupyter Markdown cell):**

- What programming applications excite you most?
- How do you encounter programming in your daily life?
- What would you like to build by the end of this course?

## 12.2 Exercise 2: Basic Calculator Operations

Practice Python as a powerful calculator with these operations:

```
# Basic arithmetic (from slides)
print("Addition:", 15 + 25)         # 40
print("Subtraction:", 50 - 18)      # 32
print("Multiplication:", 6 * 7)     # 42
print("Division:", 84 / 4)          # 21.0
print("Exponentiation:", 2 ** 10)   # 1024

# Advanced operations
print("Floor Division:", 17 // 4)   # 4 (integer result)
print("Modulus (remainder):", 17 % 4)   # 1
print("Order of operations:", 2 + 3 * 4)   # 14 (not 20!)
print("With parentheses:", (2 + 3) * 4)    # 20

# Complex calculations
print("Complex calculation:", (5 + 3) * 2 ** 3 - 10)
print("Average of 85, 90, 78:", (85 + 90 + 78) / 3)
```

## 12.3 Exercise 3: Working with Variables and Data Types

Practice the key concepts from today's slides:

```
# Data Types Practice (from slides examples)
# Numbers
student_age = 20                        # Integer
student_gpa = 3.75                      # Float
year = 2025                             # Integer
price = 29.99                           # Float

# Text (Strings)
student_name = "Alice Johnson"          # String
university = "Quinnipiac University"    # String

# True/False (Booleans)
is_enrolled = True                      # Boolean
has_scholarship = False                 # Boolean

# Display all variables with their types
print(f"Name: {student_name} (type: {type(student_name)})")
print(f"Age: {student_age} (type: {type(student_age)})")
print(f"GPA: {student_gpa} (type: {type(student_gpa)})")
print(f"Enrolled: {is_enrolled} (type: {type(is_enrolled)})")
```

## 12.4 Exercise 4: F-String Formatting Practice

Master the professional output formatting from the slides:

```
# Shopping calculation example (from slides)
price = 29.99
quantity = 3
tax_rate = 0.08

# Calculate totals
subtotal = price * quantity
tax_amount = subtotal * tax_rate
```

```
total = subtotal + tax_amount

# Professional output with f-strings
print(f"Item price: ${price:.2f}")
print(f"Quantity: {quantity}")
print(f"Subtotal: ${subtotal:.2f}")
print(f"Tax ({tax_rate*100}%): ${tax_amount:.2f}")
print(f"Total cost: ${total:.2f}")

# Personal information example
name = "Your Name Here"
age = 20
gpa = 3.75

print(f"My name is {name} and I am {age} years old.")
print(f"My current GPA is {gpa:.2f}")
print(f"GPA percentage: {(gpa/4.0)*100:.1f}%")
```

### 12.5 Exercise 5: Using the Math Library

Explore Python's mathematical superpowers (from slides preview):

```
import math

# Basic math library functions (from slides)
print(f"Square root of 16: {math.sqrt(16)}")      # 4.0
print(f"Value of pi: {math.pi}")                  # 3.14159...
print(f"Sine of 90 degrees: {math.sin(math.pi/2):.3f}")  # 1.0

# Circle calculations (from slides example)
radius = 7.5
area = math.pi * radius ** 2
print(f"Circle area with radius {radius}: {area:.2f}")

# Additional mathematical functions
print(f"\nAdditional Math Functions:")
print(f"Square root of 64: {math.sqrt(64)}")
print(f"2 to the power of 10: {math.pow(2, 10)}")
print(f"Natural log of e: {math.log(math.e):.3f}")

# Trigonometry example
angle_degrees = 30
angle_radians = math.radians(angle_degrees)
print(f"\nTrigonometry for {angle_degrees} degrees:")
print(f"  sin({angle_degrees} degrees) = {math.sin(angle_radians):.3f}"
    )
print(f"  cos({angle_degrees} degrees) = {math.cos(angle_radians):.3f}"
    )
```

### 12.6 Exercise 6: Create Your First Google Colab Notebook

Put it all together in a professional notebook using Google Colab (as emphasized in slides):

1. Visit colab.research.google.com and sign in
2. Create a new notebook and rename it to "Lecture1_YourName"

3. Add a Markdown cell with:
    - Course title: INF 605 - Introduction to Programming - Python
    - Lecture 1: Introduction to Computers and Python
    - Your name and date
    - Your programming goals from Exercise 1
4. Add code cells for Exercises 2-5 above
5. Add Markdown cells explaining what each code section does
6. Practice the three-part structure from today:
    - Part I: Your programming goals
    - Part II: Basic calculations and operations
    - Part III: Variables, data types, and f-strings
7. Save your notebook - this is the beginning of your Python portfolio!

**Pro Tip:** This Google Colab notebook demonstrates the power of cloud-based learning - you're documenting your journey while you code, accessible from anywhere!

## 13 Key Takeaways and Summary

**What We Covered Today - Our Three-Part Journey:**

**Part I: Getting to Know Each Other**

- ✓ Shared programming backgrounds and goals
- ✓ Explored programming in daily life
- ✓ Connected with course objectives and philosophy
- ✓ Learned fun Python facts and discovered its widespread use

**Part II: Programming Foundations**

- ✓ Understood programming as problem-solving and communication
- ✓ Learned computer fundamentals: hardware and software relationship
- ✓ Explored programming language evolution from machine code to Python
- ✓ Discovered why Python is perfect for beginners

**Part III: Python Programming**

- ✓ Set up Google Colab for cloud-based Python programming
- ✓ Wrote first Python programs with proper syntax
- ✓ Mastered Python data types (int, float, str, bool)
- ✓ Created and used variables with descriptive names
- ✓ Performed mathematical calculations with order of operations
- ✓ Formatted professional output using f-strings

- ✓ Preview of Python's power through libraries

**Practical Skills You Can Now Use:**

- Write Python programs with variables and data types
- Use Python as a powerful calculator for real-world problems
- Format output professionally using f-strings
- Apply proper variable naming conventions
- Use Google Colab and Jupyter Notebooks for interactive programming
- Import and use Python libraries for advanced functionality

**Looking Forward:** Today's three-part journey provides the foundation for everything we'll build in this course. You now understand:

- Programming's role in every aspect of modern life
- Why Python is the perfect first programming language
- How to think like a programmer: breaking problems into steps
- The basics of writing professional Python code
- How development tools like Google Colab make learning interactive and fun

## Preparation for Next Lecture

**For Next Class:**

**Required Practice:**

1. Complete all exercises from this handout (Exercises 1-6)
2. Create your first Google Colab notebook following Exercise 6
3. Try creating variables with different data types from the slides
4. Practice f-string formatting with your own examples
5. Read Deitel Chapter 2 (Variables and Simple Data Types)

**Technical Setup:**

- Ensure you can access Google Colab (colab.research.google.com)
- Create a Google account if you don't have one
- Bring your laptop to every class session
- If you encounter technical issues, attend office hours for help

**Next Lecture Preview:** We'll build on today's foundation with more advanced Python programming:

- Getting input from users with the input() function
- Making decisions with if statements and conditions

- Handling different types of data and conversions
- Building interactive programs that respond to user input
- More advanced string manipulation and formatting

**Congratulations!**

*You've taken your first steps into the world of programming. Every expert was once a beginner - the key is to keep practicing and never stop learning!*

**See you Wednesday for more Python adventures!**

Remember: Programming is learned by doing, not just by reading. The more you practice, the more comfortable you'll become with thinking like a programmer!