# Python Development Environment & Problem-Solving Foundations

**Tylar Campbell, PhD. Candidate (him / his)**
**School of Interactive Arts & Technology (SIAT)**
**Vanier Scholar (2022) | Making Culture Lab**

Cohort: AISE 2026

Week: 1

Day: 1

Session Code: W1D1

Duration: 3 hours

Format: Virtual (Zoom)

Type: Technical Foundations

Phase: Python Foundations (Phase 1)

## Learning Outcomes & Career Relevance

1 Professional Environment Setup

Configure VS Code and Python like industry professionals

2 Problem Decomposition

Break complex challenges into manageable, solvable pieces

3 Function Design

Create reusable, documented code that follows best practices

4 Data Type Mastery

Apply strings, numbers, and booleans to real-world scenarios

5 Portfolio Development

Build your first professional Python toolkit

6 Industry Standards

Write code that meets professional quality expectations

COLUMBIA UNIVERSITY
Justice Through Code

## The Power of Automation

Let's see what's possible when you harness Python's automation capabilities. Instead of manually typing repetitive tasks hundreds of times, you can create smart, reusable solutions.

### Manual Approach

Typing the same code 365 times for daily tasks - time-consuming and error-prone

### Python Solution

One function call replaces hundreds of lines - automated and intelligent

### Real Impact

Transform hours of work into seconds of execution with professional code

## Live Demo: Professional Development Environment

01 System Check for Python

02 Install VS Code

03 Virtual Environment Setup

```
# Windows users - open Command Prompt:
 # Press Windows Key + R, type 'cmd', press Enter
 python --version

 # Mac/Linux users - open Terminal:
 # Press Cmd + Space, type 'terminal', press Enter
 python3 --version
```

## Professional Decomposition Standards

### Single Responsibility

One function equals one clear purpose. Each function should do one thing exceptionally well.

### Descriptive Naming

Function and variable names should tell the story. Code should read like well-written prose.

### Comprehensive Documentation

Every function needs clear docstrings explaining purpose, parameters, and return values.

### Error Handling

Anticipate edge cases and provide graceful failure modes with helpful error messages.

### Modular Design

Functions should work independently and be easily combined into larger systems.

## Problem Decomposition Principles

Every complex system started as simple pieces. Problem decomposition is like planning a wedding—you don't tackle everything at once. You break it into manageable components: venue, catering, invitations, music. Each piece becomes solvable.

### Complex Problem

Overwhelming challenge that seems impossible to solve

### Break Into Pieces

Identify smaller, manageable components and relationships

### Solve Each Part

Create functions that handle individual responsibilities

### Connect Solutions

Combine functions into a complete, working system

COLUMBIA UNIVERSITY
Justice Through Code

## Breakout Session 1: Personal Finance Calculator

Time to practice breaking down a real problem into code! You'll work in groups of 3-4 people to build a Personal Finance Calculator that demonstrates professional problem decomposition.

01

### Discuss Components (5 min)

Identify budget categories, calculations needed, and user inputs as a group

02

### Implement Functions (10 min)

Each person codes individual functions with clear purposes and documentation

03

### Share & Debug (5 min)

Present solutions, identify different approaches, and help troubleshoot issues

**Remember:** One function equals one clear purpose. Use descriptive names that tell the story. TAs will rotate through rooms to provide guidance.

## BreakOut Debrief

```
"""
1. One function = One clear purpose
2. Descriptive names tell the story
3. Functions should be testable independently
4. Parameters make functions flexible
5. Return values enable function chaining
"""


# Example of good decomposition
def calculate_tax(income, tax_rate=0.2):
    """Single purpose: calculate tax"""
    return income * tax_rate

def calculate_net_income(gross_income, tax_rate=0.2):
    """Builds on other functions"""
    tax = calculate_tax(gross_income, tax_rate)

    return gross_income - tax
```

# Data Types in Professional Context

Let's connect each data type to real-world applications. This isn't just syntax—this is how professionals use these tools to solve business problems and create value.

## Strings: Text Processing

Generate email addresses, format user names, process customer data, and create dynamic content for business applications.

## Numbers: Financial Calculations

Handle money with floats, count items with integers, calculate payroll, track inventory, and process financial transactions.

## Booleans: Business Logic

Control program flow, validate user permissions, check eligibility criteria, and implement conditional business rules.

```python
# Strings: Text processing
user_name = "Alice Chen"
# Numbers: Calculations and analysis
hourly_rate = 25.50  # float for money , hours_worked = 40  # int for counting
# Booleans: Decision making
is_overtime = hours_worked > 40
```

## Live Demo : Fundamentals Deep Dive

## Breakout 2 Instructions

For the next 20 minutes, you will work on building out your personal Python toolkit. The goal is to solve real problems, not just complete exercises.

Choose problems that excite you and focus on creating meaningful solutions. TAs will be available to provide guidance and answer questions.

Core Level (Beginner)

- Create 3 working functions that solve real problems.
- Ensure basic functionality and clear purpose for each.

Stretch Level (Intermediate)

- Add comprehensive error handling to your functions.
- Anticipate edge cases and provide graceful failure modes.

Advanced Level (Expert)

- Create interdependent functions that work together as a system.
- Design for modularity and scalability.

## Peer Showcase

It's time to celebrate your creativity and hard work! We're looking for **two volunteers** to demonstrate their personal Python toolkits. This is your moment to shine and show off the amazing problems you've solved.

## Peer Feedback Protocol

As your peers present, please provide constructive feedback in the chat. Focus on what you learned, what impressed you, and any suggestions for improvement. Remember to be supportive and encouraging!

## Code Review Focus Areas

- Clear function naming conventions
- Comprehensive documentation practices
- Logical parameter design
- Creative problem-solving approaches

Your toolkits aren't just code; they're demonstrations of your problem-solving skills and your ability to apply professional patterns that get developers hired. Let's celebrate your creativity and the value you've created!

Columbia University
Justice Through Code

# Building Your Portfolio

1. **Today**
   Personal toolkit with 5+ functions, documentation, and error handling
2. **Week 2**
   Add data processing and GitHub version control to your toolkit
3. **Month 1**
   Complete portfolio with multiple projects showcasing different skills
4. **Month 3**
   Industry-ready portfolio that opens career opportunities

> **Remember:** Every expert was once a beginner. Master the five professional practices: consistent naming, comprehensive documentation, error handling, modular design, and version control.

## Portfolio Evolution Path

1

### Today

Personal toolkit with 5+ functions, documentation, and error handling

2

### Week 2

Add data processing and GitHub version control to toolkit

3

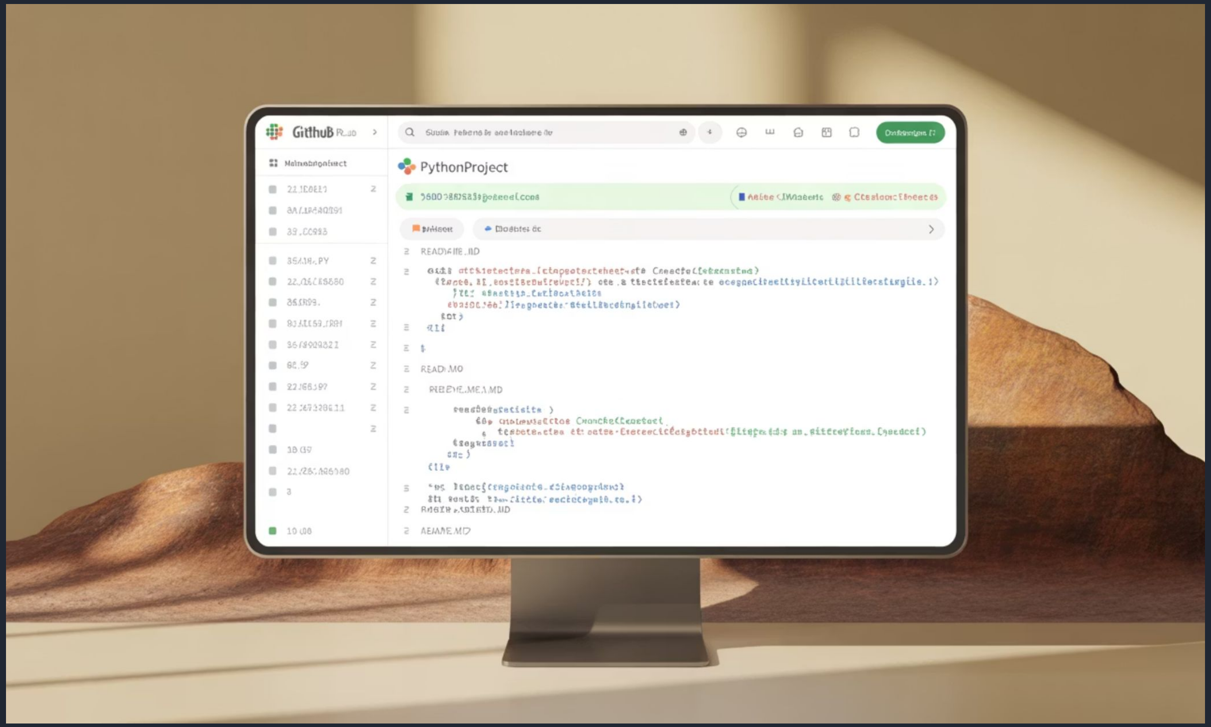
### Month 1

Complete portfolio with multiple projects showcasing different skills

4

### Month 3

Industry-ready portfolio that opens career opportunities



## Industry Best Practices: Code That Gets You Hired

### Consistent Naming Conventions

Ensure your variables, functions, and classes follow clear and predictable style e.g., snake_case for variables, PascalCase for classes this enhances readability and maintainability crucial for collaborative projects.

Checklist:

- All variables use snake_case.
- All functions use snake_case.
- All classes use PascalCase.
- Names are descriptive and avoid abbreviations

### Comprehensive Documentation

Well-documented code is easier to understand and use. Include docstrings for functions and classes, and inline comments for complex logic. Type hints improve code clarity and enable static analysis.

Checklist:

- Each function has a docstring explaining its purpose, arguments, return value.
- Complex logic is explained inline comments.
- Type hints are used for functions parameters and return values.
- README file (if applicable) clearly explains project setup and usage.

### Error Handling and Edge Cases

Robust code anticipates potential issues and handles them gracefully. Implement try- except blocks validate inputs and provide informative error messages to prevent unexpected crashes and improve user experience.

Checklist:

- Input data is validated to prevent errors.
- try-except blocks are used for operations that might fail (e.g., files I/O, network requests).
- Specific exceptions are caught, not just generic Exception.
- Informative error messages are provided to the user or logged.

COLUMBIA UNIVERSITY
Justice Through Code

## Industry Best Practices: Code That Gets You Hired

### Modular Design Principles

Breakdown complex problems into smaller, manageable functions and classes. This promotes reusability makes testing easier and allows for better organization of your code base.

Checklist:

- Code is organized into functions or classes each with a single responsibility.
- Functions are small in focus.
- Code avoids global variables where possible
- Dependencies between modules are minimized

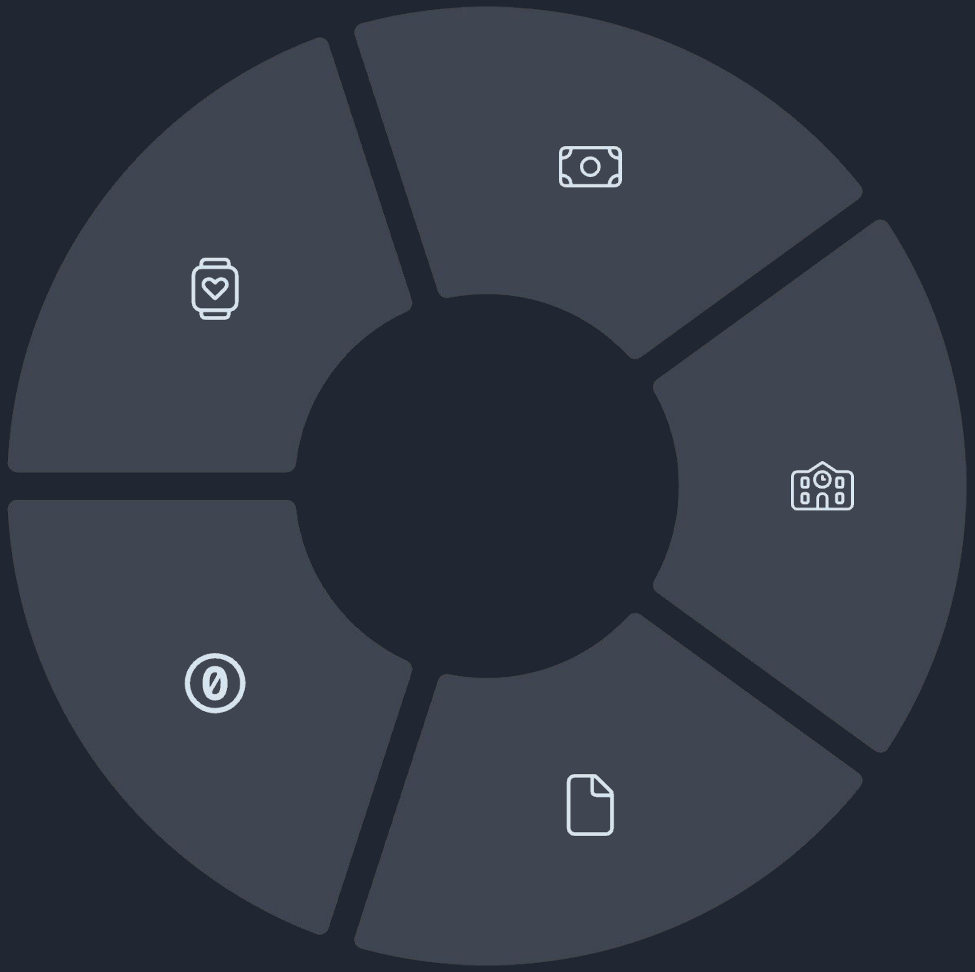
### Version Control Principles

Mastering tools like git and Github is non-negotiable for professional development. It enables tracking changes collaborating with others, and easily reverting to previous versions of your code.

Checklist:

- Project is initialized as  a Git repository..
- Regular, atomic commit with descriptive message are made.
- Code is pushed to a remote repository (e.g., GitHub).
- Branches are used for new features or bug fixes.

## Building Your Professional Toolkit



### Health & Fitness

BMI calculators, workout trackers, calorie counters, or meal planners

### Financial Tools

Budget calculators, tip computers, loan analyzers, or savings trackers

### Academic Management

GPA calculators, study schedulers, assignment trackers, or grade analyzers

### Productivity

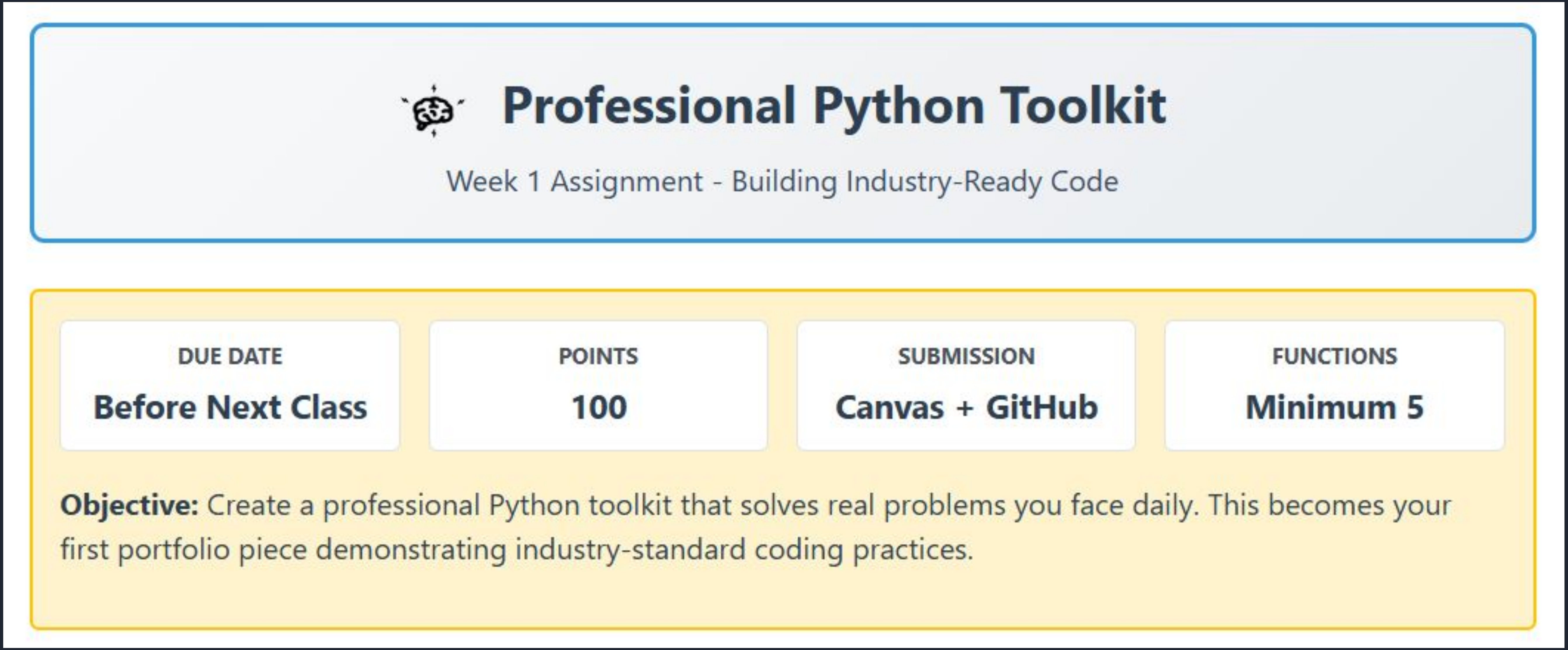
Time converters, unit converters, password generators, or task organizers

### Creative Projects

Text generators, color palette tools, or custom calculators for hobbies

COLUMBIA UNIVERSITY
Justice Through Code

## Assignment & Next Steps



### Professional Python Toolkit

Week 1 Assignment - Building Industry-Ready Code

| DUE DATE | POINTS | SUBMISSION | FUNCTIONS |
|---|---|---|---|
| Before Next Class | 100 | Canvas + GitHub | Minimum 5 |

**Objective:** Create a professional Python toolkit that solves real problems you face daily. This becomes your first portfolio piece demonstrating industry-standard coding practices.