Lecture slides - Week 3

Programming in Python - A Problem Solving Approach

Dr. Aamir Akbar

Director of both AWKUM AI Lab and AWKUM Robotics, Final Year Projects (FYPs) coordinator, and lecturer at the department of Computer Science
Abdul Wali Khan University, Mardan (AWKUM)

Contents

1. Foundations of Effective Programming
2. Exploring Python Program Execution
3. Problem Solving Approach with Examples
4. Download Source Codes

Foundations of Effective Programming

What is a problem solving approach in programming?

A problem-solving approach in programming is a systematic way of tackling complex problems using logical thinking and a structured methodology. It involves breaking down a problem into smaller, manageable parts and devising a step-by-step plan to arrive at a solution.

Key components of a problem-solving approach in programming include:

- Understanding the problem statement.
- Identifying the input and output requirements.
- Decomposing the problem into smaller subproblems.
- Developing an algorithm or pseudocode.
- Writing and testing code.
- Debugging and refining the solution.

Logical mind

Successful programming often requires a logical mindset. This means being able to think critically, reason through problems, and follow a systematic approach to arrive at solutions. Here are some tips for developing a logical mind in programming:

- Practice breaking down problems into smaller parts.
- Use flowcharts or diagrams to visualize solutions.
- Develop a clear understanding of data structures and algorithms.
- Learn from your mistakes and debugging experiences.
- Collaborate with others to solve complex problems.

Exploring Python Program Execution

Execution Flow of Python Program (Part 1/3)

When you run a Python program, it goes through a series of steps that make up its execution flow. Understanding this flow is important to write efficient and bug-free code.

1. **Source Code**: The program starts with the source code written by the developer. This code can be stored in a `.py` file.
2. **Compilation (for .py files)**: Python is primarily an interpreted language, and its interpreter can directly execute the source code without the need for a separate compilation step. When you run a Python code (`.py file`), the interpreter reads the source code and executes it line by line. However, python also offers an option for compilation to bytecode as a form of optimization. When Python compiles source code into bytecode, `it can potentially execute slightly faster` because the interpreter doesn't need to parse the source code each time it runs. Therefore, `.py` files are first compiles into bytecode, which is a lower-level representation of the code and is stored in `.pyc` files.

Execution Flow of Python Program (Part 2/3)

3. **Interpreter**: Python is an interpreted language, and its interpreter reads the bytecode or source code and executes it sequentially.
4. **Execution**: During execution, Python runs through statements, functions, and modules in the code. It loads libraries and dependencies as needed.
5. **Memory Allocation**: Python allocates memory for variables, data structures, and objects. Understanding memory management is vital for preventing memory leaks and optimizing program performance.
6. **Control Flow**: Program control flows through conditional statements (if-else), loops (for and while), and function calls. It's important to grasp how control flow affects the program's behavior.
7. **Error Handling**: Python handles exceptions (errors) gracefully with try-except blocks. Effective error handling ensures robust programs.

Execution Flow of Python Program (Part 3/3)

8. **Termination**: The program terminates when it reaches the end of the code or encounters a specific exit command. Proper termination is essential for resource cleanup.
9. **Cleanup**: After program execution, Python releases allocated resources, such as memory and file handles.

Python Bytecode and REPL

Bytecode Files (.pyc): When Python compiles source code to bytecode, it caches the bytecode in `.pyc` files. These files are stored in the `__pycache__` directory and are used for subsequent runs of the program to improve performance. Python checks the timestamps of the `.py` file and the corresponding `.pyc` file to determine if recompilation is needed.

The choice of whether Python programs are compiled to bytecode or not depends on several factors, including the need for optimization, the execution environment, and whether the code is being run interactively or from a script. In most cases, the compilation step is automatic and not something a developer needs to manage explicitly.

Interactive Mode: In interactive mode (using the Python REPL - Read-Eval-Print Loop), there is no compilation to `.pyc` files. The interpreter directly evaluates and executes commands entered by the user.

Problem Solving Approach with Examples

Problem-Solving Approach: Candy Sharing - Problem Statement

Problem Statement: Alice, Bob, and Carol want to share their Halloween candies equally. Any remaining candies will be smashed. For example, if they collectively bring home 91 candies, they'll take 30 each and smash 1.

Coding Example:

Problem-Solving Approach: Candy Sharing - Problem Statement

Problem Statement: Alice, Bob, and Carol want to share their Halloween candies equally. Any remaining candies will be smashed. For example, if they collectively bring home 91 candies, they'll take 30 each and smash 1.

Coding Example:

```python
# Variables representing the number of candies collected
alice_candies = 13
bob_candies = 17
carol_candies = 10

# Calculate the total number of candies
total_candies = alice_candies + bob_candies + carol_candies

# Determine how many candies need to be smashed
to_smash = total_candies % 3

# Print the number of candies to be smashed
print(to_smash)
```

Problem-Solving Approach: Candy Sharing - Solution

Solution Approach:

1. We start by defining variables for the number of candies collected by Alice, Bob, and Carol.
2. To calculate the total number of candies collected, we add these values together.
3. To determine how many candies need to be smashed, we use the modulo operator (%) to find the remainder when dividing the total by 3.
4. The result (`to_smash`) represents the number of candies that must be smashed to ensure an equal distribution.

This coding example demonstrates a problem-solving approach in Python to handle a scenario involving arithmetic operations and decision-making based on the remainder.

Problem-Solving Approach: Circle Area Calculation - Problem Statement

Problem Statement: You have a circle with a given diameter, and you need to calculate its area. The formula for the area of a circle is $\pi \times \text{radius}^2$, where π is approximately 3.14159.

Coding Example:

Problem-Solving Approach: Circle Area Calculation - Problem Statement

Problem Statement: You have a circle with a given diameter, and you need to calculate its area. The formula for the area of a circle is $\pi \times \text{radius}^2$, where π is approximately 3.14159.

Coding Example:

```python
# Given diameter
diameter = 10

# Create a variable called 'radius' equal to half the diameter
radius = diameter / 2

# Create a variable called 'area', using the formula
area = pi * (radius**2)

# Print the area of the circle
print(area)
```

Problem-Solving Approach: Circle Area Calculation - Solution

Solution Approach:

1. We start with the given diameter and need to calculate the area of the circle.
2. To calculate the area, we first create a variable called 'radius,' which is equal to half of the given diameter.
3. Then, we use the formula for the area of a circle, which involves squaring the radius and multiplying it by the value of π.
4. Finally, we print the calculated area of the circle.

This coding example demonstrates a problem-solving approach in Python to calculate the area of a circle based on its diameter and the mathematical formula for circle area.

Problem-Solving Approach: Variable Swapping - Problem Statement

Problem Statement: Imagine you have two variables, `a` and `b`, each storing a different value. Your task is to swap the values of these two variables without losing any data. This is often necessary in programming when you need to rearrange values or perform certain operations.

Coding Example:

Problem-Solving Approach: Variable Swapping - Problem Statement

Problem Statement: Imagine you have two variables, a and b, each storing a different value. Your task is to swap the values of these two variables without losing any data. This is often necessary in programming when you need to rearrange values or perform certain operations.

Coding Example:

```python
# Given values for a and b
a = 5
b = 8

# Print the initial values
print("a = " + str(a))
print("b = " + str(b))

# Your code to swap a and b goes here!
```

Problem-Solving Approach: Variable Swapping - Solution

Solution Approach:

1. We start with two variables, `a` and `b` each containing different values.
2. To swap these values, we have two methods:
   Method 1: Using Tuple Unpacking
   ```
                a, b = b, a
   ```
   This one-liner swaps the values of `a` and `b` without the need for a temporary variable.
   Method 2: Using a Temporary Variable
   ```
                temp = b
                b = a
                a = temp
   ```
   This method involves using a temporary variable `temp` to hold one of the values while swapping the others.
3. After applying one of these methods, we print the updated values of `a` and `b`.

Problem-Solving Approach: String Input

Problem Statement: Write a Python program that asks users to enter 2 numbers, the program will output True/False indicating whether the first number is greater than the second number.

Coding Example:

Problem-Solving Approach: String Input

Problem Statement: Write a Python program that asks users to enter 2 numbers, the program will output True/False indicating whether the first number is greater than the second number.

Coding Example:

```python
        # Get two numbers from input
        num_1 = int(input("Enter first number: "))
        num_2 = int(input("Enter second number: "))

        # Print whether num_1 is greater than num_2
        print(num_1 > num_2)
```

Problem-Solving Approach: Temperature Conversion

Problem Statement: Write a Python program that asks users to enter temperature in Celsius. The program will convert and output in Fahrenheit. The formula to convert temperature to Fahrenheit is:
$F = \left(C \times \frac{9}{5}\right) + 32$

Coding Example:

Problem-Solving Approach: Temperature Conversion

Problem Statement: Write a Python program that asks users to enter temperature in Celsius. The program will convert and output in Fahrenheit. The formula to convert temperature to Fahrenheit is: $F = \left(C \times \frac{9}{5}\right) + 32$

Coding Example:

```python
        # Get temperature in Celsius
        temp_c = float(input("Enter temperature in Celsius: "))

        # Convert temperature to Fahrenheit
        temp_f = (temp_c * 9/5) + 32

        # Print temperature in Fahrenheit
        print(f"Temperature in Fahrenheit is {temp_f}")
```