Chapter 1: Introduction to Advanced Python

The Zen of Python Reflection:

Based on the text in Section 1.2:

1. **Explicit is better than implicit:** The book highlights that in advanced "magic" projects, clean and straightforward code is preferred over or hidden behaviors. It emphasizes designing APIs and architectures that are easy to understand to aid maintenance.
2. **Simple is better than complex:** The text notes that adhering to simplicity ensures software remains understandable and extensible, favoring clarity over cleverness.
3. **Readability counts:** The book states that "code is read much more often than it is written," making readability a priority for collaborative and long-term projects.

Comparing Python Implementations (Section 1.4):

- **PyPy:** Described as an alternative implementation focused on speed. It uses a **Just-In-Time (JIT)** compiler to compile frequently executed code into machine code at runtime, making it advantageous for long-running, computationally intensive applications.
- **Jython:** An implementation that runs on the **Java Virtual Machine (JVM)**. It compiles Python code to Java bytecode, allowing seamless integration with Java libraries and frameworks. It is best used when you need to integrate Python into an existing Java ecosystem.

AST Exploration (Section 1.7):

- **Binary Operations:** In the AST (Abstract Syntax Tree), binary operations like multiplication or subtraction are represented by **BinOp** nodes. These nodes have a left operand, an op (operator), and a right operand.

Mutability and Object Identity (Section 1.4 Example):

The book uses the example of appending to a list to show that **mutable objects** (like lists) keep the same memory address (identity) even when their content changes. This confirms that variables in Python are references to objects in memory.

```
import dis
import ast
import re
import csv
import json
import sqlite3
import time
import functools
from abc import ABC, abstractmethod
from io import StringIO
import random

# ========================================
# Chapter 1: Introduction to Advanced Python
# ========================================
print("--- Chapter 1 Solutions ---")

# 1. Bytecode Inspection
def square(x):
    return x * x

print(f"Disassembly of square:")
dis.dis(square)

def multiply(a, b):
    return a * b

print(f"\nDisassembly of multiply:")
dis.dis(multiply)
```

```
# 2. Dynamic Typing
data = 10
print(f"\ndata: {data}, type: {type(data)}")
data = [1, 2, 3]
print(f"data: {data}, type: {type(data)}")
def my_func(): pass
data = my_func
print(f"data: {data}, type: {type(data)}")

# 3. AST Exploration
code_snippet = "y = (4 * 5) - 3"
tree = ast.parse(code_snippet)
print("\nAST Dump:")
print(ast.dump(tree, indent=4))

# 4. Mutability and Object Identity
my_list = [10, 20, 30]
print(f"\nOriginal List ID: {id(my_list)}")
my_list.append(40)
print(f"Appended List ID: {id(my_list)}")
```