Python PCEP™ Study Guide: Master the Fundamentals

This document serves as a comprehensive study reference for the **PCEP™ – Certified Entry-Level Python Programmer** certification exam. It focuses on the core Python concepts and practical skills essential for mastering Python fundamentals and excelling in the PCEP™ exam. Whether you're new to Python or refreshing your skills, this guide is designed to help you navigate the most critical topics.

Why This Guide?

The PCEP™ certification tests your knowledge of Python's basics, and this guide provides a concise, practical reference for key concepts. The exam emphasizes understanding and applying Python's core features, including:

- **Data Structures**: Mastering lists, slicing, tuples, and dictionaries.
- **Input and Output**: Handling user input and understanding `print()` methods.
- **Control Flow**: Implementing loops (`for` and `while`) and conditionals effectively.
- **Operator Precedence**: Evaluating expressions by understanding the order of operations.
- **Functions**: Calling functions with correct positional and keyword arguments.
- **Error-Free Syntax**: Writing clean, syntactically correct Python code.

By focusing on these areas, you'll strengthen your Python fundamentals and prepare to tackle tricky certification questions with confidence.

What You'll Learn

This guide is divided into sections with quick-reference tables, examples, and explanations of key topics:

1. **Lists and Slicing**:
    - Creating and manipulating lists.
    - Using slicing techniques for efficient data access.
2. **Tuples and Dictionaries**:
    - Understanding immutable tuples and mutable dictionaries.
    - Using dictionary methods for efficient lookups and updates.
3. **Input and Output**:
    - Taking user input and working with formatted strings.
    - Using advanced `print()` parameters like `sep` and `end`.
4. **Operator Precedence**:
    - Navigating the rules of expression evaluation.
    - Applying precedence and associativity for complex operations.
5. **Loops and Iteration**:

- Writing clean `for` and `while` loops.
- Breaking, continuing, and optimizing loops.

6. **Function Calling**:
   - Understanding positional and keyword arguments.
   - Mastering default arguments and unpacking.

How to Use This Guide

- **For Quick Reference**: Use the tables to look up methods, operators, and syntax at a glance.
- **For Practice**: Follow the examples and tweak them to test your understanding.
- **For Certification Prep**: Focus on tricky topics like operator precedence, function calling, and looping logic.

With this guide in hand, you'll be well-prepared to tackle the PCEP™ exam and build a solid foundation in Python programming.

Happy studying, and good luck on your certification journey! 🚀 🥇

Table of Python Built-in Data Structures

Data Structure	Description	Access Methods	Example
`list`	Mutable ordered collection.	`append()`, `pop()`, `insert()`, `remove()`.	`my_list = [1, 2, 3]` `my_list.append(4)` → `[1, 2, 3, 4]`
`tuple`	Immutable ordered collection.	`index()`, `count()`.	`my_tuple = (1, 2, 3)` `my_tuple[1]` → `2`
`set`	Mutable unordered collection.	`add()`, `remove()`, `discard()`.	`my_set = {1, 2, 3}` `my_set.add(4)` → `{1, 2, 3, 4}`
`frozenset`	Immutable version of `set`.	`union()`, `intersection()`, `difference()`.	`my_frozenset = frozenset([1, 2, 3])` `my_frozenset.union({4})` → `{1, 2, 3, 4}`
`dict`	Mutable mapping of key-value pairs.	`get()`, `keys()`, `values()`, `items()`.	`my_dict = {'a': 1, 'b': 2}` `my_dict['a']` → `1`
`str`	Immutable sequence of characters.	`lower()`, `upper()`, `find()`, `replace()`.	`my_str = "hello"` `my_str.upper()` → `"HELLO"`
`range`	Immutable range of integers.	`list(range())`, slicing.	`r = range(1, 5)` `list(r)` → `[1, 2, 3, 4]`

Data Structure	Description	Access Methods	Example
`bytes`	Immutable sequence of bytes.	`decode()`, slicing.	`b = b"hello"` `b.decode()` → `"hello"`
`bytearray`	Mutable sequence of bytes.	`append()`, slicing, `decode()`.	`ba = bytearray(b"hello")` `ba.append(33)` → `b"hello!"`
`bool`	Boolean values (`True`, `False`).	Logical operations: `and`, `or`, `not`.	`x = True` `x and False` → `False`
`int`	Immutable integer.	Arithmetic: `+`, `-`, `*`, `//`, `**`.	`x = 10` `x ** 2` → `100`
`float`	Immutable floating-point number.	Arithmetic: `+`, `-`, `*`, `/`.	`x = 3.14` `x * 2` → `6.28`
`complex`	Immutable complex number.	`.real`, `.imag`, `+`, `*`.	`c = 1+2j` `c.real` → `1.0`
`NoneType`	Represents the absence of value.	Used in comparisons, default values.	`x = None` `x is None` → `True`

Operator Precedence Table

Operator	Description	Precedence (High to Low)	Associativity	Example	Result
`()`	Parentheses	1 (Highest)	Left-to-right	`(1 + 2) * 3`	`9`
`**`	Exponentiation	2	Right-to-left	`2 ** 3 ** 2`	`512`
`+x, -x, ~x`	Unary operators	3	Right-to-left	`-3 + 2`	`-1`
`*, /, //, %`	Multiplication, Division, Modulus	4	Left-to-right	`5 * 2 // 3`	`3`
`+, -`	Addition, Subtraction	5	Left-to-right	`5 - 3 + 2`	`4`
`<<, >>`	Bitwise shifts	6	Left-to-right	`1 << 2`	`4`
`&`	Bitwise AND	7	Left-to-right	`5 & 3`	`1`
`^`	Bitwise XOR	8	Left-to-right	`5 ^ 3`	`6`
`|`	Bitwise OR	9	Left-to-right	`5 | 3`	`7`
`in`, `not in`, `is`, `is not`	Membership and identity operators	10	Left-to-right	`'a' in 'abc'`	`True`

Operator	Description	Precedence (High to Low)	Associativity	Example	Result
`<, <=, >, >=, ==, !=`	Comparison operators	11	Left-to-right	`5 > 3 == True`	`True`
`not`	Logical NOT	12	Right-to-left	`not False`	`True`
`and`	Logical AND	13	Left-to-right	`True and False`	`False`
`or`	Logical OR	14	Left-to-right	`True or False`	`True`
`=, +=, -=, *=, /=, //=, ...`	Assignment operators	15 (Lowest)	Right-to-left	`x = 5; x += 3`	`x = 8`

String Methods Reference

Method	Description	Example	Result
`str.upper()`	Converts all characters to uppercase.	`"hello".upper()`	`"HELLO"`
`str.lower()`	Converts all characters to lowercase.	`"HELLO".lower()`	`"hello"`
`str.strip()`	Removes leading and trailing spaces.	`" hello ".strip()`	`"hello"`
`str.split()`	Splits string into a list of words.	`"a,b,c".split(",")`	`['a', 'b', 'c']`
`str.join()`	Joins a list of strings into a single string.	`".".join(['a', 'b', 'c'])`	`"a.b.c"`
`str.replace()`	Replaces occurrences of a substring.	`"hello".replace("l", "x")`	`"hexxo"`
`str.find()`	Returns the index of the first occurrence.	`"hello".find("l")`	`2`
`str.startswith()`	Checks if the string starts with a prefix.	`"hello".startswith("he")`	`True`
`str.endswith()`	Checks if the string ends with a suffix.	`"hello".endswith("lo")`	`True`

Dictionary Methods Reference

Method	Description	Example	Result

Method	Description	Example	Result
`dict.get(key)`	Returns the value for `key` or `None`.	`my_dict.get('a')`	`1`
`dict.keys()`	Returns a view object of dictionary keys.	`my_dict.keys()`	`dict_keys(['a', 'b'])`
`dict.values()`	Returns a view object of dictionary values.	`my_dict.values()`	`dict_values([1, 2])`
`dict.items()`	Returns a view object of key-value pairs.	`my_dict.items()`	`dict_items([('a', 1)])`
`dict.pop(key)`	Removes the key and returns its value.	`my_dict.pop('a')`	`1`
`dict.update()`	Updates the dictionary with key-value pairs.	`my_dict.update({'c': 3})`	`{'a': 1, 'b': 2, 'c': 3}`

List Methods Reference

Method	Description	Example	Result
`list.append(x)`	Adds `x` to the end of the list.	`my_list.append(6)`	`[1, 2, 3, 4, 5, 6]`
`list.insert(i, x)`	Inserts `x` at index `i`.	`my_list.insert(2, 10)`	`[1, 2, 10, 3, 4, 5]`
`list.pop()`	Removes and returns the last item.	`my_list.pop()`	`[1, 2, 3, 4]` (returns `5`)
`list.remove(x)`	Removes the first occurrence of `x`.	`my_list.remove(3)`	`[1, 2, 4, 5]`
`list.index(x)`	Returns the index of `x`.	`my_list.index(4)`	`2`
`list.sort()`	Sorts the list in ascending order.	`my_list.sort()`	`[1, 2, 3, 4, 5]`
`list.reverse()`	Reverses the elements of the list.	`my_list.reverse()`	`[5, 4, 3, 2, 1]`

List Slicing Examples

Here are common examples of list slicing operations in Python:

```
# Basic Slicing
my_list = [1, 2, 3, 4, 5]
print(my_list[1:4])  # [2, 3, 4] (elements from index 1 to 3, excluding 4)

# Slicing with Steps
print(my_list[::2])  # [1, 3, 5] (every second element)
```

```
# Negative Indexing
print(my_list[-3:])  # [3, 4, 5] (last 3 elements)

# Reverse a List
print(my_list[::-1])  # [5, 4, 3, 2, 1] (reverse order)

# Empty Slicing
print(my_list[:])  # [1, 2, 3, 4, 5] (returns the entire list)

# Out-of-Bounds Slicing
print(my_list[2:10])  # [3, 4, 5] (stops at the end of the list if index exceeds
length)
```

Escape Sequences in Strings

Escape Sequence	Description	Example	Result
`\n`	Newline	`"hello\nworld"`	`hello` `world`
`\t`	Tab	`"hello\tworld"`	`hello world`
`\'`	Single quote	`"It\'s fine"`	`It's fine`
`\"`	Double quote	`"He said \"hi\""`	`He said "hi"`
`\\`	Backslash	`"C:\\path\\to\\file"`	`C:\path\to\file`

Data Type Conversion Reference

Function	Description	Example	Result
`int(x)`	Converts `x` to an integer.	`int("42")`	`42`
`float(x)`	Converts `x` to a float.	`float("3.14")`	`3.14`
`str(x)`	Converts `x` to a string.	`str(42)`	`"42"`
`list(x)`	Converts `x` to a list.	`list("abc")`	`['a', 'b', 'c']`
`tuple(x)`	Converts `x` to a tuple.	`tuple([1, 2, 3])`	`(1, 2, 3)`
`set(x)`	Converts `x` to a set.	`set([1, 2, 2, 3])`	`{1, 2, 3}`
`dict(x)`	Converts a list of pairs to dict.	`dict([('a', 1)])`	`{'a': 1}`

Logical Operators in Python

Operator	Description	Example	Result
`and`	Returns `True` if both operands are `True`.	`True and False`	`False`
`or`	Returns `True` if at least one operand is `True`.	`True or False`	`True`

Operator	Description	Example	Result
`not`	Reverses the logical state of its operand.	`not True`	`False`

Comparison Operators in Python

Operator	Description	Example	Result
`==`	Checks if two values are equal.	`5 == 5`	`True`
`!=`	Checks if two values are not equal.	`5 != 3`	`True`
`<`	Checks if left is less than right.	`3 < 5`	`True`
`>`	Checks if left is greater than right.	`5 > 3`	`True`
`<=`	Checks if left is less than or equal to right.	`3 <= 3`	`True`
`>=`	Checks if left is greater than or equal to right.	`5 >= 3`	`True`

Bitwise Operators in Python

Operator	Description	Example	Result
`&`	Bitwise AND	`5 & 3`	`1`
`|`	Bitwise OR	`5 | 3`	`7`
`^`	Bitwise XOR	`5 ^ 3`	`6`
`~`	Bitwise NOT (invert bits).	`~5`	`-6`
`<<`	Bitwise left shift	`5 << 1`	`10`
`>>`	Bitwise right shift	`5 >> 1`	`2`

Membership and Identity Operators

Operator	Description	Example	Result
`in`	Returns `True` if a value exists in a sequence.	`'a' in 'abc'`	`True`
`not in`	Returns `True` if a value does not exist in a sequence.	`'d' not in 'abc'`	`True`
`is`	Returns `True` if two references point to the same object.	`x is y`	`True` or `False`
`is not`	Returns `True` if two references do not point to the same object.	`x is not y`	`True` or `False`

Common Built-in Functions in Python

Function	Description	Example	Result

Function	Description	Example	Result
`len()`	Returns the length of an object.	`len([1, 2, 3])`	`3`
`max()`	Returns the maximum value in an object.	`max([1, 2, 3])`	`3`
`min()`	Returns the minimum value in an object.	`min([1, 2, 3])`	`1`
`sum()`	Returns the sum of elements in an object.	`sum([1, 2, 3])`	`6`
`abs()`	Returns the absolute value of a number.	`abs(-5)`	`5`
`round()`	Rounds a number to the nearest integer.	`round(3.14)`	`3`
`sorted()`	Returns a sorted list.	`sorted([3, 1, 2])`	`[1, 2, 3]`
`type()`	Returns the type of an object.	`type(42)`	`<class 'int'>`
`id()`	Returns the memory address of an object.	`id(x)`	e.g., `140158527`

Exception Handling Keywords in Python

Keyword	Description	Example
`try`	Defines a block of code to test for errors.	`try: x = 1/0`
`except`	Handles an error if one occurs.	`except ZeroDivisionError: print("Division by zero!")`
`finally`	Always executes a block of code, whether an exception occurs or not.	`finally: print("Done!")`
`raise`	Manually raises an exception.	`raise ValueError("Invalid value!")`
`else`	Executes if the `try` block succeeds (no exceptions).	`else: print("Success!")`

Mutable vs Immutable Objects

Type	Mutable?	Examples	Operations
`list`	Yes	`[1, 2, 3]`	Append, pop, insert, modify elements.
`dict`	Yes	`{'a': 1, 'b': 2}`	Add, remove keys, or modify values.
`set`	Yes	`{1, 2, 3}`	Add or remove elements.
`tuple`	No	`(1, 2, 3)`	Cannot modify once created.
`frozenset`	No	`frozenset([1, 2, 3])`	Immutable version of a `set`.
`str`	No	`"hello"`	Cannot modify characters directly.
`int`, `float`, `bool`, etc.	No	`42`, `3.14`, `True`	Operations return new objects, not in-place changes.

Loops and Control Flow

Construct	Description	Example
`for`	Iterates over a sequence (e.g., list, range).	`for i in range(3): print(i)` → `0 1 2`
`while`	Repeats as long as a condition is true.	`while x < 5: print(x); x += 1`
`break`	Exits the nearest enclosing loop.	`for i in range(5): if i == 3: break`
`continue`	Skips the rest of the loop for this iteration.	`for i in range(5): if i == 3: continue`
`pass`	Does nothing, placeholder for future code.	`if x > 5: pass`

Common List Comprehensions

Description	Syntax	Example	Result
Create a list of squares.	`[x**2 for x in range(5)]`	`[x**2 for x in range(5)]`	`[0, 1, 4, 9, 16]`
Filter even numbers.	`[x for x in range(5) if x%2==0]`	`[x for x in range(5) if x%2==0]`	`[0, 2, 4]`
Nested loops.	`[(x, y) for x in range(2) for y in range(2)]`	`[(x, y) for x in range(2) for y in range(2)]`	`[(0, 0), (0, 1), (1, 0), (1, 1)]`

Python Sorting Techniques

Method	Description	Example	Result
`sorted(iterable)`	Returns a new sorted list from the elements of `iterable`.	`sorted([3, 1, 2])`	`[1, 2, 3]`
`.sort()`	Sorts a list in-place.	`my_list.sort()`	Original list is modified.
`key` parameter in `sorted`	Sorts using a custom function or key.	`sorted(["apple", "bat"], key=len)`	`['bat', 'apple']`
`reverse=True`	Sorts in descending order.	`sorted([3, 1, 2], reverse=True)`	`[3, 2, 1]`

Python `print()` Method and Examples

Feature	Description	Example	Result
Basic Print	Prints a value or multiple values separated by a space.	`print("Hello", "World")`	`Hello World`
Separator (`sep`)	Customizes the separator between multiple values.	`print("Hello", "World", sep="-")`	`Hello-World`

Feature	Description	Example	Result
End (`end`)	Customizes the string printed at the end (default is `"\n"` for newline).	`print("Hello", end="!")`	`Hello!`
File Output	Directs the output to a file instead of the console.	`print("Hello", file=f)`	Writes `Hello` to the file `f`.
Flush (`flush`)	Forces the output to be flushed immediately (useful for real-time logs).	`print("Loading...", flush=True)`	Prints immediately without buffering.
Formatted Output	Allows formatted strings using f-strings, `.format()`, or `%`.	`print(f"Sum: {3+2}")`	`Sum: 5`
Printing Lists	Automatically converts lists to strings for printing.	`print([1, 2, 3])`	`[1, 2, 3]`

Common `print()` Examples

```
# 1. Basic Printing
print("Hello, World!")  # Output: Hello, World!

# 2. Printing Multiple Values
print("Python", "is", "fun!")  # Output: Python is fun!

# 3. Using the `sep` Parameter
print("Python", "is", "fun!", sep="-")  # Output: Python-is-fun!

# 4. Using the `end` Parameter
print("Hello", end=" ")
print("World!")  # Output: Hello World! (on the same line)

# 5. Redirecting Output to a File
with open("output.txt", "w") as f:
    print("Hello, File!", file=f)
# Writes "Hello, File!" to `output.txt`

# 6. Formatted Output with f-Strings
name = "Alice"
age = 25
print(f"My name is {name} and I am {age} years old.")
# Output: My name is Alice and I am 25 years old.

# 7. Printing Lists
my_list = [1, 2, 3]
print(my_list)  # Output: [1, 2, 3]

# 8. Using Flush for Immediate Output
import time
for i in range(5, 0, -1):
    print(f"Countdown: {i}", end="\r", flush=True)
    time.sleep(1)
```

```
print("Liftoff!")

Countdown: 5   # Overwritten after 1 second
Countdown: 4   # Overwritten after 1 second
Countdown: 3   # Overwritten after 1 second
Countdown: 2   # Overwritten after 1 second
Countdown: 1   # Overwritten after 1 second
Liftoff!       # Final printed line
```

Function Calling in Python

Function calling is a core concept in Python that allows you to execute reusable blocks of code. In the PCEP™ exam, you will encounter questions testing your understanding of positional arguments, keyword arguments, default values, and argument unpacking. This section will guide you through these concepts with examples.

Basic Function Syntax

```
def greet(name):
    print(f"Hello, {name}!")

greet("Alice")  # Output: Hello, Alice!

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

print(add(3, 5))  # Output: 8

def introduce(name, age):
    print(f"My name is {name} and I am {age} years old.")

introduce(age=30, name="Bob")  # Output: My name is Bob and I am 30 years old.

def greet(name="Guest"):
    print(f"Hello, {name}!")

greet()          # Output: Hello, Guest!
greet("Alice")   # Output: Hello, Alice!

def sum_numbers(*args):
    return sum(args)

print(sum_numbers(1, 2, 3))  # Output: 6

def print_details(**kwargs):
    for key, value in kwargs.items():
        print(f"{key}: {value}")

print_details(name="Alice", age=30)
# Output:
# name: Alice
```

```
# age: 30

def display(a, b, c):
    print(a, b, c)

args = [1, 2, 3]
display(*args)  # Output: 1 2 3

kwargs = {'a': 1, 'b': 2, 'c': 3}
display(**kwargs)  # Output: 1 2 3

def divide(a, b, /):
    return a / b

divide(10, 2)  # Valid
divide(a=10, b=2)  # Error: Must be positional

def configure(*, debug=False):
    print(f"Debug mode: {debug}")

configure(debug=True)  # Valid
configure(True)        # Error: Must use keyword
```

List of Resources

Here is a curated list of resources to help you deepen your understanding of Python and prepare for the PCEP™ certification exam:

Official Python Documentation

- Python Official Documentation
  Comprehensive reference for Python syntax, built-in functions, and libraries.

PCEP Certification Resources

- PCEP™ Exam Syllabus
  Review the official exam objectives to align your study plan.
- OpenEDG Python Institute Practice Tests
  Free practice questions provided by the Python Institute.

Books

- **“Python Crash Course” by Eric Matthes**
  Hands-on introduction to Python fundamentals and projects.
  Link to Book

Online Courses

- Udemy: Python Certification PCEP Course by Ana Uzelac
  Mock exams and practice tests designed for PCEP certification.

- Coursera: Python for Everybody
  Specialization in Python programming fundamentals.

Interactive Practice Platforms

- HackerRank
  Solve Python problems and improve coding skills.

Cheat Sheets

- Python Basics Cheat Sheet
  Quick reference for Python syntax and common functions.
- Real Python: Python Basics Cheat Sheet
  Covers Python fundamentals and best practices.

Discussion Forums

- Stack Overflow
  Ask and answer Python-related programming questions.
- Reddit: r/learnpython
  Community-driven platform for Python learners.

Mock Exams

- PCEP Practice Tests on Udemy
  High-quality tests mimicking the real PCEP exam.
- OpenEDG Mock Exams
  Free and paid mock exams directly from the Python Institute.

Use This Prompt as a Study Tool with ChatGPT or Gemini

If you're preparing for the **PCEP – Certified Entry-Level Python Programmer** exam, this prompt is designed to help you study effectively using AI-powered tools like **ChatGPT** or **Gemini**. By following the steps below, you can simulate a personalized mock exam experience with detailed explanations for every question.

How to Use the Prompt

1. **Copy the Prompt**: Copy the provided prompt text and paste it into your preferred AI assistant (e.g., ChatGPT or Gemini).

2. **Start the Session**: Once the AI is ready, the tool will present you with **one multiple-choice question at a time**.

3. **Answer the Question**: Type your answer (e.g., "A", "B", or a combination like "A and F" for multiple selections).

4. **Receive Feedback**:
    - If your answer is correct, the AI will confirm it and provide a detailed explanation.
    - If your answer is incorrect, the AI will guide you with the correct answer and a comprehensive breakdown of the concept.
5. **Continue at Your Pace**: After each question, the AI will ask if you're ready for the next one. You can choose to:
    - Move to the next question.
    - Pause your session and resume later.
    - Stop the session at any time.

Study Tool Prompt

Copy the following prompt and use it with ChatGPT or Gemini to simulate a personalized mock exam for the PCEP certification:

```
I am preparing for the PCEP – Certified Entry-Level Python Programmer exam. Help
me study by asking me one multiple-choice question at a time.

Here's how I'd like the study process to work:
1. Provide me with a **mock exam-style multiple-choice question**.
2. Wait for my answer.
3. Once I answer, evaluate whether my answer is **correct** or **incorrect**.
   - If my answer is correct, say so and explain why in detail.
   - If my answer is incorrect, tell me the correct answer and provide a
comprehensive explanation to help me understand the topic.
4. After explaining, ask me: **"Are you ready for the next question?"**
5. Only provide the next question if I confirm I'm ready.

The mock questions should:
- Cover key areas of the PCEP syllabus:
  - Python basics (data types, literals, variables, and operators).
  - Control flow (loops, conditionals, and exceptions).
  - Data collections (lists, tuples, dictionaries, and strings).
  - Functions and their behavior.
- Be multiple-choice with clear options.
- Vary in difficulty to simulate the exam experience.

Here are sample questions that match the style of questions I want to practice
with:

### **Sample Question 1**:
What is the expected result of the following code?

def subtract_and_store(value):
    return store - value

store = 10
```

```
store = subtract_and_store(3)
store = subtract_and_store(4)
print(store)

### **Sample Question 2**:

try:
  x = input("Enter the first value: ")
  a = len(x)
  y = input("Enter the second value: ")
  b = len(y) * 2
  print(a / b)
except ZeroDivisionError:
  print("You can't divide by zero!")
except ValueError:
  print("Wrong value error.")
except:
  print("Some other error.")

### Example Non-Code Question:
**Which two of the following statements about Python are true?**

A) Python is case-sensitive, meaning `Variable` and `variable` are treated as
different identifiers.
B) Python allows multiple inheritance in classes.
C) A tuple in Python is mutable, meaning its contents can be changed.
D) The `is` operator checks for value equality between two variables.
E) Python variables must be declared with a specific type (e.g., `int`, `float`)
before being used.
F) Python's `try` block can be used with `except`, `else`, and `finally` clauses.

### Correct Answers:
**A)** Python is case-sensitive, meaning `Variable` and `variable` are treated as
different identifiers.
**F)** Python's `try` block can be used with `except`, `else`, and `finally`
clauses.

Explanation of Answers:
- **A) Correct.** Python is case-sensitive, treating `Variable` and `variable` as
different identifiers.
- **B) Correct, but not relevant to this question.** Python supports multiple
inheritance but is not part of this question's focus.
- **C) Incorrect.** Tuples are immutable.
- **D) Incorrect.** The `is` operator checks for object identity, not value
equality.
- **E) Incorrect.** Python does not require explicit type declarations.
- **F) Correct.** Python's `try` block can include `except`, `else`, and `finally`
clauses for exception handling.
```