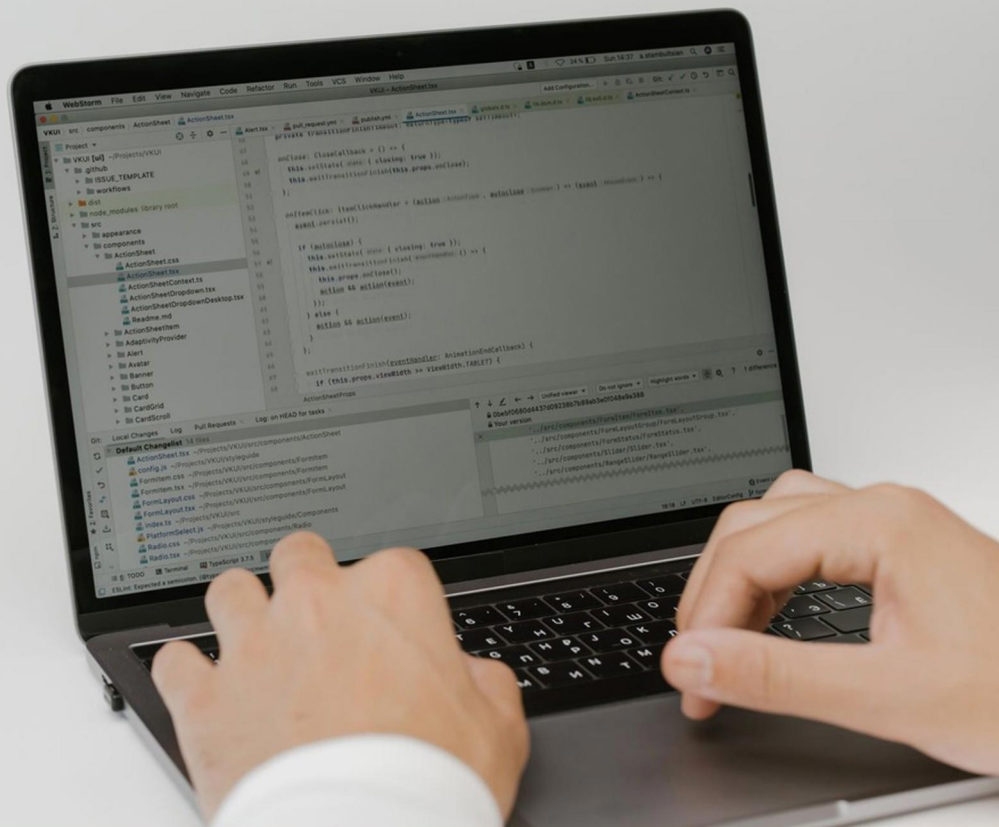


CERTIFIED PROFESSIONAL IN PYTHON PROGRAMMING LEVEL 1 (PCPP1) PRACTICE EXAM (SAMPLE)

STUDY GUIDE



SAMPLE STUDY GUIDE WITH LIMITED QUESTIONS

VISIT US ONLINE FOR
THE FULL VERSION STUDY GUIDE

<https://pcpp1.examzify.com>



Copyright © 2026 by Examzify - A Kaluba Technologies Inc. product.

ALL RIGHTS RESERVED.

No part of this book may be reproduced or transferred in any form or by any means, graphic, electronic, or mechanical, including photocopying, recording, web distribution, taping, or by any information storage retrieval system, without the written permission of the author.

Notice: Examzify makes every reasonable effort to obtain accurate, complete, and timely information about this product from reliable sources.

SAMPLE

Table of Contents

Copyright	1
Table of Contents	2
Introduction	3
How to Use This Guide	4
Questions	5
Answers	8
Explanations	10
Next Steps	15

SAMPLE

Introduction

Preparing for a certification exam can feel overwhelming, but with the right tools, it becomes an opportunity to build confidence, sharpen your skills, and move one step closer to your goals. At Examzify, we believe that effective exam preparation isn't just about memorization, it's about understanding the material, identifying knowledge gaps, and building the test-taking strategies that lead to success.

This guide was designed to help you do exactly that.

Whether you're preparing for a licensing exam, professional certification, or entry-level qualification, this book offers structured practice to reinforce key concepts. You'll find a wide range of multiple-choice questions, each followed by clear explanations to help you understand not just the right answer, but why it's correct.

The content in this guide is based on real-world exam objectives and aligned with the types of questions and topics commonly found on official tests. It's ideal for learners who want to:

- Practice answering questions under realistic conditions,
- Improve accuracy and speed,
- Review explanations to strengthen weak areas, and
- Approach the exam with greater confidence.

We recommend using this book not as a stand-alone study tool, but alongside other resources like flashcards, textbooks, or hands-on training. For best results, we recommend working through each question, reflecting on the explanation provided, and revisiting the topics that challenge you most.

Remember: successful test preparation isn't about getting every question right the first time, it's about learning from your mistakes and improving over time. Stay focused, trust the process, and know that every page you turn brings you closer to success.

Let's begin.

How to Use This Guide

This guide is designed to help you study more effectively and approach your exam with confidence. Whether you're reviewing for the first time or doing a final refresh, here's how to get the most out of your Examzify study guide:

1. Start with a Diagnostic Review

Skim through the questions to get a sense of what you know and what you need to focus on. Your goal is to identify knowledge gaps early.

2. Study in Short, Focused Sessions

Break your study time into manageable blocks (e.g. 30 – 45 minutes). Review a handful of questions, reflect on the explanations.

3. Learn from the Explanations

After answering a question, always read the explanation, even if you got it right. It reinforces key points, corrects misunderstandings, and teaches subtle distinctions between similar answers.

4. Track Your Progress

Use bookmarks or notes (if reading digitally) to mark difficult questions. Revisit these regularly and track improvements over time.

5. Simulate the Real Exam

Once you're comfortable, try taking a full set of questions without pausing. Set a timer and simulate test-day conditions to build confidence and time management skills.

6. Repeat and Review

Don't just study once, repetition builds retention. Re-attempt questions after a few days and revisit explanations to reinforce learning. Pair this guide with other Examzify tools like flashcards, and digital practice tests to strengthen your preparation across formats.

There's no single right way to study, but consistent, thoughtful effort always wins. Use this guide flexibly, adapt the tips above to fit your pace and learning style. You've got this!

Questions

SAMPLE

1. Which ConfigParser method returns a string value for a given key?
 - A. getint()
 - B. getboolean()
 - C. get()
 - D. getfloat()
2. The deleter method for a property is defined using which decorator?
 - A. @property
 - B. @propertyname.deleter
 - C. @deleter
 - D. @remove
3. What are the three arguments required by type() to create a new class?
 - A. name, bases, dict
 - B. class name, body, metaclass
 - C. name, dict, metaclass
 - D. type is not used to create classes
4. In the shallow vs deep copy example, after modifying a[2][0] = 5 on the original, which lists reflect the change in their nested lists?
 - A. a and b reflect the change; c does not.
 - B. Only a reflects the change; b and c do not.
 - C. All reflect the change.
 - D. None reflect the change.
5. How are public attributes defined according to the material?
 - A. Named with a single leading underscore
 - B. Named with a double leading underscore
 - C. Named with a double trailing underscore
 - D. Named with triple underscores

6. Which two magic methods should be implemented to allow iteration over a class using a for loop?
- A. `__init__` and `__str__`
 - B. `__iter__` and `__next__`
 - C. `__len__` and `__getitem__`
 - D. `__enter__` and `__exit__`
7. Which XPath expression would select the names of all universities located in Cambridge, MA?
- A. `//university[location="Cambridge, MA"]/name`
 - B. `//university/name[location="Cambridge, MA"]`
 - C. `/university[location="Cambridge, MA"]/name`
 - D. `//name[location="Cambridge, MA"]/university`
8. According to PEP 8, what indentation style should be used?
- A. 2 spaces per indentation level.
 - B. 4 spaces per indentation level with no tabs.
 - C. A tab character for each level.
 - D. Mixing tabs and spaces is allowed.
9. In `xml.etree.ElementTree`, what type is returned by `ET.fromstring(xml_string)`?
- A. String
 - B. Element
 - C. List
 - D. Dictionary
10. Which import form is generally discouraged in Python due to namespace pollution?
- A. `from module_name import *`
 - B. `import module_name`
 - C. `from module_name import object_name`
 - D. `import module_name as alias`

Answers

SAMPLE

1. C
2. B
3. A
4. A
5. A
6. B
7. A
8. B
9. B
10. A

SAMPLE

Explanations

SAMPLE

1. Which ConfigParser method returns a string value for a given key?

- A. getint()
- B. getboolean()
- C. get()**
- D. getfloat()

In Python's configparser, values are read as strings by default. To fetch the value exactly as stored, you use the `get()` method, which returns the value as a string for the given section and option. The other getters—`getint()`, `getboolean()`, and `getfloat()`—convert the stored string to an int, boolean, or float, respectively. So, `get()` is the right choice when you need the raw string value.

2. The deleter method for a property is defined using which decorator?

- A. @property
- B. @propertyname.deleter**
- C. @deleter
- D. @remove

The key idea is that a deleter for a property is defined by decorating a method with the deleter attribute of the property, using the property's own name. In code, you first create the property with `@property`, and then define the deleter using `@<property_name>.deleter`. This ties the deleter directly to that property and makes `del obj.prop` trigger this function. For example: `class Example: @property def value(self): return self._value @value.deleter def value(self): del self._value` Here, `del obj.value` will call the deleter, allowing you to clean up or release resources associated with value. This is why the correct approach uses the property name followed by `.deleter`.

3. What are the three arguments required by `type()` to create a new class?

- A. name, bases, dict**
- B. class name, body, metaclass
- C. name, dict, metaclass
- D. type is not used to create classes

Creating a class with `type()` is about using the built-in metaclass to assemble a new class object. The three things you pass are the class name as a string, a tuple of base classes that the new class will inherit from, and a dictionary that defines the class's attributes and methods. The dictionary becomes the class namespace, so whatever you put in there ends up as members of the class. For example, you can write `MyDynamic = type("MyDynamic", (object,), {"value": 42, "greet": lambda self: f"value is {self.value}"})` to build a simple class on the fly. The key is that the three arguments correspond to the name, the bases, and the namespace dict; this is why the correct answer lists name, bases, and dict. The other choices mix different ideas—like a separate body or a metaclass parameter—that aren't how `type()` is invoked in this form, and one option incorrectly states that `type` isn't used to create classes.

4. In the shallow vs deep copy example, after modifying `a[2][0] = 5` on the original, which lists reflect the change in their nested lists?

- A. a and b reflect the change; c does not.
- B. Only a reflects the change; b and c do not.
- C. All reflect the change.
- D. None reflect the change.

Shallow copies versus deep copies affect nested mutable objects differently. When you copy a list shallowly, Python creates a new outer list but reuses the same inner lists inside it. So the inner lists are shared between the original and the shallow copy. If you modify an element inside one of those inner lists, the change shows up in both the original and the shallow copy because they reference the same inner list object. In contrast, a deep copy duplicates every level of the structure. The inner lists become independent copies, so changing the original's nested list doesn't affect the deep copy. After doing `a[2][0] = 5` on the original, the original and its shallow copy reflect the change, while the deep copy does not.

5. How are public attributes defined according to the material?

- A. Named with a single leading underscore
- B. Named with a double leading underscore
- C. Named with a double trailing underscore
- D. Named with triple underscores

Naming conventions for attribute visibility in Python is what this item is testing. In the material, a public attribute is indicated by naming it with a single leading underscore. That simple prefix signals to users that the attribute is part of the class's interface and can be accessed from outside, but it's commonly treated as something that users should use with caution because it's more of a hint about intent than a strict access control. This contrasts with other patterns: a double leading underscore triggers name mangling, making external access more difficult and helping to avoid accidental name clashes in subclasses. A double trailing underscore is used for special, Python-defined attributes and methods (often called dunder methods), which have their own defined roles in the language. Triple underscores aren't a standard convention in Python for attribute naming.

6. Which two magic methods should be implemented to allow iteration over a class using a for loop?

- A. `__init__` and `__str__`
- B. `__iter__` and `__next__`
- C. `__len__` and `__getitem__`
- D. `__enter__` and `__exit__`

Iteration in Python relies on the iterator protocol. To let a for loop go through your class, you need to provide a way for Python to obtain an iterator and then fetch items one by one. This is done by implementing two methods: `__iter__` and `__next__`. The `__iter__` method returns an iterator object (often the object itself), and `__next__` returns the next value each time it's called, raising `StopIteration` when there are no more items. The other options serve different purposes—initialization, string representation, or context management—or in the case of `__len__` and `__getitem__`, offer an older style of sequence access, not the standard for loop iteration. So implementing `__iter__` and `__next__` is the correct approach for enabling for-loop iteration.

7. Which XPath expression would select the names of all universities located in Cambridge, MA?

- A. `//university[location="Cambridge, MA"]/name`
- B. `//university/name[location="Cambridge, MA"]`
- C. `/university[location="Cambridge, MA"]/name`
- D. `//name[location="Cambridge, MA"]/university`

The idea is to navigate to the university elements and filter them by a child value, then move to their name child. Using a double slash searches the entire document for university elements. The predicate `[location="Cambridge, MA"]` keeps only those universities whose location child equals Cambridge, MA. Finally, `/name` selects the name child of each matching university, giving you the names themselves. This is why this expression best matches the goal: it finds all relevant universities anywhere in the document and returns their names. The other forms don't fit: placing the predicate on the name element would filter name nodes (which typically don't have a location), starting with a single slash assumes the document root is a university element, and selecting `/university/name[...]` would apply the predicate to the name node rather than filtering universities by their location.

8. According to PEP 8, what indentation style should be used?

- A. 2 spaces per indentation level.
- B. 4 spaces per indentation level with no tabs.
- C. A tab character for each level.
- D. Mixing tabs and spaces is allowed.

PEP 8 sets the indentation style to keep Python code readable and consistent across editors. It specifies using four spaces per indentation level and says not to use tab characters. This approach avoids the rendering inconsistencies that can happen when tabs of different widths are mixed with spaces, which can lead to indentation errors in Python. So, four spaces per level with no tabs is the recommended practice. The other options either propose an incorrect width, rely on a tab, or allow mixing tabs and spaces, all of which conflict with the standard.

9. In `xml.etree.ElementTree`, what type is returned by `ET.fromstring(xml_string)`?

- A. String
- B. Element
- C. List
- D. Dictionary

The thing `ET.fromstring` parses the XML into is an `Element` object that represents the root of the XML tree. This is an instance of `xml.etree.ElementTree.Element`, not a string, list, or dictionary. You can access the root's tag with `.tag`, its attributes with `.attrib`, and its text with `.text`, and you can traverse its children by iterating or using methods like `.find` and `.findall`. If you need a full tree object, you would wrap the root element with `ET.ElementTree(root)`.

10. Which import form is generally discouraged in Python due to namespace pollution?

- A. from module_name import *
- B. `import module_name`
- C. `from module_name import object_name`
- D. `import module_name as alias`

Bringing in all names from a module with a star import is generally discouraged because it dumps every public symbol into your current namespace. This hides where each name came from, increases the risk of overwriting existing names, and makes the code harder to read and maintain. Tools like linters and IDEs also struggle to determine where a symbol originated, which can lead to mistakes and less helpful code navigation. In contrast, importing the module itself (`import module_name`) keeps a clear boundary between your code and the module, since you access names as `module_name.something`. This explicit qualification reduces confusion about where names come from. If you only need a few objects, using `from module_name import object_name` is safer because it imports only those specific items, though you still should be mindful of potential name conflicts. Using an alias (`import module_name as alias`) is also fine, as it preserves the module's namespace while offering a shorter or clearer name for use in your code.

SAMPLE

Next Steps

Congratulations on reaching the final section of this guide. You've taken a meaningful step toward passing your certification exam and advancing your career.

As you continue preparing, remember that consistent practice, review, and self-reflection are key to success. Make time to revisit difficult topics, simulate exam conditions, and track your progress along the way.

If you need help, have suggestions, or want to share feedback, we'd love to hear from you. Reach out to our team at hello@examzify.com.

Or visit your dedicated course page for more study tools and resources:

<https://pcpp1.examzify.com>

We wish you the very best on your exam journey. You've got this!

SAMPLE