PCAP CERTIFICATION / WHAT HAS CHANGED

A clearer path to your PCAP™ certification



Your guide to PCAP-31-03 vs. PCAP-31-04

Same certification. Broader skills. A more complete approach to Python development.

PCAP-31-04 takes associate-level Python certification further with an expanded, more comprehensive syllabus that reflects how Python programs are designed, tested, debugged, and maintained in modern development environments.

The updated exam goes beyond writing correct code. It introduces important software development practices, including automated testing, collaborative workflows, and responsible use of AI-assisted programming tools.

Discover what has changed, what remains essential, and how to build on your existing knowledge to prepare confidently for the updated assessment.

QUESTIONS	SECTIONS	PASSING SCORE
40 → 42	**5 → 6**	**70% → 72%**
Two additional questions	A dedicated sixth section	Updated passing standard

The Big Picture

Same credential. A broader definition of Python proficiency.

PCAP-31-04 maintains its associate-level focus while expanding the skills candidates are expected to demonstrate. The revised assessment places greater emphasis on practical programming, modern Python features, structured problem-solving, software quality, and professional development workflows.

KEEP

The programming foundation

Modules, packages, exceptions, strings, classes, inheritance, comprehensions, functions, and file operations remain essential.

ADD

Modern Python development practices

Package dependency management, structural pattern matching, advanced comprehensions, filesystem operations, testing, debugging, collaboration, and AI-assisted development receive explicit assessment coverage.

For candidates already preparing for PCAP-31-03: Your existing knowledge remains valuable. Use this guide to identify the new objectives, strengthen areas with expanded coverage, and adjust your preparation without starting from scratch.

The exams at a glance

A quick side-by-side comparison of the exam experience.

EXAM DETAIL	PCAP-31-03	PCAP-31-04
Questions	40	42
Exam duration	65 minutes	65 minutes
Syllabus structure	5 blocks	6 sections
Passing score	70%	72%
Certification level	Associate	Associate
Prerequisites	None, PCEP recommended	None, PCEP recommended
Credential validity	Lifetime	5 years
Credential awarded	PCAP™	PCAP™

What this means

A more comprehensive exam, with greater emphasis on practical development

PCAP-31-04 introduces two additional questions and a sixth examination section while retaining the same 65-minute testing duration.

The syllabus has been significantly revised to reflect the knowledge and skills required to create, organize, test, debug, and maintain intermediate Python programs. Although the core

language topics remain important, the updated assessment now explicitly covers software development activities that were not separately assessed in PCAP-31-03.

The passing requirement has also increased from 70% to 72%, reflecting the updated competency standard established for the revised certification assessment.

The examination remains focused on associate-level Python programming rather than advanced software engineering or specialized professional development.

What does not change

Your PCAP certification and previously earned credentials

Passing either exam version awards the same PCAP™ certification.

The introduction of PCAP-31-04 does not invalidate previously earned credentials or require existing certification holders to retake the examination.

Credentials already issued remain valid according to their original terms. Certifications issued with lifetime validity retain that status, while certifications issued with a five-year validity period retain their original expiration dates.

The introduction of a new examination version does not retroactively change the validity of an existing credential.

Why credential validity matters

The five-year credential validity period helps ensure that PCAP certification continues to represent relevant, current knowledge of Python programming and software development practices.

Python and its development ecosystem continue to evolve. New language capabilities, development tools, testing practices, and AI-assisted workflows increasingly influence how software is created and maintained.

The validity period balances recognition of professional achievement with the need to keep technical credentials aligned with modern programming practices.

We encourage certification holders to continue developing their skills throughout their careers and to consider updated examination versions as opportunities to demonstrate ongoing professional growth.

Important dates

PCAP-31-03 will be retired on November 30, 2027

PCAP-31-03 will remain available during the transition period and will be officially **retired (sunsetted) on November 30, 2027**. After that date, candidates will no longer be able to take the PCAP-31-03 exam.

Candidates currently preparing for PCAP-31-03 can continue following its syllabus and take the exam before the retirement date. Those beginning their certification journey are encouraged to prepare for PCAP-31-03, which reflects the updated and expanded assessment scope.

The retirement of PCAP-31-03 does not affect the validity of certifications already earned.

Important for exam day

Check the exact exam code

Make sure your syllabus, learning resources, and preparation products correspond to the exam version you intend to take. If you have already purchased a voucher, confirm its exam version, validity, and redemption terms in your candidate account or with your voucher provider.

Planning to take PCAP-31-03? Make sure you complete your exam no later than November 30, 2027. Voucher expiration dates do not extend the availability of a retired exam version. Do not assume that exam versions or vouchers are interchangeable.

How the syllabus has changed

The original five-block structure has become a six-section structure.

Four existing sections retain their names, although their assessment coverage and relative weights have been revised. The fifth section has been renamed and expanded, while an entirely new sixth section introduces software testing, debugging, collaborative development, and AI-assisted programming.

PCAP-31-03 – OLD EXAM BLOCKS	PCAP-31-04 – NEW EXAM SECTIONS
Block 1: Modules and Packages 6 questions · 12%	**Section 1: Modules and Packages** 5 questions · 12%
Block 2: Exceptions 5 questions · 14%	**Section 2: Exceptions** 5 questions · 12%
Block 3: Strings 8 questions · 18%	**Section 3: Strings** 4 questions · 9.5%
Block 4: Object-Oriented Programming 12 questions · 34%	**Section 4: Object-Oriented Programming** 12 questions · 28.5%
Block 5: Miscellaneous (List Comprehensions, Lambdas, Closures, I/O) 9 questions · 22%	**Section 5: Advanced Python Constructs and I/O** 10 questions · 24%
–	**Section 6: Testing, Debugging, Collaborative Development, and AI Assistance** 6 questions · 14%
40 questions · 100%	42 questions · 100%

The weights above are the officially published scoring weights. They are not necessarily identical to each section's proportion of the question count.

What stays familiar

Five essential foundations carry forward

1. **Modules and packages**: Importing modules, working with namespaces, creating reusable modules and packages, and understanding Python's module-search mechanisms.
2. **Exceptions**: Handling, raising, inspecting, and defining exceptions, including exception hierarchies and structured exception-handling statements.
3. **Strings**: Character encoding, Unicode, string immutability, comparisons, classification, searching, and text processing.
4. **Object-oriented programming**: Classes, objects, attributes, methods, initialization, introspection, inheritance, polymorphism, and object relationships.
5. **Advanced Python constructs and I/O**: Comprehensions, lambda functions, map(), filter(), closures, streams, and file operations.

These topics remain central to associate-level Python programming. Candidates who have already developed these skills have a substantial foundation for preparing for PCAP-31-04.

The most visible change

A dedicated sixth section: Modern Python Development Practices

The most significant structural change is the introduction of *Section 6: Testing, Debugging, Collaborative Development, and AI Assistance*.

This new section accounts for **6 questions** and **14% of the examination**.

It reflects the fact that Python proficiency involves more than understanding language syntax and writing code that produces the expected result. Developers must also be able to verify their work, identify defects, collaborate effectively, and use modern development tools responsibly.

The new section includes three equally weighted areas:

- Testing and Debugging – **2 questions** (How to design and interpret unit tests, use assertions, recognize defects, read tracebacks, and verify corrections.)
- Teamwork and Collaboration – **2 questions** (How repositories, branches, commits, pull requests, code reviews, documentation, automated tests, and continuous integration support shared software development.)
- AI-Assisted Development – **2 questions** (How to use AI tools to generate and improve Python code while independently checking correctness, security, maintainability, and compliance with requirements.)

This is not a separate AI certification. The section evaluates how associate-level Python programmers use AI assistance as part of an ordinary software development workflow while retaining responsibility for the final code.

What to add to your study plan

These skills are newly introduced, substantially expanded, or more explicitly developed in the PCAP-31-04 assessment objectives. Use the self-check questions to determine what you already understand and which topics deserve additional preparation.

New and newly introduced coverage

TOPIC	WHAT TO STUDY & SELF-CHECK
EXTERNAL PACKAGES AND DEPENDENCIES	Understand PyPI and distinguish external packages from standard-library and local modules. Use `pip` and `python -m pip` to install, list, and remove packages. Create and interpret `requirements.txt` files and install project dependencies from them. **SELF-CHECK** Can you explain how to recreate a project's required packages using a requirements file?
STRUCTURAL PATTERN MATCHING	Use `match` and `case` to compare values against alternatives, include a default case, and distinguish basic pattern matching from an `if`/`elif`/`else` statement. **SELF-CHECK** Can you trace a `match` statement and determine which case executes?
SET AND DICTIONARY COMPREHENSIONS	Construct set and dictionary comprehensions in addition to list comprehensions. Transform values, filter elements, and choose the appropriate collection type for the result. **SELF-CHECK** Can you convert a list comprehension into a set or dictionary comprehension and explain how the result changes?
FILESYSTEM OPERATIONS WITH `pathlib`	Create and combine `Path` objects, distinguish relative and absolute paths, navigate directory contents using `iterdir()`, and use `read_text()`, `write_text()`, `read_bytes()`, and `write_bytes()`.

	SELF-CHECK Can you use `pathlib` to read a text file from a specified directory and save processed output to another file?
UNIT TESTING	Understand test cases, test suites, expected and actual results, and test isolation. Design tests using typical, boundary, and invalid inputs. Use `assert` to verify results and rerun tests after changes. **SELF-CHECK** Can you design three test cases for a function and explain what each one verifies?
DEBUGGING AND ERROR ANALYSIS	Distinguish syntax, runtime, and logic errors. Read tracebacks, inspect variable values, trace function calls, locate defects, and verify corrections. **SELF-CHECK** Can you use a traceback and the program's execution flow to identify the cause of an error?
VERSION CONTROL AND SHARED REPOSITORIES	Understand how repositories, branches, commits, and meaningful commit messages support collaborative development. Recognize merge conflicts and the purpose of pull or merge requests. **SELF-CHECK** Can you explain why developers create branches and how changes are safely integrated into a shared codebase?
CODE REVIEWS AND DEVELOPMENT COMMUNICATION	Understand how issue descriptions, documentation, code reviews, and constructive feedback help teams communicate requirements, defects, changes, and decisions. **SELF-CHECK** Can you identify useful review feedback and explain how a developer should respond to it?
AUTOMATED TESTING AND CONTINUOUS INTEGRATION	Explain how automated tests and continuous integration help validate changes, detect regressions, and maintain reliable shared code.

	SELF-CHECK Can you explain why a team runs automated checks before merging a change?
AI-ASSISTED PYTHON DEVELOPMENT	Use clear requirements, context, examples, and constraints when requesting AI assistance. Apply AI tools to draft, explain, complete, refactor, document, and test Python code. **SELF-CHECK** Can you describe an effective AI request for improving an existing Python function?
VALIDATING AI-GENERATED CODE	Independently review generated code for correctness, readability, maintainability, security, and compliance with requirements. Execute tests and verify that suggested modules, APIs, and methods actually exist. **SELF-CHECK** Can you identify why AI-generated code should not be accepted without testing and human review?
SECURE AND RESPONSIBLE AI USE	Apply human-in-the-loop practices, protect credentials and sensitive information, follow organizational requirements, and retain responsibility for approving generated code. **SELF-CHECK** Can you explain what information should not be shared with an AI tool and what checks are required before using its output?

Retained skills and coverage

TOPIC	WHAT TO STUDY & SELF-CHECK
MODULE IMPORTS AND NAMESPACES	Use `import`, `from ... import`, aliases, and fully qualified names. Recognize wildcard-import risks and name collisions between imported objects. **SELF-CHECK** Can you trace how Python resolves names when two imported modules contain objects with identical names?
MODULE SEARCH PATHS	Inspect modules using `dir()` and understand how `sys.path`, package structure, and the current environment affect module discovery. Recognize import failures and circular-import problems. **SELF-CHECK** Can you explain why a module fails to import even though its file exists?
USER-DEFINED MODULES AND PACKAGES	Organize reusable code into modules and packages. Understand `__name__`, `__main__`, `__init__.py`, nested packages, and `__pycache__`. **SELF-CHECK** Can you determine how a module behaves when executed directly versus imported into another program?
EXCEPTION HANDLERS AND OBJECTS	Order exception handlers correctly, group exception types in a tuple, capture exception objects using `except ... as`, and inspect their `args` property. **SELF-CHECK** Can you determine which handler catches a particular exception and what information the exception object contains?
EXCEPTION-HANDLING FLOW	Combine `try`, `except`, `else`, and `finally`. Trace handled and unhandled exceptions and determine which statements execute. **SELF-CHECK** Can you predict whether `else` and `finally` execute in different exception scenarios?

RAISING AND PROPAGATING EXCEPTIONS	Use `raise`, re-raise exceptions, distinguish a bare `raise` from explicitly raising an exception object, and trace propagation across function boundaries. **SELF-CHECK** Can you follow an exception through several function calls and determine where it is handled?
CUSTOM EXCEPTIONS	Define custom exception classes, provide explanatory information, and organize related exceptions into a hierarchy. **SELF-CHECK** Can you create a custom exception for an invalid application-specific condition and handle it appropriately?
UNICODE AND CHARACTER ENCODING	Understand ASCII, Unicode, UTF-8, code points, and the distinction between text and binary data. Use `ord()` and `chr()` for character-code-point conversion. **SELF-CHECK** Can you explain why a character can occupy more than one byte when encoded using UTF-8?
STRING COMPARISONS AND IMMUTABILITY	Trace string slicing, concatenation, repetition, and comparisons. Explain how Unicode code points and letter case affect lexicographical ordering. **SELF-CHECK** Can you predict the result of a string comparison and explain why string methods do not modify the original object?
STRING VALIDATION	Use classification methods such as `isalnum()`, `isalpha()`, `isdigit()`, `islower()`, `isspace()`, and `isupper()`. Combine classification results in Boolean expressions. **SELF-CHECK** Can you determine whether a string meets a set of validation rules?
STRING SEARCHING	Distinguish `find()`, `index()`, `rfind()`, and `rindex()`. Use start and stop arguments, `count()`, `startswith()`, and `endswith()`. Recognize when searches return `-1` or raise an exception.

	SELF-CHECK Can you select the correct search method and predict its result when a substring is absent?
OBJECT-ORIENTED DESIGN PRINCIPLES	Explain abstraction, encapsulation, inheritance, and polymorphism. Understand how these principles support reusable, maintainable code. **SELF-CHECK** Can you identify which OOP principle is demonstrated by a short Python example?
CLASS AND INSTANCE ATTRIBUTES	Distinguish shared class variables from instance-specific values. Trace attribute assignment, shadowing, namespace inspection with `__dict__`, and name mangling. **SELF-CHECK** Can you predict attribute values across several instances after modifying class and instance variables?
INSTANCE METHODS	Define methods using `self`, pass additional arguments, modify object state, override `__str__()`, and identify invalid method signatures. **SELF-CHECK** Can you explain how Python supplies the instance when a method is called through an object?
OBJECT AND CLASS INTROSPECTION	Use `hasattr()`, `getattr()`, `__name__`, `__module__`, and `__bases__` to inspect objects, classes, and inheritance relationships. **SELF-CHECK** Can you determine whether an object provides an attribute and retrieve it dynamically?
INHERITANCE AND METHOD RESOLUTION	Build single- and multiple-inheritance hierarchies, override methods, use `super()`, and interpret method resolution order (MRO), including diamond-shaped inheritance. **SELF-CHECK** Can you predict which method executes in a multiple-inheritance hierarchy and explain why?

OBJECT INITIALIZATION	Define `__init__()`, use positional and keyword arguments, provide sensible defaults, validate initialization data, and call superclass initializers. Recognize risks associated with mutable default arguments. **SELF-CHECK** Can you identify an unsafe initializer and correct its handling of default values?
NESTED COMPREHENSIONS	Construct and trace comprehensions with multiple `for` clauses, conditions, and nested collections. Recognize how iteration order and filtering affect results. **SELF-CHECK** Can you flatten a nested list and explain the order of values produced?
LAMBDA FUNCTIONS AND ITERATORS	Define and use lambdas with `map()` and `filter()`. Recognize that these functions return iterators in Python 3 and that results may need conversion to collections. **SELF-CHECK** Can you predict the values produced by a `map()` or `filter()` expression?
CLOSURES	Define nested functions, return inner functions, and understand how closures preserve access to enclosing variables. **SELF-CHECK** Can you create a simple function that remembers a value supplied during an earlier call?
FILE MODES AND STREAMS	Understand streams, file handles, standard input/output/error, text and binary modes, and access modes including `r`, `w`, `a`, `x`, `b`, `t`, and `+`. **SELF-CHECK** Can you select a suitable file mode for reading, appending, creating, or updating data?
SAFE TEXT AND BINARY I/O	Use `with` to manage file resources, read data using `read()`, `readline()`, and `readlines()`, write text and binary content, and work with `bytearray`. Handle common file-related exceptions.

SELF-CHECK Can you read and write a binary file safely and explain why a context manager is useful?

Topics that are no longer explicitly assessed

PCAP-31-04 revises the emphasis of several existing syllabus areas. Some operations and functions specifically identified in the PCAP-31-03 assessment objectives are no longer individually listed in the new detailed objectives.

This does not mean that these features have been removed from Python or that all related knowledge is irrelevant.

TOPIC	WHAT HAS CHANGED
SPECIFIC `math` MODULE FUNCTIONS	Functions such as `ceil()`, `floor()`, `trunc()`, `factorial()`, `hypot()`, and `sqrt()` were explicitly listed in PCAP-31-03. They are no longer individually identified in the PCAP-31-04 detailed assessment objectives.
SPECIFIC `random` MODULE FUNCTIONS	Functions including `random()`, `seed()`, `choice()`, and `sample()` are no longer individually listed in the revised detailed objectives.
SELECTED STRING TRANSFORMATION METHODS	The previous syllabus explicitly named methods such as `join()`, `split()`, and `strip()`. The revised detailed string objectives emphasize validation, comparisons, searching, ordering, and text representation.
`errno` AND SPECIFIC FILE ERROR CODES	The previous syllabus explicitly identified `errno` and its values. The revised objectives focus more broadly on safe file handling and common file-related exceptions.
EXPLICIT FILE CLOSING	The previous syllabus named `close()` directly. PCAP-31-04 places greater emphasis on using `with` statements to manage file resources safely.

Important: The new blueprint's candidate profile still refers to some standard-library and encoding operations that are not individually named in its detailed objectives. Candidates should follow the complete official syllabus and any clarification issued by the Python Institute.

How to use this checklist

If you are transitioning from PCAP-31-03 preparation, begin by identifying the topics you already understand.

Your existing knowledge of modules, exceptions, strings, object-oriented programming, comprehensions, lambda functions, closures, and file I/O remains relevant.

Next, focus on the most significant additions:

- External dependency management using pip and requirements.txt.
- Structural pattern matching with match and case.
- Set and dictionary comprehensions.
- Filesystem operations using pathlib.
- Unit testing, assertions, and systematic debugging.
- Repositories, version control, code reviews, and continuous integration.
- AI-assisted programming and independent validation of generated code.

Then revisit the expanded requirements for inheritance, MRO, object initialization, exception propagation, string searching, and safe file operations.

Practice reading, tracing, constructing, reviewing, and debugging intermediate Python programs. Also practice identifying development problems and selecting appropriate solutions.

Your goal is not simply to know more Python syntax. It is to understand how to build reliable, maintainable Python software and how to contribute effectively to modern development workflows.

Reference syllabi

- PCAP-31-03 examination syllabus: https://pythoninstitute.org/pcap-31-03-exam-syllabus
- PCAP-31-04 examination syllabus: https://pythoninstitute.org/pcap-exam-syllabus

Which exam version should you choose?

Both exam versions lead to the same PCAP certification. The right choice depends on your current preparation, available resources, and examination timeline.

YOUR SITUATION	OUR RECOMMENDATION
I'm beginning my associate-level Python studies.	Prepare for PCAP-31-04. The revised syllabus provides broader coverage of Python programming and introduces important modern software development practices.
I've already been studying for PCAP-31-03.	Continue building on your existing knowledge. If you are nearly ready and PCAP-31-03 remains available, you may choose to complete that version. Otherwise, use this guide to identify the additional topics needed for PCAP-31-04.
I've purchased a PCAP-31-03 voucher.	Confirm the voucher's exam version, expiration date, and redemption terms. Check that your intended exam remains available before scheduling.
I'm already PCAP certified.	Your existing PCAP certification remains valid according to its original terms, and retaking the examination is not required. However, we strongly recommend taking PCAP-31-04 to update and validate your skills against the expanded syllabus, strengthen your knowledge of modern Python development practices, and demonstrate your commitment to continuous learning and professional growth.

Our recommendation: If you are beginning your associate-level certification journey now, choose PCAP-31-04. If you have already invested substantial time in PCAP-31-03 preparation, assess your readiness and examination timeline before deciding whether to transition.