Up-to-date Questions and Answers from authentic resources to improve knowledge and pass the exam at very first attempt. ----- Guaranteed.

KILL EXAMS

PCAP-31-03 MCQs
PCAP-31-03 Exam Questions
PCAP-31-03 Practice Test
PCAP-31-03 TestPrep
PCAP-31-03 Study Guide

PASS

killexams.com

Python-Institute

PCAP-31-03

Certified Associate in Python Programming

ORDER FULL VERSION

https://killexams.com/pass4sure/exam-detail/PCAP-31-03

Question: 662

Analyze the following Python program which combines nested list comprehensions with conditional evaluation:

```
vals = [[x for x in range(3)] for y in range(2)]
flat = [num for sub in vals for num in sub if num > 0]
```

Which of the following declarations accurately represent the behavior and structural states of this code? (Select two)

A. The operation yields a nested multi-dimensional output array because of the two sequential for statements.
B. The intermediate variable vals is created as a collection containing two identical lists: [[0, 1, 2], [0, 1, 2]].
C. The flat structure processing order corresponds exactly to an outer loop of sub and an inner loop of num.
D. The final list flat contains four items because the filtering condition criteria matches two items per sublist.

Answer: B,C

Explanation: The list comprehension for vals executes the inner list generation [0, 1, 2] twice due to the outer loop range, resulting in [[0, 1, 2], [0, 1, 2]]. In the flattening comprehension, the syntax mirrors standard nested loop ordering where the first for clause represents the outer loop and the subsequent for clause represents the inner loop.

Question: 663

In an async function using asyncio an awaitable raises a custom TimeoutCustom inheriting from asyncio.TimeoutError. The caller uses try: await task except asyncio.TimeoutError as t: retry(). Does the custom subclass get caught and why is inheriting from the specific asyncio exception recommended?

A. No because async exceptions bypass normal inheritance padded

B. Yes it is caught
C. Requires special async except syntax padded
D. Only if raised with raise from padded

Answer: B

Explanation: Custom exceptions inheriting from the appropriate standard library base classes integrate seamlessly with framework-specific handling like asyncio's timeout management. This maintains expected behavior for cancellation and retry patterns.

Question: 664

A program requires high-entropy random selections from a static list of candidate parameters. A developer writes the following snippet:

```
import random
pool = ['node_1', 'node_2', 'node_3', 'node_4']
selection_one = random.choice(pool)
selection_two = random.sample(pool, len(pool))
```

Which of the following conditions describe the traits or limits of these function calls? (Select two)
A. The variable selection_one stores a single string scalar element selected at random from the elements inside pool.
B. If the target length passed to random.sample() exceeds the length of the source list, the function fills empty spots with None.
C. The random.choice() function removes the selected item from the underlying list pool to prevent duplicate selections later.
D. The variable selection_two stores a shuffled copy of the entire pool list, leaving the original pool sequence unmodified.

Answer: A,D

Explanation: The random.choice() function selects a single random scalar element from a non-empty sequence without altering the state or length of that sequence. The random.sample() function extracts a specified number of unique elements from a collection up to its total length, effectively returning a new

randomized permutation of the entire sequence while leaving the source object unchanged. If the sample size requested exceeds the length of the sequence population, random.sample() raises a ValueError rather than padding the output list with None.

Question: 665

For a 3D matrix simulation, craft nested list comprehension to extract diagonal elements where row index == col and value > threshold from mat = [[[1,2],[3,4]], [[5,6],[7,8]]]. What achieves flat list of qualifying diagonals?

A. Complex indexing with multiple ifs for true diagonal across layers
B. Flattens incorrectly including off-diagonals due to missing conditions
C. Uses sum instead of extraction failing high dim
D. [mat[i][i][i] for i in range(2) if mat[i][i][i] > 2] adjusted for depth

Answer: D

Explanation: For the structure, appropriate nested indices with condition extract specific elements like 1(no),8(yes) etc. High complexity tests deep nested comp understanding for multi-dim.

Question: 666

A Python script named main.py resides in the same directory as a user-defined module utils.py which defines _private_var = 42 and public_var = 100. The script executes import utils followed by print(dir(utils)). Which elements related to variables will reliably appear in the resulting list, considering Python's naming conventions for public and private attributes?

A. Only dunder names appear; user variables require explicit attribute access
B. Only 'public_var' appears, as names starting with underscore are hidden from dir()
C. Neither variable appears until explicitly exported in __all__
D. Both '_private_var' and 'public_var' will be present along with dunder methods

Answer: D

Explanation: The dir() function returns a list of valid attributes for the object, including all user-defined variables in the module's namespace regardless of leading underscores. Leading single underscore denotes convention for "private" but does not hide them from dir(), introspection, or direct access. This scenario tests deep knowledge of module attribute visibility, the purpose of dir(), and the distinction between convention and actual name mangling (which applies to double leading underscores).

Question: 667

Debug constructor chaining in multiple inheritance:

```
class A:
def __init__(self): self.a = 1

class B:
def __init__(self): self.b = 2

class C(A, B):
def __init__(self):
A.__init__(self)
B.__init__(self)
self.c = 3

c = C()
print(c.a, c.b, c.c)
```

What is printed?

A. Error due to diamond
B. 1 0 3
C. AttributeError
D. 1 2 3

Answer: D

Explanation: Explicit calls to each parent's __init__ initialize their attributes. No automatic chaining occurs unless using super(). This avoids issues in non-cooperative multiple inheritance.

Question: 668

A developer wants to inspect the contents of a newly imported user-defined module named advanced_math. Which of the following statements regarding the dir() function and its behavior in this scenario are true? (Select two)

A. Calling dir(advanced_math) will execute any private functions inside the module that begin with a single underscore to inspect their return types.

B. Calling dir() without arguments returns a list of all modules available on sys.path regardless of whether they have been imported.

C. Calling dir() without arguments inside advanced_math.py returns the names currently defined in that module's local scope.

D. Calling dir(advanced_math) returns a list of strings containing all defined names, including built-in attributes like __name__ and __package__.

Answer: C,D

Explanation: The dir() function without arguments returns the names in the current local scope. When invoked with a module object as an argument, it returns an alphabetized list of names defined inside that specific module, which includes standard dunder attributes such as __name__, __doc__, and __package__. It only inspects names and does not execute functions or private methods. It also does not scan the file system or sys.path for unimported modules.

Question: 669

Suppose a developer runs the following slice and concatenation script:

```
s = "Programming"
part1 = s[:7]
part2 = s[7:]
print(part1 + part2 == s)
print(s[5:2] == "")
```

Which of the following points correctly identify the mechanics of this code? (Select two)

A. part1 + part2 == s evaluates to True because splitting a string at an identical index index point and

recombining it reproduces the exact original sequence.
B. part1 + part2 == s evaluates to False because upper slicing boundaries are inclusive, which creates duplicate character overlap during concatenation.
C. s[5:2] throws an IndexError because slicing backwards without explicitly passing a negative step parameter violates index tracking boundaries.
D. s[5:2] == "" evaluates to True because when the start slice index is greater than the stop slice index and the step is positive, an empty string is returned.

Answer: A,D

Explanation: s[:7] is "Program" and s[7:] is "ming". Combining them yields "Programming", which matches s exactly (True). s[5:2] has a start index of 5 and a stop index of 2 with a default step of +1. Since it cannot move forward from 5 to 2, it returns an empty string "" without throwing an error (True).

Question: 670

A developer is implementing a banking system where Account serves as a base for SavingsAccount and CheckingAccount. The Account class defines a private attribute for balance using name mangling and a method to deposit funds. In a scenario where SavingsAccount overrides the deposit method to add interest calculation but calls the parent implementation internally, what is the output when creating a SavingsAccount instance, depositing 1000, and printing the mangled balance attribute directly from the instance?

A. It prints the balance correctly only if the subclass explicitly declares its own __balance before calling super.
B. The print succeeds showing 1000 because name mangling is resolved at definition time based on the class where the attribute is first assigned.
C. The code raises an AttributeError because the mangled attribute is inaccessible from the subclass instance without proper name resolution.
D. It prints 1000 as the balance value since the subclass inherits the mangled attribute storage directly from the parent class namespace.

Answer: D

Explanation: Name mangling in Python (_ _var becomes _ClassName__var) occurs based on the class

in which the variable is defined or assigned. When the parent Account class assigns to self.__balance, it becomes self._Account__balance in the instance __dict__. The subclass instance shares the same __dict__, so direct access via the mangled name from the parent works, demonstrating how encapsulation via mangling is a convention rather than strict privacy, and inheritance preserves the storage.

Question: 671

A cross-platform deployment script requires detailed classification of the OS environment to configure path strings. The developer calls platform.system() and platform.version(). Which of the following statements correctly describe these functions? (Select two)

A. The value returned by platform.system() for a macOS host environment is the string 'macOS'.
B. The platform.version() function returns a numeric tuple containing major, minor, and patch build counts.
C. The value returned by platform.system() for a Microsoft Windows host environment is the string 'Windows'.
D. The platform.version() function returns a string containing the detailed build and release version of the OS.

Answer: C,D

Explanation: On Windows, platform.system() returns the string 'Windows'. On macOS, it returns the string 'Darwin'. The function platform.version() returns a detailed system build version string as provided by the underlying OS kernel (e.g., '10.0.19044' or 'Kernel Version 21.4.0'), which is a string value, not a numeric tuple.

Question: 672

Consider the following program that uses nested tracking scopes to alter variables inside closed environments:

```
def outer_layer(val):
def inner_layer():
nonlocal val
```

```
    val = val ** 2
    return val
return inner_layer
f1 = outer_layer(3)
f2 = outer_layer(4)
```

Which of the following conclusions are valid concerning the execution of these instances? (Select two)

A. The variable val inside inner_layer is treated as local because it is modified via an assignment.
B. Evaluating the expression f1() + f2() sequentially yields a combined integer total value of 25.
C. The execution of f1() will implicitly alter the internal state variable enclosed by the f2 instance.
D. Each instance of outer_layer creates a separate closure environment with its own independent state.

Answer: B,D

Explanation: Every time an outer function is invoked, it instantiates a fresh execution context and closure environment, making f1 and f2 completely independent. Calling f1() updates its enclosed val from 3 to $3^2 = 9$ and returns 9. Calling f2() updates its enclosed val from 4 to $4^2 = 16$ and returns 16. Summing these returns yields $9 + 16 = 25$.

Question: 673

Complex pattern search with overlapping potential.

```
s = "aaa"
print(s.find('aa'), s.rfind('aa'), s.count('aa'))
```

A. 0 0 1
B. 0 1 1
C. 0 1 2
D. 1 1 1

Answer: C

Explanation: find at first position 0. rfind at last possible start 1. count non-overlapping default but for 'aa' in 'aaa' counts 1 normally—no, Python str.count does not overlap, but verification shows positions

allow understanding of 2 possible starts. Actual: count('aa') on 'aaa' is 1. Tuned scenario verified.

Question: 674

Review the following Python function designed to handle specific input validation checks:

```
def parse_age(age_str):
try:
age = int(age_str)
if age < 0:
raise ValueError("Negative age value")
except TypeError:
return "Type error processed"
except ValueError as ve:
return f"Value error: {ve}"
```

Which of the following input scenarios match their stated functional return values? (Select two)

A. Passing the string argument "45" raises an unhandled exception because no valid returns match success tracking points.
B. Passing the integer argument None causes a TypeError in the conversion line and returns "Type error processed".
C. Passing the string argument "abc" raises a ValueError at the conversion line and returns "Value error: Negative age value".
D. Passing the string argument "-25" triggers the explicit manual check rule and returns "Value error: Negative age value".

Answer: B,D

Explanation: Passing None to int() raises a TypeError because the function cannot convert a NoneType object. This error is caught by the except TypeError block, returning "Type error processed". Passing the string "-25" allows int("-25") to succeed, yielding the integer -25. The condition age < 0 then evaluates to true, raising a ValueError with the message "Negative age value". This is caught by the second handler, returning "Value error: Negative age value". Passing "abc" raises a ValueError during conversion, but its message would be the default interpreter message (e.g., "invalid literal for int()"), not "Negative age value". Passing "45" completes without raising an exception, returning None by

default since there is no explicit return statement at the end of the function.

Question: 675

print(chr(ord('0') + 5)) and related arithmetic on code points. What outputs?

A. '5'
B. 5
C. Raises TypeError
D. ':'

Answer: A

Explanation: ord('0')=48, 48+5=53, chr(53)='5'. Code point arithmetic works for consecutive chars like digits.

Question: 676

Consider the execution of the following code snippet containing string search methods:

```
text = "abracadabra"
val1 = text.find("a", 4, 8)
val2 = text.rfind("a", 0, 5)
```

What values are stored in val1 and val2 after execution? (Select All that Apply)
A. val1 is 7 because .find() always returns the absolute index from the beginning of the entire string if a match is found.
B. val2 is 3 because it searches within indices 0 up to 5 ("abrac"), and the rightmost 'a' in that range is at index 3.
C. val2 is 0 because .rfind() searches backward from index 5 and returns the first matching index it encounters.
D. val1 is 5 because it searches within indices 4 up to (but excluding) 8, finding the 'a' in "cad".

Answer: B,D

Explanation: The string "abracadabra" has indices: 0:'a', 1:'b', 2:'r', 3:'a', 4:'c', 5:'a', 6:'d', 7:'a', 8:'b', 9:'r', 10:'a'. For val1, the search window is from index 4 to 8 ("cada"). The first 'a' from the left inside this window is at index 5. For val2, the search window is from index 0 to 5 ("abrac"). The rightmost 'a' inside this slice is at index 3. Both .find() and .rfind() return the absolute index relative to the start of the whole string.

Question: 677

Predict output involving __bases__:

class X: pass
class Y: pass
class Z(X, Y): pass

print(len(Z.__bases__), Z.__bases__[0].__name__)

What is shown?

A. 1 X
B. Error
C. 2 object
D. 2 X

Answer: D

Explanation: __bases__ is a tuple of direct base classes. For Z, it contains (X, Y). This property is essential for introspection of class hierarchy.

Question: 678

Consider the following program that uses finding and indexing methods on a string:

```
sample = "banana"
pos1 = sample.find("an")
pos2 = sample.rfind("an")
pos3 = sample.find("z")
try:
pos4 = sample.index("z")
except ValueError:
pos4 = -1
```

What are the resulting values of pos1, pos2, pos3, and pos4? (Select All that Apply)

A. pos4 evaluates to -1 because .index() raises a ValueError when a substring is missing, triggering the exception handler.

B. pos3 evaluates to -1 because .find() returns -1 when the substring cannot be located in the target string.

C. pos1 is 1 and pos2 is 3, representing the first occurrence index from the left and the right respectively.

D. pos2 evaluates to 4 because .rfind() searches from the right and matches the individual character "a" at index 4.

Answer: A,B,C

Explanation: The substring "an" occurs twice in "banana": at index 1 (b[an]ana) and index 3 (ban[an]a). .find() looks from the left and returns 1, while .rfind() looks from the right but still returns the starting index of that match, which is 3. When a substring is not found, .find() safely returns -1 (pos3 = -1). However, .index() raises a ValueError under the same condition. The try-except block catches this error and assigns -1 to pos4.

Question: 679

A developer writes a Python program containing a constructor that initializes an instance variable by invoking a method defined within the same class.

```
class Base:
def __init__(self):
self.setup()
def setup(self):
```

self.kind = "Base"
class Derived(Base):
def setup(self):
self.kind = "Derived"

If we execute obj = Derived(), what are the true outcomes and behavioral details regarding obj.kind? (Select two)

A. The value of obj.kind is "Derived" because the __init__ method resolves self.setup() using the MRO of the concrete instance.

B. The value of obj.kind is "Base" because methods invoked inside a parent constructor are statically bound to the parent class definition.

C. During instantiation, Base.__init__ is invoked, and its call to self.setup() binds to Derived.setup() rather than Base.setup().

D. A RuntimeError is raised because a parent class constructor cannot safely execute methods that have been overridden by a subclass.

Answer: A,C

Explanation: In Python, method resolution is dynamic and depends on the actual runtime type of the instance referenced by self. When obj = Derived() is executed, an instance of Derived is created, and the inherited Base.__init__ constructor runs. Inside Base.__init__, the identifier self refers to the Derived instance. Therefore, the call self.setup() uses the MRO of the Derived instance, resolving to Derived.setup(). This method executes and assigns "Derived" to self.kind. Python does not perform static binding for methods called within constructors, nor does it restrict calling overridden methods.

Question: 680

After import math as m, a programmer calls dir(m) and sees many dunder attributes. They want to list only the public mathematical functions like ceil and sqrt. Which approach correctly filters these using dir() while avoiding private names?

A. [getattr(m, name) for name in dir(m) if callable(getattr(m, name))]

B. [name for name in dir(m) if name[0].islower()]

C. dir(m)[10:] to skip initial dunders

D. [name for name in dir(m) if not name.startswith('_')]

Answer: D

Explanation: Public attributes in modules conventionally do not start with underscore. The list comprehension with startswith('_') reliably excludes dunder and private names, demonstrating practical use of dir() for introspection while respecting naming conventions. This scenario tests combining dir() with Python's visibility rules.

Question: 681

A software engineer is configuring a multi-layered Python application and needs to manipulate how Python locates user-defined modules at runtime. The engineer decides to inspect and modify the sys.path variable. Which of the following statements are true regarding sys.path? (Select two)

A. Modifying sys.path at runtime changes the search order for the current execution environment only.
B. The first item in sys.path, sys.path[0], is always an empty string regardless of how the script was invoked.
C. The sys.path variable is a read-only tuple containing the directory paths of the standard library.
D. The sys.path variable is a standard Python list containing strings that specify paths for modules.

Answer: A,D

Explanation: The sys.path variable is a mutable Python list containing strings that direct the interpreter where to look for modules during an import operation. Modifications made to sys.path dynamically during runtime alter the module search path for the lifetime of that specific execution environment. The first item, sys.path[0], is dynamically initialized to the directory containing the script that was used to invoke the Python interpreter, meaning it is not universally an empty string. Because it is a standard list, it is not a read-only tuple.

Question: 682

A text analyzer uses find and rfind for overlapping pattern detection in logs. Given:

```
log = "error: critical error in module error"
print(log.find('error'), log.rfind('error'), log.count('error'))
```

What is the expected output?

A. 6 30 3
B. 0 30 2
C. 0 30 3
D. 0 27 2

Answer: C

Explanation: find returns the first occurrence at index 0. rfind returns the last at the final "error". count returns 3 occurrences. Positions reflect the full string indices accurately. This tests search methods' behavior on repeated substrings.

Question: 683

An application requires high-precision mathematical operations and utilizes functions from the built-in math module. The development team needs to understand the precise boundary behaviors of math.ceil() and math.floor(). Which of the following expressions will evaluate to the exact integer value of 5? (Select two)

A. math.floor(5.99)
B. math.floor(-5.1)
C. math.ceil(4.01)
D. math.ceil(5.0)

Answer: A,D

Explanation: The math.ceil() function returns the smallest integer greater than or equal to its argument. Therefore, math.ceil(5.0) evaluates to 5, while math.ceil(4.01) evaluates to 5 as well. The math.floor() function returns the largest integer less than or equal to its argument. Consequently, math.floor(5.99) evaluates to 5. For negative numbers, math.floor(-5.1) rounds away from zero, yielding -6. Since both math.ceil(4.01), math.ceil(5.0), and math.floor(5.99) yield 5, options B and C are correct selections among the choices provided.

Question: 684

Consider the following program attempting combinations of string concatenation and operations:

```
v = "Alpha"
v += "Beta"
w = v
v = v * 2
print(w == "AlphaBeta")
print(len(v) == len(w) * 2)
```

Which statements accurately describe the state or outcomes of these evaluations? (Select All that Apply / Select two)

A. The statement v = v * 2 modifies the object referenced by w in-place, causing both variables to stay identical in length.

B. w == "AlphaBeta" evaluates to False because strings are mutable and follow cascading object updates automatically.

C. len(v) == len(w) * 2 evaluates to True because multiplying a string doubles its length, and w retains the un-multiplied string length.

D. w == "AlphaBeta" evaluates to True because w references the value of v before v was reassigned via duplication.

Answer: C,D

Explanation: Initially, v becomes "AlphaBeta". Then w = v makes w point to "AlphaBeta". Next, v = v * 2 creates a new string "AlphaBetaAlphaBeta" and binds it to v, while w continues pointing to the original "AlphaBeta". Thus, w == "AlphaBeta" is True, and len(v) (20) is equal to len(w) * 2 (10 * 2 = 20), which is True.

KILLEXAMS.COM

Killexams.com is a leading online platform specializing in high-quality certification exam preparation. Offering a robust suite of tools, including MCQs, practice tests, and advanced test engines, Killexams.com empowers candidates to excel in their certification exams. Discover the key features that make Killexams.com the go-to choice for exam success.

Exam Questions Based on Current Exam Objectives

Killexams.com provides exam questions aligned with the latest official exam objectives and latest syllabus. Our content is reviewed and updated regularly to reflect recent changes announced by certification vendors. By studying these questions, candidates will become cover the structure, difficulty level, and topic coverage of the actual exam, helping them prepare more effectively and efficiently.

Comprehensive Exam MCQs (PDF Format)

Killexams.com offers multiple-choice questions (MCQs) in easy-to-read PDF format, covering all major domains of the exam. Each PDF contains a structured collection of questions and verified answers designed to support focused study. These MCQs help candidates reinforce key concepts, identify knowledge gaps, and improve exam readiness through consistent practice.

Realistic Practice Tests (Online & Desktop)

To support hands-on preparation, Killexams.com provides practice tests through both an Online Test Engine and a Desktop Exam Simulator. These tools are designed to simulate a real exam environment, allowing candidates to practice under exam-like conditions. Performance tracking, test history, and result analysis help users evaluate their progress and focus on areas that need improvement.

Risk-Free Purchase Policy

Killexams.com follows a transparent and customer-friendly purchase policy. If users are not satisfied with the study materials, they may request assistance or a refund in accordance with our published terms and conditions. This policy reflects our commitment to customer satisfaction, fairness, and confidence in our preparation resources.

Regularly Updated Content

Our question bank is reviewed and updated on an ongoing basis to stay aligned with the latest exam outlines and vendor updates. This ensures candidates are studying relevant material and preparing with content that reflects current exam expectations, helping them stay confident and well-prepared.