DUMPSARENA

GIAC Python Coder (GPYC)

GIAC GPYC

Version Demo

Total Demo Questions: 13

Total Premium Questions: 138

Buy Premium PDF

https://dumpsarena.co

sales@dumpsarena.co

sales@dumpsarena.co
dumpsarena.co

QUESTION NO: 1

A user enters unexpected data into program. Which functionality can the programmer use to present an understandable error message to the user?

A. Casting

B. Exception handling

C. Dictionaries

D. Regular expressions

ANSWER: B

Explanation:

Exception handling is the appropriate functionality for responding to unexpected user input with an understandable message. In Python, code that may fail because input cannot be converted, parsed, opened, or otherwise processed is placed in a `try` block. A corresponding `except` block catches the anticipated exception and can display a clear, user-focused message, such as “Please enter a whole number,” rather than exposing a traceback or allowing the application to terminate unexpectedly.

For example, converting input with `int(input())` can raise `ValueError` when the entered value is not a valid integer. Catching that specific exception lets the program communicate the issue clearly and, where appropriate, prompt the user again. This approach separates normal program logic from error-recovery logic and supports a more reliable, usable application. Python’s documentation recommends handling specific exceptions when possible, because the program can then respond appropriately to conditions it expects may occur. See the Python tutorial on errors and exceptions and the built-in exceptions reference.

QUESTION NO: 2

Which of the following statements about the `with open()` construct in Python are correct? (Choose two)

A. It closes the file when the block is exited.

B. It can close the file when an exception occurs in the block.

C. It reads the entire file automatically when the block begins.

D. It makes the file object available to every Python module.

E. It prevents the file from being opened in write mode.

ANSWER: A B

Explanation:

The `with` statement is used with a context manager. A file object returned by `open()` is a context manager, so using `with open("report.txt") as report:` ensures that the file is closed when execution leaves the block, including when an exception occurs inside the block.

The variable following `as` refers to the opened file object and can be used to call methods such as `read()`, `write()`, and `readline()`. See the Python tutorial on reading and writing files and the with statement reference.

QUESTION NO: 3

An operator able to perform bitwise shifts is coded as (select two answers)

A. - -

B. ++

C. <<

D. > >

ANSWER: C D

Explanation:

Python performs bitwise shifting with the `<<` and `>>` operators. The `<<` operator is the left-shift operator: it moves the bits in an integer value to the left by the specified number of positions. For non-negative integers, shifting left by `n` positions has the same numerical effect as multiplying by `2**n`. For example, `5 << 1` produces `10`, because the binary representation of 5 is shifted one place to the left. The `>>` operator is the right-shift operator: it moves an integer's bits to the right by the specified number of positions. For non-negative integers, shifting right by `n` positions corresponds to floor division by `2**n`; for example, `20 >> 2` produces `5`. Python defines these operations for integers, including its arbitrary-precision `int` type, and documents their behavior in the expressions reference. See the Python shifting-operations reference and the Python integer bitwise-operations documentation.

QUESTION NO: 4

During a password guessing attack, which HTTP request method would a Python program most commonly call to submit a usemame and password to a target website?

A. OPTIONS

B. POST

C. CONNECT

ANSWER: B

Explanation:

POST is the HTTP method most commonly used to submit a username and password to a website's login endpoint. In a typical form-based authentication workflow, the browser—or an authorized Python testing program—places credentials in the request body and sends them to the application using POST. Python HTTP clients such as `requests` support this pattern through `requests.post()`, including form-encoded data via the `data` argument or JSON credentials via the `json` argument. A security tester should only automate authentication attempts against systems for which they have explicit authorization, and should respect the target's defined testing scope and rate limits. The HTTP semantics specification defines POST as a method for requesting that a server process the content enclosed in the request, which matches login form submission. The Python Requests documentation provides the corresponding `post()` interface used by Python programs to send these requests. See MDN Web Docs: POST and Requests API: requests.post.

QUESTION NO: 5

What is the output of the following when executed in a Python shell?

```
>>> listoflists = [[1,2],[3,4]]
>>> copyoflists = list(listoflists)
>>> copyoflists.append([5,6])
>>> copyoflists
[[1, 2], [3, 4], [5, 6]]
>>> copyoflists[0].append(2.5)
>>> copyoflists
[[1, 2, 2.5], [3, 4], [5, 6]]
>>> listoflists
```

A. [[1, 2, 2.5], [3, 4], [5, 6]]

B. [[1, 2], [3, 4], [5, 6]]

C. [[1, 2], [3, 4]]

D. [[1, 2, 2.5], [3, 4]]

ANSWER: D

Explanation:

The output is `[[1, 2, 2.5], [3, 4]]`. The two variables initially refer to the same outer list, so modifying the first nested list through either variable changes the shared object. In particular, appending `2.5` to the first inner list mutates that inner list in place. Since both names still lead to that same nested list, the mutation is visible when the original outer-list variable is displayed.

The later operation that forms a value including `[5, 6]` creates and assigns a new outer list to the variable on the left of that assignment rather than modifying the original outer list in place. It does not undo the earlier in-place append, and it does not add the new final inner list to the original object. Therefore, the original list retains its two inner lists, with the first one expanded to include `2.5`. Python's assignment model binds names to objects, while methods such as `list.append()` modify a list directly; see the Python list documentation and the Python assignment statement reference.

QUESTION NO: 6

Which of the following will be the value of the variable y?

```
>>> x = {'a': [1, 2, [3,4]], 'b': [[5, 6], 7]}
>>> y = x ['b'] [0] [1]
```

A. 7

B. y has no value. The following error occurred: IndexError: list index out of range

C. 6

D. y has no value. The following error occurred: KeyError: 'b'

ANSWER: C

Explanation:

The value of `y` is `6`. The expression shown accesses an existing element using Python's zero-based indexing rules: the first item in a sequence has index `0`, so the item at index `1` is the second item. In the sequence used by the code, that second value is `6`, and assignment stores the retrieved value in `y`. Python evaluates the lookup before completing the assignment; because the referenced container and index are valid in the displayed code, evaluation succeeds and `y` receives the resulting integer. This behavior follows the standard sequence subscription model described in the Python language reference: an integer index selects an item from a sequence, with indices beginning at zero. The Python tutorial also demonstrates this convention with list indexing and shows that a list expression can be used to retrieve a specific element directly. See the Python language reference on subscriptions and the Python tutorial's list documentation.

QUESTION NO: 7

What is the output of the following command typed in Python interactive mode?

>>> print("and" in "And now for something completely different")

A. SyntaxError: invalid syntax

B. True

C. False

ANSWER: A

Explanation:

The correct result is `SyntaxError: invalid syntax` because the expression as written uses typographic "smart quotes" (" and ") around the text. Python string literals must be delimited with valid Python quote characters: the ASCII single quote (`'`), double quote (`"`), or their triple-quoted forms. The curly quotation marks in the command are not recognized as string delimiters by the Python parser, so Python cannot parse the expression and raises a syntax error before it can evaluate the `in` membership test or call `print()`. In a typical current Python interpreter, the diagnostic may be more specific, such as reporting an invalid character " (U+201C), but it is still a `SyntaxError`. This distinction matters when code has been copied from word processors, formatted web pages, or chat applications, which may automatically replace straight quotes with typographic quotes. Python's lexical specification defines string literals using quote characters such as `'` and `"`; see the Python lexical analysis documentation. The Python documentation for SyntaxError further explains that this exception is raised when the parser encounters invalid syntax.

QUESTION NO: 8

Which of the following statements about Python's `zipfile` module are correct? (Choose two)

A. `ZipFile.namelist()` returns the names of archive members.

B. `zipfile` can be used to create ZIP archives.

C. `zipfile` can read only uncompressed text files.

D. Extracting an archive automatically prevents unsafe member paths.

ANSWER: A B

Explanation:

The `zipfile` module reads and writes ZIP archive files. A `ZipFile` object can list archive members with `namelist()` and can extract archive contents with methods such as `extract()` and `extractall()`. It can also add files when the archive is opened in an appropriate write or append mode.

Programs should validate archive member names and extraction destinations when handling untrusted ZIP files. Archive entries can contain path components that attempt to write outside the intended extraction directory. See the Python zipfile documentation.

QUESTION NO: 9

Given that mylist = [1, 3, 2, 1, 4, 5, 3] how do you remove all occurrences of the number 1 from the list in Python?

A. mylist=[x for x in mylist if x!=1]

B. mylist.remove(1, mylist.count(1))

C. mylist.replace(1,“”)

D. mylist.remove(1)

ANSWER: A

Explanation:

`mylist=[x for x in mylist if x!=1]` is correct because a list comprehension constructs a new list by evaluating every element in `mylist` and retaining only those for which the condition `x != 1` is true. Starting with `[1, 3, 2, 1, 4, 5, 3]`, it excludes both instances of `1` and assigns the resulting list, `[3, 2, 4, 5, 3]`, back to `mylist`. This approach is reliable when every matching value must be removed, including repeated values at arbitrary positions. It also clearly expresses the intended filtering operation in one statement: iterate through the existing list and keep values other than `1`. Python’s documentation describes list comprehensions as a concise way to create lists by applying an expression to items from an iterable, optionally subject to a condition. The condition here provides exactly the required selection criterion. See the Python list-comprehensions tutorial and the Python language reference for comprehensions.

QUESTION NO: 10

What would be the result of the following code in Python?

```
registry_key = reg_handle.open("Microsoft\Windows NT\CurrentVersion")

map(lambda x: x.path(), registry_key.subkeys())
```

A. The path of each subkey under Microsoft\Windows NT\CurrentVersion

B. A tuple containing the name, value, and type of each subkey under Microsoft\Windows NT \CurrentVersion

C. The full registry path of the CurrentVersion key

D. A handle to the Microsoft\Windows NT\CurrentVersion

E. A tuple containing the name, data, and type of each value under Microsoft\Windows NT\CurrentVersion

ANSWER: E

Explanation:

The code enumerates registry values in the `Microsoft\Windows NT\CurrentVersion` registry key. For every index passed to Python’s `winreg.EnumValue()` function, the function returns a three-item tuple in the form `(value_name, value_data, value_type)`. The first item identifies the registry value’s name, the second contains its stored data, and the third is the Windows registry type constant describing that data, such as `REG_SZ` or `REG_DWORD`. Therefore, the output produced during enumeration is a tuple containing the name, data, and type for each value held directly by that key. This is the standard result contract of `EnumValue`; callers commonly obtain the number of values first with `QueryInfoKey()` and then enumerate indices from zero through that count. Python’s official documentation specifies both the tuple structure and

the behavior of registry-value enumeration. See the Python winreg.EnumValue documentation and the related winreg.QueryInfoKey documentation.

QUESTION NO: 11

Which of the following statements about the `csv` module are correct? (Choose two)

A. `csv.reader()` can iterate over rows from a CSV file.

B. CSV files must use commas as delimiters.

C. `newline=''` is recommended when opening a CSV file for use with the module.

D. `csv.writer()` can write only dictionaries.

ANSWER: A C

Explanation:

Python's `csv` module provides reader and writer objects for processing CSV data. `csv.reader()` reads rows from a CSV source, while `csv.writer()` writes rows to a CSV destination. The module handles common CSV concerns such as delimiters and quoted fields.

When opening a CSV file for use with the module, Python recommends specifying `newline=''`. This allows the `csv` module to handle newline processing correctly. See the Python csv module documentation.

QUESTION NO: 12

Which of the following statements about a TCP server socket created with Python's `socket` module are correct? (Choose two)

A. bind() associates the server socket with a local address and port.

B. listen() enables a stream socket to accept incoming connections.

C. accept() returns only the bytes sent by the client.

D. sendto() must be called before a TCP server can accept a connection.

E. recvfrom() is required for every TCP server connection.

ANSWER: A B

Explanation:

A TCP server normally calls `bind()` to associate the socket with a local address and port, then calls `listen()` to enable the socket to accept incoming connections. Afterward, `accept()` waits for and returns a new connected socket together with the client address.

The original listening socket remains available to accept additional connections. Data for an accepted TCP connection is exchanged through the socket object returned by `accept()`, using methods such as `recv()` and `sendall()`. See the Python socket.listen() documentation and the socket.accept() documentation.

QUESTION NO: 13

A script uses the `json` module to process data received from a web service. Which of the following statements are correct? (Choose two)

A. json.loads() converts JSON text into Python objects.

B. json.dumps() converts a Python object into a JSON string.

C. json.loads() writes JSON directly to a file object.

D. json.dumps() can only serialize Python strings.

E. JSON objects are converted to Python tuples by default.

ANSWER: A B

Explanation:

`json.loads()` deserializes a JSON-formatted string, bytes object, or byte array into corresponding Python objects. For example, a JSON object normally becomes a Python dictionary and a JSON array becomes a Python list.

`json.dumps()` serializes a Python object to a JSON-formatted string. This is useful when preparing structured data for storage or transmission. The related `json.load()` and `json.dump()` functions operate on file-like objects rather than directly on a JSON string. See the Python json module documentation.