Answers

Question 1: The answers are A and D

#A

The imported module is named m, and it is located in a package named p. As the from phrase is in use, invoking the f() function doesn't require the use of a qualified name.

#D

The imported module is named m, and it is located in a package named p. Invoking the f() function requires the use of a qualified name, e.g, p.m.f()

#B

The names listed after the import instruction cannot specify a function name (in this case, the f() function).

#C

A package (in this case, the p package) cannot be imported

Question 2: The answers are A and C

#A

The phrase import a.b determines the fact that b is a module name, hence the above statement is true

#B

No, you can't do that - the function name, in this context, has to be a qualified name.

#C

The phrase import a.b determines the fact that a is either a package or a module.

#D

Omitting a makes the invocation incorrect - a qualified name must be used in this case.

Question 3: The answer is B

dir() is obviously a function; when invoked with an argument as a module, it returns a list containing the module attributes' names.

Question 4: The answer is C

While sys is a module containing facilities which allow the user to diagnose certain features of the OS environment, path is a list of strings that specifies the search path for modules. It is initialized upon program start-up, and path[0] shows the name of the directory containing the script that has been used to invoke the Python interpreter.

Question 5: The answers are C and D

#A

This function comes from the platform module, and it returns the real processor name, e.g., "Intel (R) Core (TM) i5-7200U CPU @ 2.50GHz."

#B

This function comes from the random module, and it initializes the random value generator.

#C

The function evaluates the square root of the argument.

#D

The function finds the length of the hypotenuse.

Question 6: The answers are B and D

#A

The module provides a set of objects used to build a GUI environment.

#B

The module contains the following functions: machine() (it returns s string which describes the machine type, e.g., "x86_64") and system() (it returns a string which provides the OS name, e.g., "Linux")

#C

The module contains mathematical functions.

#D

It contains the choice() function (it returns a random element chosen from a sequence)

Question 7: The answer is A

Let's analyze the example step by step:

1. math.floor(-1.5) − > -2
2. math.ceil(-1.5) − > -1
3. (-2) + (-1) − > -3
4. abs(-3) − > 3

Question 8: The answer is C

Initializing a number generator with random.seed(0) twice will set it to exactly the same state, hence the values returned by two subsequent random.choice() invocations will be the sam. This means that the result of the subtraction operation between them must be equal to zero.

Question 9: The answers are B and D

#A

Incorrect - the folder is created and filled with files automatically.

#B

The variable can be used to determine whether the code has been run standalone (when the variable contains '__main__' or whether it has been imported as a module (when it contains the module name)).

#C

Incorrect - the semi-compiled version-dependent Python byte-code is stored in files marked with the .pyc extension.

#D

The presence of the file informs Python that the directory content should be treated as a package. Moreover, if the file contains valid Python code, it can be used to initialize the package.

Question 10: The answers are A and D

#A

The semi-compiled code is stored in .pyc extension.

#B

Incorrect - Python manages it itself and needs no help with it.

#C

Incorrect - Python creates it itself, don't do it for it..

#D

Python creates it itself during package compilation.

Question 11: The answer is B

#A

A very weird idea.

#B

Correct. Moreover, it is searched through first - you can rely on this.

#C

Obviously not true.

#D

Life would be very complicated if it were true.

Question 12: The answer is C

If you want Python to treat your variable as private, you should start its name with two underscores (double underscores). This is the only variable name here that meets this requirement.

Answers

Question 1: The answers are A and C

#A

Indexing a list may raise the IndexError exception when an index value exceeds the list boundaries.

#B

Slicing a string is safe as illegal indices don't raise exceptions (such operations result in an empty string being created instead).

#C

Invoking the int() function may raise the ValueError exception when the argument doesn't represent a valid integer number.

#D

Incrementing integer values is always safe.

Question 2: The answers are B and C

#A

Exceptions caused by reckless dictionary lookups are recognized as KeyError.

#B

KeyError may be raised when you look up a non-existent within a dictionary.

#C

AssertionError may be raised when the assert instruction is executed, and its condition evaluates to False.

#D

Fortunately, there is no such an exception.

Question 3: The answer is A

Dividing by 0.0 raises the ZeroDivisionError exception, so the control falls into the first except branch, which sets the z variable to -1, resulting in the else branch being bypassed. In effect, -1 is outputted to the screen.

Question 4: The answer is C

Dividing by 3.0 doesn't raise any exception, so the control falls into the else branch, which sets the z variable to -2. In effect, -2 is outputted to the screen.

Question 5: The answer is C

Let's analyze the example:

1. The fun(0) invocation raises the ZeroDivisionError exception;

2. The exception is caught by the except: branch, and the AssertionError is raised;

3. As the AssertError is a subclass of Error, it is handled by the except Exception: branch;

4. In effect the x variable is set to -1, and this is the value that will be printed.

Question 6: The answer is D

This is how the control flows through the code:

1. mid_level() is invoked with the argument set to -1

2. Assertion fails and the AssertionError exception is raised;

3. The exception is handled by the raise instruction is executed and the AssertionError is raised again;

4. Because AssertionError is not a subclass of RuntimeError, the control falls into the default except: branch, which sets x to - 2;

5. Finally, -2 is printed to the screen.

Question 7: The answer is B

Trying to access a non-existent element of a tuple raises the IndexError exception, which is a subclass of Exception. That is why the control falls into the only "named" except branch. The exception's args tuple will carry the following text: 'tuple index out of range'.

Question 8: The answer is A

Although 314e-2 is close to 3.141592, these two values are different, so the .index() method fails, raising the ValueError exception. The exception's args tuple will carry the following text: '3.14 is not in list' (which is evidently true).

Question 9: The answers are C and D

#A

There is no line in the code which can produce such output.

#B

This message cannot be printed because print(“success”) is located inside the else branch which is bypassed.

#C

This message will appear on the screen as a result of the print(accident) function invocation. As the Accident class has its __str__() function overridden in order to return the string “problem: every time, printing this class’s object will show this text on the screen.

#D

This message is printed as there are no obstacles to executing the print() invocation.

Question 10: The answers are A and D

#A

Because Failure is derived from IndexError, exceptions of this kind will fall into the except IndexError branch, and that is why this line appears on the screen.

#B

The __str__() method is not utilized in this code, hence there is no chance to see this message printed.

#C

There is no line in the code which will print this text on the screen.

#D

There is nothing in the code which can prevent this message from appearing on the screen.