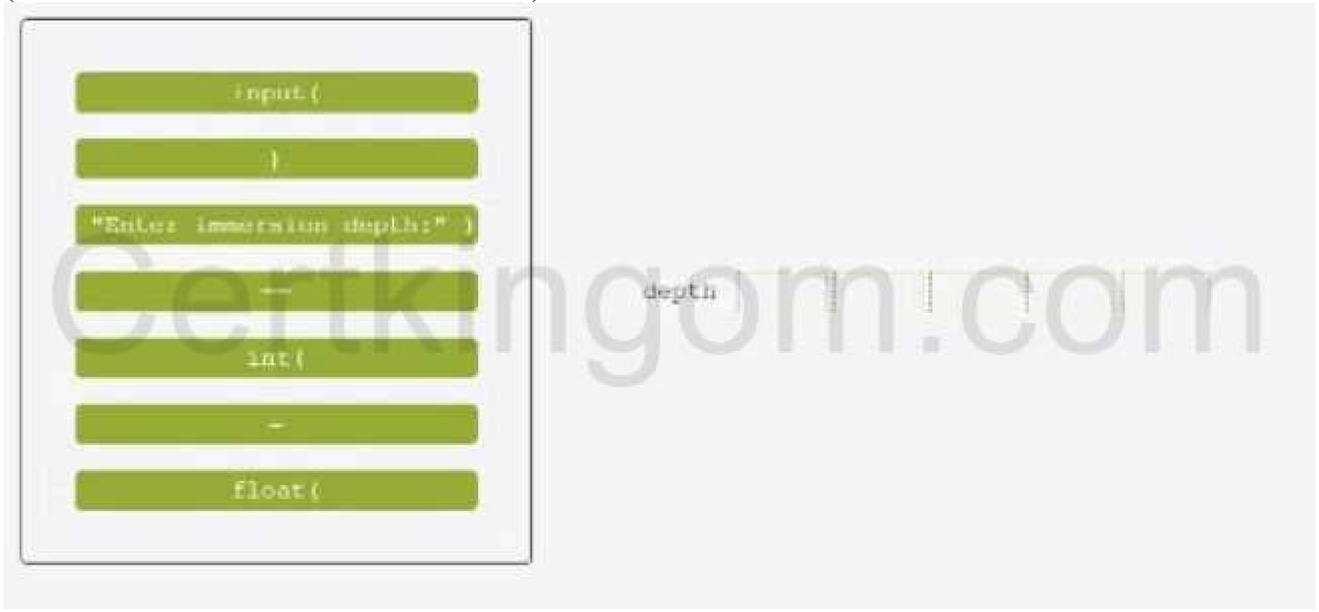
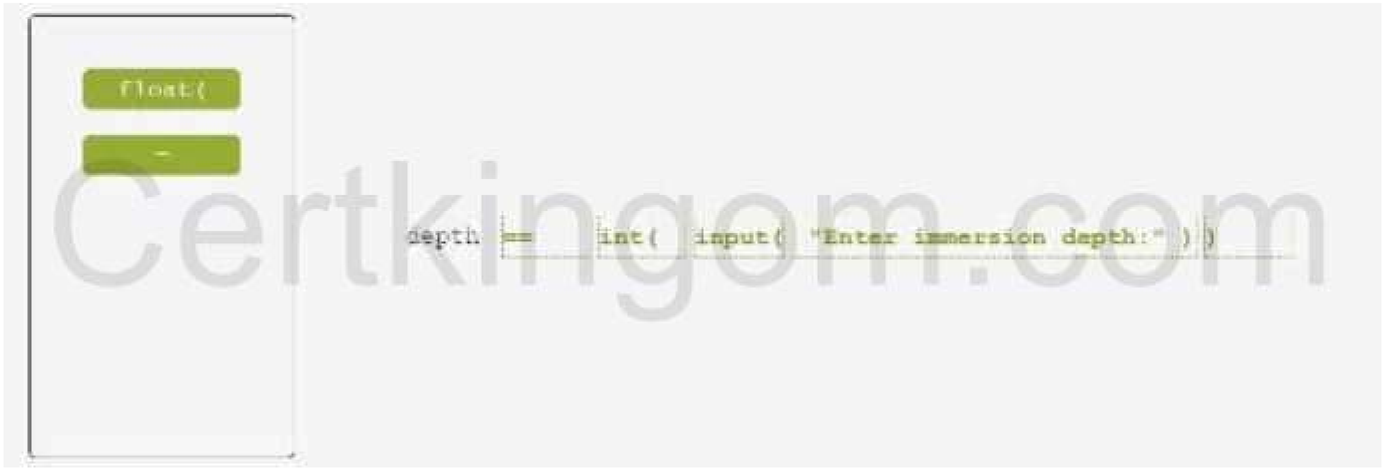
QUESTION 1
DRAG DROP
Insert the code boxes in the correct positions in order to build a line of code which asks the user for an integer value and assigns it to the depth variable.
(Note: some code boxes will not be used.)



Answer:



```
depth == int( input( "Enter immersion depth:" ) )
```

Explanation:
One possible way to insert the code boxes in the correct positions in order to build a line of code which asks the user for an integer value and assigns it to the depth variable is:
depth = int(input("Enter the immersion depth: "))
This line of code uses the input function to prompt the user for a string value, and then uses the int function to convert that string value into an integer number. The result is then assigned to the variable depth.

You can find more information about the input and int functions in Python in the following references:
[Python input() Function]
[Python int() Function]

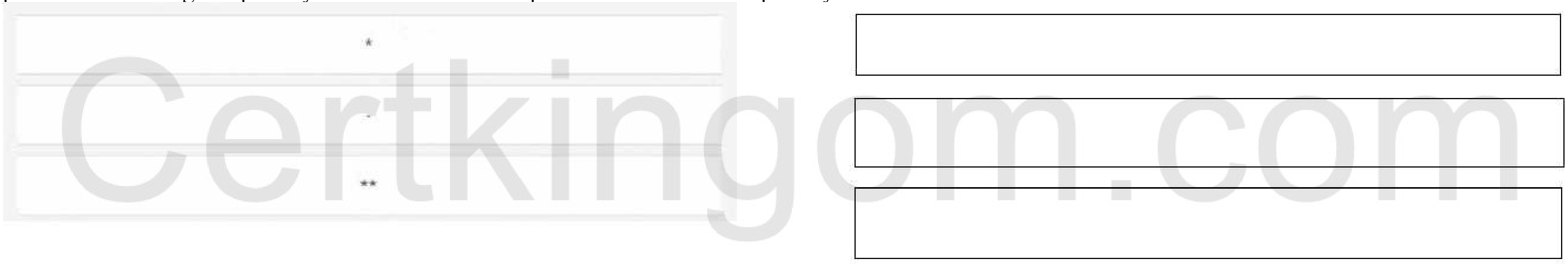
QUESTION 2
A set of rules which defines the ways in which words can be coupled in sentences is called:

A. lexis
B. syntax
C. semantics
D. dictionary

Answer: B

Explanation:
Syntax is the branch of linguistics that studies the structure and rules of sentences in natural languages. Lexis is the vocabulary of a language. Semantics is the study of meaning in language. A dictionary is a collection of words and their definitions, synonyms, pronunciations, etc.
Reference: [Python Institute - Entry-Level Python Programmer Certification]

QUESTION 3
DRAG DROP
Arrange the binary numeric operators in the order which reflects their priorities, where the top-most position has the highest priority and the bottom-most position has the lowest priority.



Answer:

Explanation:
The correct order of the binary numeric operators in Python according to their priorities is:
Exponentiation (**)
Multiplication (*) and Division (/, //, %)
Addition (+) and Subtraction (-)
This order follows the standard mathematical convention of operator precedence, which can be remembered by the acronym PEMDAS (Parentheses, Exponents, Multiplication/Division, Addition/Subtraction). Operators with higher precedence are evaluated before those with lower precedence, but operators with the same precedence are evaluated from left to right. Parentheses can be used to change the order of evaluation by grouping expressions.
For example, in the expression 2 + 3 * 4 ** 2, the exponentiation operator (**) has the highest priority, so it is evaluated first, resulting in 2 + 3 * 16. Then, the multiplication operator (*) has the next highest priority, so it is evaluated next, resulting in 2 + 48. Finally, the addition operator (+) has the lowest priority, so it is evaluated last, resulting in 50.
You can find more information about the operator precedence in Python in the following references:
6. Expressions " Python 3.11.5 documentation
Precedence and Associativity of Operators in Python - Programiz
Python Operator Priority or Precedence Examples Tutorial

QUESTION 4
Which of the following expressions evaluate to a non-zero result? (Select two answers.)

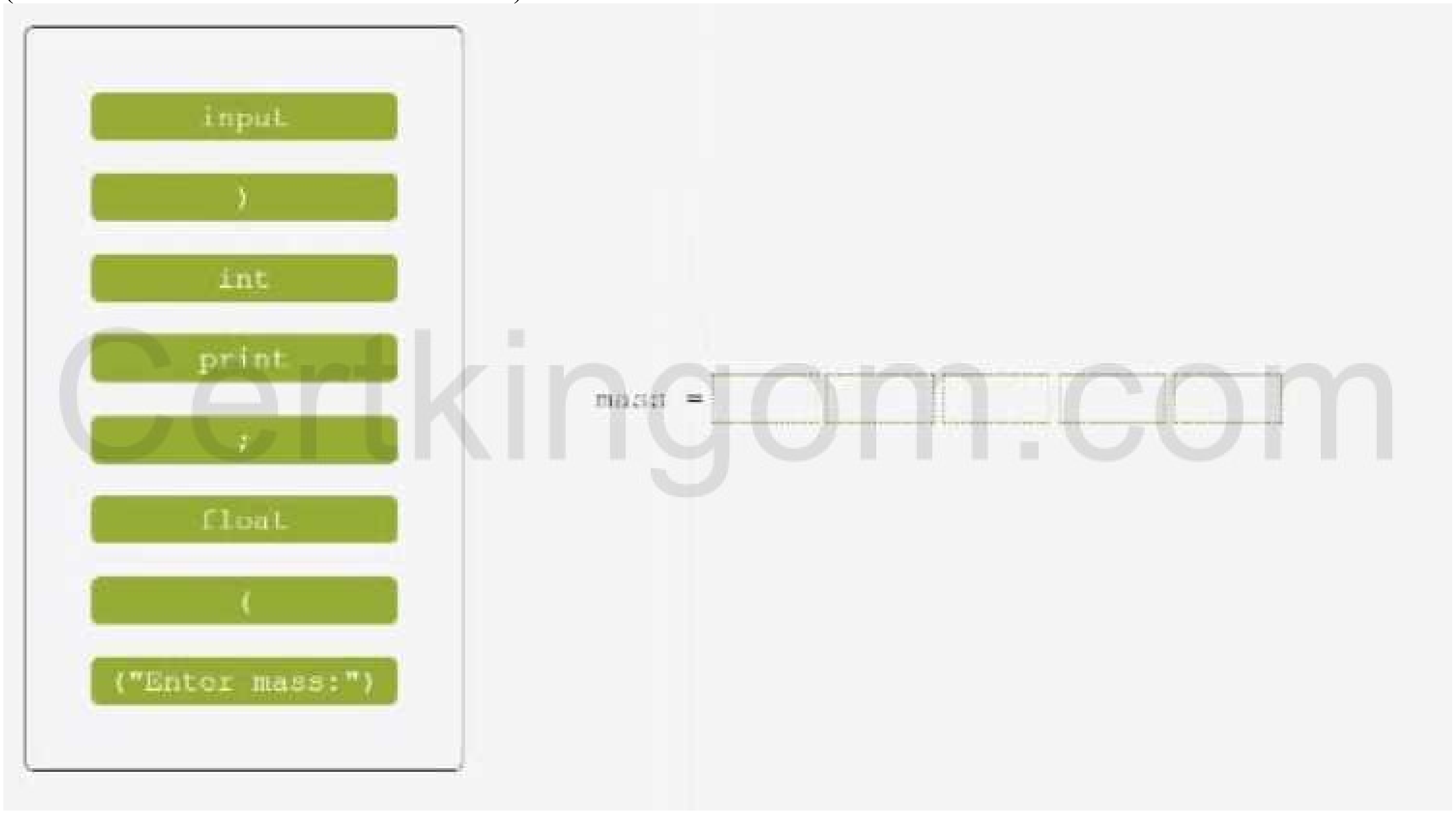
A. 2 ** 3 / A - 2
B. 4 / 2 * * 3 - 2
C. 1 * * 3 / 4 - 1
D. 1 * 4 // 2 ** 3

Answer: AB

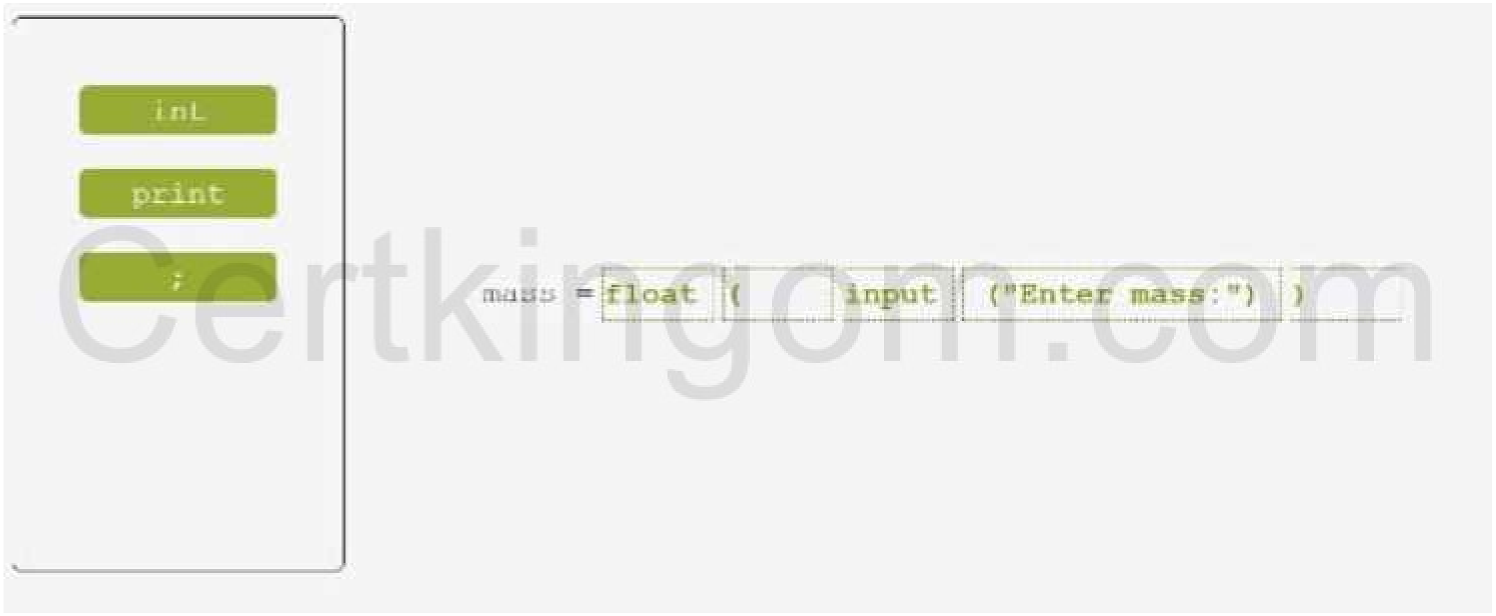
Explanation:
In Python, the ** operator is used for exponentiation, the / operator is used for floating-point

division, and the // operator is used for integer division. The order of operations is parentheses, exponentiation, multiplication/division, and addition/subtraction. Therefore, the expressions can be evaluated as follows:
A) 2 ** 3 / A - 2 = 8 / A - 2 (assuming A is a variable that is not zero or undefined) B. 4 / 2 * * 3 - 2 = 4 / 8 - 2 = 0.5 - 2 = -1.5 C. 1 * * 3 / 4 - 1 = 1 / 4 - 1 = 0.25 - 1 = -0.75 D. 1 * 4 // 2 ** 3 = 4 // 8 = 0
Only expressions A and B evaluate to non-zero results.
Reference: [Python Institute - Entry-Level Python Programmer Certification]

QUESTION 5
DRAG DROP
Insert the code boxes in the correct positions in order to build a line of code which asks the user for a float value and assigns it to the mass variable.
(Note: some code boxes will not be used.)



Answer:



```
mass = float ( input ("Enter mass:") )
```

Explanation:
One possible way to insert the code boxes in the correct positions in order to build a line of code that asks the user for a float value and assigns it to the mass variable is:
mass = float(input("Enter the mass: "))
This line of code uses the input function to prompt the user for a string value, and then uses the float function to convert that string value into a floating-point number. The result is then assigned to the variable mass.
You can find more information about the input and float functions in Python in the following references:
[Python input() Function]
[Python float() Function]

QUESTION 6
Python Is an example of which programming language category?

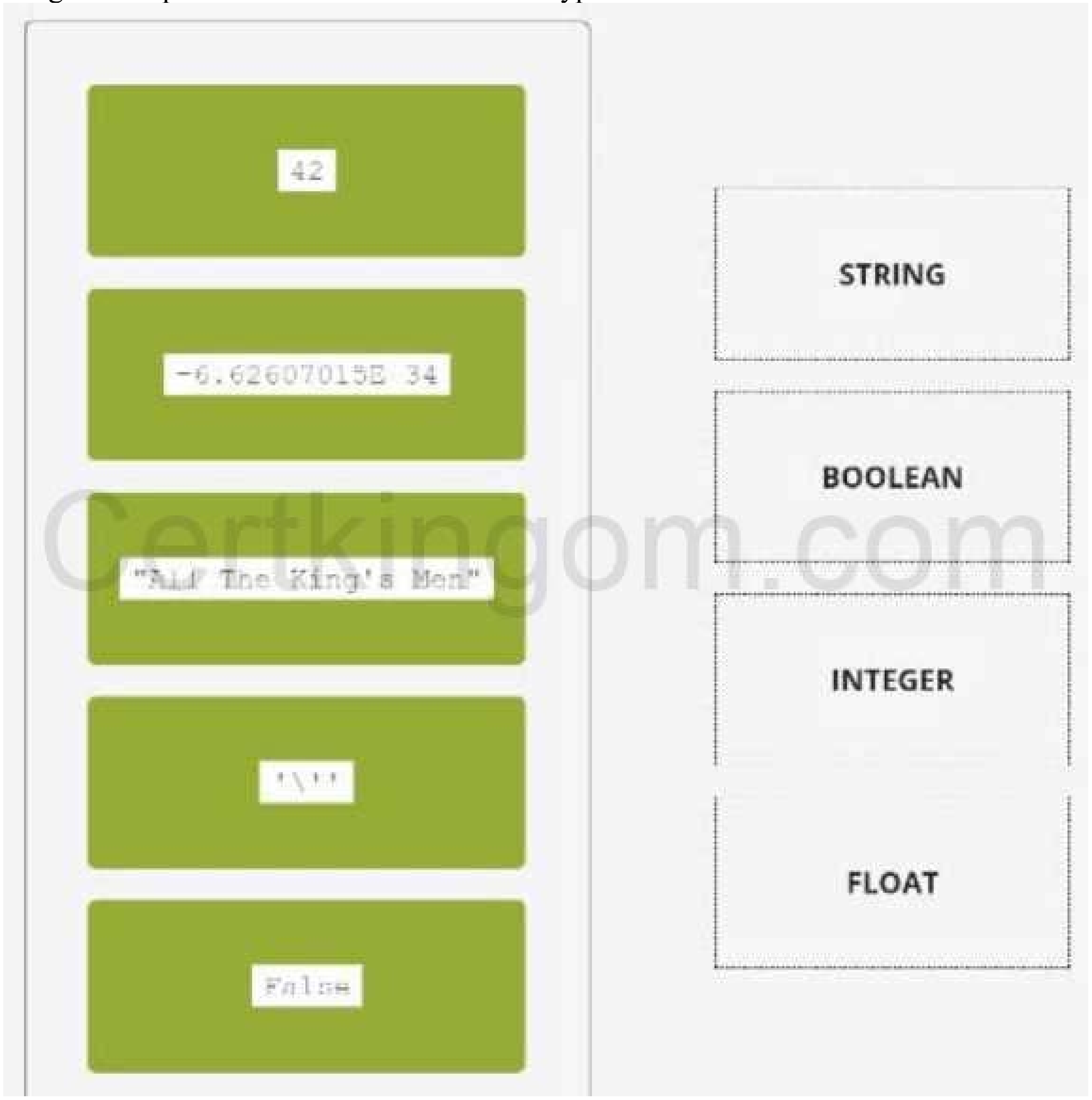
A. interpreted
B. assembly
C. compiled
D. machine

Answer: A

Explanation:
Python is an interpreted programming language, which means that the source code is translated into executable code by an interpreter at runtime, rather than by a compiler beforehand. Interpreted languages are more flexible and portable than compiled languages, but they are also slower and less

efficient. Assembly and machine languages are low-level languages that are directly executed by the hardware, while compiled languages are high-level languages that are translated into machine code by a compiler before execution.
Reference: [Python Institute - Entry-Level Python Programmer Certification]

QUESTION 7
DRAG DROP
Drag and drop the literals to match their data type names.

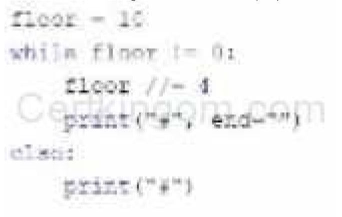


Answer:

Explanation:
One possible way to drag and drop the literals to match their data type names is:

STRING: œAll The Kings Men
BOOLEAN: False
INTEGER: 42
FLOAT: -6.62607015E-34
A literal is a value that is written exactly as it is meant to be interpreted by the Python interpreter. A data type is a category of values that share some common characteristics or operations. Python has four basic data types: string, boolean, integer, and float.
A string is a sequence of characters enclosed by either single or double quotes. A string can represent text, symbols, or any other information that can be displayed as text. For example, œAll The Kings Men is a string literal that represents the title of a novel.
A boolean is a logical value that can be either True or False. A boolean can represent the result of a comparison, a condition, or a logical operation. For example, False is a boolean literal that represents the opposite of True.
An integer is a whole number that can be positive, negative, or zero. An integer can represent a count, an index, or any other quantity that does not require fractions or decimals. For example, 42 is an integer literal that represents the answer to life, the universe, and everything.
A float is a number that can have a fractional part after the decimal point. A float can represent a measurement, a ratio, or any other quantity that requires precision or approximation. For example, -6.62607015E-34 is a float literal that represents the Planck constant in scientific notation.
You can find more information about the literals and data types in Python in the following references:
[Python Data Types]
[Python Literals]
[Python Basic Syntax]

QUESTION 8
How many hashes (+) does the code output to the screen?



```
floor = 10
while floor != 0:
    floor //= 4
    print("#", end="")
else:
    print("#")
```

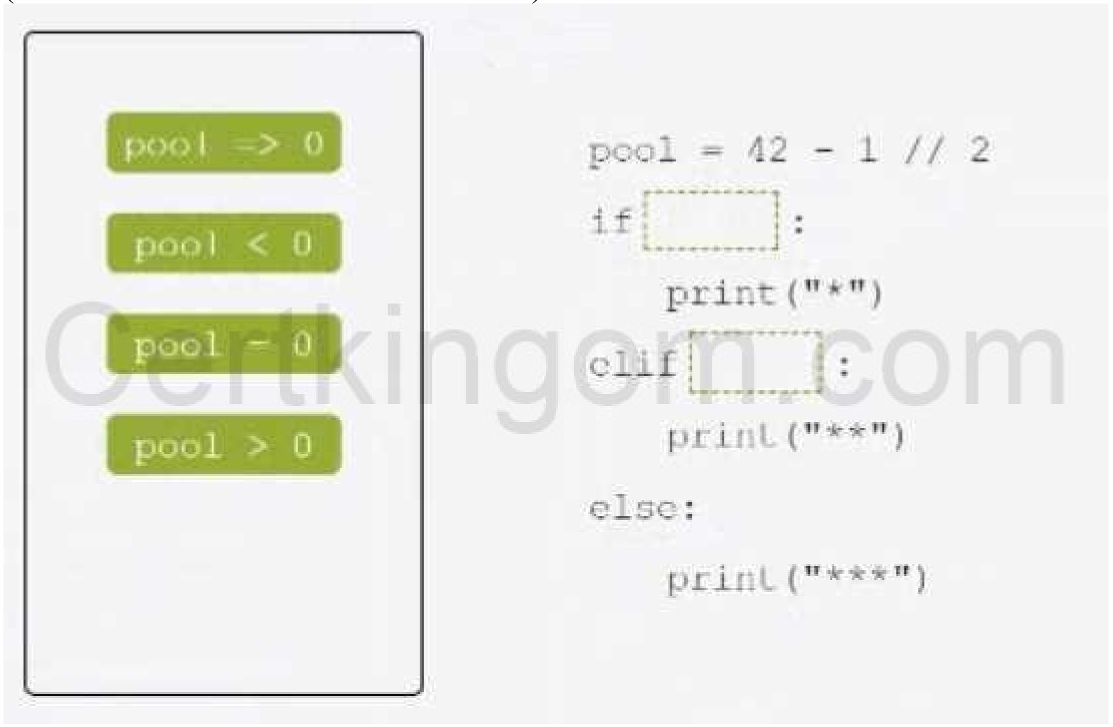
A. one
B. zero (the code outputs nothing)
C. five
D. three

Answer: C

Explanation:
The code snippet that you have sent is a loop that checks if a variable œfloor is less than or equal to 0 and prints a string accordingly. The code is as follows:
floor = 5 while floor > 0: print(œ+) floor = floor - 1

The code starts with assigning the value 5 to the variable œfloor . Then, it enters a while loop that repeats as long as the condition œfloor > 0 is true. Inside the loop, the code prints a œ+ symbol to the screen, and then subtracts 1 from the value of œfloor . The loop ends when œfloor becomes 0 or negative, and the code exits.
The code outputs five œ+ symbols to the screen, one for each iteration of the loop. Therefore, the correct answer is C. five.
Reference: [Python Institute - Entry-Level Python Programmer Certification]

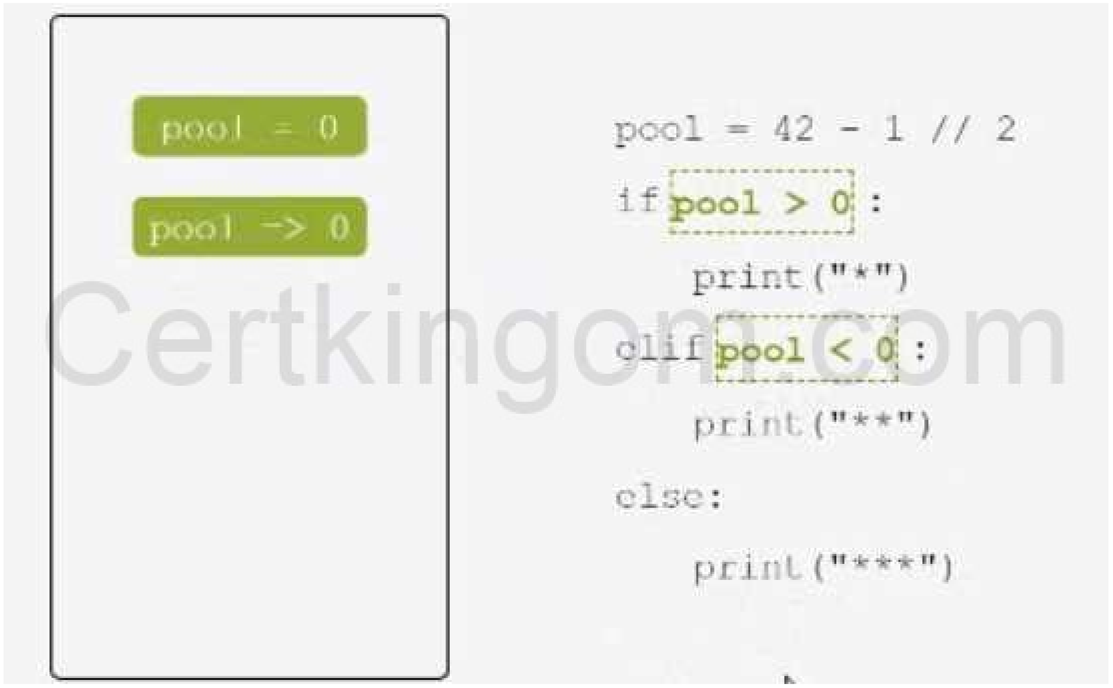
QUESTION 9
DRAG DROP
Drag and drop the conditional expressions to obtain a code which outputs * to the screen.
(Note: some code boxes will not be used.)



```
pool => 0
pool < 0
pool = 0
pool > 0
```

```
pool = 42 - 1 // 2
if [      ]:
    print("*")
elif [      ]:
    print("**")
else:
    print("***")
```

Answer:



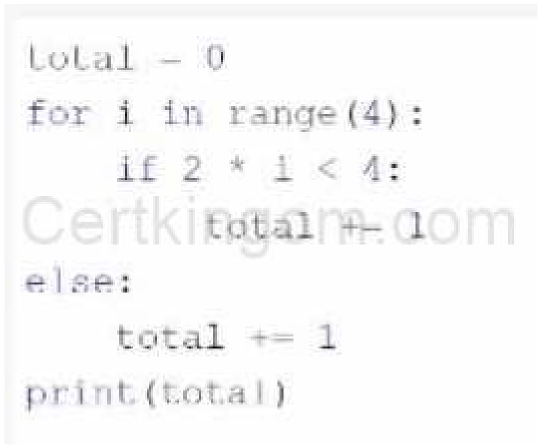
```
pool = 42 - 1 // 2
if pool > 0 :
    print("*")
elif pool < 0 :
    print("**")
else:
    print("***")
```

Explanation:
One possible way to drag and drop the conditional expressions to obtain a code which outputs * to the screen is:

```
if pool > 0:
print("*")
elif pool < 0:
print("**")
else:
print("***")
```

This code uses the if, elif, and else keywords to create a conditional statement that checks the value of the variable pool. Depending on whether the value is greater than, less than, or equal to zero, the code will print a different pattern of asterisks to the screen. The print function is used to display the output. The code is indented to show the blocks of code that belong to each condition. The code will output * if the value of pool is positive, ** if the value of pool is negative, and *** if the value of pool is zero.
You can find more information about the conditional statements and the print function in Python in the following references:
[Python If ¦ Else]
[Python Print Function]
[Python Basic Syntax]

QUESTION 10
What happens when the user runs the following code?



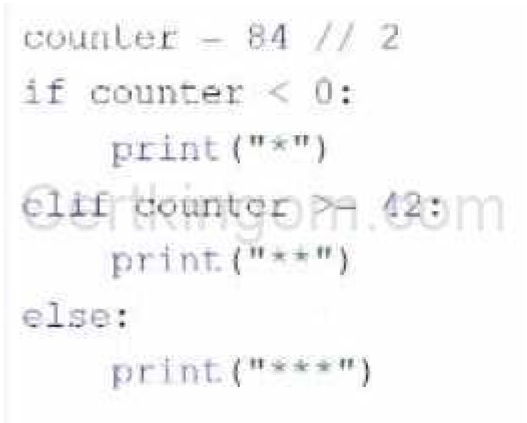
```
total = 0
for i in range(4):
    if 2 * i < 4:
        total += 1
else:
    total += 1
print(total)
```

A. The code outputs 3.
B. The code outputs 2.
C. The code enters an infinite loop.
D. The code outputs 1.

Answer: B

Explanation:
The code snippet that you have sent is calculating the value of a variable œtotal   based on the values in the range of 0 to 3. The code is as follows:
total = 0 for i in range(0, 3): if i % 2 == 0: total = total + 1 else: total = total + 2 print(total)
The code starts with assigning the value 0 to the variable œtotal   . Then, it enters a for loop that iterates over the values 0, 1, and 2 (the range function excludes the upper bound). Inside the loop, the code checks if the current value of œi   is even or odd using the modulo operator (%). If œi   is even, the code adds 1 to the value of œtotal   . If œi   is odd, the code adds 2 to the value of œtotal   . The loop ends when œi   reaches 3, and the code prints the final value of œtotal   to the screen.
The code outputs 2 to the screen, because the value of œtotal   changes as follows:
When i = 0, total = 0 + 1 = 1
When i = 1, total = 1 + 2 = 3
When i = 2, total = 3 + 1 = 4
When i = 3, the loop ends and total = 4 is printed
Therefore, the correct answer is B. The code outputs 2.
Reference: [Python Institute - Entry-Level Python Programmer Certification]

QUESTION 11
What is the expected output of the following code?



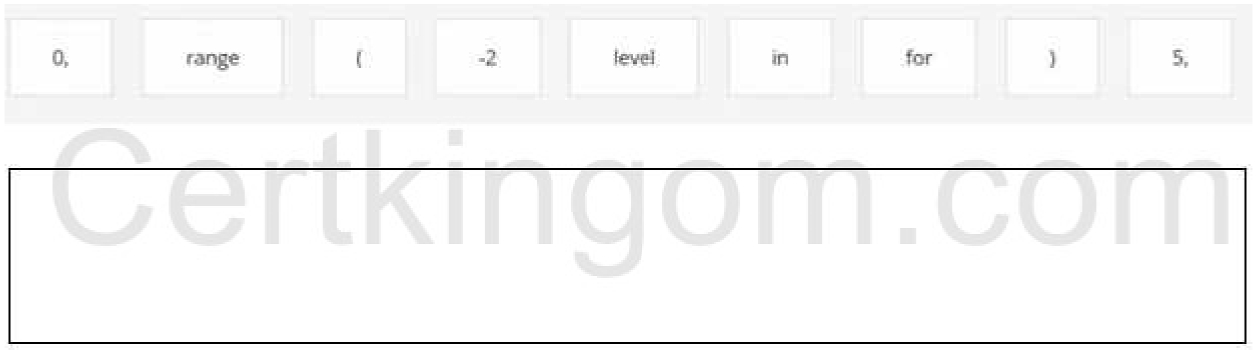
```
counter = 84 // 2
if counter < 0:
    print("*")
elif counter >= 42:
    print("**")
else:
    print("***")
```

A. The code produces no output.
B. * * *
C. * *
D. *

Answer: C

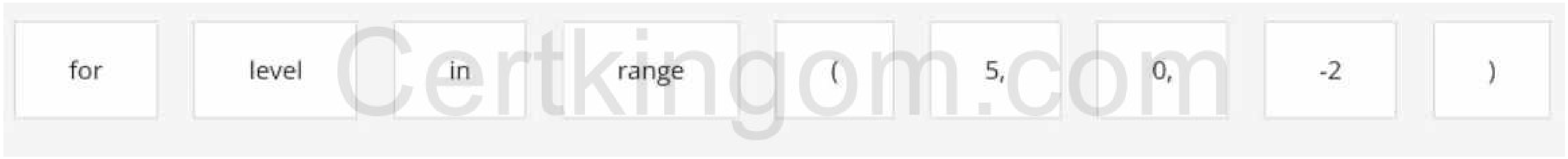
Explanation:
The code snippet that you have sent is a conditional statement that checks if a variable œcounter   is less than 0, greater than or equal to 42, or neither. The code is as follows:
if counter < 0: print(œœ) elif counter >= 42: print(œ) else: print(   )
The code starts with checking if the value of œcounter   is less than 0. If yes, it prints a single asterisk () to the screen and exits the statement. If no, it checks if the value of œcounter   is greater than or equal to 42. If yes, it prints three asterisks () to the screen and exits the statement. If no, it prints two asterisks () to the screen and exits the statement.
The expected output of the code depends on the value of œcounter . If the value of œcounter   is 10, as shown in the image, the code will print two asterisks (**) to the screen, because 10 is neither less than 0 nor greater than or equal to 42. Therefore, the correct answer is C. * *
Reference: [Python Institute - Entry-Level Python Programmer Certification]

QUESTION 12
DRAG DROP
Arrange the code boxes in the correct positions in order to obtain a loop which executes its body with the level variable going through values 5, 1, and 1 (in the same order).



0, range (-2 level in for) 5,

Answer:

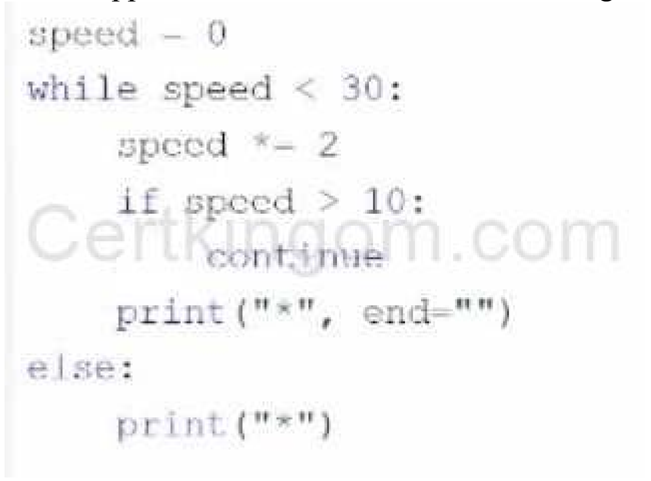


for level in range (5, 0, -2)

Explanation:

QUESTION 13

What happens when the user runs the following code?



```
speed = 0
while speed < 30:
    speed *= 2
    if speed > 10:
        continue
    print("*", end="")
else:
    print("*")
```

A. The program outputs three asterisks (***)to the screen.
B. The program outputs one asterisk (*) to the screen.
C. The program outputs five asterisks (*****) to the screen.
D. The program enters an infinite loop.

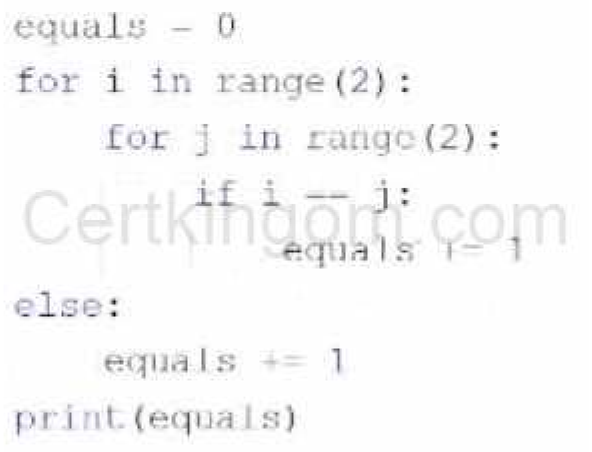
Answer: D

Explanation:

The code snippet that you have sent is a while loop with an if statement and a print statement inside

it. The code is as follows:
while True: if counter < 0: print(œœ) else: print(**   )
The code starts with entering a while loop that repeats indefinitely, because the condition œTrue   is always true. Inside the loop, the code checks if the value of œcounter   is less than 0. If yes, it prints a single asterisk () to the screen. If no, it prints three asterisks (**) to the screen. However, the code does not change the value of œcounter   inside the loop, so the same condition is checked over and over again. The loop never ends, and the code enters an infinite loop.
The program outputs either one asterisk () or three asterisks (**) to the screen repeatedly, depending on the initial value of œcounter   . Therefore, the correct answer is D. The program enters an infinite loop.
Reference: [Python Institute - Entry-Level Python Programmer Certification]

QUESTION 14
What is the expected output of the following code?



```
equals = 0
for i in range(2):
    for j in range(2):
        if i == j:
            equals += 1
else:
    equals += 1
print(equals)
```

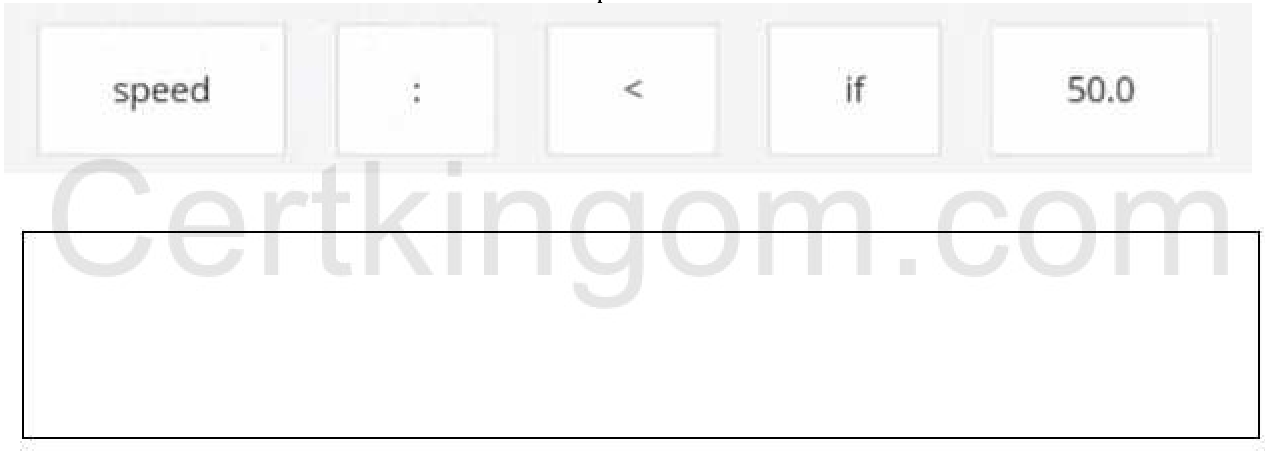
A. The code outputs nothing.
B. 3
C. 1
D. 4

Answer: C

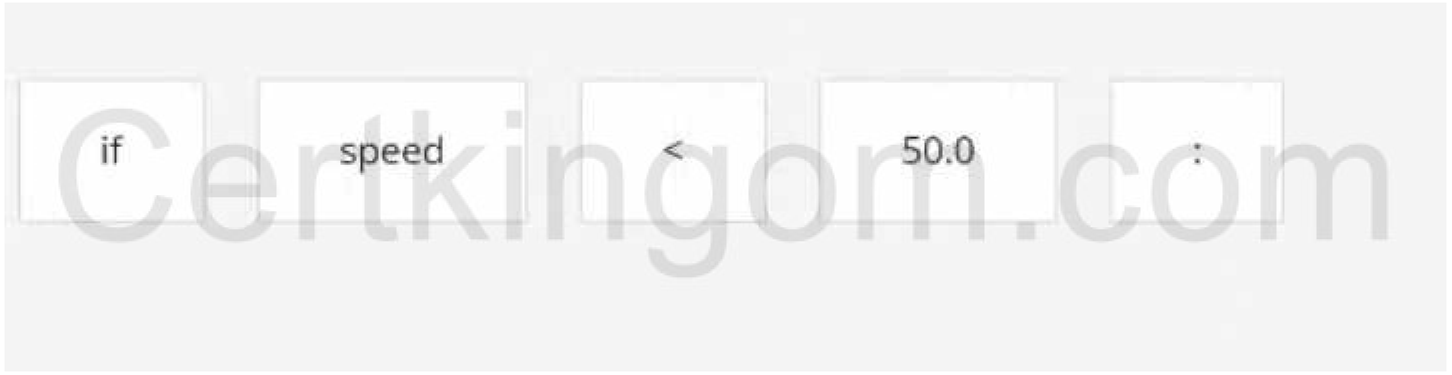
Explanation:
The code snippet that you have sent is checking if two numbers are equal and printing the result.
The code is as follows:
num1 = 1 num2 = 2 if num1 == num2: print(4) else: print(1)
The code starts with assigning the values 1 and 2 to the variables œnum1   and œnum2   respectively. Then, it enters an if statement that compares the values of œnum1   and œnum2   using the equality operator (==). If the values are equal, the code prints 4 to the screen. If the values are not equal, the code prints 1 to the screen.
The expected output of the code is 1, because the values of œnum1   and œnum2   are not equal.

Therefore, the correct answer is C. 1.
Reference: [Python Institute - Entry-Level Python Programmer Certification]

QUESTION 15
DRAG DROP
Arrange the code boxes in the correct positions to form a conditional instruction which guarantees that a certain statement is executed when the speed variable is less than 50.0.



Answer:

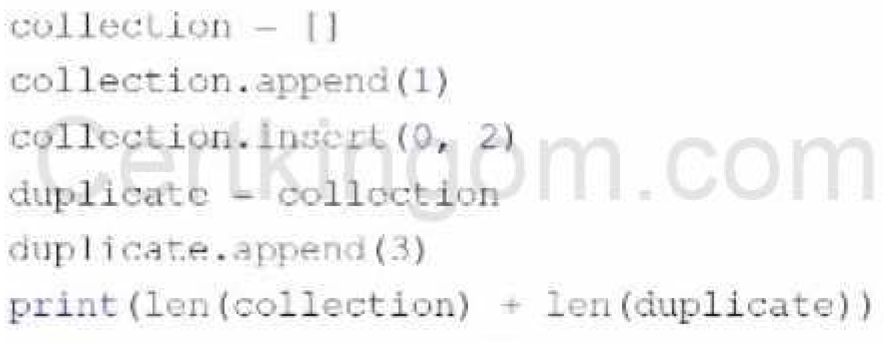


Explanation:
One possible way to arrange the code boxes in the correct positions to form a conditional instruction which guarantees that a certain statement is executed when the speed variable is less than 50.0 is:

```
if speed < 50.0:
print("The speed is low.")
```

This code uses the if keyword to create a conditional statement that checks the value of the variable speed. If the value is less than 50.0, then the code will print œThe speed is low.   to the screen. The print function is used to display the output. The code is indented to show the block of code that belongs to the if condition.
You can find more information about the if statement and the print function in Python in the following references:
Python If ¦ Else
Python Print Function

QUESTION 16
What is the expected output of the following code?



```
collection = []
collection.append(1)
collection.insert(0, 2)
duplicate = collection
duplicate.append(3)
print(len(collection) + len(duplicate))
```

A. 5
B. 4
C. 6
D. The code raises an exception and outputs nothing.

Answer: D

Explanation:
The code snippet that you have sent is trying to print the combined length of two lists, œcollection and œduplicate . The code is as follows:
collection = [] collection.append(1) collection.insert(0, 2) duplicate = collection duplicate.append(3) print(len(collection) + len(duplicate))
The code starts with creating an empty list called œcollection and appending the number 1 to it. The list now contains [1]. Then, the code inserts the number 2 at the beginning of the list. The list now contains [2, 1]. Then, the code creates a new list called œduplicate and assigns it the value of œcollection . However, this does not create a copy of the list, but rather a reference to the same list object. Therefore, any changes made to œduplicate will also affect œcollection , and vice versa. Then, the code appends the number 3 to œduplicate . The list now contains [2, 1, 3], and so does œcollection . Finally, the code tries to print the sum of the lengths of œcollection and œduplicate .
However, this causes an exception, because the len function expects a single argument, not two. The code does not handle the exception, and therefore outputs nothing.
The expected output of the code is nothing, because the code raises an exception and terminates.
Therefore, the correct answer is D. The code raises an exception and outputs nothing.
Reference: [Python Institute - Entry-Level Python Programmer Certification]

QUESTION 17
Assuming that the following assignment has been successfully executed:
My_list “ [1, 1, 2, 3]
Select the expressions which will not raise any exception.
(Select two expressions.)

A. my_list[-10]
B. my_list|my_Li1st | 3| I
C. my list [6]
D. my_List- [0:1]

Answer: B, D

Explanation:
The code snippet that you have sent is assigning a list of four numbers to a variable called œmy_list .
The code is as follows:
my_list = [1, 1, 2, 3]
The code creates a list object that contains the elements 1, 1, 2, and 3, and assigns it to the variable œmy_list . The list can be accessed by using the variable name or by using the index of the elements. The index starts from 0 for the first element and goes up to the length of the list minus one for the last element. The index can also be negative, in which case it counts from the end of the list. For example, my_list[0] returns 1, and my_list[-1] returns 3.
The code also allows some operations on the list, such as slicing, concatenation, repetition, and membership. Slicing is used to get a sublist of the original list by specifying the start and end index. For example, my_list[1:3] returns [1, 2]. Concatenation is used to join two lists together by using the + operator. For example, my_list + [4, 5] returns [1, 1, 2, 3, 4, 5]. Repetition is used to create a new list by repeating the original list a number of times by using the * operator. For example, my_list * 2 returns [1, 1, 2, 3, 1, 1, 2, 3]. Membership is used to check if an element is present in the list by using the in operator. For example, 2 in my_list returns True, and 4 in my_list returns False.
The expressions that you have given are trying to access or manipulate the list in different ways. Some of them are valid, and some of them are invalid and will raise an exception. An exception is an error that occurs when the code cannot be executed properly. The expressions are as follows:
A) my_list[-10]: This expression is trying to access the element at the index -10 of the list. However, the list only has four elements, so the index -10 is out of range. This will raise an IndexError exception and output nothing.
B) my_list|my_Li1st | 3| I: This expression is trying to perform a bitwise OR operation on the list and some other operands. The bitwise OR operation is used to compare the binary representation of two numbers and return a new number that has a 1 in each bit position where either number has a 1. For example, 3 | 1 returns 3, because 3 in binary is 11 and 1 in binary is 01, and 11 | 01 is 11. However, the bitwise OR operation cannot be applied to a list, because a list is not a number. This will raise a TypeError exception and output nothing.
C) my list [6]: This expression is trying to access the element at the index 6 of the list. However, the list only has four elements, so the index 6 is out of range. This will raise an IndexError exception and output nothing.
D) my_List- [0:1]: This expression is trying to perform a subtraction operation on the list and a sublist. The subtraction operation is used to subtract one number from another and return the difference. For example, 3 - 1 returns 2. However, the subtraction operation cannot be applied to a list, because a list is not a number. This will raise a TypeError exception and output nothing.

Only two expressions will not raise any exception. They are:
B) my_list|my_Li1st | 3| I: This expression is not a valid Python code, but it is not an expression that tries to access or manipulate the list. It is just a string of characters that has no meaning. Therefore, it will not raise any exception, but it will also not output anything.
D) my_List- [0:1]: This expression is a valid Python code that uses the slicing operation to get a sublist of the list. The slicing operation does not raise any exception, even if the start or end index is out of range. It will just return an empty list or the closest possible sublist. For example, my_list[0:10] returns [1, 1, 2, 3], and my_list[10:20] returns []. The expression my_List- [0:1] returns the sublist of the list from the index 0 to the index 1, excluding the end index. Therefore, it returns [1]. This expression will not raise any exception, and it will output [1].
Therefore, the correct answers are B. my_list|my_Li1st | 3| I and D. my_List- [0:1].
Reference: [Python Institute - Entry-Level Python Programmer Certification]

QUESTION 18
What is true about tuples? (Select two answers.)

A. Tuples are immutable, which means that their contents cannot be changed during their lifetime.
B. The len { } function cannot be applied to tuples.
C. An empty tuple is written as { } .
D. Tuples can be indexed and sliced like lists.

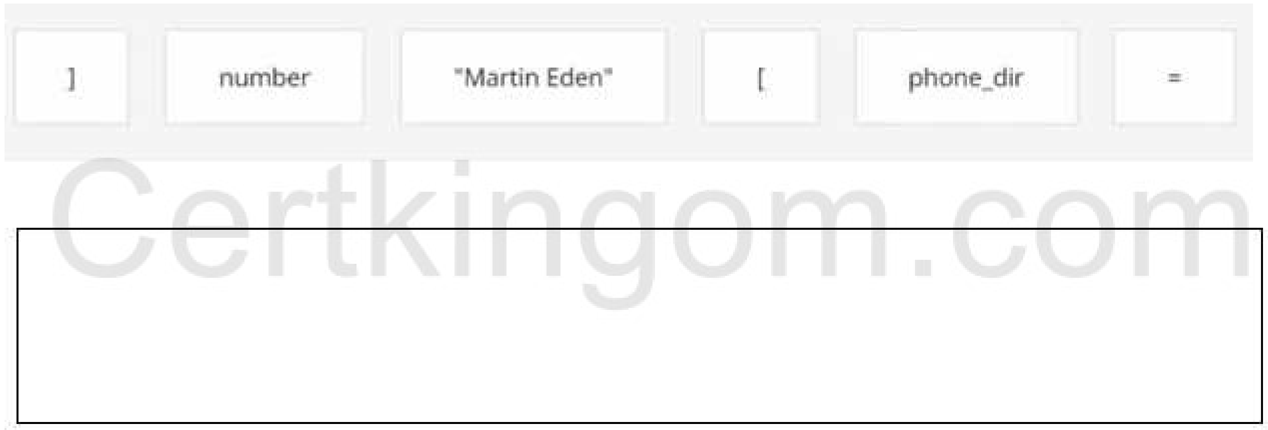
Answer: A, D

Explanation:
Tuples are one of the built-in data types in Python that are used to store collections of data. Tuples have some characteristics that distinguish them from other data types, such as lists, sets, and dictionaries. Some of these characteristics are:
Tuples are immutable, which means that their contents cannot be changed during their lifetime. Once a tuple is created, it cannot be modified, added, or removed. This makes tuples more stable and reliable than mutable data types. However, this also means that tuples are less flexible and dynamic than mutable data types. For example, if you want to change an element in a tuple, you have to create a new tuple with the modified element and assign it to the same variable12
Tuples are ordered, which means that the items in a tuple have a defined order and can be accessed by using their index. The index of a tuple starts from 0 for the first item and goes up to the length of the tuple minus one for the last item. The index can also be negative, in which case it counts from the end of the tuple. For example, if you have a tuple t = ("a", "b", "c"), then t[0] returns "a", and t[-1] returns "c"12
Tuples can be indexed and sliced like lists, which means that you can get a single item or a sublist of a tuple by using square brackets and specifying the start and end index. For example, if you have a tuple t = ("a", "b", "c", "d", "e"), then t[2] returns "c", and t[1:4] returns ("b", "c", "d"). Slicing does not raise any exception, even if the start or end index is out of range. It will just return an empty tuple or the closest possible sublist12
Tuples can contain any data type, such as strings, numbers, booleans, lists, sets, dictionaries, or even

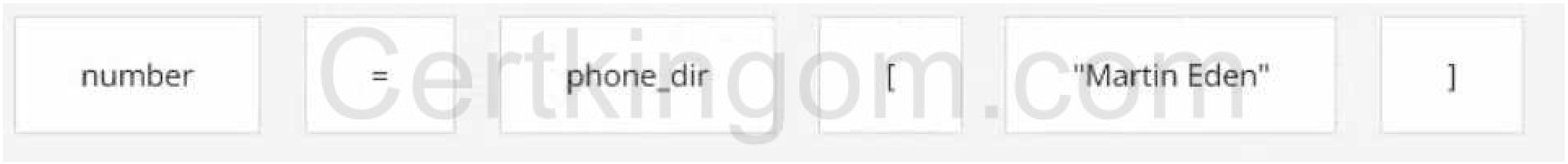
other tuples. Tuples can also have duplicate values, which means that the same item can appear more than once in a tuple. For example, you can have a tuple t = (1, 2, 3, 1, 2), which contains two 1s and two 2s12
Tuples are written with round brackets, which means that you have to enclose the items in a tuple with parentheses. For example, you can create a tuple t = ("a", "b", "c") by using round brackets. However, you can also create a tuple without using round brackets, by just separating the items with commas. For example, you can create the same tuple t = "a", "b", "c" by using commas. This is called tuple packing, and it allows you to assign multiple values to a single variable12
The len() function can be applied to tuples, which means that you can get the number of items in a tuple by using the len() function. For example, if you have a tuple t = ("a", "b", "c"), then len(t) returns 312
An empty tuple is written as (), which means that you have to use an empty pair of parentheses to create a tuple with no items. For example, you can create an empty tuple t = () by using empty parentheses. However, if you want to create a tuple with only one item, you have to add a comma after the item, otherwise Python will not recognize it as a tuple. For example, you can create a tuple with one item t = ("a",) by using a comma12
Therefore, the correct answers are
A. Tuples are immutable, which means that their contents cannot
be changed during their lifetime. and D. Tuples can be indexed and sliced like lists.
Reference: Python Tuples - W3SchoolsTuples in Python - GeeksforGeeks

---

QUESTION 19
DRAG DROP
Assuming that the phonc_dir dictionary contains namemumber pairs, arrange the code boxes to create a valid line of code which retrieves Martin Eden's phone number, and assigns it to the number variable.



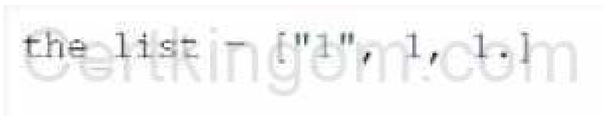
Answer:



```
number    =    phone_dir    [    "Martin Eden"    ]
```

Explanation:
number = phone_dir["Martin Eden"]
This code uses the square brackets notation to access the value associated with the key œMartin Eden in the phone_dir dictionary. The value is then assigned to the variable number. A dictionary is a data structure that stores key-value pairs, where each key is unique and can be used to retrieve its corresponding value. You can find more information about dictionaries in Python in the following references:
[Python Dictionaries - W3Schools]
[Python Dictionary (With Examples) - Programiz]
[5.5. Dictionaries ” How to Think Like a Computer Scientist ¦]

QUESTION 20
Assuming that the following assignment has been successfully executed:



```
the_list = ["1", 1, 1.]
```

Which of the following expressions evaluate to True? (Select two expressions.)

A. the_List.index {"1"} in the_list
B. 1.1 in the_list |1:3 |
C. len (the list [0:2]} <3
D. the_list. index {'1'} -- 0

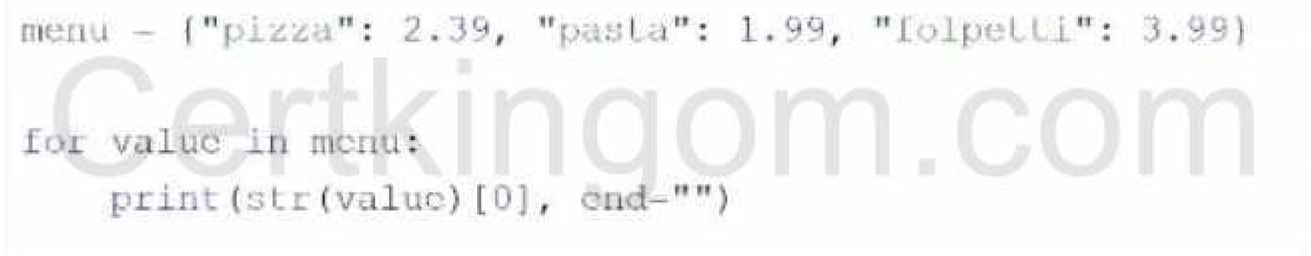
Answer: CD

Explanation:
The code snippet that you have sent is assigning a list of four values to a variable called œthe_list .
The code is as follows:
the_list = [˜1, 1, 1, 1]
The code creates a list object that contains the values ˜1, 1, 1, and 1, and assigns it to the variable œthe_list . The list can be accessed by using the variable name or by using the index of the values. The index starts from 0 for the first value and goes up to the length of the list minus one for the last value. The index can also be negative, in which case it counts from the end of the list. For example, the_list[0] returns ˜1, and the_list[-1] returns 1.
The expressions that you have given are trying to evaluate some conditions on the list and return a boolean value, either True or False. Some of them are valid, and some of them are invalid and will raise an exception. An exception is an error that occurs when the code cannot be executed properly.

The expressions are as follows:
A) the_List.index {œ1   } in the_list: This expression is trying to check if the index of the value ˜1 in the list is also a value in the list. However, this expression is invalid, because it uses curly brackets instead of parentheses to call the index method. The index method is used to return the first occurrence of a value in a list. For example, the_list.index(˜1) returns 0, because ˜1 is the first value in the list. However, the_list.index {œ1   } will raise a SyntaxError exception and output nothing.
B) 1.1 in the_list |1:3 |: This expression is trying to check if the value 1.1 is present in a sublist of the list. However, this expression is invalid, because it uses a vertical bar instead of a colon to specify the start and end index of the sublist. The sublist is obtained by using the slicing operation, which uses square brackets and a colon to get a part of the list. For example, the_list[1:3] returns [1, 1], which is the sublist of the list from the index 1 to the index 3, excluding the end index. However, the_list |1:3 | will raise a SyntaxError exception and output nothing.
C) len (the list [0:2]} <3: This expression is trying to check if the length of a sublist of the list is less than 3. This expression is valid, because it uses the len function and the slicing operation correctly. The len function is used to return the number of values in a list or a sublist. For example, len(the_list) returns 4, because the list has four values. The slicing operation is used to get a part of the list by using square brackets and a colon. For example, the_list[0:2] returns [˜1, 1], which is the sublist of the list from the index 0 to the index 2, excluding the end index. The expression len (the list [0:2]} <3 returns True, because the length of the sublist [˜1, 1] is 2, which is less than 3.
D) the_list. index {˜1} " 0: This expression is trying to check if the index of the value ˜1 in the list is equal to 0. This expression is valid, because it uses the index method and the equality operator correctly. The index method is used to return the first occurrence of a value in a list. For example, the_list.index(˜1) returns 0, because ˜1 is the first value in the list. The equality operator is used to compare two values and return True if they are equal, or False if they are not. For example, 0 == 0 returns True, and 0 == 1 returns False. The expression the_list. index {˜1} " 0 returns True, because the index of ˜1 in the list is 0, and 0 is equal to 0.
Therefore, the correct answers are C. len (the list [0:2]} <3 and D. the_list. index {˜1} " 0.
Reference: Python List Methods - W3Schools5. Data Structures " Python 3.11.5 documentationList methods in Python - GeeksforGeeks

---

QUESTION 21
What is the expected output of the following code?



```
menu = {"pizza": 2.39, "pasta": 1.99, "folpetti": 3.99}

for value in menu:
    print(str(value)[0], end="")
```

A. The code is erroneous and cannot be run.
B. ppt
C. 213

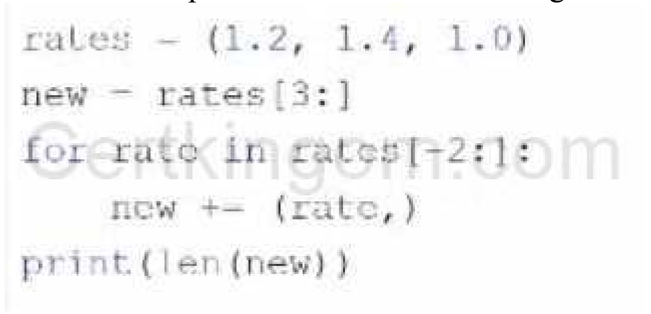
D. pizzapastafolpetti

Answer: B

Explanation:
The code snippet that you have sent is using the slicing operation to get parts of a string and concatenate them together. The code is as follows:
pizza = œpizza   pasta = œpasta   folpetti = œfolpetti   print(pizza[0] + pasta[0] + folpetti[0])
The code starts with assigning the strings œpizza , œpasta , and œfolpetti   to the variables pizza, pasta, and folpetti respectively. Then, it uses the print function to display the result of concatenating the first characters of each string. The first character of a string can be accessed by using the index 0 inside square brackets. For example, pizza[0] returns œp . The concatenation operation is used to join two or more strings together by using the + operator. For example, œa   + œb   returns œab . The code prints the result of pizza[0] + pasta[0] + folpetti[0], which is œp   + œp   + œf , which is œppt .
The expected output of the code is ppt, because the code prints the first characters of each string.
Therefore, the correct answer is B. ppt.
Reference: Python String Slicing - W3SchoolsPython String Concatenation - W3Schools

---

QUESTION 22
What is the expected result of the following code?



```
rates = (1.2, 1.4, 1.0)
new = rates[3:]
for rate in rates[-2:]:
    new += (rate,)
print(len(new))
```

A. 5
B. 2
C. 1
D. The code will cause an unhandled

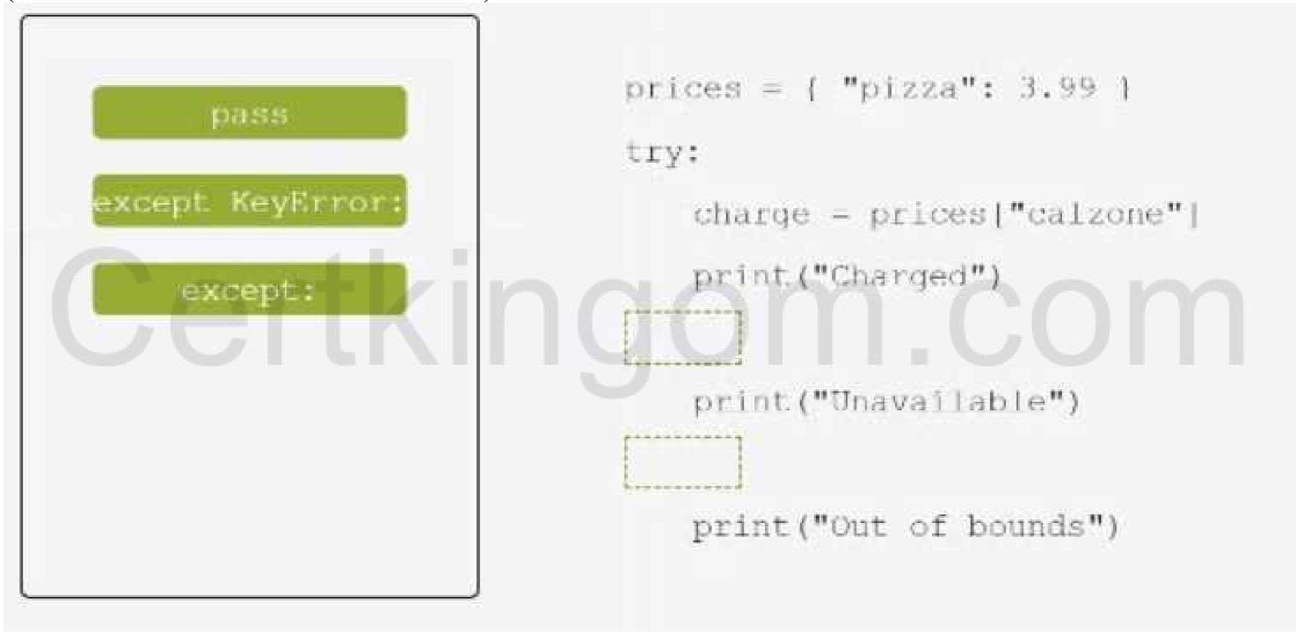
Answer: D

Explanation:
The code snippet that you have sent is trying to use a list comprehension to create a new list from an existing list. The code is as follows:
my_list = [1, 2, 3, 4, 5] new_list = [x for x in my_list if x > 5]
The code starts with creating a list called œmy_list   that contains the numbers 1, 2, 3, 4, and 5. Then, it tries to create a new list called œnew_list   by using a list comprehension. A list comprehension is a concise way of creating a new list from an existing list by applying some expression or condition to each element. The syntax of a list comprehension is:

new_list = [expression for element in old_list if condition]
The expression is the value that will be added to the new list, which can be the same as the element or a modified version of it. The element is the variable that takes each value from the old list. The condition is an optional filter that determines which elements will be included in the new list. For example, the following list comprehension creates a new list that contains the squares of the even numbers from the old list:
old_list = [1, 2, 3, 4, 5, 6] new_list = [x ** 2 for x in old_list if x % 2 == 0]
new_list = [4, 16, 36]
The code that you have sent is trying to create a new list that contains the elements from the old list that are greater than 5. However, there is a problem with this code. The problem is that none of the elements in the old list are greater than 5, so the condition is always false. This means that the new list will be empty, and the expression will never be evaluated. However, the expression is not valid, because it uses the variable x without defining it. This will cause a NameError exception, which is an error that occurs when a variable name is not found in the current scope. The code does not handle the exception, and therefore it will terminate with an error message.
The expected result of the code is an unhandled exception, because the code tries to use an undefined variable in an expression that is never executed. Therefore, the correct answer is D. The code will cause an unhandled exception.
Reference: Python - List Comprehension - W3SchoolsPython - List Comprehension - GeeksforGeeksPython Exceptions: An Introduction “ Real Python

---

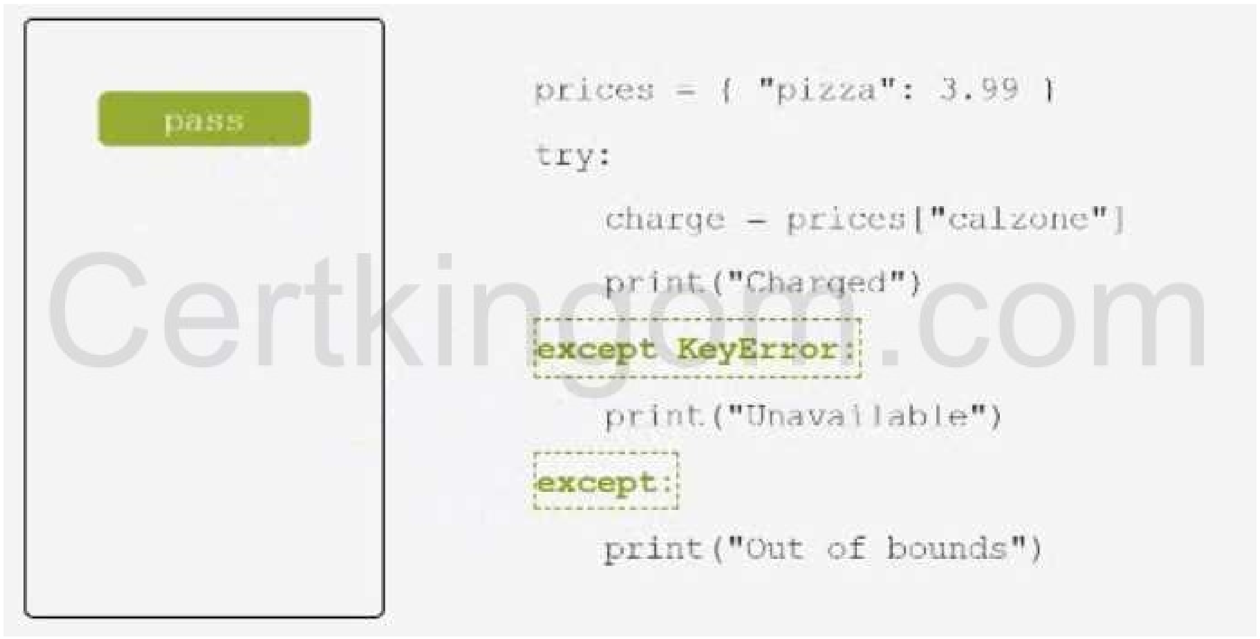
QUESTION 23
DRAG DROP
Drag and drop the code boxes in order to build a program which prints Unavailable to the screen.
(Note: one code box will not be used.)



```
pass
except KeyError:
except:
```

```
prices = { "pizza": 3.99 }
try:
    charge = prices["calzone"]
    print("Charged")
[      ]
    print("Unavailable")
[      ]
    print("Out of bounds")
```

Answer:

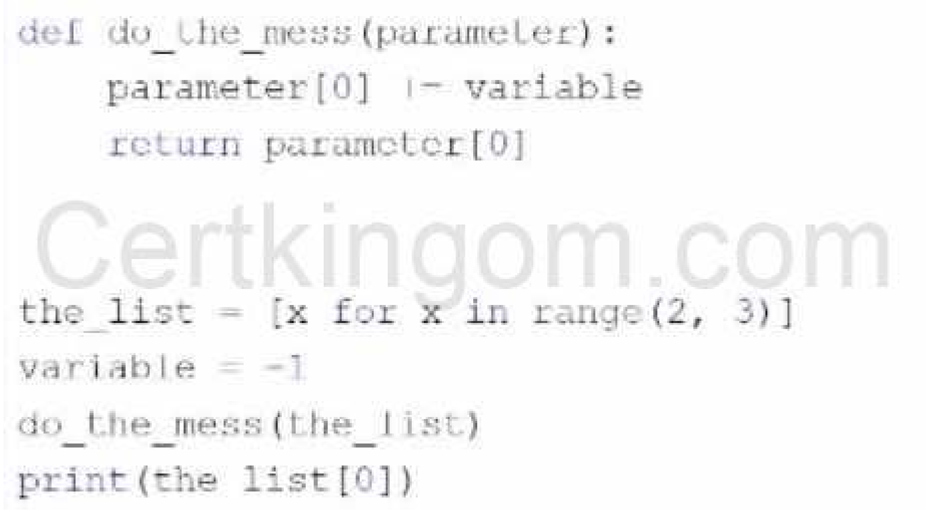


```
prices = { "pizza": 3.99 }
try:
    charge = prices["calzone"]
    print("Charged")
except KeyError:
    print("Unavailable")
except:
    print("Out of bounds")
```

Explanation:

QUESTION 24

What is the expected result of running the following code?



```
def do_the_mess(parameter):
    parameter[0] != variable
    return parameter[0]


the_list = [x for x in range(2, 3)]
variable = -1
do_the_mess(the_list)
print(the_list[0])
```

A. The code prints 1 .
B. The code prints 2
C. The code raises an unhandled exception.
D. The code prints 0

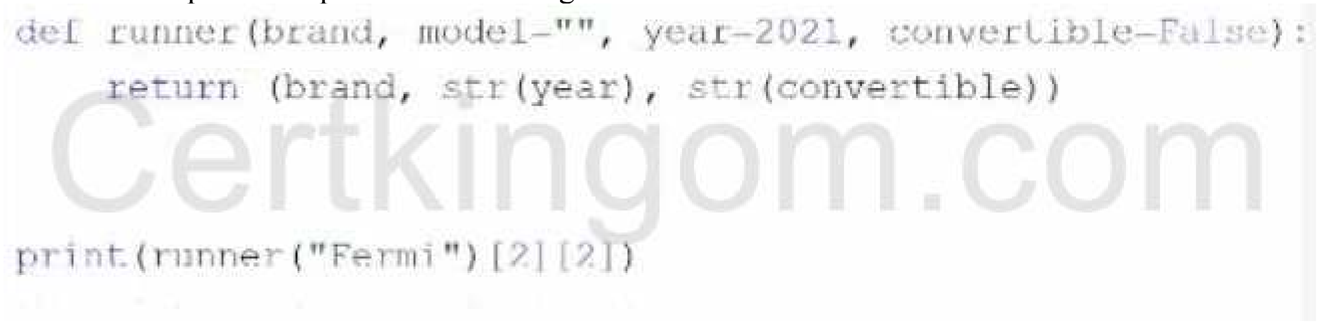
Answer: C

Explanation:

The code snippet that you have sent is trying to use the index method to find the position of a value in a list. The code is as follows:
the_list = [1, 2, 3, 4, 5] print(the_list.index(6))
The code starts with creating a list called œthe_list that contains the numbers 1, 2, 3, 4, and 5. Then, it tries to print the result of calling the index method on the list with the argument 6. The index method is used to return the first occurrence of a value in a list. For example, the_list.index(1) returns 0, because 1 is the first value in the list.
However, the code has a problem. The problem is that the value 6 is not present in the list, so the index method cannot find it. This will cause a ValueError exception, which is an error that occurs when a function or operation receives an argument that has the right type but an inappropriate value. The code does not handle the exception, and therefore it will terminate with an error message.
The expected result of the code is an unhandled exception, because the code tries to find a value that does not exist in the list. Therefore, the correct answer is C. The code raises an unhandled exception.
Reference: Python List index() Method - W3SchoolsPython Exceptions: An Introduction " Real Python

---

QUESTION 25
What is the expected output of the following code?



```
def runner(brand, model="", year=2021, convertible=False):
    return (brand, str(year), str(convertible))

print(runner("Fermi")[2][2])
```

A. 1
B. The code raises an unhandled exception.
C. False
D. ('Fermi ', '2021', 'False')

Answer: D

Explanation:
The code snippet that you have sent is defining and calling a function in Python. The code is as follows:
def runner(brand, model, year): return (brand, model, year)
print(runner(œFermi))
The code starts with defining a function called œrunner with three parameters: œbrand , œmodel , and œyear . The function returns a tuple with the values of the parameters. A tuple is a data type in Python that can store multiple values in an ordered and immutable way. A tuple is created by using parentheses and separating the values with commas. For example, (1, 2, 3) is a tuple with three

values.
Then, the code calls the function œrunner with the value œFermi for the œbrand parameter and prints the result. However, the function expects three arguments, but only one is given. This will cause a TypeError exception, which is an error that occurs when a function or operation receives an argument that has the wrong type or number. The code does not handle the exception, and therefore it will terminate with an error message.
However, if the code had handled the exception, or if the function had used default values for the missing parameters, the expected output of the code would be ('Fermi ', ˜2021, ˜False). This is because the function returns a tuple with the values of the parameters, and the print function displays the tuple to the screen. Therefore, the correct answer is D. ('Fermi ', ˜2021, ˜False).
Reference: Python Functions - W3SchoolsPython Tuples - W3SchoolsPython Exceptions: An Introduction " Real Python

---

QUESTION 26
What is true about exceptions and debugging? (Select two answers.)

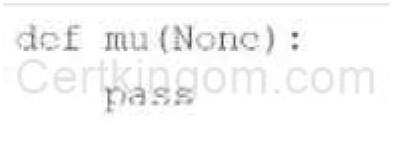
A. A tool that allows you to precisely trace program execution is called a debugger.
B. If some Python code is executed without errors, this proves that there are no errors in it.
C. One try-except block may contain more than one except branch.
D. The default (anonymous) except branch cannot be the last branch in the try-except block.

Answer: A, C

Explanation:
Exceptions and debugging are two important concepts in Python programming that are related to handling and preventing errors. Exceptions are errors that occur when the code cannot be executed properly, such as syntax errors, type errors, index errors, etc. Debugging is the process of finding and fixing errors in the code, using various tools and techniques. Some of the facts about exceptions and debugging are:
A tool that allows you to precisely trace program execution is called a debugger. A debugger is a program that can run another program step by step, inspect the values of variables, set breakpoints, evaluate expressions, etc. A debugger can help you find the source and cause of an error, and test possible solutions. Python has a built-in debugger module called pdb, which can be used from the command line or within the code. There are also other third-party debuggers available for Python, such as PyCharm, Visual Studio Code, etc12
If some Python code is executed without errors, this does not prove that there are no errors in it. It only means that the code did not encounter any exceptions that would stop the execution. However, the code may still have logical errors, which are errors that cause the code to produce incorrect or unexpected results. For example, if you write a function that is supposed to calculate the area of a circle, but you use the wrong formula, the code may run without errors, but it will give you the wrong answer. Logical errors are harder to detect and debug than syntax or runtime errors, because they do not generate any error messages. You have to test the code with different inputs and outputs, and compare them with the expected results34

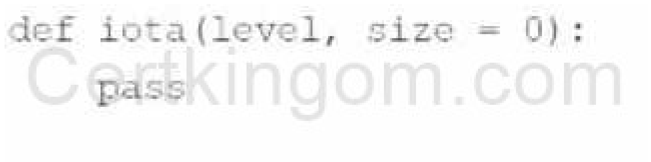
One try-except block may contain more than one except branch. A try-except block is a way of handling exceptions in Python, by using the keywords try and except. The try block contains the code that may raise an exception, and the except block contains the code that will execute if an exception occurs. You can have multiple except blocks for different types of exceptions, or for different actions to take. For example, you can write a try-except block like this:
try: # some code that may raise an exception except ValueError: # handle the ValueError exception except ZeroDivisionError: # handle the ZeroDivisionError exception except: # handle any other exception
This way, you can customize the error handling for different situations, and provide more informative messages or alternative solutions5
The default (anonymous) except branch can be the last branch in the try-except block. The default except branch is the one that does not specify any exception type, and it will catch any exception that is not handled by the previous except branches. The default except branch can be the last branch in the try-except block, but it cannot be the first or the only branch. For example, you can write a tryexcept block like this:
try: # some code that may raise an exception except ValueError: # handle the ValueError exception except: # handle any other exception
This is a valid try-except block, and the default except branch will be the last branch. However, you cannot write a try-except block like this:
try: # some code that may raise an exception except: # handle any exception
This is an invalid try-except block, because the default except branch is the only branch, and it will catch all exceptions, even those that are not errors, such as KeyboardInterrupt or SystemExit. This is considered a bad practice, because it may hide or ignore important exceptions that should be handled differently or propagated further. Therefore, you should always specify the exception types that you want to handle, and use the default except branch only as a last resort5
Therefore, the correct answers are
A. A tool that allows you to precisely trace program execution is
called a debugger. and C. One try-except block may contain more than one except branch.
Reference: Python Debugger " Python pdb - GeeksforGeeksHow can I see the details of an exception in Pythons debugger?Python Debugging (fixing problems)Python - start interactive debugger when exception would be otherwise thrownPython Try Except [Error Handling and Debugging " Programming with Python for Engineers]

QUESTION 27
Which of the following functions can be invoked with two arguments?
A)



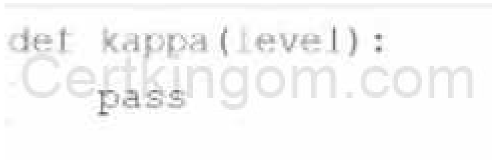
```
def mu(None):
    pass
```

B)



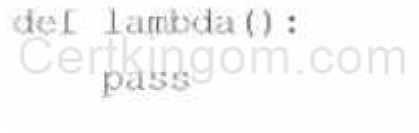
```
def iota(level, size = 0):
    pass
```

C)



```
def kappa(level):
    pass
```

D)



```
def lambda():
    pass
```

A. Option A
B. Option B
C. Option C
D. Option D

Answer: B

Explanation:
The code snippets that you have sent are defining four different functions in Python. A function is a block of code that performs a specific task and can be reused in the program. A function can take zero or more arguments, which are values that are passed to the function when it is called. A function can also return a value or None, which is the default return value in Python.
To define a function in Python, you use the def keyword, followed by the name of the function and parentheses. Inside the parentheses, you can specify the names of the parameters that the function will accept. After the parentheses, you use a colon and then indent the code block that contains the statements of the function. For example:
def function_name(parameter1, parameter2): # statements of the function return value
To call a function in Python, you use the name of the function followed by parentheses. Inside the parentheses, you can pass the values for the arguments that the function expects. The number and order of the arguments must match the number and order of the parameters in the function definition, unless you use keyword arguments or default values. For example:
function_name(argument1, argument2)
The code snippets that you have sent are as follows:
A) def my_function(): print(œHello   )
B) def my_function(a, b): return a + b
C) def my_function(a, b, c): return a * b * c
D) def my_function(a, b=0): return a - b
The question is asking which of these functions can be invoked with two arguments. This means that

the function must have two parameters in its definition, or one parameter with a default value and one without. The default value is a value that is assigned to a parameter if no argument is given for it when the function is called. For example, in option D, the parameter b has a default value of 0, so the function can be called with one or two arguments.
The only option that meets this criterion is option B. The function in option B has two parameters, a and b, and returns the sum of them. This function can be invoked with two arguments, such as my_function(2, 3), which will return 5.
The other options cannot be invoked with two arguments. Option A has no parameters, so it can only be called with no arguments, such as my_function(), which will print œHello . Option C has three parameters, a, b, and c, and returns the product of them. This function can only be called with three arguments, such as my_function(2, 3, 4), which will return 24. Option D has one parameter with a default value, b, and one without, a, and returns the difference of them. This function can be called with one or two arguments, such as my_function(2) or my_function(2, 3), which will return 2 or -1, respectively.
Therefore, the correct answer is B. Option B.

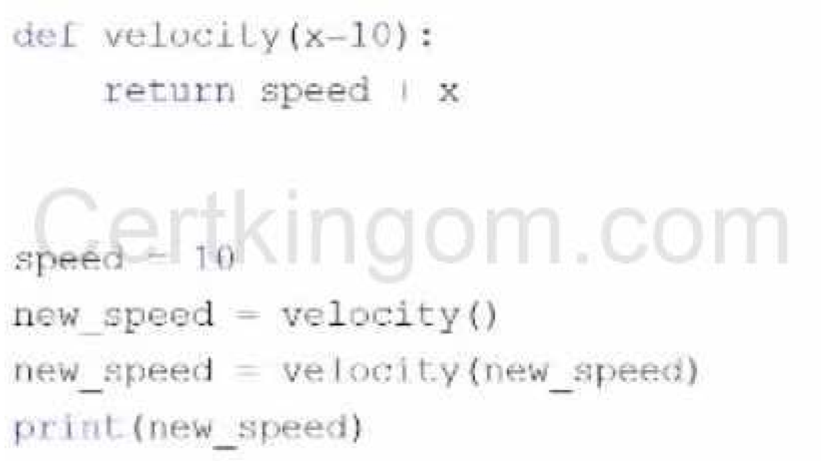
QUESTION 28
Which of the following are the names of Python passing argument styles?
(Select two answers.)

A. keyword
B. reference
C. indicatory
D. positional

Answer: A, D

Explanation:
Keyword arguments are arguments that are specified by using the name of the parameter, followed by an equal sign and the value of the argument. For example, print (sep='-', end='!') is a function call with keyword arguments. Keyword arguments can be used to pass arguments in any order, and to provide default values for some arguments1.
Positional arguments are arguments that are passed in the same order as the parameters of the function definition. For example, print ('Hello', 'World') is a function call with positional arguments. Positional arguments must be passed before any keyword arguments, and they must match the number and type of the parameters of the function2.
Reference: 1: 5 Types of Arguments in Python Function Definitions | Built In 2: python - Whats the pythonic way to pass arguments between functions ¦

QUESTION 29
What is the expected result of the following code?



```
def velocity(x=10):
    return speed + x


speed = 10
new_speed = velocity()
new_speed = velocity(new_speed)
print(new_speed)
```

A. The code is erroneous and cannot be run.
B. 20
C. 10
D. 30

Answer: A

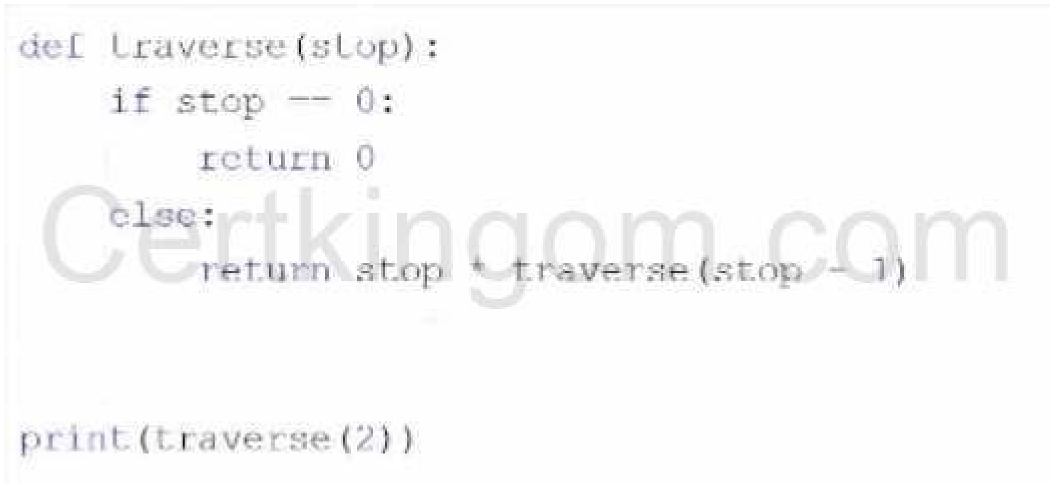
Explanation:
The code snippet that you have sent is trying to use the global keyword to access and modify a global variable inside a function. The code is as follows:
speed = 10 def velocity(): global speed speed = speed + 10 return speed
print(velocity())
The code starts with creating a global variable called œspeed and assigning it the value 10. A global variable is a variable that is defined outside any function and can be accessed by any part of the code. Then, the code defines a function called œvelocity that takes no parameters and returns the value of œspeed after adding 10 to it. Inside the function, the code uses the global keyword to declare that it wants to use the global variable œspeed , not a local one. A local variable is a variable that is defined inside a function and can only be accessed by that function. The global keyword allows the function to modify the global variable, not just read it. Then, the code adds 10 to the value of œspeed and returns it. Finally, the code calls the function œvelocity and prints the result.
However, the code has a problem. The problem is that the code uses the global keyword inside the function, but not outside. The global keyword is only needed when you want to modify a global variable inside a function, not when you want to create or access it outside a function. If you use the global keyword outside a function, you will get a SyntaxError exception, which is an error that occurs when the code does not follow the rules of the Python language. The code does not handle the exception, and therefore it will terminate with an error message.
The expected result of the code is an unhandled exception, because the code uses the global keyword incorrectly. Therefore, the correct answer is
A. The code is erroneous and cannot be run.
Reference: Python Global Keyword - W3SchoolsPython Exceptions: An Introduction " Real Python

The code is erroneous because it is trying to call the œvelocity   function without passing any parameter, which will raise a TypeError exception. The œvelocity   function requires one parameter œx , which is used to calculate the return value of œspeed   multiplied by œx . If no parameter is passed, the function will not know what value to use for œx .
The code is also erroneous because it is trying to use the œnew_speed   variable before it is defined. The œnew_speed   variable is assigned the value of 20 after the first function call, but it is used as a parameter for the second function call, which will raise a NameError exception. The variable should be defined before it is used in any expression or function call.
Therefore, the code will not run and will not produce any output.
The correct way to write the code would be:

```
# Define the speed variable
speed = 10
# Define the velocity function
def velocity(x):
return speed * x
# Define the new_speed variable
new_speed = 20
# Call the velocity function with new_speed as a parameter
print(velocity(new_speed))
```

Copy
This code will print 200, which is the result of 10 multiplied by 20.
Reference:
[Python Programmer Certification (PCPP) “ Level 1]
[Python Programmer Certification (PCPP) “ Level 2]
[Python Programmer Certification (PCPP) “ Level 3]
[Python: Built-in Exceptions]
[Python: Defining Functions]
[Python: More on Variables and Printing]

QUESTION 30
What is the expected output of the following code?



```
def traverse(stop):
    if stop == 0:
        return 0
    else:
        return stop * traverse(stop - 1)


print(traverse(2))
```

A. 2
B. 0
C. 3
D. 1

Answer: D

Explanation:
The code snippet that you have sent is using the count method to count the number of occurrences of a value in a list. The code is as follows:
my_list = [1, 2, 3, 4, 5] print(my_list.count(1))
The code starts with creating a list called œmy_list   that contains the numbers 1, 2, 3, 4, and 5. Then, it uses the print function to display the result of calling the count method on the list with the argument 1. The count method is used to return the number of times a value appears in a list. For example, my_list.count(1) returns 1, because 1 appears once in the list.
The expected output of the code is 1, because the code prints the number of occurrences of 1 in the list. Therefore, the correct answer is D. 1.
Reference: Python List count() Method - W3Schools