1. Who developed Python Programming Language?
a) Wick van Rossum
b) Rasmus Lerdorf
c) Guido van Rossum
d) Niene Stom

Answer: c
Explanation: Python language is designed by a Dutch programmer Guido van Rossum in the Netherlands.

2. Which type of Programming does Python support?
a) object-oriented programming
b) structured programming
c) functional programming
d) all of the mentioned

Answer: d
Explanation: Python is an interpreted programming language, which supports object-oriented, structured, and functional programming.

3. Is Python case sensitive when dealing with identifiers?
a) no
b) yes
c) machine dependent
d) none of the mentioned

Answer: b
Explanation: Case is always significant while dealing with identifiers in python.

4. Which of the following is the correct extension of the Python file?
a) .python
b) .pl
c) .py
d) .p

Answer: c
Explanation: '.py' is the correct extension of the Python file. Python programs can be written in any text editor. To save these programs we need to save in files with file extension '.py'.

5. Is Python code compiled or interpreted?
a) Python code is both compiled and interpreted
b) Python code is neither compiled nor interpreted
c) Python code is only compiled
d) Python code is only interpreted

Answer: a
Explanation: Many languages have been implemented using both compilers and interpreters, including C, Pascal, and Python.

6. All keywords in Python are in _____
a) Capitalized
b) lower case
c) UPPER CASE
d) None of the mentioned

Answer: d
Explanation: Most keywords are in lowercase, but some like True, False, and None are capitalized.

7. What will be the value of the following Python expression?

```
print(4 + 3 % 5)
```

a) 7
b) 2
c) 4
d) 1

Answer: a
Explanation: In Python, the modulus operator % has higher precedence than addition +. So, the expression is evaluated as 4 + (3 % 5), which is 4 + 3 = 7.

8. Which of the following is used to define a block of code in Python language?

a) Indentation
b) Key
c) Brackets
d) All of the mentioned

Answer: a
Explanation: In Python, to define a block of code we use indentation. Indentation refers to whitespaces at the beginning of the line.

9. Which keyword is used for function in Python language?

a) Function
b) def
c) Fun
d) Define

Answer: b
Explanation: The def keyword is used to create, (or define) a function in python.

10. Which of the following character is used to give single-line comments in Python?

a) //
b) #
c) !
d) /*

Answer: b
Explanation: To write single-line comments in Python use the Hash character (#) at the beginning of the line. It is also called number sign or pound sign. To write multi-line comments, close the text between triple quotes.
Example: “”” comment
text “””

11. What will be the output of the following Python code?

```
i = 1
while True:
    if i%3 == 0:
        break
    print(i)

    i + = 1
```

a) 1 2 3
b) SyntaxError
c) 1 2
d) none of the mentioned

Answer: b
Explanation: The output will be a SyntaxError because i + = 1 is invalid syntax in Python. There should be no space between + and =. The correct syntax is i += 1.

12. Which of the following functions can help us to find the version of python that we are currently working on?
a) sys.version(1)
b) sys.version(0)
c) sys.version()
d) sys.version

Answer: d
Explanation: The function sys.version can help us to find the version of python that we are currently working on. It also contains information on the build number and compiler used. For example, 3.5.2, 2.7.3 etc. this function also returns the current date, time, bits etc along with the version.
13. Python supports the creation of anonymous functions at runtime, using a construct called _____
a) pi
b) anonymous
c) lambda
d) none of the mentioned

Answer: c
Explanation: In Python, lambda functions are anonymous, meaning they don't have a name. They are defined using the lambda keyword and can take any number of arguments but only have one expression. Lambdas

are useful for creating small, throwaway functions quickly without formally defining them using def.

14. What is the order of precedence in python?

a) Exponential, Parentheses, Multiplication, Division, Addition, Subtraction

b) Exponential, Parentheses, Division, Multiplication, Addition, Subtraction

c) Parentheses, Exponential, Multiplication, Addition, Division, Subtraction

d) Parentheses, Exponential, Multiplication, Division, Addition, Subtraction

Answer: d

Explanation: Python follows the PEMDAS rule (similar to BODMAS): Parentheses, Exponentiation, Multiplication/Division, then Addition/Subtraction. Operators at the same level are evaluated left to right.

15. What will be the output of the following Python code snippet if x=1?

```
x<<2
```

a) 4

b) 2

c) 1

d) 8

Answer: a

Explanation: The binary form of 1 is 0001. The expression x<<2 implies we are performing bitwise left shift on x. This shift yields the value: 0100, which is the binary form of the number 4.

16. What does pip stand for python?

a) Pip Installs Python

b) Pip Installs Packages
c) Preferred Installer Program
d) All of the mentioned

Answer: c
Explanation: pip is a package manager for python. Which is also called Preferred Installer Program.

17. Which of the following is true for variable names in Python?
a) underscore and ampersand are the only two special characters allowed
b) unlimited length
c) all private members must have leading and trailing underscores
d) none of the mentioned

Answer: b
Explanation: Python allows variable names of unlimited length. Private members usually have only a leading underscore, not both leading and trailing. The ampersand (&) is not permitted in variable names; only the underscore (_) is allowed as a special character.

18. What are the values of the following Python expressions?

```
print(2**(3**2))
print((2**3)**2)
print(2**3**2)
```

a) 512, 64, 512
b) 512, 512, 512
c) 64, 512, 64
d) 64, 64, 64

Answer: a
Explanation: Expression 1 is evaluated as: 2**9, which is equal to 512.

Expression 2 is evaluated as 8**2, which is equal to 64. The last expression is evaluated as 2**(3**2). This is because the associativity of ** operator is from right to left. Hence the result of the third expression is 512.

19. Which of the following is the truncation division operator in Python?

a) |
b) //
c) /
d) %

Answer: b
Explanation: // is the operator for truncation division. It is called so because it returns only the integer part of the quotient, truncating the decimal part. For example: 20//3 = 6.

20. What will be the output of the following Python code?

```
l=[1, 0, 2, 0, 'hello', '', []]
print(list(filter(bool, l)))
```

a) [1, 0, 2, ‘hello’, ”, []]
b) Error
c) [1, 2, ‘hello’]
d) [1, 0, 2, 0, ‘hello’, ”, []]

Answer: c
Explanation: The function filter(bool, l) removes all false elements from the list l, such as 0, ”, and []. The remaining true elements — 1, 2, and ‘hello’ — are returned as a new list.

21. Which of the following functions is a built-in function in python?

a) factorial()
b) print()

c) seed()
d) sqrt()

Answer: b
Explanation: The function seed is a function which is present in the random module. The functions sqrt and factorial are a part of the math module. The print function is a built-in function which prints a value directly to the system output.

22. Which of the following is the use of id() function in python?

a) Every object doesn't have a unique id
b) Id returns the identity of the object
c) All of the mentioned
d) None of the mentioned

Answer: b
Explanation: The id() function in Python returns the identity of an object. This identity is a unique integer (or memory address) that remains constant for the object during its lifetime. Every object in Python has a unique id, which helps in comparing object references.

23. The following python program can work with ____ parameters.

```
def f(x):
    def f1(*args, **kwargs):
        print("Sanfoundry")
        return x(*args, **kwargs)
    return f1
```

a) any number of
b) 0
c) 1
d) 2

Answer: a
Explanation: The decorator function f defines f1 which uses *args and **kwargs to accept any number of positional and keyword arguments. This allows the decorated function to work with any number of parameters, making the decorator flexible.

24. What will be the output of the following Python function?

```
print(min(max(False,-3,-4), 2,7))
```

a) -4
b) -3
c) 2
d) False

Answer: d
Explanation: The max(False, -3, -4) evaluates to 0 because False is treated as 0, and 0 is greater than -3 and -4. Then, min(0, 2, 7) returns 0. Since 0 is equivalent to False, the output is False.

25. Which of the following is not a core data type in Python programming?

a) Tuples
b) Lists
c) Class
d) Dictionary

Answer: c
Explanation: Class is a user-defined data type.

26. What will be the output of the following Python expression if x=56.236?

```
print("%.2f"%x)
```

a) 56.236
b) 56.23

c) 56.0000
d) 56.24

Answer: d
Explanation: The expression shown above rounds off the given number to the number of decimal places specified. Since the expression given specifies rounding off to two decimal places, the output of this expression will be 56.24. Had the value been x=56.234 (last digit being any number less than 5), the output would have been 56.23.

27. Which of these is the definition for packages in Python?

a) A set of main modules
b) A folder of python modules
c) A number of files containing Python definitions and statements
d) A set of programs making use of Python modules

28. What will be the output of the following Python function?

```
print(len(["hello",2, 4, 6]))
```

a) Error
b) 6
c) 4
d) 3

Answer: c
Explanation: The len() function returns the number of elements in the list, regardless of their types. In this case, the list [“hello”, 2, 4, 6] contains four elements, so len() returns 4.

29. What will be the output of the following Python code?

```
x = 'abcd'
for i in x:
   print(i.upper())
```

a)
a
B
C
D
b) a b c d
c) error
d)
A
B
C
D

Answer: d
Explanation: In this code, x = ‘abcd’ is iterated over, and for each character i, i.upper() is called. The upper() method returns a new string where all characters are converted to uppercase. Each uppercase character is then printed on a new line. Therefore, the output is A, B, C, D, one per line.

30. What is the order of namespaces in which Python looks for an identifier?

a) Python first searches the built-in namespace, then the global namespace and finally the local namespace
b) Python first searches the built-in namespace, then the local namespace and finally the global namespace
c) Python first searches the local namespace, then the global namespace and finally the built-in namespace
d) Python first searches the global namespace, then the local namespace and finally the built-in namespace

Answer: c

Explanation: When Python encounters an identifier (like a variable or function name), it follows the LEGB rule to resolve it. It first looks in the Local namespace (inside the current function), then in the Enclosing namespace (if it's a nested function), followed by the Global namespace (top-level of the module), and finally the Built-in namespace (predefined functions like len(), sum(), etc.). So, the correct search order for namespaces is: local → global → built-in.

31. What will be the output of the following Python code snippet?

```
for i in [1, 2, 3, 4][::-1]:
    print(i, end=' ')
```

a) 4 3 2 1

b) error

c) 1 2 3 4

d) none of the mentioned

Answer: a

Explanation: The expression [1, 2, 3, 4][::-1] uses slicing with a step of -1 to reverse the list. So the list becomes [4, 3, 2, 1]. The for loop iterates over this reversed list and prints each element, with end=' ' ensuring the output is on one line with spaces in between.

32. What will be the output of the following Python statement?

```
print("a"+"bc")
```

a) bc

b) abc

c) a

d) bca

Answer: b
Explanation: In Python, the + operator is used for string concatenation. “a” + “bc” joins the two strings together into a single string “abc”.

33. Which function is called when the following Python program is executed?

```
f = foo()
format(f)
```

a) str()
b) format()
c) __str__()
d) __format__()

Answer: d
Explanation: When format(f) is executed, Python internally invokes the special method f.__format__(). This method controls how the object is formatted. The __str__() method is used by str(f), not format(f).

34. Which one of the following is not a keyword in Python language?

a) pass
b) eval
c) assert
d) nonlocal

Answer: b
Explanation: eval can be used as a variable.

35. What will be the output of the following Python code?

```
class tester:
    def __init__(self, id):
        self.id = str(id)
        id="224"
```

```
temp = tester(12)
print(temp.id)
```

a) 12
b) 224
c) None
d) Error

Answer: a
Explanation: When the tester class is instantiated with temp = tester(12), the __init__ method is called. The id argument is passed as 12, and inside the __init__ method, self.id is assigned the string value of id, which is “12”. However, the local variable id is reassigned to “224”, but this change does not affect self.id, which retains the value “12”.

36. What will be the output of the following Python program?

```
def foo(x):
    x[0] = ['def']
    x[1] = ['abc']
    return id(x)
q = ['abc', 'def']
print(id(q) == foo(q))
```

a) Error
b) None
c) False
d) True

Answer: d
Explanation: The list q is passed by reference to the function foo, so both x and q refer to the same object in memory. The id() function returns the memory address of the object, which remains unchanged. Therefore, id(q) == foo(q) evaluates to True.

37. Which module in the python standard library parses options received from the command line?

a) getarg
b) getopt
c) main
d) os

Answer: b
Explanation: The getopt module in Python's standard library is used to parse command-line options and arguments. It allows the script to accept flags and parameters (like -h or –help) similar to those in shell scripts. For example:

```
import getopt, sys
opts, args = getopt.getopt(sys.argv[1:], "h", ["help"])
```

This line parses short option -h and long option –help.

38. What will be the output of the following Python program?

```
z=set('abc')
z.add('san')
z.update(set(['p', 'q']))
print(z)
```

a) {'a', 'c', 'c', 'p', 'q', 's', 'a', 'n'}
b) {'abc', 'p', 'q', 'san'}
c) {'a', 'b', 'c', 'p', 'q', 'san'}
d) {'a', 'b', 'c', ['p', 'q'], 'san}

Answer: c
Explanation: The code shown first adds the element 'san' to the set z. The set z is then updated and two more elements, namely, 'p' and 'q' are added to it. Hence the output is: {'a', 'b', 'c', 'p', 'q', 'san'}

39. What arithmetic operators cannot be used with strings in Python?
a) *
b) –
c) +
d) All of the mentioned

Answer: b
Explanation: + is used to concatenate and * is used to multiply strings.
40. What will be the output of the following Python code?

```
print("abc. DEF".capitalize())
```

a) Abc. def
b) abc. def
c) Abc. Def
d) ABC. DEF

Answer: a
Explanation: The first letter of the string is converted to uppercase and the others are converted to lowercase.
41. Which of the following statements is used to create an empty set in Python?
a) ( )
b) [ ]
c) { }
d) set()

Answer: d
Explanation: { } creates a dictionary not a set. Only set() creates an empty set.
42. What will be the value of 'result' in following Python program?

```
list1 = [1,2,3,4]
```

```
list2 = [2,4,5,6]
list3 = [2,6,7,8]
result = list()
result.extend(i for i in list1 if i not in (list2+list3) and i not in result)
result.extend(i for i in list2 if i not in (list1+list3) and i not in result)
result.extend(i for i in list3 if i not in (list1+list2) and i not in result)
print(result)
```

a) [1, 3, 5, 7, 8]
b) [1, 7, 8]
c) [1, 2, 4, 7, 8]
d) error

Answer: a
Explanation: Here, 'result' is a list which is extending three times. When first time 'extend' function is called for 'result', the inner code generates a generator object, which is further used in 'extend' function. This generator object contains the values which are in 'list1' only (not in 'list2' and 'list3').
Same is happening in second and third call of 'extend' function in these generator object contains values only in 'list2' and 'list3' respectively.
So, 'result' variable will contain elements which are only in one list (not more than 1 list).

43. To add a new element to a list we use which Python command?

a) list1.addEnd(5)
b) list1.addLast(5)
c) list1.append(5)
d) list1.add(5)

Answer: c
Explanation: We use the function append to add an element to the list.

44. What will be the output of the following Python code?

```
print('*', "abcde".center(6), '*', sep='')
```

a) * abcde *
b) *abcde *
c) * abcde*
d) * abcde *

Answer: b
Explanation: Padding is done towards the right-hand-side first when the final string is of even length.

45. What will be the output of the following Python code?

```
list1 = [1, 3]
list2 = list1
list1[0] = 4
print(list2)
```

a) [1, 4]
b) [1, 3, 4]
c) [4, 3]
d) [1, 3]

Answer: c
Explanation: In the code, list2 = list1 creates a reference to the same list in memory. So when list1[0] is changed to 4, list2 also reflects that change. The output is [4, 3].

46. Which one of the following is the use of function in python?

a) Functions don’t provide better modularity for your application
b) you can’t also create your own functions
c) Functions are reusable pieces of programs
d) All of the mentioned

Answer: c
Explanation: Functions are reusable pieces of programs. They allow you to give a name to a block of statements, allowing you to run that block using the specified name anywhere in your program and any number of times.

47. Which of the following Python statements will result in the output: 6?

```
A = [[1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]]
```

a) A[2][1]
b) A[1][2]
c) A[3][2]
d) A[2][3]

Answer: b
Explanation: The output that is required is 6, that is, row 2, item 3. This position is represented by the statement: A[1][2].

48. What is the maximum possible length of an identifier in Python?

a) 79 characters
b) 31 characters
c) 63 characters
d) none of the mentioned

Answer: d
Explanation: In Python, identifiers can be of any length. There is no fixed maximum, though extremely long names are not practical.

49. What will be the output of the following Python program?

```
i = 0
while i < 5:
```

```
    print(i)
    i += 1
    if i == 3:
        break
else:
    print(0)
```

a) error
b)
0
1
2
3
0
c)
0
1
2
d) none of the mentioned

Answer: c
Explanation: In this code, the while loop iterates from i = 0 to i = 2, printing the values of i. When i becomes 3, the if i == 3 condition is met, and the break statement is executed, which terminates the loop early. Since the loop was terminated using break, the else block is not executed. Therefore, the output is 0, 1, and 2.

50. What will be the output of the following Python code?

```
x = 'abcd'
for i in range(len(x)):
    print(i)
```

a) error
b) 1 2 3 4
c) a b c d
d) 0 1 2 3

Answer: d
Explanation: i takes values 0, 1, 2 and 3.

51. What are the two main types of functions in Python?

a) System function
b) Custom function
c) Built-in function & User defined function
d) User function

Answer: c
Explanation: Built-in functions and user defined ones. The built-in functions are part of the Python language. Examples are: dir(), len() or abs(). The user defined functions are functions created with the def keyword.

52. What will be the output of the following Python program?

```
def addItem(listParam):
    listParam += [1]

mylist = [1, 2, 3, 4]
addItem(mylist)
print(len(mylist))
```

a) 5
b) 8
c) 2
d) 1

Answer: a
Explanation: The function addItem uses += [1] to modify the list passed to it. Since lists are mutable and passed by reference, mylist is modified directly. After appending 1, it becomes [1, 2, 3, 4, 1], so its length is 5.

53. Which of the following is a Python tuple?

a) {1, 2, 3}
b) {}
c) [1, 2, 3]
d) (1, 2, 3)

Answer: d
Explanation: A tuple in Python is an immutable sequence type and is defined using round brackets ( ). For example, (1, 2, 3) is a tuple.

54. What will be the output of the following Python code snippet?

```
z=set('abc$de')
print('a' in z)
```

a) Error
b) True
c) False
d) No output

Answer: b
Explanation: The code shown above is used to check whether a particular item is a part of a given set or not. Since ‘a’ is a part of the set z, the output is true. Note that this code would result in an error in the absence of the quotes.

55. What will be the output of the following Python expression?

```
print(round(4.576))
```

a) 4
b) 4.6
c) 5
d) 4.5

Answer: c
Explanation: The round() function rounds the number to the nearest integer by default. Since 4.576 is closer to 5 than 4, round(4.576) returns 5. Therefore, the output is 5.

56. Which of the following is a feature of Python DocString?

a) In Python all functions should have a docstring
b) Docstrings can be accessed by the __doc__ attribute on objects
c) It provides a convenient way of associating documentation with Python modules, functions, classes, and methods
d) All of the mentioned

Answer: d
Explanation: Python has a nifty feature called documentation strings, usually referred to by its shorter name docstrings. DocStrings are an important tool that you should make use of since it helps to document the program better and makes it easier to understand.

57. What will be the output of the following Python code?

```
print("Hello {0[0]} and {0[1]}".format(('foo', 'bin')))
```

a) Hello (‘foo’, ‘bin’) and (‘foo’, ‘bin’)
b) Error
c) Hello foo and bin
d) None of the mentioned

Answer: c
Explanation: The output of the code is Hello foo and bin. Here, the format

string uses index-based access to retrieve elements from the tuple passed as a single argument. {0[0]} and {0[1]} access the first and second elements of the tuple respectively.

58. What is output of print(math.pow(3, 2))?

a) 9.0
b) None
c) 9
d) None of the mentioned

Answer: a
Explanation: math.pow() returns a floating point number.

59. Which of the following is the use of id() function in python?

a) Every object in Python doesn't have a unique id
b) In Python Id function returns the identity of the object
c) None of the mentioned
d) All of the mentioned

Answer: b
Explanation: Each object in Python has a unique id. The id() function returns the object's id.

60. What will be the output of the following Python code?

```
x = [[0], [1]]
print((' '.join(list(map(str, x))),))
```

a) 01
b) [0] [1]
c) ('01')
d) ('[0] [1]',)

Answer: d
Explanation: In this code, map(str, x) converts each inner list [0] and [1]

into strings: “[0]” and “[1]”. Then ‘ ‘.join(…) creates the string “[0] [1]”. Finally, it is wrapped in a tuple using the comma syntax, resulting in (‘[0] [1]’,).

61. The process of pickling in Python includes _____

a) conversion of a Python object hierarchy into byte stream
b) conversion of a datatable into a list
c) conversion of a byte stream into Python object hierarchy
d) conversion of a list into a datatable

Answer: a
Explanation: Pickling is the process of serializing a Python object, that is, conversion of a Python object hierarchy into a byte stream. The reverse of this process is known as unpickling.

62. What will be the output of the following Python code?

```
def foo():
    try:
        return 1
    finally:
        return 2
k = foo()
print(k)
```

a) error, there is more than one return statement in a single try-finally block
b) 3
c) 2
d) 1

Answer: c
Explanation: In Python, if both the try block and the finally block contain

return statements, the return in the finally block overrides the one in the try block. So, even though return 1 is in the try, the function ends up returning 2 because of the finally block.

1. Is Python case sensitive when dealing with identifiers?
a) yes
b) no
c) machine dependent
d) none of the mentioned

Answer: a
Explanation: Yes, Python is case sensitive. For example, Variable, variable, and VARIABLE are all treated as different identifiers.
2. What is the maximum possible length of an identifier?
a) 31 characters
b) 63 characters
c) 79 characters
d) none of the mentioned

Answer: d
Explanation: In Python, identifiers can be of any length. There is no fixed maximum, though extremely long names are not practical.
3. Which of the following is not allowed in Python?
a) _a = 1
b) __a = 1
c) __str__ = 1
d) none of the mentioned

Answer: d
Explanation: All the given statements are valid in Python and will run

without errors. However, using names like __str__ can reduce readability or interfere with built-in functionality.

4. Which of the following is an invalid variable?
a) my_string_1
b) 1st_string
c) foo
d) _

Answer: b
Explanation: Variable names in Python cannot start with a digit. Since 1st_string begins with the digit 1, it is invalid. The other options follow Python's variable naming rules, where names can start with letters or underscores and can include digits after the first character.
5. Why are local variable names beginning with an underscore discouraged?
a) they are used to indicate a private variables of a class
b) they confuse the interpreter
c) they are used to indicate global variables
d) they slow down execution

Answer: a
Explanation: As Python has no concept of private variables, leading underscores are used to indicate variables that must not be accessed from outside the class.
6. Which of the following is not a keyword in Python?
a) eval
b) assert
c) nonlocal

d) pass

Answer: a
Explanation: eval is a built-in function, not a keyword. The others (assert, nonlocal, and pass) are Python keywords.

7. All keywords in Python are in _____

a) lower case
b) UPPER CASE
c) Capitalized
d) None of the mentioned

Answer: d
Explanation: Most keywords are in lowercase, but some like True, False, and None are capitalized.

8. Which of the following is true for variable names in Python?

a) unlimited length
b) all private members must have leading and trailing underscores
c) underscore and ampersand are the only two special characters allowed
d) none of the mentioned

Answer: a
Explanation: Python allows variable names of unlimited length. Private members usually have only a leading underscore, not both leading and trailing. The ampersand (&) is not permitted in variable names; only the underscore (_) is allowed as a special character.

9. Which of the following is an invalid statement?

a) abc = 1,000,000
b) a b c = 1000 2000 3000
c) a,b,c = 1000, 2000, 3000

d) a_b_c = 1,000,000

Answer: b
Explanation: In Python, variable names cannot have spaces between them, so a b c = 1000 2000 3000 is invalid syntax.

10. Which of the following cannot be a variable name in Python?

a) __init__
b) in
c) it
d) on

Answer: b
Explanation: **in** is a reserved keyword in Python used for membership testing and loops, so it cannot be used as a variable name. The other options (__init__, it, on) are valid identifiers.

1. Which is the correct operator for power(x^y) in Python?

a) x^y
b) x**y
c) x^^y
d) None of the mentioned

Answer: b
Explanation: In python, the power operator is x**y. For example, 2**3=8 results in 8.

2. Which one of these is floor division?

a) /
b) //
c) %
d) None of the mentioned

Answer: b
Explanation: The // operator in Python performs floor division, which returns the largest integer less than or equal to the division result. For example, 5 // 2 results in 2, not 2.5. The / operator, on the other hand, performs true division and returns a float (5 / 2 = 2.5). To get the integer result without the fractional part, use //.

3. What is the order of precedence in python?

i) Parentheses
ii) Exponential
iii) Multiplication
iv) Division
v) Addition
vi) Subtraction
a) i,ii,iii,iv,v,vi
b) ii,i,iii,iv,v,vi
c) ii,i,iv,iii,v,vi
d) i,ii,iii,v,vi,iv

Answer: a
Explanation: Python follows the PEMDAS rule (similar to BODMAS): Parentheses, Exponentiation, Multiplication/Division, then Addition/Subtraction. Operators at the same level are evaluated left to right.

4. What is the answer to this expression, 22 % 3 is?

a) 7
b) 1
c) 0
d) 5

Answer: b
Explanation: The modulus operator (%) returns the remainder when one number is divided by another. In this case, 22 % 3 gives the remainder 1 (since 22 divided by 3 is 7 with a remainder of 1).

5. Mathematical operations be directly performed on a string in Python without conversion.

a) True
b) False

Answer: b
Explanation: In Python, you cannot perform mathematical operations like addition, subtraction, multiplication, or division directly on a string. To perform arithmetic operations, you must first convert the string to a numerical type (e.g., int or float). Otherwise, Python will raise a TypeError.

6. Operators with the same precedence are evaluated in which manner?

a) Left to Right
b) Right to Left
c) Can't say
d) None of the mentioned

Answer: c
Explanation: In Python language, most of the operators with the same precedence are evaluated with left to right such as a lot of binary operators. However, exponent operator, unary operators, ternary, and assignment operators are evaluated from right to left.

7. What is the output of this expression, 3*13?**

a) 27
b) 9

c) 3
d) 1

Answer: c
Explanation: In Python, the exponentiation operator (**) has higher precedence than multiplication (*). Therefore, the expression is evaluated as 1 ** 3 first, which equals 1. Then, 3 * 1 is evaluated, giving the final result of 3.

8. Which one of the following has the same precedence level?

a) Addition and Subtraction
b) Multiplication, Division and Addition
c) Multiplication, Division, Addition and Subtraction
d) Addition and Multiplication

Answer: a
Explanation: In Python, Addition (+) and Subtraction (-) operators have the same precedence level. Similarly, Multiplication (*) and Division (/) operators also share the same precedence level, but they are evaluated before addition and subtraction due to their higher precedence.

9. The expression int(x) implies that the value of variable x is converted to integer.

a) True
b) False

Answer: a
Explanation: The expression int(x) explicitly converts the value of variable x to the integer data type. This is an example of explicit type conversion in Python.

10. Which one of the following has the highest precedence in the expression?

a) Exponential
b) Addition
c) Multiplication
d) Parentheses

Answer: d
Explanation: The Parentheses () have the highest precedence in Python, followed by Exponentiation **, then Multiplication and Division, and finally Addition and Subtraction. This order is known as PEMDAS (Parentheses, Exponentiation, Multiplication/Division, Addition/Subtraction).

1. Which of the following is not a core data type in Python?

a) List
b) Dictionary
c) Tuple
d) Class

Answer: d
Explanation: Classes are user-defined data types, while lists, dictionaries, and tuples are built-in (core) data types in Python.

2. Given a function that does not return any value, what is the default return value when it is executed in the Python shell?

a) int
b) bool
c) void
d) None

Answer: d
Explanation: If a function does not explicitly return a value, Python returns None by default. The type of None is NoneType.

3. What will be the output of the following Python code?

```
str = "hello"
print(str[:2])
```

a) he
b) lo
c) olleh
d) hello

Answer: a
Explanation: The code str[:2] slices the string from the beginning (index 0) up to, but not including, index 2. This results in the substring “he”.

4. Which of the following will run without errors?

a) round(45.8)
b) round(6352.898,2,5)
c) round() d) round(7463.123,2,1)

Answer: a
Explanation: The round() function in Python takes one or two values. It gives an error if more than two values are passed. You can type help(round) in the Python shell to see more about how it works.

5. What is the return type of the id() function in Python?

a) int
b) float
c) bool
d) dict

Answer: a
Explanation: The id() function returns a unique integer that shows the

memory address of an object. You can use help(id) in the Python shell to learn more.

6. In Python, variable types are not explicitly declared—they are inferred at runtime. Consider the following incomplete operation:

```
x = 13 ? 2
```

The objective is to ensure that x has an integer value. Select all options that achieve this (Python 3.x):

a) x = 13 // 2
b) x = int(13 / 2)
c) x = 13 % 2
d) All of the mentioned

Answer: d
Explanation:

- **x = 13 // 2** → Floor division returns **6** (an integer).
- **x = int(13 / 2)** → Regular division returns **6.5**, and **int()** converts it to **6**.
- **x = 13 % 2** → Modulus returns the remainder **1**, which is also an integer.

All three assign an integer value to x, so all are valid in Python 3.x.

7. What error occurs when you execute the following Python code snippet?

apple = mango
a) SyntaxError
b) NameError
c) ValueError
d) TypeError

Answer: b
Explanation: The variable mango is not defined before it is used, so Python raises a NameError indicating that the name is not recognized.

8. What will be the output of the following Python code snippet?

```
def example(a):
   a = a + '2'
    a = a*2
   return a
example("hello")
```

a) Indentation Error
b) Cannot perform mathematical operation on strings
c) hello2
d) hello2hello2

Answer: a
Explanation: The line a = a*2 has an extra space at the beginning, which makes the indentation uneven. Python needs all lines in a block to be properly indented. This causes an Indentation Error, and the code won't run.

9. What data type is the object below?

L = [1, 23, 'hello', 1]
a) list
b) dictionary
c) array
d) tuple

Answer: a
Explanation: The variable L is enclosed in square brackets [], which defines a list in Python. Lists can hold multiple data types, such as integers and strings.

10. In Python, which core data type is used to store values in the form of key–value pairs?

a) list
b) tuple
c) class
d) dictionary

Answer: d
Explanation: A dictionary is a built-in Python data type that stores data in the form of key–value pairs. It allows fast lookups and efficient data organization based on unique keys.

11. Which of the following will cause a SyntaxError in Python?

a) ‘”Once upon a time…”, she said.’
b) “He said, ‘Yes!'”
c) ‘3\’
d) ”’That’s okay”’

Answer: c
Explanation: ‘3\’ is not valid because the backslash escapes the closing quote, but there is no character after it. This makes Python throw a SyntaxError.

12. The following is displayed by a print function call. Select all of the function calls that result in this output.

```
tom
dick
harry
```

a)

```
print('''tom
\ndick
\nharry''')
```

b) print(”’tomdickharry”’)
c) print(‘tom\ndick\nharry’)
d)
print('tom
dick
harry')

Answer: c
Explanation: The \n adds a new line. So, print(‘tom\ndick\nharry’) will display the words on separate lines.

13. What is the average value of the following Python code snippet?

```
grade1 = 80
grade2 = 90
average = (grade1 + grade2) / 2
print(average)
```

a) 85.0
b) 85.1
c) 95.0
d) 95.1

Answer: a
Explanation: The expression (grade1 + grade2) / 2 becomes (80 + 90) / 2 = 170 / 2, which evaluates to 85.0. In Python 3, the / operator performs floating-point division, so the result includes a decimal.

14. Which of the following will print this output?
hello-how-are-you
a) print(‘hello’, ‘how’, ‘are’, ‘you’)
b) print(‘hello’, ‘how’, ‘are’, ‘you’ + ‘-‘ * 4)

c) print(‘hello-‘ + ‘how-are-you’)
d) print(‘hello’ + ‘-‘ + ‘how’ + ‘-‘ + ‘are’ + ‘you’)

Answer: c
Explanation: The code:

```
print('hello-' + 'how-are-you')
```

correctly concatenates the strings with hyphens, resulting in the output hello-how-are-you. Other variations either insert spaces or miss a hyphen between words.

15. What is the return type of trunc() in Python?

a) int
b) bool
c) float
d) None

Answer: a
Explanation: The trunc() function removes the decimal part of a number and returns an integer. For example, trunc(4.7) returns 4.

1. What is the output of print 0.1 + 0.2 == 0.3?

a) True
b) False
c) Machine dependent
d) Error

Answer: b
Explanation: Due to the limitations of floating-point representation in binary, neither 0.1, 0.2, nor 0.3 can be represented precisely. This leads to a small rounding error when adding 0.1 and 0.2, causing the result to

be slightly different from 0.3. Therefore, 0.1 + 0.2 == 0.3 evaluates to False.

2. Which of the following is not a complex number?

a) k = 2 + 3j
b) k = complex(2, 3)
c) k = 2 + 3l
d) k = 2 + 3J

Answer: c
Explanation: In Python, complex numbers are represented using j or J (e.g., 2 + 3j or complex(2, 3)). However, l (or L) is used to denote a long integer in some contexts, not a complex number. Therefore, k = 2 + 3l is not a complex number.

3. What is the type of inf?

a) Boolean
b) Integer
c) Float
d) Complex

Answer: c
Explanation: In Python, inf represents infinity, which is a special case of floating-point numbers. It can be created using float('inf') or directly as inf. Thus, the type of inf is float.

4. What does ~4 evaluate to?

a) -5
b) -4
c) -3
d) +3

Answer: a
Explanation: In Python, the ~ operator is the bitwise NOT operator. It inverts the bits of the number. The expression ~x is equivalent to -(x + 1). For x = 4, ~4 evaluates to -5.

5. What does ~~~~~~5 evaluate to?

a) +5
b) -11
c) +11
d) -5

Answer: a
Explanation: The ~ operator inverts the bits of a number and is equivalent to -(x + 1). When applied twice (~~x), it cancels out, returning the original number. Since ~x is applied six times, it effectively results in the same value as x. Therefore, ~~~~~~5 evaluates to 5.

6. Which of the following is incorrect?

a) x = 30963
b) x = 0x4f5
c) x = 19023
d) x = 03964

Answer: d
Explanation: In Python, numbers starting with a 0 are considered octal numbers (base 8). However, octal numbers can only contain digits from 0 to 7. Since 9 is not a valid digit in an octal number, 03964 is incorrect.

7. What is the result of cmp(3, 1)?

a) 1
b) 0
c) True

d) False

Answer: a
Explanation: The cmp(x, y) function compares two values, returning 1 if x > y, 0 if x == y, and -1 if x < y. Since 3 > 1, the result of cmp(3, 1) is 1.

8. Which of the following is incorrect in Python?

a) float(‘inf’)
b) float(‘nan’)
c) float(’56’+’78’)
d) float(’12+34’)

Answer: d
Explanation: In Python, float() can convert strings representing valid floating-point numbers, such as ‘inf’, ‘nan’, and ’56’ + ’78’ (which becomes ‘5678’). However, the string ’12+34’ includes the + sign, which is not valid for direct conversion to a float. Therefore, float(’12+34’) will raise a ValueError.

9. What is the result of round(0.5) – round(-0.5)?

a) 1.0
b) 2.0
c) 0.0
d) Value depends on Python version

Answer: d
Explanation: The behavior of the **round()** function is different in Python 2 and Python 3. In Python 2, it rounds off numbers away from 0 when the number to be rounded off is exactly halfway through. round(0.5) is 1 and round(-0.5) is -1 whereas in Python 3, it rounds off numbers towards nearest even number when the number to be rounded off is exactly halfway through.

In the above output, you can see that the round() functions on 0.5 and -0.5 are moving towards 0 and hence "**round(0.5) – (round(-0.5)) = 0 – 0 = 0**". Also note that the round(2.5) is 2 (which is an even number) whereas round(3.5) is 4 (which is an even number).

10. What does 3 ^ 4 evaluate to?

a) 81
b) 12
c) 0.75
d) 7

Answer: d
Explanation: In Python, the ^ operator is the binary XOR (exclusive OR) operator, not exponentiation. The result of 3 ^ 4 is the bitwise XOR of the binary representations of 3 (0011) and 4 (0100), which results in 0111 (decimal 7). Thus, 3 ^ 4 evaluates to 7.

1. The value of the expressions 4/(3*(2-1)) and 4/3*(2-1) is the same.

a) True
b) False

Answer: a
Explanation: Although the presence of parenthesis does affect the order of precedence, in the case shown above, it is not making a difference. The result of both of these expressions is 1.333333333. Hence the statement is true.

2. What will be the value of the following Python expression?

```
print(4 + 3 % 5)
```

a) 4
b) 7

c) 2
d) 0

Answer: b
Explanation: In Python, the modulus operator % has higher precedence than addition +. So, the expression is evaluated as 4 + (3 % 5), which is 4 + 3 = 7.

3. Evaluate the expression given below if A = 16 and B = 15.

```
A % B // A
```

a) 0.0
b) 0
c) 1.0 d) 1

Answer: b
Explanation: The expression A % B // A becomes 16 % 15 // 16, which simplifies to 1 // 16. Using floor division, the result is 0.

4. Which of the following operators has its associativity from right to left?

a) +
b) //
c) %
d) **

Answer: d
Explanation: Most operators in Python are left-associative, but the exponentiation operator ** is right-associative. So, 2 ** 3 ** 2 is evaluated as 2 ** (3 ** 2).

5. What will be the value of x in the following Python expression?

```
x = int(43.55+2/2)
print(x)
```

a) 43
b) 44
c) 22
d) 23

Answer: b
Explanation: The expression evaluates as int(43.55 + 1) → int(44.55), which results in 44 due to explicit conversion (truncates the decimal part).

6. What is the value of the following expression?

```
print(2+4.00, 2**4.0)
```

a) (6.0, 16.0)
b) (6.00, 16.00)
c) (6, 16)
d) (6.00, 16.0)

Answer: a
Explanation: In Python, 2 + 4.00 results in 6.0 (float), and 2 ** 4.0 results in 16.0 because any operation involving a float yields a float. So, the final result is (6.0, 16.0).

7. Which of the following is the truncation division operator?

a) /
b) %
c) //
d) |

Answer: c
Explanation: // is the truncation (floor) division operator in Python. It returns only the integer part of the result by discarding the decimal part. For example, 20 // 3 gives 6.

8. What are the values of the following Python expressions?

```
print(2**(3**2))
print((2**3)**2)
print(2**3**2)
```

a) 64, 512, 64
b) 64, 64, 64
c) 512, 512, 512
d) 512, 64, 512

Answer: d
Explanation: Expression 1 is evaluated as 2**9, which is equal to 512. Expression 2 is evaluated as 8**2, which is equal to 64. The last expression is evaluated as 2**(3**2). This is because the associativity of ** operator is from right to left. Hence the result of the third expression is 512.

9. What is the value of the following expression?

```
print(8/4/2, 8/(4/2))
```

a) 1.0 4.0
b) 1.0 1.0
c) 4.0 1.0
d) 4.0 4.0

Answer: a
Explanation: The first part 8/4/2 is evaluated left to right: 2.0 / 2 = 1.0. The second part 8/(4/2) becomes 8/2 = 4.0. Hence, the result is 1.0 4.0.

10. What is the value of the following expression?

```
print(float(22//3+3/3))
```
a) 8
b) 8.0
c) 8.3
d) 8.33

Answer: b
Explanation: The given expression print(float(22 // 3 + 3 / 3)) involves both integer and float operations. First, the floor division 22 // 3 evaluates to 7 because it divides 22 by 3 and discards the decimal part. Next, the division 3 / 3 evaluates to 1.0 since it is a floating-point operation. When these two values are added — 7 + 1.0 — the result is 8.0, because adding an integer and a float results in a float. Finally, applying float() to 8.0 does not change its value. Therefore, the final output of the expression is 8.0.

1. What will be the output of the following Python expression?

```
print(4.00/(2.0+2.0))
```
a) Error
b) 1.0
c) 1.00
d) 1

Answer: b
Explanation: The expression evaluates as 4.00 / 4.0, which results in 1.0. Therefore, the output is 1.0.

2. What will be the value of X in the following Python expression?

```
X = 2+9*((3*12)-8)/10
print(X)
```

a) 30.0
b) 30.8
c) 28.4
d) 27.2

Answer: d
Explanation: The expression shown above is evaluated as: 2+9*(36-8)/10, which simplifies to give 2+9*(2.8), which is equal to 2+25.2 = 27.2. Hence the result of this expression is 27.2.

3. Which of the following expressions involves coercion when evaluated in Python?

a) 4.7 – 1.5
b) 7.9 * 6.3
c) 1.7 % 2
d) 3.4 + 4.6

Answer: c
Explanation: Coercion is the implicit (automatic) conversion of operands to a common type. Coercion is automatically performed on mixed-type expressions. The expression 1.7 % 2 is evaluated as 1.7 % 2.0 (that is, automatic conversion of int to float).

4. What will be the output of the following Python expression?

```
print(24//6%3, 24//4//2)
```

a) 1 3
b) 0 3
c) 1 0
d) 3 1

Answer: a
Explanation: The expression 24 // 6 % 3, 24 // 4 // 2 evaluates to (1, 3)

because the operations are performed from left to right, with 24 // 6 giving 4, followed by 4 % 3 resulting in 1, and 24 // 4 giving 6, followed by 6 // 2 resulting in 3.

5. Which among the following list of operators has the highest precedence?

```
+, -, **, %, /, <<, >>, |
```

a) <<, >>
b) **
c) |
d) %

Answer: b
Explanation: The highest precedence is that of the exponentiation operator, that is of **.

6. What will be the value of the following Python expression?

```
print(float(4+int(2.39)%2))
```

a) 5.0
b) 5
c) 4.0
d) 4

Answer: c
Explanation: The above expression is an example of explicit conversion. It is evaluated as: float(4+int(2.39)%2) = float(4+2%2) = float(4+0) = 4.0. Hence the result of this expression is 4.0.

7. Which of the following expressions is an example of type conversion?

a) 4.0 + float(3)
b) 5.3 + 6.3
c) 5.0 + 3

d) 3 + 7

Answer: a
Explanation: Type conversion is nothing but explicit conversion of operands to a specific type. Options 5.3 + 6.3 and 5.0 + 3 are examples of implicit conversion whereas option 4.0 + float(3) is an example of explicit conversion or type conversion.

8. Which of the following expressions results in an error?

a) float(‘10’)
b) int(‘10’)
c) float(’10.8’)
d) int(’10.8’)

Answer: d
Explanation: All of the expressions involve explicit type conversion. However, int(‘10.8’) results in an error because the int() function cannot convert a string with a decimal point into an integer. To convert a string with a decimal, it must first be converted to a float.

9. What will be the value of the following Python expression?

```
print(4+2**5//10)
```

a) 3
b) 7
c) 77
d) 0

Answer: b
Explanation: The order of precedence is: **, //, +. The expression 4+2**5//10 is evaluated as 4+32//10, which is equal to 4+3 = 7. Hence the result of the expression is 7.

10. The expression 22**3 is evaluates as: (2**2)**3.**

a) True

b) False

Answer: b

Explanation: The expression (2**2)**3 results in 4**3 = 64, but the expression 2**2**3 in Python is evaluated as 2**(2**3), which equals 2**8 = 256. This is because the associativity of the exponentiation operator (**) in Python is from right to left, not left to right.

1. What will be the output of the following Python code snippet if x=1?

```
x<<2
```

a) 8

b) 1

c) 2

d) 4

Answer: d

Explanation: The binary form of 1 is 0001. The expression x << 2 performs a bitwise left shift on x, which shifts the bits two positions to the left. This results in 0100, which is the binary form of 4. Therefore, the result is 4.

2. What will be the output of the following Python expression?

```
print(bin(29))
```

a) 0b10111

b) 0b11101

c) 0b11111

d) 0b11011

Answer: b
Explanation: The binary representation of the number 29 is 11101. In Python, the bin() function returns the binary representation of an integer prefixed with 0b, so the output will be 0b11101.

3. What will be the value of x in the following Python expression, if the result of that expression is 2?

```
x>>2
```

a) 8
b) 4
c) 2
d) 1

Answer: a
Explanation: When the value of x is equal to 8 (1000), then x>>2 (bitwise right shift) yields the value 0010, which is equal to 2. Hence the value of x is 8.

4. What will be the output of the following Python expression?

```
print(int(1011))
```

a) 1011
b) 11
c) 13
d) 1101

Answer: a
Explanation: The result of the expression shown will be 1011. This is because we have not specified the base in this expression. Hence it automatically takes the base as 10.

5. To find the decimal value of 1111, that is 15, we can use the function:
a) int(1111,10)
b) int('1111',10)
c) int(1111,2)
d) int('1111',2)

Answer: d
Explanation: The expression int('1111', 2) converts the binary string '1111' to its decimal equivalent, which is 15. The expression int('1111', 10) would interpret '1111' as a decimal number and return 1111.
6. What will be the output of the following Python expression if x=15 and y=12?

```
x & y
```

a) b1101
b) 0b1101
c) 12
d) 1101

Answer: c
Explanation: The symbol '&' represents bitwise AND. This operation gives 1 if both bits are equal to 1, otherwise, it gives 0. The binary form of 15 is 1111 and that of 12 is 1100. Performing the bitwise AND operation results in 1100, which is equal to 12.
7. Which of the following expressions results in an error?
a) int(1011)
b) int('1011',23)
c) int(1011,2)
d) int('1011')

Answer: c
Explanation: The expression int(1011,2) results in an error because the first argument should be a string. If it was written as int('1011',2), there would be no error.

8. Which of the following represents the bitwise XOR operator?

a) &
b) ^
c) |
d) !

Answer: b
Explanation: The ^ operator represents the bitwise XOR operation. & represents bitwise AND, | represents bitwise OR, and ! is typically used as a logical NOT in many programming languages (not a bitwise operator).

9. What is the value of the following Python expression?

```
print(bin(0x8))
```

a) 0bx1000
b) 8
c) 1000
d) 0b1000

Answer: d
Explanation: The prefix 0x specifies that the value is in hexadecimal. When we convert 0x8 to its binary form, we get 0b1000.

10. What will be the output of the following Python expression?

```
print(0x35 | 0x75)
```

a) 115
b) 116
c) 117

d) 118

Answer: c
Explanation: The binary value of 0x35 is 110101 and that of 0x75 is 1110101. Performing the bitwise OR operation on these values results in 1110101, which is equal to 117. Hence, the result of the expression is 117.

1. It is not possible for the two's complement value to be equal to the original value in any case.

a) True
b) False

Answer: b
Explanation: In most cases, the two's complement of a binary number is different from the original value. However, there is a special case in which the two's complement is equal to the original value. For example, in an 8-bit system, the binary number 10000000 represents -128. Taking the two's complement of this number results in the same binary value: 10000000. This happens because -128 is the minimum value that can be represented in 8-bit two's complement format, and it does not have a positive counterpart. Hence, the statement is false.

2. The one's complement of 110010101 is:

a) 001101010
b) 110010101
c) 001101011
d) 110010100

Answer: a
Explanation: The one's complement of a value is obtained by simply

changing all the 1's to 0's and all the 0's to 1's. Hence the one's complement of 110010101 is 001101010.

3. Bitwise _____ gives 1 if either of the bits is 1 and 0 when both of the bits are 1.

a) OR
b) AND
c) XOR
d) NOT

Answer: c
Explanation: Bitwise XOR gives 1 if the two bits are different, and 0 if they are the same. So, it returns 1 when one bit is 1 and the other is 0.

4. What will be the output of the following Python expression?

```
print(4^12)
```

a) 2
b) 4
c) 8
d) 12

Answer: c
Explanation: ^ is the XOR operator. The binary form of 4 is 0100 and that of 12 is 1100. Therefore, 0100^1100 is 1000, which is equal to 8.

5. Any odd number on being AND-ed with _____ always gives 1.
Hint: Any even number on being AND-ed with this value always gives 0.

a) 10
b) 2
c) 1
d) 0

Answer: c
Explanation: Any odd number on being AND-ed with 1 always gives 1. Any even number on being AND-ed with this value always gives 0.

6. What will be the value of the following Python expression?

```
print(bin(10-2)+bin(12^4))
```

a) 0b10000
b) 0b10001000
c) 0b1000b1000
d) 0b10000b1000

Answer: d
Explanation: The value of the expression bin(10 – 2) + bin(12 ^ 4) is '0b1000b1000' because bin(10 – 2) results in '0b1000' and bin(12 ^ 4) also results in '0b1000', and their concatenation gives '0b1000b1000'.

7. Which of the following expressions can be used to multiply a given number 'a' by 4?

a) a<<2
b) a<<4
c) a>>2
d) a>>4

Answer: a
Explanation: Let us consider an example wherein a=2. The binary form of 2 is 0010. When we left shift this value by 2, we get 1000, the value of which is 8. Hence if we want to multiply a given number 'a' by 4, we can use the expression: a<<2.

8. What will be the output of the following Python code if a=10 and b =20?

```
a=10
b=20
```

```
a=a^b
b=a^b
a=a^b
print(a,b)
```

a) 10 20
b) 10 10
c) 20 10 d) 20 20

Answer: c
Explanation: The code shown above is used to swap the contents of two memory locations using bitwise X0R operator. Hence the output of the code shown above is: 20 10.

9. What is the two's complement of -44?

a) 1011011
b) 11010100
c) 11101011
d) 10110011

Answer: b
Explanation: The binary form of -44 is 00101100. The one's complement of this value is 11010011. On adding one to this we get: 11010100 (two's complement).

10. What will be the output of the following Python expression?

```
print(~100)
```

a) 101
b) -101
c) 100
d) -100

Answer: b
Explanation: The expression ~100 uses the bitwise NOT operator, which inverts all the bits of the number. In Python, this is equivalent to -(100 + 1), which results in -101. Therefore, the output of ~100 is -101.

1. What will be the output of the following Python code snippet?

```
print(bool('False'))
print(bool())
```

a)
  True
  True

b)
  False
  True

c)
  False
  False

d)
  True
  False

Answer: d
Explanation: The expression bool('False') returns True because any non-empty string, including 'False', is considered True in Python. The second expression bool() returns False because calling bool() with no argument is the same as passing None, which is considered False.

2. What will be the output of the following Python code snippet?

```
print(['hello', 'morning'][bool('')])
```

a) error
b) no output
c) hello
d) morning

Answer: c
Explanation: The expression [‘hello’, ‘morning’][bool(”)] evaluates the Boolean value of an empty string, which is False or 0, and selects the element at index 0 of the list — resulting in ‘hello’.

3. What will be the output of the following Python code snippet?

```
print(not(3>4))
print(not(1&1))
```

a)
True
True
b)
True
False
c)
False
True
d)
False
False

Answer: b
Explanation: The expression not(3 > 4) returns True because 3 > 4 is False, and not(False) is True. The expression not(1 & 1) returns False because 1 & 1 equals 1, which is true, and not(1) is False.

4. What will be the output of the following Python code?

```
print(['f', 't'][bool('spam')])
```

a) t
b) f
c) No output
d) Error

Answer: a
Explanation: The string ‘spam’ is a non-empty string, so bool(‘spam’) returns True which is equivalent to 1. The expression becomes [‘f’, ‘t’][1], which evaluates to ‘t’.

5. What will be the output of the following Python code?

```
l=[1, 0, 2, 0, 'hello', '', []]
print(list(filter(bool, l)))
```

a) Error
b) [1, 0, 2, 0, ‘hello’, ”, []]
c) [1, 0, 2, ‘hello’, ”, []]
d) [1, 2, ‘hello’]

Answer: d
Explanation: The function filter(bool, l) removes all false elements from the list l, such as 0, ”, and []. The remaining true elements — 1, 2, and ‘hello’ — are returned as a new list.

6. What will be the output of the following Python code if the system date is 21st June, 2017 (Wednesday)?

```
[] or {}
{} or []
```

a)

[]
{}
b)
[]
[]
c)
{}
[]
d)
{}
{}

Answer: c
Explanation: In both expressions, Python evaluates operands from left to right using the or operator. Since both the left and right operands ([] and {}) are false, the or operator returns the last evaluated operand. Therefore, the first expression returns {} and the second returns [].

7. What will be the output of the following Python code?

```
class Truth:
        pass
x=Truth()
print(bool(x))
```

a) pass
b) True
c) False
d) error

Answer: b
Explanation: If a class does not implement a __bool__() or __len__() method, then all its instances are considered true by default, so bool(x) returns True.

8. What will be the output of the following Python code?

```
if (9 < 0) and (0 < -9):
    print("hello")
elif (9 > 0) or False:
    print("good")
else:
    print("bad")
```

a) error
b) hello
c) good
d) bad

Answer: c
Explanation: The first condition is False because both 9 < 0 and 0 < -9 are false. The second condition (9 > 0) or False is True because 9 > 0 is true. Therefore, the program prints “good”.

9. Which of the following Boolean expressions is not logically equivalent to the other three?
a) not(-6<0 or-6>10)
b) -6>=0 and -6<=10
c) not(-6<10 or-6==10)
d) not(-6>10 or-6==10)

Answer: d
Explanation: The expression not(-6<0 or -6>10) returns the output False. The expression -6>=0 and -6<=10 returns the output False.

The expression not(-6<10 or -6==10) returns the output False.
The expression not(-6>10 or -6==10) returns the output True.

10. What will be the output of the following Python code snippet?

```
print(not(10<20) and not(10>30))
```

a) True
b) False
c) Error
d) No output

Answer: b
Explanation: The expression not(10 < 20) evaluates to False, and not(10 > 30) evaluates to True. The AND operation between False and True results in False.

1. What will be the output of the following Python code snippet?

```
X=”hi”
print(“05d”%X)
```

a) 00000hi
b) 000hi
c) hi000
d) error

Answer: d
Explanation: The code snippet shown above results in an error because the above formatting option works only if ‘X’ is a number. Since in the above case ‘X’ is a string, an error is thrown.

2. What will be the output of the following Python code snippet?

```
X = "san-foundry"
print("%56s" % X)
```

a) 45 blank spaces before san-foundry
b) 56 blank spaces before san and foundry
c) 56 blank spaces after san-foundry
d) no change

Answer: a
Explanation: The formatting option %56s aligns the string to the right within a field of width 56. If the string “san-foundry” is 11 characters long, it will be preceded by (56 – 11 = 45) blank spaces.

3. What will be the output of the following Python code?

```
x=456
print("%-06d"%x)
```

a) 000456
b) 456000
c) 456
d) error

Answer: c
Explanation: The format specifier %-06d means “left-align the integer in a field of width 6”. The 0 (zero-padding) is ignored when used with -. So, the number 456 is printed left-aligned with spaces on the right, resulting in ‘456 ‘. Visually, it appears as 456.

4. What will be the output of the following Python code?

```
X=345
print(“%06d”%X)
```

a) 345000
b) 000345
c) 000000345
d) 345000000

Answer: b
Explanation: The format specifier %06d means the integer should be printed with a total width of 6 characters, padded with leading zeroes if necessary. Since 345 has 3 digits, it is printed as 000345.

5. Which of the following formatting options can be used in order to add ‘n’ blank spaces after a given string ‘S’?

a) print(“-ns”%S)
b) print(“-ns”%S)
c) print(“%ns”%S)
d) print(“%-ns”%S)

Answer: d
Explanation: The format specifier “%-ns” left-aligns the string S in a field of width n. This causes n – len(S) blank spaces to be added after the string to reach the specified width.

6. What will be the output of the following Python code?

```
X = -122
print("-%06d"%X)
```

a) −000122
b) 000122
c) −−00122
d) −00122

Answer: c
Explanation: The expression “%06d” % X formats -122 as -00122, with leading zeros to make the total width 6. Since there’s an extra – in the string, it gets added as a literal character, resulting in –00122. So the output is –00122.

7. What will be the output of the following Python code?

```
x=34
```

```
print(“%f”%x)
```
a) 34.00
b) 34.0000
c) 34.000000
d) 34.00000000

Answer: c
Explanation: The output of the code is 34.000000. By default, the %f format specifier in Python displays floating-point numbers with 6 digits after the decimal point. Since no precision is explicitly set, x = 34 is printed as 34.000000.

8. What will be the output of the following Python expression?

```
x=56.236
print("%.2f"%x)
```
a) 56.00
b) 56.24
c) 56.23
d) 0056.236

Answer: b
Explanation: The expression shown above rounds off the given number to the number of decimal places specified. Since the expression given specifies rounding off to two decimal places, the output of this expression will be 56.24. Had the value been x=56.234 (last digit being any number less than 5), the output would have been 56.23.

9. What will be the output of the following Python expression?

```
x=22.19
print("%5.2f"%x)
```
a) 22.1900
b) 22.00000

c) 22.19
d) 22.20

Answer: c
Explanation: The format “%5.2f” means the number should be printed with 2 digits after the decimal point, and a minimum width of 5 characters (including the decimal point). So 22.19 fits perfectly and is printed as-is.

10. The expression shown below results in an error.

```
print("-%5d0",989)
```

a) True
b) False

Answer: b
Explanation: The expression shown above does not result in an error. The output of this expression is -%5d0 989. Hence this statement is incorrect.

1. What will be the output of the following Python code snippet?

```
print('%d %s %g you' %(1, 'hello', 4.0))
```

a) Error
b) 1 hello you 4.0
c) 1 hello 4 you
d) 1 4 hello you

Answer: c
Explanation: In the format string ‘%d %s %g you’, %d formats the integer 1, %s formats the string ‘hello’, and %g formats the float 4.0 in a compact form (removing unnecessary decimal zeros), so 4.0 becomes 4. The resulting output is ‘1 hello 4 you’.

2. The output of which of the codes shown below will be: “There are 4 blue birds.”?

a) ‘There are %g %d birds.’ %4 %blue
b) ‘There are %d %s birds.’ %(4, “blue”)
c) ‘There are %s %d birds.’ %[4, “blue”]
d) ‘There are %d %s birds.’ 4, “blue”

Answer: b
Explanation: The code ‘There are %d %s birds.’ % (4, “blue”) produces the output: “There are 4 blue birds.” When inserting more than one value using old-style formatting, you must group them as a tuple using parentheses, and all string literals like “blue” must be quoted.

3. What will be the output of the python code shown below for various styles of format specifiers?


```
x=1234
res='integers:...%d...%-6d...%06d' %(x, x, x)
print(res)
```

a) ‘integers:…1234…1234 …001234’
b) ‘integers…1234…1234…123400’
c) ‘integers:… 1234…1234…001234’
d) ‘integers:…1234…1234…001234’

Answer: a
Explanation: The format specifiers work as follows: %d prints the integer normally (1234), %-6d prints the integer left-aligned within a 6-character width (1234 with two trailing spaces), and %06d prints the integer padded with zeros to a width of 6 (001234). Combining these results produces the string: ‘integers:…1234…1234 …001234’.

4. What will be the output of the following Python code snippet?

```
x=3.3456789
print('%f | %e | %g' %(x, x, x))
```

a) Error
b) ‘3.3456789 | 3.3456789+00 | 3.345678’
c) ‘3.345678 | 3.345678e+0 | 3.345678’
d) ‘3.345679 | 3.345679e+00 | 3.34568’

Answer: d
Explanation: In Python, %f formats to 6 decimal places by default, %e uses exponential notation, and %g automatically switches between %f and %e depending on the value’s magnitude and precision. So x = 3.3456789 is formatted as: 3.345679 | 3.345679e+00 | 3.34568.

5. What will be the output of the following Python code snippet?

```
x=3.3456789
print('%-6.2f | %05.2f | %+06.1f' %(x, x, x))
```

a) ‘3.35 | 03.35 | +003.3’
b) ‘3.3456789 | 03.3456789 | +03.3456789’
c) Error
d) ‘3.34 | 03.34 | 03.34+’

Answer: a
Explanation: The code shown above rounds the floating point value to two decimal places. In this code, a variety of addition formatting features such as zero padding, total field width etc. Hence the output of this code is: ‘3.35 | 03.35 | +003.3’.

6. What will be the output of the following Python code snippet?

```
x=3.3456789
print('%s' %x, str(x))
```

a) Error
b) (‘3.3456789’, ‘3.3456789’)

c) (3.3456789, 3.3456789)
d) (‘3.3456789’, 3.3456789)

Answer: b
Explanation: Both ‘%s’ % x and str(x) convert the float x to its string representation. Therefore, the output is a tuple containing two identical strings: (‘3.3456789’, ‘3.3456789’).

7. What will be the output of the following Python code snippet?

```
print('%(qty)d more %(food)s' %{'qty':1, 'food': 'spam'})
```

a) Error
b) No output
c) ‘1 more foods’
d) ‘1 more spam’

Answer: d
Explanation: This code uses old-style string formatting with a dictionary. The placeholder %(qty)d expects an integer and is replaced by 1, and %(food)s expects a string and is replaced by ‘spam’. Thus, the output is ‘1 more spam’.

8. What will be the output of the following Python code snippet?

```
a = 'hello'
q = 10
print(vars())
```

a) {‘a’ : ‘hello’, ‘q’ : 10, ……..plus built-in names set by Python….}
b) {……Built in names set by Python……}
c) {‘a’ : ‘hello’, ‘q’ : 10}
d) Error

Answer: a
Explanation: The vars() function returns the __dict__ attribute of the

current local scope as a dictionary, which includes user-defined variables (like a and q in this case), as well as built-in names and functions set by Python. Hence the output will be a dictionary containing the variables a and q, plus other built-in names.

9. What will be the output of the following Python code?

```
s='{0}, {1}, and {2}'
print(s.format('hello', 'good', 'morning'))
```

a) hello good and morning
b) hello, good, morning
c) hello, good, and morning
d) Error

Answer: c
Explanation: In the given code, the str.format() method is used to insert values into placeholders defined by curly braces {0}, {1}, and {2}. The values ‘hello’, ‘good’, and ‘morning’ are inserted in that order, with the commas and “and” appearing exactly as written in the string, resulting in: “hello, good, and morning”.

10. What will be the output of the following Python code?

```
s='%s, %s & %s'
print(s%('mumbai', 'kolkata', 'delhi'))
```

a) mumbai kolkata & delhi
b) Error
c) No output
d) mumbai, kolkata & delhi

Answer: d
Explanation: The %s format specifier in the string is replaced by the corresponding values from the tuple (‘mumbai’, ‘kolkata’, ‘delhi’). The

commas and “&” are preserved as they are in the string. Therefore, the output is: “mumbai, kolkata & delhi”.

11. What will be the output of the following Python code?

```
t = '%(a)s, %(b)s, %(c)s'
print(t % dict(a='hello', b='world', c='universe'))
```

a) ‘hello, world, universe’
b) ‘hellos, worlds, universes’
c) Error
d) hellos, world, universe

Answer: a
Explanation: The string t uses old-style string formatting with named placeholders %(a)s, %(b)s, and %(c)s. When the dictionary {‘a’:’hello’, ‘b’:’world’, ‘c’:’universe’} is applied, each placeholder is replaced by the corresponding value, resulting in ‘hello, world, universe’.

12. What will be the output of the following Python code?

```
print('{a}, {0}, {abc}'.format(10, a=2.5, abc=[1, 2]))
```

a) Error
b) ‘2.5, 10, [1, 2]’
c) 2.5, 10, 1, 2
d) ’10, 2.5, [1, 2]’

Answer: b
Explanation: The format string uses both positional and keyword arguments: {a} maps to 2.5, {0} maps to the first positional argument 10, and {abc} maps to the list [1, 2]. Thus, the output is ‘2.5, 10, [1, 2]’.

13. What will be the output of the following Python code?

```
print('{0:.2f}'.format(1.234))
```

a) ‘1’
b) ‘1.234’

c) ‘1.23’
d) ‘1.2’

Answer: c
Explanation: The format specifier {0:.2f} means format the first argument as a floating-point number with 2 digits after the decimal point. So, 1.234 is rounded to 1.23.

14. What will be the output of the following Python code?

```
print('%x %d' %(255, 255))
```

a) ‘ff, 255’
b) ‘255, 255’
c) ‘15f, 15f’
d) Error

Answer: a
Explanation: In the given code, %x is used to convert 255 to its hexadecimal representation (ff), and %d is used to display 255 as a decimal number. Hence the output is: “ff, 255”.

15. The output of the two codes shown below is the same.

```
i. print('{0:.2f}'.format(1/3.0))
ii. print('%.2f'%(1/3.0))
```

a) True
b) False

Answer: a
Explanation: The two codes shown above represent the same operation but in different formats. The output of both of these functions is: ‘0.33’. Hence the statement is true.

1. What will be the output of the following Python code?

```
l=list('HELLO')
print('first={0[0]}, third={0[2]}'.format(l))
```

a) ‘first=H, third=L’
b) ‘first=0, third=2’
c) Error
d) ‘first=0, third=L’

Answer: a
Explanation: The list l contains the characters [‘H’, ‘E’, ‘L’, ‘L’, ‘O’]. Using {0[0]} and {0[2]} inside .format(l) accesses the first and third elements of the list. So the output is ‘first=H, third=L’.

2. What will be the output of the following Python code?

```
l=list('HELLO')
p=l[0], l[-1], l[1:3]
print('a={0}, b={1}, c={2}'.format(*p))
```

a) Error
b) “a=’H’, b=’O’, c=(E, L)”
c) “a=H, b=O, c=[‘E’, ‘L’]”
d) Junk value

Answer: c
Explanation: The tuple p is (‘H’, ‘O’, [‘E’, ‘L’]), created from the list l = [‘H’, ‘E’, ‘L’, ‘L’, ‘O’]. Using *p unpacks it into three arguments for format(), resulting in: ‘a=H, b=O, c=[‘E’, ‘L’]’.

3. The formatting method {1:<10} represents the _____ positional argument, _____ justified in a 10 character wide field.

a) first, right
b) second, left
c) first, left

d) second, right

Answer: b
Explanation: In {1:<10}, the 1 refers to the second positional argument (indexing starts at 0), and < indicates left justification. The 10 specifies the total field width, so the second argument is left-justified in a 10-character space.

4. What will be the output of the following Python code?

```
print(hex(255), int('FF', 16), 0xFF)
```

a) [0xFF, 255, 16, 255]
b) (‘0xff’, 155, 16, 255)
c) Error
d) (‘0xff’, 255, 255)

Answer: d
Explanation: hex(255) returns the string ‘0xff’, int(‘FF’, 16) converts the hexadecimal string ‘FF’ to integer 255, and 0xFF is the hexadecimal literal for 255. So the output is (‘0xff’, 255, 255).

5. The output of the two codes shown below is the same.

```
i. print(bin((2**16)-1))
ii. print('{}'.format(bin((2**16)-1)))
```

a) True
b) False

Answer: a
Explanation: Both expressions produce the same result. bin((2**16)-1) gives the binary string ‘0b1111111111111111’, and ‘{}’.format(bin((2**16)-1)) just formats and returns the same string. So the outputs are identical.

6. What will be the output of the following Python code?

```
print('{a}{b}{a}'.format(a='hello', b='world'))
```

a) ‘hello world’
b) ‘hello’ ‘world’ ‘hello’
c) ‘helloworldhello’
d) ‘hello’ ‘hello’ ‘world’

Answer: c
Explanation: In the format string ‘{a}{b}{a}’, the placeholders {a} and {b} are filled with the values ‘hello’ and ‘world’, respectively. So the result is ‘hello’ + ‘world’ + ‘hello’, which gives ‘helloworldhello’.

7. What will be the output of the following Python code?

```
D=dict(p='san', q='foundry')
print('{p}{q}'.format(**D))
```

a) Error
b) sanfoundry
c) san foundry
d) {‘san’, ‘foundry’}

Answer: b
Explanation: The format(**D) syntax unpacks the dictionary D and passes its keys as named arguments to the format() method. {p} is replaced by ‘san’ and {q} by ‘foundry’, resulting in the output ‘sanfoundry’.

8. What will be the output of the following Python code?

```
print('The {} side {1} {2}'.format('bright', 'of', 'life'))
```

a) Error
b) ‘The bright side of life’
c) ‘The {bright} side {of} {life}’
d) No output

Answer: a
Explanation: In Python's str.format() method, you cannot mix automatic field numbering ({}) with manual field numbering ({1}, {2}) in the same format string. The line 'The {} side {1} {2}' tries to do both, which raises a ValueError.

9. What will be the output of the following Python code?

```
print('{0:f}, {1:2f}, {2:05.2f}'.format(1.23456, 1.23456, 1.23456))
```

a) Error
b) '1.234560, 1.22345, 1.23'
c) No output
d) '1.234560, 1.234560, 01.23'

Answer: d
Explanation: The code uses three float format specifiers:

- {0:f} formats the number with default 6 decimal places: 1.234560
- {1:2f} also shows 6 decimal places (the 2 is the width, but it's too small to affect the output)
- {2:05.2f} formats the number to 2 decimal places and pads it to 5 characters with leading zeros: 01.23

So, the final output is: '1.234560, 1.234560, 01.23'.

10. What will be the output of the following Python code?

```
print('%.2f%s' % (1.2345, 99))
```

a) '1.2345', '99'
b) '1.2399'
c) '1.234599'
d) 1.23, 99

Answer: b
Explanation: The format %.2f ensures that the floating point number is

rounded to two decimal places. In this case, the number 1.2345 becomes 1.23 after rounding, and when the string 99 is appended, the result is '1.2399', not the expected simple concatenation. Thus, the rounding and formatting lead to the result '1.2399'.

11. What will be the output of the following Python code?

```
print('%s' %((1.23,),))
```

a) '(1.23,)'
b) 1.23,
c) (,1.23)
d) '1.23'

Answer: a
Explanation: In the code provided, the format string %s is used, which expects a string representation of its argument. The argument ((1.23,),) is a tuple containing another tuple (1.23,). When %s is used, Python converts the entire outer tuple to a string, resulting in the output '(1.23,)'. This is the string representation of the tuple containing a single float tuple.

12. What will be the output of the following two codes?

```
i. print('{0}'.format(4.56))
ii. print('{0}'.format([4.56,]))
```

a) '4.56', '4.56,'
b) '4.56', '[4.56]'
c) 4.56, [4.56,]
d) 4.56, 4.56,

Answer: b
Explanation: In the first code, {0} is used to format the floating-point number 4.56, and it is directly displayed as 4.56. In the second code, {0} is

used to format a list [4.56,]. The list is converted to its string representation, which is “[4.56]”. Hence, the output is ‘4.56’, ‘[4.56]’.

1. What will be the output of the following Python code?

```
def mk(x):
    def mk1():
        print("Decorated")
        x()
    return mk1
def mk2():
    print("Ordinary")
p = mk(mk2)
p()
```

a)
Decorated
Decorated

b)
Ordinary
Ordinary

c)
Ordinary
Decorated

d)
Decorated
Ordinary

Answer: d
Explanation: The function mk is a decorator that modifies the behavior of

mk2. When p() is called, it first prints “Decorated” (from mk1) and then calls mk2, printing “Ordinary”. Hence, the output is:
Decorated
Ordinary

2. In the following Python code, which function is the decorator?

```
def mk(x):
    def mk1():
        print("Decorated")
        x()
    return mk1
def mk2():
    print("Ordinary")
p = mk(mk2)
p()
```

a) p()
b) mk()
c) mk1()
d) mk2()

Answer: b
Explanation: The decorator in the code is the function mk(). It takes a function mk2() as an argument, modifies its behavior by printing “Decorated”, and returns a new function mk1() that includes the decorated functionality.

3. The _____ symbol along with the name of the decorator function can be placed above the definition of the function to be decorated works as an alternate way for decorating a function.

a) #

b) $
c) @
d) &

Answer: c
Explanation: The @ symbol is used to apply a decorator to a function in Python. It is an alternate, more readable way to decorate a function, as opposed to manually calling the decorator inside the function definition.

4. What will be the output of the following Python code?

```
def ordi():
        print("Ordinary")
ordi
ordi()
```

a)
Address
Ordinary
b)
Error
Address
c)
Ordinary
Ordinary
d)
Ordinary
Address

Answer: a
Explanation: The code first prints the memory address of the function ordi because calling ordi without parentheses just references the

function object itself. When ordi() is called with parentheses, it executes the function and prints “Ordinary”.
Address
Ordinary

5. The two snippets of the following Python codes are equivalent.

```
CODE 1
  @f
def f1():
      print(“Hello”)
CODE 2
  def f1():
       print(“Hello”)
f1 = f(f1)
```

a) True
b) False

Answer: a
Explanation: Both snippets are functionally equivalent. The @f decorator syntax in CODE 1 is just an alternate and cleaner way of applying the decorator, whereas in CODE 2, the decorator is applied manually by reassigning f1 to f(f1). Both result in f1 being decorated by f.

6. What will be the output of the following Python function?

```
def f(p, q):
        return p%q
print(f(0, 2))
print(f(2, 0))
```

a)
0

0
b)
Zero Division Error
Zero Division Error
c)
0
Zero Division Error
d)
Zero Division Error
0

Answer: c
Explanation: The first call f(0, 2) returns 0, since 0 % 2 = 0. The second call f(2, 0) raises a ZeroDivisionError, as division by zero is not allowed in Python. To avoid the error, you can handle it with a try-except block.

7. What will be the output of the following Python code?

```
def f(x):
    def f1(a, b):
        print("hello")
        if b==0:
            print("NO")
            return
        return f(a, b)
    return f1
@f
def f(a, b):
    return a%b
f(4,0)
```

a)
hello
NO
b)
hello
Zero Division Error
c) NO
d) hello

Answer: a
Explanation: In the code shown above, we have used a decorator in order to avoid the Zero Division Error. Hence the output of this code is:
hello
NO

8. What will be the output of the following Python code?

```
def f(x):
    def f1(*args, **kwargs):
        print("*"* 5)
        x(*args, **kwargs)
        print("*"* 5)
    return f1
def a(x):
    def f1(*args, **kwargs):
        print("%"* 5)
        x(*args, **kwargs)
        print("%"* 5)
    return f1
@f
@a
def p(m):
```

```
    print(m)
p("hello")
```

a)
*****
%%%%%
hello
%%%%%
*****

b) Error

c) *****%%%%%hello%%%%%*****

d) hello

Answer: a

Explanation: The decorators @f and @a wrap the function p. @f prints ***** before and after calling @a, and @a prints %%%%% before and after calling p. The final output is

*****
%%%%%
hello
%%%%%
*****

9. The following python code can work with ____ parameters.

```
def f(x):
    def f1(*args, **kwargs):
        print("Sanfoundry")
        return x(*args, **kwargs)
    return f1
```

a) 2

b) 1

c) any number of

d) 0

Answer: c
Explanation: The decorator function f defines f1 which uses *args and **kwargs to accept any number of positional and keyword arguments. This allows the decorated function to work with any number of parameters, making the decorator flexible.

10. What will be the output of the following Python code?

```
def f(x):
    def f1(*args, **kwargs):
        print("*", 5)
        x(*args, **kwargs)
        print("*", 5)
    return f1
@f
def p(m):
    p(m)
print("hello")
```

a)
*****
hello
b)
*****
*****
hello
c) *****
d) hello

Answer: d
Explanation: The function p is decorated with @f, which wraps it with f1.

However, there's a mistake in the code: p(m) inside the function p calls itself recursively, but no parameter is passed to it. Therefore, the recursion is not executed as expected, and only the print("hello") statement is executed, printing hello.

11. A function with parameters cannot be decorated.

a) True
b) False

Answer: b
Explanation: Any function, irrespective of whether or not it has parameters can be decorated. Hence the statement is false.

12. Identify the decorator in the snippet of code shown below.

```
def sf():
    pass
sf = mk(sf)
@f
def sf():
    return
```

a) @f
b) f
c) sf()
d) mk

Answer: d
Explanation: In the code, the decorator is applied using @f for the function sf, but the actual decorator function that is modifying sf is mk, since sf = mk(sf) is explicitly written before the @f decorator. The @f line represents the decorator application, while mk is the actual decorator function.

13. What will be the output of the following Python code?

```
class A:
    @staticmethod
    def a(x):
        print(x)
A.a(100)
```

a) Error
b) Warning
c) 100
d) No output

Answer: c
Explanation: In this code, a is defined as a static method of class A using the @staticmethod decorator. Static methods do not require an instance of the class to be called. Therefore, A.a(100) is a valid call to the static method a, which prints the value 100 passed to it.

14. What will be the output of the following Python code?

```
def d(f):
    def n(*args):
        return '$' + str(f(*args))
    return n
@d
def p(a, t):
    return a + a*t
print(p(100,0))
```

a) 100
b) $100
c) $0
d) 0

Answer: b
Explanation: The function p(a, t) calculates a + a*t. With a = 100 and t = 0, the result of p(100, 0) is 100 + 100*0 = 100. The decorator @d then adds a dollar sign ($) before the result, so the output is $100.

15. What will be the output of the following Python code?

```
def c(f):
    def inner(*args, **kargs):
        inner.co += 1
        return f(*args, **kargs)
    inner.co = 0
    return inner
@c
def fnc():
    pass
if __name__ == '__main__':
    fnc()
    fnc()
    fnc()
    print(fnc.co)
```

a) 4
b) 3
c) 0
d) 1

Answer: b
Explanation: In this code, the decorator @c is applied to the function fnc. The decorator defines an attribute co on the inner function, which is initially set to 0. Each time fnc is called, inner.co is incremented by 1. The function fnc() is called three times, so inner.co will be incremented three times, resulting in a final value of 3. Thus, the output of print(fnc.co) is 3.

1. What will be the output of the following Python code?

```
x = ['ab', 'cd']
for i in x:
   i.upper()
print(x)
```

a) [‘ab’, ‘cd’]
b) [‘AB’, ‘CD’]
c) [None, None]
d) none of the mentioned

Answer: a
Explanation: In the code, i.upper() is called inside the loop, but upper() creates a new string and does not modify the original string in place. Since the result of i.upper() is not assigned back to any variable or element, the original list x remains unchanged. Therefore, the output is [‘ab’, ‘cd’].

2. What will be the output of the following Python code?

```
x = ['ab', 'cd']
for i in x:
   x.append(i.upper())
print(x)
```

a) [‘AB’, ‘CD’]
b) [‘ab’, ‘cd’, ‘AB’, ‘CD’]
c) [‘ab’, ‘cd’]
d) none of the mentioned

Answer: d
Explanation: The output will be an infinite loop because new elements are being appended to the list during iteration.

3. What will be the output of the following Python code?

```
i = 1
while True:
    if i%3 == 0:
        break
    print(i)

    i + = 1
```

a) 1 2
b) 1 2 3
c) SyntaxError
d) none of the mentioned

Answer: c
Explanation: The output will be a SyntaxError because there is a space between + and = in the i += 1 statement.

4. What will be the output of the following Python code?

```
i = 1
while True:
    if i%0O7 == 0:
        break
    print(i)
    i += 1
```

a)
1
2
3
4
5
6

b)
1
2
3
4
5
6
7
c) error
d) none of the mentioned

Answer: a
Explanation: The code exits the loop when i % 0O7 == 0 (i.e., when i reaches 7 because 0O7 is the octal representation of 7). Therefore, the loop runs until i becomes 7.

5. What will be the output of the following Python code?

```
i = 5
while True:
    if i%0O11 == 0:
        break
    print(i)
    i += 1
```

a)
5
6
7
8
9
10
b)

5
6
7
8
c)
5
6
d) error

Answer: b
Explanation: 0O11 is the octal representation of the number 9. The loop will break when i reaches 9 because i % 9 == 0. The loop prints the values from 5 to 8 before it breaks, as i reaches 9 where the condition i % 0O11 == 0 is true.

6. What will be the output of the following Python code?

```
i = 5
while True:
   if i%0O9 == 0:
      break
   print(i)
   i += 1
```

a) 5 6 7 8
b) 5 6 7 8 9
c) 5 6 7 8 9 10 11 12 13 14 15 ….
d) error

Answer: d
Explanation: In Python, octal numbers are represented by a prefix 0o or 0O followed by digits from 0 to 7. Since 9 is not a valid octal digit, the

code will result in a SyntaxError when trying to interpret 0O9 as an octal number.

7. What will be the output of the following Python code?

```
i = 1
while True:
    if i%2 == 0:
        break
    print(i)
    i += 2
```

a) 1
b) 1 2
c) 1 2 3 4 5 6 …
d) 1 3 5 7 9 11 …

Answer: d
Explanation: In the given code, i starts from 1 and increases by 2 in each iteration (i += 2). This ensures that i will always be an odd number. The loop will keep printing the odd numbers and will never break, since the condition i % 2 == 0 (which checks if i is even) will never be true. Therefore, the output will be an infinite sequence of odd numbers starting from 1.

8. What will be the output of the following Python code?

```
i = 2
while True:
    if i%3 == 0:
        break
    print(i)
    i += 2
```

a)
2

4
6
8
10
.....
b)
2
4
c)
2
3
d) error

Answer: b
Explanation: The variable i starts at 2 and increments by 2 in each iteration (i += 2). The loop continues printing i until i % 3 == 0 (i.e., when i is divisible by 3).

- In the first iteration, i = 2, and it is printed.
- In the second iteration, i = 4, and it is printed.
- In the third iteration, i = 6, which is divisible by 3, so the loop breaks.

Thus, the output will be 2 4.

9. What will be the output of the following Python code?

```
i = 1
while False:
    if i%2 == 0:
        break
    print(i)
    i += 2
```

a) 1
b) 1 3 5 7 …
c) 1 2 3 4 …
d) none of the mentioned

Answer: d
Explanation: The while False: condition means that the loop will never execute because False is always false. Therefore, the code inside the loop, including the print(i) and the increment of i, will never be executed. As a result, there will be no output.

10. What will be the output of the following Python code?

```
True = False
while True:
    print(True)
    break
```

a) True
b) False
c) ERROR
d) none of the mentioned

Answer: c
Explanation: True is a reserved keyword in Python, and its value cannot be reassigned. Attempting to assign True = False will result in a SyntaxError. Therefore, the code will not execute and will raise an error.

1. What will be the output of the following Python code?

```
i = 0
while i < 5:
    print(i)
    i += 1
    if i == 3:
```

```
        break
else:
    print(0)
```

a)
0
1
2
0

b)
0
1
2
c) error
d) none of the mentioned

Answer: b
Explanation: In this code, the while loop iterates from i = 0 to i = 2, printing the values of i. When i becomes 3, the if i == 3 condition is met, and the break statement is executed, which terminates the loop early. Since the loop was terminated using break, the else block is not executed. Therefore, the output is 0, 1, and 2.

2. What will be the output of the following Python code?

```
i = 0
while i < 3:
    print(i)
    i += 1
else:
    print(0)
```

a)

0
1
2
3
0
b)
0
1
2
0
c)
0
1
2
d) error

Answer: b
Explanation: The while loop runs while i < 3, so it prints 0, 1, and 2. Once i becomes 3, the condition i < 3 becomes false, and the loop ends. Since the loop ends normally (without a break), the else block is executed, printing 0. Therefore, the final output is 0, 1, 2, and 0.

3. What will be the output of the following Python code?

```
x = "abcdef"
while i in x:
    print(i, end=" ")
```

a) a b c d e f
b) abcdef
c) i i i i i i …
d) error

Answer: d
Explanation: In the given code, i is not defined before it is used in the while loop. This will raise a NameError because Python does not know what i refers to. To fix this, you would need to define i before using it in the loop.

4. What will be the output of the following Python code?

```
x = "abcdef"
i = "i"
while i in x:
    print(i, end=" ")
```

a) no output
b) i i i i i i …
c) a b c d e f
d) abcdef

Answer: a
Explanation: The condition while i in x: checks if the string “i” exists in the string “abcdef”. Since “i” is not present in “abcdef”, the condition is False from the start, so the loop does not execute, resulting in no output.

5. What will be the output of the following Python code?

```
x = "abcdef"
i = "a"
while i in x:
    print(i, end = " ")
```

a) no output
b) i i i i i i …
c) a a a a a a …
d) a b c d e f

Answer: c
Explanation: The condition while i in x: checks if the string “a” is present in the string “abcdef”. Since “a” is present in “abcdef”, the loop will keep printing “a” endlessly. The value of i doesn’t change inside the loop, so the condition will always be True, and the loop will run indefinitely, printing “a” each time.

6. What will be the output of the following Python code?

```
x = "abcdef"
i = "a"
while i in x:
    print('i', end = " ")
```

a) no output
b) i i i i i i …
c) a a a a a a …
d) a b c d e f

Answer: b
Explanation: In this code, i is a string with the value “a”, and the while loop checks if “i” (the string) is in x (which is “abcdef”). Since the string “i” is not in “abcdef”, the condition will always be False, and the loop will print the string ‘i’ endlessly.

7. What will be the output of the following Python code?

```
x = "abcdef"
i = "a"
while i in x:
    x = x[:-1]
    print(i, end = " ")
```

a) i i i i i i
b) a a a a a a
c) a a a a a

d) none of the mentioned

Answer: b
Explanation: In the code, x starts as “abcdef” and in each iteration, x = x[:-1] removes the last character from the string. Since “a” is still in the string during each iteration, it continues to print “a”. The loop prints “a” until the string x is reduced to just “a”, and it prints a total of six “a”s.

8. What will be the output of the following Python code?

```
x = "abcdef"
i = "a"
while i in x[:-1]:
    print(i, end = " ")
```

a) a a a a a
b) a a a a a a
c) a a a a a a …
d) a

Answer: c
Explanation: In the code, x[:-1] means that the string x is sliced, excluding the last character. So, x[:-1] will be “abcde”. The loop checks if “a” is in “abcde”, which is true. However, since the value of x is not modified within the loop, it keeps printing “a” repeatedly, causing an infinite loop. Thus, the loop will continuously print “a” indefinitely.

9. What will be the output of the following Python code?

```
x = "abcdef"
i = "a"
while i in x:
    x = x[1:]
    print(i, end = " ")
```

a) a a a a a a
b) a
c) no output
d) error

Answer: b
Explanation: In the code, x = x[1:] slices the string x, removing the first character in each iteration. The loop continues as long as the string x contains the character “a”. Initially, x = “abcdef”, so the loop starts by printing “a”. After each iteration, x becomes “bcdef”, “cdef”, and so on, until “a” is no longer in x, at which point the loop terminates. Therefore, only a single “a” is printed.

10. What will be the output of the following Python code?

```
x = "abcdef"
i = "a"
while i in x[1:]:
    print(i, end = " ")
```

a) a a a a a a
b) a
c) no output
d) error

Answer: c
Explanation: In the code, x[1:] slices the string x starting from the second character. So, x[1:] becomes “bcdef”. The loop checks if i (which is “a”) is in the sliced string “bcdef”, but “a” is not found in “bcdef”. As a result, the condition i in x[1:] is False, and the loop does not execute, producing no output.

1. What will be the output of the following Python code?

```
x = 'abcd'
```

```
for i in x:
   print(i)
   x.upper()
```

a)
a
B
C
D

b)
a
b
c
d
c)
A
B
C
D
d) error

Answer: b
Explanation: In the code, the string x = ‘abcd’ is iterated over character by character using a for loop. The x.upper() method returns a new string with all characters in uppercase, but it doesn’t modify x in place because strings in Python are immutable. Therefore, the loop prints each character in its original lowercase form (a, b, c, d), and the changes made by x.upper() do not affect the output of the loop.

2. What will be the output of the following Python code?

```
x = 'abcd'
```

```
for i in x:
    print(i.upper())
```

a)
a
b
c
d
b)
A
B
C
D
c)
a
B
C
D
d) error

Answer: b
Explanation: In this code, x = ‘abcd’ is iterated over, and for each character i, i.upper() is called. The upper() method returns a new string where all characters are converted to uppercase. Each uppercase character is then printed on a new line. Therefore, the output is A, B, C, D, one per line.

3. What will be the output of the following Python code?

```
x = 'abcd'
for i in range(x):
    print(i)
```

a) a b c d
b) 0 1 2 3
c) error
d) none of the mentioned

Answer: c
Explanation: The code attempts to use range(x), where x is a string (‘abcd’). However, the range() function expects an integer as its argument, not a string. Since range(str) is not valid, this will result in a TypeError. Hence, the code will raise an error.

4. What will be the output of the following Python code?

```
x = 'abcd'
for i in range(len(x)):
    print(i)
```

a)
a
b
c
d
b)
0
1
2
3
c)
1
2
3
4

d) error

Answer: b
Explanation: x = ‘abcd’ has 4 characters, so len(x) is 4. The loop for i in range(len(x)) iterates over range(4), which gives the values 0, 1, 2, 3. Each of these values is printed on a new line.
So, the output is:
0
1
2
3

5. What will be the output of the following Python code?

```
x = 'abcd'
for i in range(len(x)):
   print(i.upper())
```

a) a b c d
b) 0 1 2 3
c) error
d) 1 2 3 4

Answer: c
Explanation: The code raises an error because i is an integer from range(len(x)), and integers do not have the upper() method, which is only valid for strings.

6. What will be the output of the following Python code snippet?

```
x = 'abcd'
for i in range(len(x)):
   i.upper()
print (x)
```

a) a b c d
b) 0 1 2 3
c) error
d) none of the mentioned

Answer: c
Explanation: The code results in an error because i is an integer, and integers do not support the upper() method, which is only available for strings.

7. What will be the output of the following Python code snippet?

```
x = 'abcd'
for i in range(len(x)):
   x[i].upper()
print (x)
```

a) abcd
b) ABCD
c) error
d) none of the mentioned

Answer: a
Explanation: The string method upper() returns a new uppercase string but does not modify the original string. Since the result of x[i].upper() isn't stored, x remains unchanged. Hence, the output is 'abcd'.

8. What will be the output of the following Python code snippet?

```
x = 'abcd'
for i in range(len(x)):
   i[x].upper()
print (x)
```

a) abcd
b) ABCD

c) error
d) none of the mentioned

Answer: c
Explanation: The given code produces an error because i is an integer, and the expression i[x] attempts to subscript (index) an integer, which is not allowed in Python. Integers are not subscriptable types, so this results in a TypeError. If the expression had been x[i].upper() instead, it would have been valid, though it still wouldn't change the original string.

9. What will be the output of the following Python code snippet?

```
x = 'abcd'
for i in range(len(x)):
   x = 'a'
   print(x)
```

a) a
b)
abcd
abcd
abcd
abcd
c)
a
a
a
a
d) none of the mentioned

Answer: c
Explanation: The code will output 'a' four times because inside the loop,

the variable x is reassigned to the string ‘a’ on each iteration, and then ‘a’ is printed. This happens for all four iterations of the loop.

10. What will be the output of the following Python code snippet?

```
x = 'abcd'
for i in range(len(x)):
   print(x)
   x = 'a';
```

a) a
b)
abcd
abcd
abcd
abcd
c)
a
a
a
a
d) none of the mentioned

Answer: d
Explanation: The code will output ‘abcd’ once and then ‘a’ three times. This is because the value of x is set to ‘a’ after the first print statement, and on each subsequent iteration, it prints the updated value of x, which is ‘a’. Therefore, the correct output is ‘abcd’, followed by ‘a’ three times, which doesn’t match any of the given options.

1. What will be the output of the following Python code?

```
x = 123
for i in x:
   print(i)
```

a) 1 2 3
b) 123
c) error
d) none of the mentioned

Answer: c
Explanation: In Python, the for loop is used to iterate over iterable objects like strings, lists, tuples, etc. However, x = 123 is an integer, and integers are not iterable. So when Python tries to iterate over x, it raises a TypeError.

Error Message:

```
TypeError: 'int' object is not iterable
```

2. What will be the output of the following Python code?

```
d = {0: 'a', 1: 'b', 2: 'c'}
for i in d:
    print(i)
```

a)
0
1
2
b)
a
b
c
c)
0 a
1 b
2 c

d) none of the mentioned

Answer: a
Explanation: The for loop iterates over the keys of the dictionary by default. Since the dictionary d has keys 0, 1, and 2, those are printed. To print values, you'd need to use d.values().

3. What will be the output of the following Python code?

```
d = {0: 'a', 1: 'b', 2: 'c'}
for x, y in d:
    print(x, y)
```

a)
0
1
2
b)
a
b
c
c)
0 a
1 b
2 c
d) none of the mentioned

Answer: d
Explanation: The code will raise an error. Iterating over a dictionary directly yields keys, not key-value pairs. To unpack two variables x, y in the loop, you should iterate using .items() like this:

```
for x, y in d.items():
    print(x, y)
```

So, the current code is invalid and results in a ValueError.

4. What will be the output of the following Python code?

```
d = {0: 'a', 1: 'b', 2: 'c'}
for x, y in d.items():
    print(x, y)
```

a)
0
1
2
b)
a
b
c
c)
0 a
1 b
2 c
d) none of the mentioned

Answer: c
Explanation: The d.items() method returns key-value pairs from the dictionary. In each iteration, x gets the key and y gets the corresponding value. Hence, the output is
0 a
1 b
2 c
.

5. What will be the output of the following Python code?

```
d = {0: 'a', 1: 'b', 2: 'c'}
for x in d.keys():
```

```
    print(d[x])
```
a)
0
1
2
b)
a
b
c
c)
0 a
1 b
2 c
d) none of the mentioned

Answer: b
Explanation: The for x in d.keys() loop iterates over the keys (0, 1, 2) of the dictionary d. For each key, d[x] fetches the corresponding value (‘a’, ‘b’, ‘c’), which are then printed. Hence, the output is:
a
b
c

6. What will be the output of the following Python code?

```
d = {0: 'a', 1: 'b', 2: 'c'}
for x in d.values():
    print(x)
```

a)
0
1
2

b)
a
b
c
c)
0 a
1 b
2 c
d) none of the mentioned

Answer: b
Explanation: The d.values() method returns all the values in the dictionary. The for loop iterates over ‘a’, ‘b’, and ‘c’, and prints them. So the output is:
a
b
c

7. What will be the output of the following Python code?

```
d = {0: 'a', 1: 'b', 2: 'c'}
for x in d.values():
    print(d[x])
```

a)
0
1
2
b)
a
b
c
c)

0 a
1 b
2 c
d) none of the mentioned

Answer: d
Explanation: The loop goes through the values in the dictionary (‘a’, ‘b’, ‘c’), and tries to use each value as a key (i.e., d[‘a’], d[‘b’], d[‘c’]). But these strings are not keys in the dictionary, so it causes a KeyError.

8. What will be the output of the following Python code?

```
d = {0, 1, 2}
for x in d.values():
    print(x)
```

a) 0 1 2
b) None None None
c) error
d) none of the mentioned

Answer: c
Explanation: The variable d is a set, not a dictionary. Sets in Python do not have a .values() method. Trying to call d.values() on a set raises an AttributeError.

9. What will be the output of the following Python code?

```
d = {0, 1, 2}
for x in d:
    print(x)
```

a)
0
1
2

b)
{0, 1, 2}
{0, 1, 2}
{0, 1, 2}
c) error
d) none of the mentioned

Answer: a
Explanation: The code creates a set d = {0, 1, 2} and loops through it using a for loop. Sets in Python are unordered, but the loop prints each unique element once. Therefore, the output will be the numbers 0, 1, and 2, each on a new line (order may vary).

10. What will be the output of the following Python code?

```
d = {0, 1, 2}
for x in d:
    print(d.add(x))
```

a)
0
1
2
b)
0 1 2
0 1 2
0 1 2
...
c)
None
None
None

d) None of the mentioned

Answer: c
Explanation: The variable x takes the values 0, 1, and 2. The set.add() method returns None, even if the element is already present in the set. Since print(d.add(x)) prints the return value of the add() method, the output will be None three times.

11. What will be the output of the following Python code?

```
for i in range(0):
    print(i)
```

a) 0
b) no output
c) error
d) none of the mentioned

Answer: b
Explanation: The range(0) function produces an empty sequence, meaning it generates no numbers to iterate over. As a result, the body of the for loop is never executed, and the program produces no output.

1. What will be the output of the following Python code?

```
for i in range(2.0):
    print(i)
```

a) 0.0 1.0
b) 0 1
c) error
d) none of the mentioned

Answer: c
Explanation: The range() function in Python requires integer arguments. Passing a float like 2.0 results in a TypeError because a float cannot be

interpreted as an integer. Therefore, the code will raise an error and not execute the loop.

2. What will be the output of the following Python code?

```
for i in range(int(2.0)):
    print(i)
```

a)
0.0
1.0
b)
0
1
c) error
d) none of the mentioned

Answer: b
Explanation: The expression int(2.0) converts the float 2.0 to the integer 2. So, range(int(2.0)) becomes range(2), which produces the sequence 0, 1.

3. What will be the output of the following Python code?

```
for i in range(float('inf')):
    print (i)
```

a) 0.0 0.1 0.2 0.3 …
b) 0 1 2 3 …
c) 0.0 1.0 2.0 3.0 …
d) none of the mentioned

Answer: d
Explanation: The code attempts to use float('inf') as an argument to range(). However, range() requires integer arguments, and passing a float

— even one representing infinity — will raise a TypeError. Specifically, Python will say: “TypeError: ‘float’ object cannot be interpreted as an integer”

4. What will be the output of the following Python code?

```
for i in range(int(float('inf'))):
    print (i)
```

a) 0.0 0.1 0.2 0.3 …
b) 0 1 2 3 …
c) 0.0 1.0 2.0 3.0 …
d) none of the mentioned

Answer: d
Explanation: The code attempts to convert float(‘inf’) (positive infinity) to an integer using int(float(‘inf’)). This results in an OverflowError because Python cannot convert infinite floating-point values to integers.

5. What will be the output of the following Python code snippet?

```
for i in [1, 2, 3, 4][::-1]:
    print(i, end=' ')
```

a) 1 2 3 4
b) 4 3 2 1
c) error
d) none of the mentioned

Answer: b
Explanation: The expression [1, 2, 3, 4][::-1] uses slicing with a step of -1 to reverse the list. So the list becomes [4, 3, 2, 1]. The for loop iterates over this reversed list and prints each element, with end=’ ‘ ensuring the output is on one line with spaces in between.

6. What will be the output of the following Python code snippet?

```
for i in ''.join(reversed(list('abcd'))):
```

```
    print (i)
```

a)
a
b
c
d
b)
d
c
b
a
c) error
d) none of the mentioned

Answer: b
Explanation: The code does the following:

- list('abcd') → converts the string to a list: ['a', 'b', 'c', 'd'].
- reversed(…) → returns an iterator that yields the elements in reverse: ['d', 'c', 'b', 'a'].
- ".join(…) → joins the reversed characters into a string: 'dcba'.

The for loop iterates over the characters in 'dcba', printing each on a new line.
Output:
d
c
b
a
Each character is printed on a separate line because print(i) by default ends with a newline.
7. What will be the output of the following Python code snippet?

```
for i in 'abcd'[::-1]:
    print (i)
```

a)
a
b
c
d
b)
d
c
b
a
c) error
d) none of the mentioned

Answer: b
Explanation: The expression ‘abcd'[::-1] uses slicing with a step of -1 to reverse the string, resulting in ‘dcba’. The for loop then iterates over this reversed string, printing each character on a new line.
Output:
D
c
b
a

8. What will be the output of the following Python code snippet?

```
for i in '':
    print (i)
```

a) None
b) (nothing is printed)
c) error

d) none of the mentioned

Answer: b
Explanation: The string “” is an empty string, meaning it contains no characters. When a for loop tries to iterate over it, there’s nothing to process, so the loop body doesn’t run at all. As a result, the program doesn’t produce any output—nothing is printed, and no error or message like None appears. It simply moves on without doing anything.

9. What will be the output of the following Python code snippet?

```
x = 2
for i in range(x):
    x += 1
    print (x)
```

a)
0
1
2
3
b)
0
1
c)
3
4
d) error

Answer: c
Explanation: The loop runs twice because range(x) is evaluated at the beginning when x is 2. Inside the loop, x is incremented by 1 each time,

so the first iteration prints 3 and the second prints 4. Hence, the output is 3 and 4.

10. What will be the output of the following Python code snippet?

```
x = 2
for i in range(x):
   x -= 2
   print (x)
```

a)
0
1
2
3
4
...
b)
0
-2
c) 0
d) error

Answer: b
Explanation: Initially, x = 2. The loop runs for range(x) where x = 2, so the loop will iterate twice. In each iteration, x is decreased by 2 (x -= 2), and the value of x is printed.

- 1st iteration: x = 2 – 2 = 0, prints 0.
- 2nd iteration: x = 0 – 2 = -2, prints -2.

Thus, the output is 0 and -2.

1. What will be the output of the following Python code?

```
for i in range(10):
```

```
    if i == 5:
        break
    else:
        print(i)
else:
    print("Here")
```

a)
0
1
2
3
4
Here

b)
0
1
2
3
4
5
Here

c)
0
1
2
3
4

d) error

Answer: c
Explanation: In this code, the loop runs from 0 to 9, and when i equals 5, the break statement is executed, causing the loop to exit. The else block will not execute because the loop is terminated by the break statement before it completes normally. Therefore, the loop prints numbers 0 through 4, and the else block is skipped.
Output:
0
1
2
3
4

2. What will be the output of the following Python code?

```
for i in range(5):
    if i == 5:
        break
    else:
        print(i)
else:
    print("Here")
```

a)
0
1
2
3
4
Here
b)
0
1

2
3
4
5
Here
c)
0
1
2
3
4
d) error

Answer: a
Explanation: The for loop iterates over range(5), which gives values from 0 to 4. When i == 5, the break statement would be triggered, but that condition is never met because the loop only goes up to 4. Since there is no break within the loop, the else part will execute, printing “Here.” The loop prints the numbers 0, 1, 2, 3, and 4 before exiting, followed by “Here”.

3. What will be the output of the following Python code?

```
x = (i for i in range(3))
for i in x:
    print(i)
```

a)
0
1
2
b) error
c)

0
1
2
0
1
2
d) none of the mentioned

Answer: a
Explanation: The code creates a generator expression (i for i in range(3)) that generates values from 0 to 2. The for loop iterates over the generator, printing each value. Since a generator only produces values on demand, it will output 0, 1, and 2 once as it exhausts the generator.

4. What will be the output of the following Python code?

```
x = (i for i in range(3))
for i in x:
    print(i)
for i in x:
    print(i)
```

a)
0
1
2
b) error
c)
0
1
2
0
1

2
d) none of the mentioned

Answer: a
Explanation: The code creates a generator object x which yields values 0, 1, and 2. In the first for loop, the generator is exhausted after it iterates over all the values. A generator can only be traversed once. Once it's exhausted, the second for loop will not produce any output, resulting in no values being printed in the second loop.

5. What will be the output of the following Python code?

```
string = "my name is x"
for i in string:
    print (i, end=", ")
```

a) m, y, , n, a, m, e, , i, s, , x,
b) m, y, , n, a, m, e, , i, s, , x
c) my, name, is, x,
d) error

Answer: a
Explanation: The for loop iterates over each character in the string "my name is x", and the print(i, end=", ") statement prints each character followed by a comma and a space. Since the loop processes each character individually, the output includes each character of the string separated by a comma and space.
Output:

```
m, y,  , n, a, m, e,  , i, s,  , x,
```

6. What will be the output of the following Python code?

```
string = "my name is x"
for i in string.split():
    print (i, end=", ")
```

a) m, y, , n, a, m, e, , i, s, , x,
b) m, y, , n, a, m, e, , i, s, , x
c) my, name, is, x,
d) error

Answer: c
Explanation: The split() method splits the string “my name is x” into words, resulting in the list [‘my’, ‘name’, ‘is’, ‘x’]. The for loop then iterates over each word in the list, printing each word followed by a comma and space.

7. What will be the output of the following Python code snippet?

```
a = [0, 1, 2, 3]
for a[-1] in a:
    print(a[-1])
```

a)
0
1
2
3
b)
0
1
2
2
c)
3
3
3
3

d) error

Answer: b
Explanation: The code modifies the last element of the list a during each iteration of the loop. Initially, a[-1] is set to 0, then updated to 1, and finally 2. In each iteration, the new value of a[-1] is printed, resulting in the output 0, 1, 2, 2.

8. What will be the output of the following Python code snippet?

```
a = [0, 1, 2, 3]
for a[0] in a:
    print(a[0])
```

a)
0
1
2
3
b)
0
1
2
2
c)
3
3
3
3
d) error

Answer: a
Explanation: In this code, the first element of the list a[0] is updated

during each iteration of the for loop. Initially, a = [0, 1, 2, 3], and during the loop, a[0] is assigned each value from the list. The loop will print the updated value of a[0] each time. Since the first value of the loop is 0, it prints 0, and then the loop continues updating and printing a[0] with each iteration, ultimately printing 0, 1, 2, 3.

9. What will be the output of the following Python code snippet?

```
a = [0, 1, 2, 3]
i = -2
for i not in a:
   print(i)
   i += 1
```

a) -2 -1
b) 0
c) error
d) none of the mentioned

Answer: c
Explanation: The statement for i not in a: is invalid syntax in Python. The for loop requires an iterable (e.g., for i in a:), but i not in a is a condition, not an iterable. Hence, Python raises a SyntaxError.

10. What will be the output of the following Python code snippet?

```
string = "my name is x"
for i in ' '.join(string.split()):
   print (i, end=", ")
```

a) m, y, , n, a, m, e, , i, s, , x,
b) m, y, , n, a, m, e, , i, s, , x
c) my, name, is, x,
d) error

Answer: a
Explanation: The string.split() breaks the sentence into words: [‘my’, ‘name’, ‘is’, ‘x’]. Then ‘ ‘.join(…) combines them back into a single string “my name is x”. The for loop iterates over this string one character at a time, printing each character followed by a comma and space.

1. What will be the output of the following Python statement?

```
print("a"+"bc")
```

a) a
b) bc
c) bca
d) abc

Answer: d
Explanation: In Python, the + operator is used for string concatenation. “a” + “bc” joins the two strings together into a single string “abc”.

2. What will be the output of the following Python statement?

```
print("abcd"[2:])
```

a) a
b) ab
c) cd
d) dc

Answer: c
Explanation: The expression “abcd”[2:] uses slicing to extract the substring starting from index 2 to the end. In the string “abcd”, index 2 corresponds to ‘c’, so the result is ‘cd’.

3. The output of executing string.ascii_letters can also be achieved by:

a) string.ascii_lowercase_string.digits
b) string.ascii_lowercase+string.ascii_uppercase

c) string.letters
d) string.lowercase_string.uppercase

4. What will be the output of the following Python code?

```
str1 = 'hello'
str2 = ','
str3 = 'world'
print(str1[-1:])
```

a) olleh
b) hello
c) h
d) o

Answer: d
Explanation: In Python, str1[-1:] slices the string starting from the last character (-1) to the end. For str1 = ‘hello’, the last character is ‘o’.

5. What arithmetic operators cannot be used with strings?

a) +
b) *
c) –
d) All of the mentioned

Answer: c
Explanation: In Python, the + operator is used to concatenate strings, and the * operator is used to repeat strings. However, the – (minus) operator is not supported for strings and will raise a TypeError if used.

6. What will be the output of the following Python code?

```
print(r"\nhello")
```

a) a new line and hello
b) \nhello

c) the letter r and then hello
d) error

Answer: b
Explanation: The prefix r makes the string a raw string, meaning escape sequences like \n are not interpreted. So print(r"\nhello") outputs the string exactly as written, including the backslash.
Output:

```
\nhello
```

7. What will be the output of the following Python statement?

```
print('new' 'line')
```

a) Error
b) Output equivalent to print 'new\nline'
c) newline
d) new line

Answer: c
Explanation: In Python, adjacent string literals without any operator are automatically concatenated. So 'new' 'line' becomes 'newline'. No error occurs, and no whitespace is added unless explicitly included.

8. What will be the output of the following Python statement?

```
print('x\97\x98')
```

a) Error
b)
97
98
c) x\97
d) \x97\x98

Answer: c
Explanation: The output of the code print('x\97\x98') is x\97. Here, \97 is not a valid escape sequence, so it's treated as the literal characters \, 9, and 7. On the other hand, \x98 is a valid hexadecimal escape sequence, which may produce an unprintable character, often shown as .

9. What will be the output of the following Python code?

```
str1="helloworld"
print(str1[::-1])
```

a) dlrowolleh
b) hello
c) world
d) helloworld

Answer: a
Explanation: The slice notation [::-1] reverses the string. In the case of str1 = "helloworld", applying [::-1] will reverse the string and result in "dlrowolleh".

10. What will be the output of the following Python code?

```
print(0xA + 0xB + 0xC)
```

a) 0xA0xB0xC
b) Error
c) 0x22
d) 33

Answer: d
Explanation: 0xA and 0xB and 0xC are hexadecimal integer literals representing the decimal values 10, 11 and 12 respectively. Their sum is 10 + 11 + 12 = 33.

1. What will be the output of the following Python code?

```
class father:
```

```
    def __init__(self, param):
        self.o1 = param

class child(father):
    def __init__(self, param):
        self.o2 = param

obj = child(22)
print("%d %d" % (obj.o1, obj.o2))
```

a) None None
b) None 22
c) 22 None
d) Error is generated

Answer: d
Explanation: The code will generate an error because in the child class's __init__ method, the father class's __init__ method is not called. As a result, self.o1 is never created. To fix this, you need to call the parent class's __init__ method using super().__init__(param) in the child class's __init__.

2. What will be the output of the following Python code?

```
class tester:
    def __init__(self, id):
        self.id = str(id)
        id="224"

temp = tester(12)
print(temp.id)
```

a) 224
b) Error
c) 12
d) None

Answer: c
Explanation: When the tester class is instantiated with temp = tester(12), the __init__ method is called. The id argument is passed as 12, and inside the __init__ method, self.id is assigned the string value of id, which is “12”. However, the local variable id is reassigned to “224”, but this change does not affect self.id, which retains the value “12”.

3. What will be the output of the following Python code?

```
example = "snow world"
print("%s" % example[4:7])
```

a) ” wo”
b) ” world”
c) “sn ”
d) ” rl”

Answer: a
Explanation: In the string “snow world”, the slice example[4:7] refers to the characters starting at index 4 and ending just before index 7. Since the index range is exclusive of the end, it retrieves the substring ” wo”, which includes a space followed by the letters “w” and “o”.

4. What will be the output of the following Python code?

```
example = "snow world"
print(example[3] = 's')
```

a) snow
b) snow world
c) Error

d) snos world

Answer: c
Explanation: In Python, strings are immutable, meaning you cannot modify individual characters of a string. The code example[3] = ‘s’ tries to assign a new value to a specific index of the string, which will raise an error. This results in a TypeError.

5. What will be the output of the following Python code?

```
print(max("what are you"))
```

a) error
b) u
c) t
d) y

Answer: d
Explanation: The max() function, when applied to a string, returns the character with the highest ASCII value. In the string “what are you”, the character ‘y’ has the highest ASCII value, so it is returned.

6. Given a string example=”hello” what is the output of example.count(‘l’)?

a) 2
b) 1
c) None
d) 0

Answer: a
Explanation: The method count(‘l’) returns the number of times the character ‘l’ appears in the string “hello”. In this case, ‘l’ occurs twice, so the output is 2.

7. What will be the output of the following Python code?

```
example = "helle"
print(example.find("e"))
```

a) Error
b) -1
c) 1
d) 0

Answer: c
Explanation: The find() method returns the index of the first occurrence of the specified substring. In the string “helle”, the first occurrence of the letter ‘e’ is at index 1.

8. What will be the output of the following Python code?

```
example = "helle"
print(example.rfind("e"))
```

a) -1
b) 4
c) 3
d) 1

Answer: b
Explanation: The rfind() method returns the highest index of the specified substring in the string. In the string “helle”, the last occurrence of ‘e’ is at index 4.

9. What will be the output of the following Python code?

```
example="helloworld"
print(example[::-1].startswith("d"))
```

a) dlrowolleh
b) True
c) -1

d) None

Answer: b
Explanation: example[::-1] reverses the string “helloworld” to “dlrowolleh”. Then, .startswith(“d”) checks if the reversed string starts with ‘d’, which it does—so the output is True.

10. To concatenate two strings to a third what statements are applicable?

a) s3 = s1 . s2
b) s3 = s1.add(s2)
c) s3 = s1.__add__(s2)
d) s3 = s1 * s2

Answer: c
Explanation: The __add__() method is a special method in Python used to define the behavior of the + operator. It can be used for concatenating two strings. The statement s3 = s1.__add__(s2) is equivalent to s3 = s1 + s2.

1. What will be the output of the following Python statement?

```
print(chr(ord('A')))
```

a) A
b) B
c) a
d) Error

Answer: a
Explanation: The ord() function returns the Unicode code point (integer) of the character ‘A’, which is 65. The chr() function converts that Unicode value back to its corresponding character. So chr(ord(‘A’)) returns ‘A’.

2. What will be the output of the following Python statement?

```
print(chr(ord('b')+1))
```

a) a
b) b
c) c
d) A

Answer: c
Explanation: The ord('b') returns the Unicode code point of 'b', which is 98. Adding 1 gives 99, and chr(99) returns 'c'.

3. Which of the following statement prints hello\example\test.txt?

a) print("hello\example\test.txt")
b) print("hello\\example\\test.txt")
c) print("hello\"example\"test.txt")
d) print("hello"\example"\test.txt")

Answer: b
Explanation: In Python, the backslash \ is an escape character, so to print a file path like hello\example\test.txt, each backslash must be escaped as \\. Therefore, the correct statement is print("hello\\example\\test.txt"), which produces the intended output with backslashes displayed properly.

4. Suppose s is "\t\tWorld\n", what is s.strip()?

a) \t\tWorld\n
b) \t\tWorld\n
c) \t\tWORLD\n
d) World

Answer: d
Explanation: In the string s = "\t\tWorld\n", the strip() method removes leading and trailing whitespace characters such as tabs (\t) and newlines

(\n). As a result, s.strip() returns “World”. This is useful for cleaning up user input or formatting strings.

5. The format function, when applied on a string returns _____

a) Error
b) int
c) bool
d) str

Answer: d
Explanation: The format() function in Python returns a string where placeholders in the original string are replaced with specified values. It’s commonly used for string formatting.

6. What will be the output of the “hello” +1+2+3?

a) hello123
b) hello
c) Error
d) hello6

Answer: c
Explanation: In Python, you cannot directly concatenate a string with integers using +. “hello” + 1 + 2 + 3 will raise a TypeError because “hello” is a string and the others are integers.

7. What will be the output of the following Python code?

```
print("D", end = ' ')
print("C", end = ' ')
print("B", end = ' ')
print("A", end = ' ')
```

a) DCBA
b) A, B, C, D
c) D C B A

d) D, C, B, A will be displayed on four lines

Answer: c
Explanation: The end=’ ‘ argument in the print() function prevents the default newline and replaces it with a space. So, all values are printed on the same line, separated by spaces.

8. What will be the output of the following Python statement?(python 3.xx)

```
print(format("Welcome", "10s"), end = '#')
print(format(111, "4d"), end = '#')
print(format(924.656, "3.2f"))
```

a) Welcome# 111#924.66
b) Welcome#111#924.66
c) Welcome#111#.66
d) Welcome # 111#924.66

Answer: d
Explanation: The output of the given Python code formats the string “Welcome” with a width of 10 characters, padding it with spaces. The number 111 is formatted with a width of 4 characters, and the float 924.656 is rounded to two decimal places. The formatted outputs are then printed with # as separators, resulting in “Welcome # 111#924.66”.

9. What will be displayed by print(ord(‘b’) – ord(‘a’))?

a) 0
b) 1
c) -1
d) 2

Answer: b
Explanation: The ord() function returns the ASCII value of a character. The

ASCII value of ‘b’ is 98 and that of ‘a’ is 97. Therefore, the expression ord(‘b’) – ord(‘a’) equals 98 – 97, which is 1.

10. Say s=”hello” what will be the return value of type(s)?

a) int
b) bool
c) str
d) String

Answer: c
Explanation: In Python, strings are represented by the str type. The type() function returns the type of an object, and for the string “hello”, it will return <class ‘str’>, which is the str type.

1. What is “Hello”.replace(“l”, “e”)?

a) Heeeo
b) Heelo
c) Heleo
d) None

Answer: a
Explanation: The replace() method in Python returns a new string where all occurrences of the substring “l” are replaced with “e”. In the string “Hello”, this means every “l” will be replaced with “e”, resulting in the string “Heeeo”.

2. To retrieve the character at index 3 from string s=”Hello” what command do we execute (multiple answers allowed)?

a) s[]
b) s.getitem(3)
c) s.__getitem__(3)
d) s.getItem(3)

Answer: c
Explanation: In Python, __getitem__() is the method that allows access to an element of a sequence, such as a string, at a specific index. You can use s.__getitem__(3) to retrieve the character at index 3 of the string s. However, the more common and preferred way is to use the square brackets, like s[3].

3. To return the length of string s what command do we execute?

a) len(s) OR s.__len__()
b) len(s) only
c) size(s)
d) s.size()

Answer: a
Explanation: In Python, the len(s) function is the standard way to get the length of a string s. Internally, len(s) calls the __len__() method of the string object. While it's more common to use len(s), using s.__len__() directly will also return the length of the string.

4. If a class defines the __str__(self) method, for an object obj for the class, you can use which command to invoke the __str__ method.

a) obj.__str__()
b) str(obj)
c) print obj
d) all of the mentioned

Answer: d
Explanation: If a class defines the __str__(self) method, it can be invoked in multiple ways:

- obj.__str__() directly calls the method.
- str(obj) automatically invokes __str__().

- print(obj) also calls __str__() internally to get the string representation.

So, all options are valid ways to invoke the __str__() method.

5. To check whether string s1 contains another string s2, use _____

a) s1.__contains__(s2)
b) s1.contains(s2)
c) s1.in(s2)
d) s2.in(s1)

Answer: a
Explanation: The method s1.__contains__(s2) checks if s2 is present in s1. It is the underlying method used when using the in operator in Python.

6. Suppose i is 5 and j is 4, i + j is same as _____

a) i.__add(j)
b) i.__add__(j)
c) i.__Add(j)
d) i.__ADD(j)

Answer: b
Explanation: In Python, the __add__ method is the special method used to implement the behavior of the + operator. When you use i + j, Python internally calls i.__add__(j). It's important to use the double underscore (__add__) and not a single or capitalized version.

7. What will be the output of the following Python code?

```
class Count:
   def __init__(self, count = 0):
      self.__count = count

c1 = Count(2)
c2 = Count(2)
```

```
print(id(c1) == id(c2), end = " ")

s1 = "Good"
s2 = "Good"
print(id(s1) == id(s2))
```

a) True False
b) True True
c) False True
d) False False

Answer: c
Explanation: In the case of objects, id(c1) and id(c2) will be different, even if the objects have the same value. However, Python optimizes memory usage for immutable objects like strings, so id(s1) and id(s2) will be the same.

8. What will be the output of the following Python code?

```
class Name:
   def __init__(self, firstName, mi, lastName):
      self.firstName = firstName
      self.mi = mi
      self.lastName = lastName

firstName = "John"
name = Name(firstName, 'F', "Smith")
firstName = "Peter"
name.lastName = "Pan"
print(name.firstName, name.lastName)
```

a) Peter Pan
b) John Pan
c) Peter Smith

d) John Smith

Answer: b
Explanation: The code defines a Name class, and creates an instance name with the first name “John” and last name “Smith”. Although the firstName variable outside the class is changed to “Peter”, the instance variable name.firstName remains “John”. After updating name.lastName to “Pan”, the final output is “John Pan”.

9. What function do you use to read a string?

a) input(“Enter a string”)
b) eval(input(“Enter a string”))
c) enter(“Enter a string”)
d) eval(enter(“Enter a string”))

Answer: a
Explanation: The input() function is used to take user input as a string in Python. It reads the line of text typed by the user and returns it as a string.

10. Suppose x is 345.3546, what is format(x, “10.3f”) (_ indicates space).

a) __345.355
b) ___345.355
c) ____345.355
d) _____345.354

Answer: b
Explanation: The format(x, “10.3f”) specifies that the total width should be 10 characters (including spaces and the decimal point), with 3 digits after the decimal point. The number 345.3546 will be rounded to 345.355, and the result will be right-aligned with spaces padding the left side to make the total width 10 characters.

1. What will be the output of the following Python code?

```
print("abc DEF".capitalize())
```

a) abc def
b) ABC DEF
c) Abc def
d) Abc Def

Answer: c
Explanation: The .capitalize() method in Python returns a copy of the string with the first character converted to uppercase and the rest to lowercase. So, “abc DEF”.capitalize() results in “Abc def”.

2. What will be the output of the following Python code?

```
print("abc. DEF".capitalize())
```

a) abc. def
b) ABC. DEF
c) Abc. def
d) Abc. Def

Answer: c
Explanation: The capitalize() method in Python modifies a string such that only the first character is converted to uppercase while the rest are converted to lowercase. When applied to “abc. DEF”, it changes the string to “Abc. def”.

3. What will be the output of the following Python code?

```
print("abcdef".center())
```

a) cd
b) abcdef
c) error

d) none of the mentioned

Answer: c
Explanation: The center() method in Python requires at least one argument, which specifies the total width of the resulting string after centering. If you call it without any arguments, like in print(“abcdef”.center()), it raises a TypeError.

4. What will be the output of the following Python code?

```
print("abcdef".center(0))
```

a) cd
b) abcdef
c) error
d) none of the mentioned

Answer: b
Explanation: The center(width) method centers the string in a field of given width. If the width is less than or equal to the string length, the original string is returned unchanged. So, “abcdef”.center(0) returns “abcdef”.

5. What will be the output of the following Python code?

```
print('*', "abcdef".center(7), '*')
```

a) * abc def *
b) * abcdef *
c) *abcdef *
d) * abcdef*

Answer: b
Explanation: The str.center(width) method centers a string within a given total width by padding it with spaces (by default). If the total width is

greater than the string's length, spaces are added equally to both sides as much as possible. If the extra space is odd, left side gets more space.

6. What will be the output of the following Python code?

```
print('*', "abcdef".center(7), '*', sep='')
```

a) * abcdef *
b) * abcdef *
c) *abcdef *
d) * abcdef*

Answer: d
Explanation: The string "abcdef" has 6 characters, and using .center(7) adds one extra space to center it in a field of width 7. Since the padding is uneven, Python places the extra space on the left. With sep='', the final output is * abcdef*.

7. What will be the output of the following Python code?

```
print('*', "abcde".center(6), '*', sep='')
```

a) * abcde *
b) * abcde *
c) *abcde *
d) * abcde*

Answer: c
Explanation: The string "abcde" has 5 characters, and .center(6) adds one space to make it fit a total width of 6. Since the padding is uneven and the total width is even, Python adds the extra space to the right. With sep='', the output becomes *abcde *.

8. What will be the output of the following Python code?

```
print("abcdef".center(7, 1))
```

a) 1abcdef
b) abcdef1

c) abcdef
d) error

Answer: d
Explanation: The center() method takes two parameters: the total width of the final string and an optional fill character (which must be a string of length 1). Here, 1 is an integer, not a string. Passing an integer as a fill character raises a TypeError.

9. What will be the output of the following Python code?

```
print("abcdef".center(7, '1'))
```

a) 1abcdef
b) abcdef1
c) abcdef
d) error

Answer: a
Explanation: The string “abcdef” has 6 characters, and .center(7, ‘1’) pads it to a width of 7 using ‘1’ as the fill character. Since only 1 padding character is needed, and the total width is odd, Python adds the extra character to the left, resulting in ‘1abcdef’.

10. What will be the output of the following Python code?

```
print("abcdef".center(10, '12'))
```

a) 12abcdef12
b) abcdef1212
c) 1212abcdef
d) error

Answer: d
Explanation: The center() method in Python requires the fill character to

be exactly one character long. Since ’12’ is two characters, this raises a TypeError.

1. What will be the output of the following Python code?

```
print("xyyzxyzxzxyy".count('yy'))
```

a) 2
b) 0
c) error
d) none of the mentioned

Answer: a
Explanation: The .count(‘yy’) method counts non-overlapping occurrences of ‘yy’ in the string “xyyzxyzxzxyy”. It appears twice: once at the beginning (“xyy”) and once at the end (“xyy”).

2. What will be the output of the following Python code?

```
print("xyyzxyzxzxyy".count('yy', 1))
```

a) 2
b) 0
c) 1
d) none of the mentioned

Answer: a
Explanation: The method .count(‘yy’, 1) counts occurrences of ‘yy’ starting from index 1 onward in the string “xyyzxyzxzxyy”. Both ‘yy’ substrings are still within this range, so the result is 2.

3. What will be the output of the following Python code?

```
print("xyyzxyzxzxyy".count('yy', 2))
```

a) 2
b) 0
c) 1

d) none of the mentioned

Answer: c
Explanation: The method .count('yy', 2) starts searching for 'yy' from index 2 of the string "xyyzxyzxzxyy". At this point, only the second occurrence of 'yy' (at the end) is within range, so the count is 1.

4. What will be the output of the following Python code?

```
print("xyyzxyzxzxyy".count('xyy', 0, 100))
```

a) 2
b) 0
c) 1
d) error

Answer: a
Explanation: The given Python code counts the number of times the substring 'xyy' appears in the string "xyyzxyzxzxyy" between indices 0 and 100. Although 100 exceeds the string's length, Python handles it gracefully. The substring 'xyy' appears twice, so the output is 2.

5. What will be the output of the following Python code?

```
print("xyyzxyzxzxyy".count('xyy', 2, 11))
```

a) 2
b) 0
c) 1
d) error

Answer: b
Explanation: The method .count('xyy', 2, 11) searches for the substring 'xyy' in "xyyzxyzxzxyy", starting from index 2 and ending just before index 11. Since 'xyy' is not found between positions 2 and 10 (the end index 11 is exclusive), the count is 0.

6. What will be the output of the following Python code?

```
print("xyyzxyzxzxyy".count('xyy', -10, -1))
```

a) 2
b) 0
c) 1
d) error

Answer: b
Explanation: The method .count('xyy', -10, -1) searches for the substring 'xyy' in "xyyzxyzxzxyy", starting at position -10 (which is the same as position 4) and ending at position -1 (which is the last character). In this range, 'xyy' is not found, so the count is 0.

7. What will be the output of the following Python code?

```
print('abc'.encode())
```

a) abc
b) 'abc'
c) b'abc'
d) h'abc'

Answer: c
Explanation: The .encode() method converts a string into a bytes object. In this case, 'abc'.encode() returns b'abc', which is a bytes representation of the string.

8. What is the default value of encoding in encode()?

a) ascii
b) qwerty
c) utf-8
d) utf-16

Answer: c
Explanation: In Python, the encode() method converts a string into bytes using a specified encoding. If no encoding is explicitly provided, it defaults to UTF-8, which is a widely used and versatile character encoding standard.

9. What will be the output of the following Python code?

```
print("xyyzxyzxzxyy".endswith("xyy"))
```

a) 1
b) True
c) 3
d) 2

Answer: b
Explanation: The string “xyyzxyzxzxyy” does end with the substring “xyy”, so the endswith() method returns True.

10. What will be the output of the following Python code?

```
print("xyyzxyzxzxyy".endswith("xyy", 0, 2))
```

a) 0
b) 1
c) True
d) False

Answer: d
Explanation: The .endswith(“xyy”, 0, 2) method checks if the substring “xyy” is found at the end of the string “xyyzxyzxzxyy”, within the specified range from index 0 to index 2. Since the substring “xyy” doesn’t appear in that range (the range covers only “xy”), the result is False.

1. What will be the output of the following Python code?

```
print("ab\tcd\tef".expandtabs())
```

a) ab cd ef
b) abcdef
c) ab\tcd\tef
d) ab cd ef

Answer: a
Explanation: The expandtabs() function replaces each tab character (\t) in the string with spaces. By default, the tab size is 8 spaces. However, it fills up to the next multiple of 8 from the current character position.

2. What will be the output of the following Python code?

```
print("ab\tcd\tef".expandtabs(4))
```

a) ab cd ef
b) abcdef
c) ab\tcd\tef
d) ab cd ef

Answer: d
Explanation: The expandtabs() function in Python replaces tab characters (\t) in a string with spaces. By default, each tab is expanded to 8 spaces, but you can specify a custom tab size. For example, expandtabs(4) replaces each tab with up to 4 spaces based on the current character position in the string.

3. What will be the output of the following Python code?

```
print("ab\tcd\tef".expandtabs('+'))
```

a) ab+cd+ef
b) ab++++++++cd++++++++ef
c) ab cd ef
d) none of the mentioned

Answer: d
Explanation: The expandtabs() method expects an integer as an argument to specify the number of spaces to replace each tab character. Passing a string like ‘+’ causes a TypeError, making “none of the mentioned” the correct answer.

4. What will be the output of the following Python code?

```
print("abcdef".find("cd") == "cd" in "abcdef")
```

a) True
b) False
c) Error
d) None of the mentioned

Answer: b
Explanation: The code checks if the index of “cd” in “abcdef” is equal to whether “cd” is in “abcdef”. The find() method returns the position of “cd” (which is 2), but in checks if “cd” exists and gives a True or False result. Since 2 == True is False, the output is False.

5. What will be the output of the following Python code?

```
print("abcdef".find("cd"))
```

a) True
b) 2
c) 3
d) None of the mentioned

Answer: b
Explanation: The find() method in Python returns the index of the first occurrence of the specified substring. In this case, “cd” starts at index 2 in the string “abcdef”, so the output will be 2.

6. What will be the output of the following Python code?

```
print("ccdcddcd".find("c"))
```
a) 4
b) 0
c) Error
d) True

Answer: b
Explanation: The find() method returns the index of the first occurrence of the substring. In this case, the first occurrence of “c” is at index 0 in the string “ccdcddcd”, so the output will be 0.

7. What will be the output of the following Python code?

```
print("Hello {0} and {1}".format('foo', 'bin'))
```
a) Hello foo and bin
b) Hello {0} and {1} foo bin
c) Error
d) Hello 0 and 1

Answer: a
Explanation: The str.format() method replaces {0} and {1} with the first and second arguments respectively. So ‘foo’ replaces {0} and ‘bin’ replaces {1} resulting in “Hello foo and bin”.

8. What will be the output of the following Python code?

```
print("Hello {1} and {0}".format('bin', 'foo'))
```
a) Hello foo and bin
b) Hello bin and foo
c) Error
d) None of the mentioned

Answer: a
Explanation: The given Python code uses the format() method to insert

values into a string using positional indexes. {1} refers to the second argument (‘foo’) and {0} refers to the first argument (‘bin’). Therefore, the output is “Hello foo and bin”.

9. What will be the output of the following Python code?

```
print("Hello {} and {}".format('foo', 'bin'))
```

a) Hello foo and bin
b) Hello {} and {}
c) Error
d) Hello and

Answer: a
Explanation: The format() method is used to substitute the {} placeholders with the values passed as arguments. In this case, {} will be replaced by ‘foo’ and ‘bin’. Therefore, the output will be: Hello foo and bin.

10. What will be the output of the following Python code?

```
print("Hello {name1} and {name2}".format('foo', 'bin'))
```

a) Hello foo and bin
b) Hello {name1} and {name2}
c) Error
d) Hello and

Answer: c
Explanation: The code provided leads to an error because the placeholders {name1} and {name2} expect values to be passed with named arguments. In the given example, only positional arguments (‘foo’ and ‘bin’) are provided, causing a mismatch. To avoid the error, named arguments should be used in the format() method.

1. What will be the output of the following Python code?

```
print("Hello {name1} and {name2}".format(name1='foo', name2='bin'))
```

a) Hello foo and bin
b) Hello {name1} and {name2}
c) Error
d) Hello and

Answer: a
Explanation: The .format() method in Python replaces placeholders in a string with specified values. Here, {name1} and {name2} are named placeholders that get replaced by ‘foo’ and ‘bin’ respectively. As a result, the output is: “Hello foo and bin”.

2. What will be the output of the following Python code?

```
print("Hello {0!r} and {0!s}".format('foo', 'bin'))
```

a) Hello foo and foo
b) Hello ‘foo’ and foo
c) Hello foo and ‘bin’
d) Error

Answer: b
Explanation: The output of the code is Hello ‘foo’ and foo. Here, {0!r} uses repr() which includes quotes around the string, while {0!s} uses str() which prints the string without quotes. Only the first argument ‘foo’ is used in both placeholders.

3. What will be the output of the following Python code?

```
print("Hello {0} and {1}".format(('foo', 'bin')))
```

a) Hello foo and bin
b) Hello (‘foo’, ‘bin’) and (‘foo’, ‘bin’)
c) Error

d) None of the mentioned

Answer: c
Explanation: The .format() method expects each placeholder index to match a corresponding argument. Here, only one argument is passed—a tuple (‘foo’, ‘bin’) but the format string tries to access two separate arguments ({0} and {1}). Since there’s no second top-level argument, this results in an IndexError.

4. What will be the output of the following Python code?

```
print("Hello {0[0]} and {0[1]}".format(('foo', 'bin')))
```

a) Hello foo and bin
b) Hello (‘foo’, ‘bin’) and (‘foo’, ‘bin’)
c) Error
d) None of the mentioned

Answer: a
Explanation: The output of the code is Hello foo and bin. Here, the format string uses index-based access to retrieve elements from the tuple passed as a single argument. {0[0]} and {0[1]} access the first and second elements of the tuple respectively.

5. What will be the output of the following Python code snippet?

```
print('The sum of {0} and {1} is {2}'.format(2, 10, 12))
```

a) The sum of 2 and 10 is 12
b) Error
c) The sum of 0 and 1 is 2
d) None of the mentioned

Answer: a
Explanation: The output of the code is The sum of 2 and 10 is 12. Here, the format() method simply replaces the placeholders {0}, {1}, and {2} with the values 2, 10, and 12, respectively. Integers can be directly passed as arguments to the format() function.

6. What will be the output of the following Python code snippet?

```
print('The sum of {0:b} and {1:x} is {2:o}'.format(2, 10, 12))
```

a) The sum of 2 and 10 is 12
b) The sum of 10 and a is 14
c) The sum of 10 and a is c
d) Error

Answer: b
Explanation: The .format() method uses format specifiers to convert the given integers into different bases. {0:b} converts 2 to binary (10), {1:x} converts 10 to hexadecimal (a), and {2:o} converts 12 to octal (14). Therefore, the output is: “The sum of 10 and a is 14”.

7. What will be the output of the following Python code snippet?

```
print('{:,}'.format(1112223334))
```

a) 1,112,223,334
b) 111,222,333,4
c) 1112223334
d) Error

Answer: a
Explanation: In this code, the .format() method is used with the :, format specifier, which adds commas to separate every three digits from the right. So, the number 1112223334 is formatted as 1,112,223,334, with commas added for readability.

8. What will be the output of the following Python code snippet?

```
print('{:,}'.format('1112223334'))
```
a) 1,112,223,334
b) 111,222,333,4
c) 1112223334
d) Error

Answer: d
Explanation: The :, format specifier expects a numeric value (either an integer or a float), but here the input is a string (‘1112223334’). Since the format specifier cannot handle strings for numeric formatting, this results in a TypeError. Therefore, the output is an error.

9. What will be the output of the following Python code snippet?

```
print('{:$}'.format(1112223334))
```
a) 1,112,223,334
b) 111,222,333,4
c) 1112223334
d) Error

Answer: d
Explanation: In Python, the $ symbol is not a valid format specifier for numbers in the .format() method. As a result, attempting to use {:$} leads to a ValueError because $ is not recognized as a valid formatting option.

10. What will be the output of the following Python code snippet?

```
print('{:#}'.format(1112223334))
```
a) 1,112,223,334
b) 111,222,333,4
c) 1112223334
d) Error

Answer: c
Explanation: The # format specifier in Python is used for special formatting, such as adding a prefix for binary, octal, or hexadecimal numbers. However, when applied to a plain integer, it doesn't alter the number. In this case, it prints the number as it is, without any changes. Therefore, the output is 1112223334.

1. What will be the output of the following Python code?

```
print('{0:.2}'.format(1/3))
```

a) 0.333333
b) 0.33
c) 0.333333:.2
d) Error

Answer: b
Explanation: The format specifier {0:.2} is used to format the first argument (1/3) to a precision of 2 significant digits. In this case, 1/3 ≈ 0.3333, and when rounded to 2 significant digits, it becomes 0.33. So, the output is 0.33.

2. What will be the output of the following Python code?

```
print('{0:.2%}'.format(1/3))
```

a) 0.33
b) 0.33%
c) 33.33%
d) 33%

Answer: c
Explanation: The format specifier {0:.2%} converts the number to a percentage by multiplying it by 100 and appending a % sign. It also

rounds the result to 2 decimal places. Since 1/3 ≈ 0.3333, it becomes 33.33% when formatted this way.

3. What will be the output of the following Python code?

```
print('ab12'.isalnum())
```

a) True
b) False
c) None
d) Error

Answer: a
Explanation: The .isalnum() method returns True if the string contains only alphanumeric characters (letters and digits) and no spaces or special characters. Since ‘ab12’ contains only letters and digits, the result is True.

4. What will be the output of the following Python code?

```
print('ab,12'.isalnum())
```

a) True
b) False
c) None
d) Error

Answer: b
Explanation: The .isalnum() method returns True only if all characters in the string are letters or digits. In ‘ab,12’, the comma , is neither a letter nor a digit, so .isalnum() returns False.

5. What will be the output of the following Python code?

```
print('ab'.isalpha())
```

a) True
b) False
c) None
d) Error

Answer: a
Explanation: The isalpha() method in Python returns True if all characters in the string are alphabetic (i.e., letters from A–Z or a–z) and there is at least one character. In this case, ‘ab’ contains only letters, so the result is True.

6. What will be the output of the following Python code?

```
print('a B'.isalpha())
```

a) True
b) False
c) None
d) Error

Answer: b
Explanation: The isalpha() method returns True only if all characters in the string are alphabetic and there is at least one character. In the string ‘a B’, there is a space character, which is not alphabetic. Hence, ‘a B’.isalpha() returns False.

7. What will be the output of the following Python code snippet?

```
print('0xa'.isdigit())
```

a) True
b) False
c) None
d) Error

Answer: b
Explanation: The .isdigit() method returns True only if all characters in the string are decimal digits (0–9). In ‘0xa’, the characters ‘x’ and ‘a’ are not digits, so the method returns False. Hexadecimal representations like ‘0xa’ are not treated as numeric digits by .isdigit().

8. What will be the output of the following Python code snippet?

```
print(''.isdigit())
```

a) True
b) False
c) None
d) Error

Answer: b
Explanation: The .isdigit() method returns True only if all characters in the string are digits and there is at least one character. Since the string is empty (”), there are no characters to check, so .isdigit() returns False.

9. What will be the output of the following Python code snippet?

```
print('my_string'.isidentifier())
```

a) True
b) False
c) None
d) Error

Answer: a
Explanation: The .isidentifier() method returns True if the string is a valid Python identifier, meaning it can be used as a variable name. ‘my_string’ follows the rules: it starts with a letter, contains only letters, digits, or underscores, and doesn’t use any reserved keywords. So, the result is True.

10. What will be the output of the following Python code snippet?

```
print('__foo__'.isidentifier())
```
a) True
b) False
c) None
d) Error

Answer: a
Explanation: The .isidentifier() method in Python checks whether a string is a valid identifier. A string like '__foo__' is considered valid because it starts with an underscore and contains only alphanumeric characters or underscores. Special names like this are often used internally in Python but are still valid identifiers.

1. What will be the output of the following Python code snippet?

```
print('for'.isidentifier())
```
a) True
b) False
c) None
d) Error

Answer: a
Explanation: The .isidentifier() method checks whether a string is a valid identifier according to Python's syntax rules — it doesn't check whether the string is a keyword. 'for' follows the syntax rules for identifiers, so .isidentifier() returns True, even though 'for' is a Python keyword and can't actually be used as a variable name.

2. What will be the output of the following Python code snippet?

```
print('abc'.islower())
```
a) True
b) False

c) None
d) Error

Answer: a
Explanation: The .islower() method returns True if all alphabetic characters in the string are lowercase and there is at least one alphabetic character. Since ‘abc’ contains only lowercase letters, the result is True.

3. What will be the output of the following Python code snippet?

```
print('a@ 1,'.islower())
```

a) True
b) False
c) None
d) Error

Answer: a
Explanation: The .islower() method returns True if all alphabetic characters in the string are lowercase, ignoring non-alphabetic characters like @, space, 1, and ,. Since there are no uppercase letters and the alphabetic characters are lowercase, the result is True.

4. What will be the output of the following Python code snippet?

```
print('11'.isnumeric())
```

a) True
b) False
c) None
d) Error

Answer: a
Explanation: The .isnumeric() method returns True if all characters in the

string are numeric characters. Since ’11’ contains only numeric digits, the method returns True.

5. What will be the output of the following Python code snippet?

```
print('1.1'.isnumeric())
```

a) True
b) False
c) None
d) Error

Answer: b
Explanation: The .isnumeric() method returns True only if all characters in the string are numeric characters. Since the string ‘1.1’ contains a dot . which is not numeric, the method returns False.

6. What will be the output of the following Python code snippet?

```
print('1@ a'.isprintable())
```

a) True
b) False
c) None
d) Error

Answer: a
Explanation: The .isprintable() method returns True if all characters in the string are printable, including letters, digits, punctuation, and whitespace. Since ‘1@ a’ contains only printable characters, the result is True.

7. What will be the output of the following Python code snippet?

```
print(''''''.isspace())
```

a) True
b) False
c) None

d) Error

Answer: b
Explanation: The string ”””’ is an empty string (no characters). The .isspace() method returns True only if all characters in the string are whitespace characters. Since the string is empty and contains no characters at all, .isspace() returns False.

8. What will be the output of the following Python code snippet?

```
print('\t'.isspace())
```

a) True
b) False
c) None
d) Error

Answer: a
Explanation: The .isspace() method returns True if all characters in the string are whitespace characters, which include spaces, tabs (\t), newlines, and similar. Since ‘\t’ is a tab character, it counts as whitespace, so the result is True.

9. What will be the output of the following Python code snippet?

```
print('HelloWorld'.istitle())
```

a) True
b) False
c) None
d) Error

Answer: b
Explanation: The .istitle() method checks if each word in the string starts with an uppercase letter followed by lowercase letters. In ‘HelloWorld’,

Python treats it as a single word with multiple uppercase letters, which violates the title case rule. Therefore, the output is False.

10. What will be the output of the following Python code snippet?

```
print('Hello World'.istitle())
```

a) True
b) False
c) None
d) Error

Answer: a
Explanation: The .istitle() method returns True if each word in the string starts with an uppercase letter followed by lowercase letters. In 'Hello World', both "Hello" and "World" follow this rule, so the method returns True.

1. What will be the output of the following Python code?

```
print('Hello!2@#World'.istitle())
```

a) True
b) False
c) None
d) error

Answer: a
Explanation: The .istitle() method checks if each word starts with an uppercase letter followed by lowercase letters. Non-alphabetic characters like !2@# do not affect this check and are treated as separators or ignored. In 'Hello!2@#World', both "Hello" and "World" start with uppercase letters followed by lowercase letters, so .istitle() returns True.

2. What will be the output of the following Python code?

```
print('1Rn@'.lower())
```
a) n
b) 1rn@
c) rn
d) r

Answer: b
Explanation: The .lower() method converts all uppercase letters in the string to their lowercase equivalents. Other characters, such as digits (1) and special characters (@), remain unchanged. So '1Rn@'.lower() becomes '1rn@'.

3. What will be the output of the following Python code?

```
print('''
 \tfoo'''.lstrip())
```
a) \tfoo
b) foo
c) foo
d) none of the mentioned

Answer: b
Explanation: The .lstrip() method removes all leading whitespace characters, including spaces, tabs (\t), and newlines. In the given string, the leading newline and tab before foo are removed, leaving just 'foo'.

4. What will be the output of the following Python code?

```
print('xyyzxxyxyy'.lstrip('xyy'))
```
a) error
b) zxxyxyy
c) z
d) zxxy

Answer: b
Explanation: The .lstrip(‘xyy’) method removes all leading characters that are in the set {‘x’, ‘y’} from the start of the string until it encounters a character not in this set. Here, the string ‘xyyzxxyxyy’ starts with ‘x’, ‘y’, ‘y’, which are removed. The next character is ‘z’, which is not in the set, so stripping stops. The remaining string is ‘zxxyxyy’.

5. What will be the output of the following Python code?

```
print('xyxxyyzxxy'.lstrip('xyy'))
```

a) zxxy
b) xyxxyyzxxy
c) xyxzxxy
d) none of the mentioned

Answer: a
Explanation: The .lstrip(‘xyy’) method removes all leading characters that are in the set ‘x’, ‘y’ (duplicates don’t affect). It keeps stripping characters from the left until it encounters a character not in this set. In the string ‘xyxxyyzxxy’, it strips the leading ‘x’, ‘y’, ‘x’, ‘x’, ‘y’, ‘y’ until it reaches ‘z’, which is not in the set. The remaining string is ‘zxxy’.

6. What will be the output of the following Python code?

```
print('cba'.maketrans('abc', '123'))
```

a) {97: 49, 98: 50, 99: 51}
b) {65: 49, 66: 50, 67: 51}
c) 321
d) 123

Answer: a
Explanation: The maketrans(‘abc’, ‘123’) method returns a translation table (a dictionary) mapping the Unicode code points of characters in ‘abc’ to those in ‘123’.

- ‘a’ → ASCII 97 maps to ‘1’ → ASCII 49
- ‘b’ → ASCII 98 maps to ‘2’ → ASCII 50
- ‘c’ → ASCII 99 maps to ‘3’ → ASCII 51

So the result is {97: 49, 98: 50, 99: 51}.

7. What will be the output of the following Python code?

```
print('a'.maketrans('ABC', '123'))
```

a) {97: 49, 98: 50, 99: 51}
b) {65: 49, 66: 50, 67: 51}
c) {97: 49}
d) 1

Answer: b
Explanation: The maketrans() method is a static method, so calling ‘a’.maketrans(‘ABC’, ‘123’) is the same as str.maketrans(‘ABC’, ‘123’). It creates a translation table mapping Unicode code points of uppercase ‘A’ (65), ‘B’ (66), and ‘C’ (67) to ‘1’ (49), ‘2’ (50), and ‘3’ (51) respectively. The original string ‘a’ does not affect the output.

8. What will be the output of the following Python code?

```
print('abcdef'.partition('cd'))
```

a) (‘ab’, ‘ef’)
b) (‘abef’)
c) (‘ab’, ‘cd’, ‘ef’)
d) 2

Answer: c
Explanation: The .partition() method splits the string into a tuple of three parts:

- The part before the separator (‘cd’)
- The separator itself (‘cd’)
- The part after the separator

For ‘abcdef’.partition(‘cd’), it splits as (‘ab’, ‘cd’, ‘ef’).

9. What will be the output of the following Python code?

```
print('abcdefcdgh'.partition('cd'))
```

a) (‘ab’, ‘cd’, ‘ef’, ‘cd’, ‘gh’)
b) (‘ab’, ‘cd’, ‘efcdgh’)
c) (‘abcdef’, ‘cd’, ‘gh’)
d) error

Answer: b
Explanation: The .partition(‘cd’) method splits the string at the first occurrence of the separator ‘cd’. It returns a tuple of three parts:

- The part before the first ‘cd’ → ‘ab’
- The separator itself → ‘cd’
- The rest of the string after the first ‘cd’ → ‘efcdgh’

So, the output is (‘ab’, ‘cd’, ‘efcdgh’).

10. What will be the output of the following Python code?

```
print('abcd'.partition('cd'))
```

a) (‘ab’, ‘cd’, ”)
b) (‘ab’, ‘cd’)
c) error
d) none of the mentioned

Answer: a
Explanation: The .partition(‘cd’) method splits the string at the first occurrence of ‘cd’. It returns a tuple with three parts:

- The part before ‘cd’ → ‘ab’
- The separator itself → ‘cd’
- The part after ‘cd’ → since ‘cd’ is at the end, this is an empty string ”

Thus, the output is (‘ab’, ‘cd’, ”).

1. What will be the output of the following Python code snippet?

```
print('cd'.partition('cd'))
```

a) (‘cd’)
b) (”)
c) (‘cd’, ”, ”)
d) (”, ‘cd’, ”)

Answer: d
Explanation: When using .partition(‘cd’) on the string ‘cd’, the separator ‘cd’ matches the entire string. The .partition() method returns a tuple:

- The part before the match → ” (nothing before ‘cd’)
- The separator itself → ‘cd’
- The part after the match → ” (nothing after ‘cd’)

So, the result is (”, ‘cd’, ”).

2. What will be the output of the following Python code snippet?

```
print('abef'.partition('cd'))
```

a) (‘abef’)
b) (‘abef’, ‘cd’, ”)
c) (‘abef’, ”, ”)
d) error

Answer: c
Explanation: When the separator ‘cd’ is not found in the string ‘abef’, the .partition() method returns a tuple:

- The original string as the first element → ‘abef’
- An empty string as the second element (no separator found) → ”
- Another empty string as the third element → ”

So, the output is (‘abef’, ”, ”).

3. What will be the output of the following Python code snippet?

```
print('abcdef12'.replace('cd', '12'))
```

a) ab12ef12
b) abcdef12
c) ab12efcd
d) none of the mentioned

Answer: a
Explanation: The .replace(‘cd’, ’12’) method replaces all occurrences of ‘cd’ with ’12’ in the string ‘abcdef12’. Since ‘cd’ appears once, it is replaced, resulting in ‘ab12ef12’. The method processes the string from left to right.

4. What will be the output of the following Python code snippet?

```
print('abef'.replace('cd', '12'))
```

a) abef
b) 12
c) error
d) none of the mentioned

Answer: a
Explanation: The .replace(‘cd’, ’12’) method looks for the substring ‘cd’ in ‘abef’, but since it is not found, the original string is returned unchanged. No error is raised.

5. What will be the output of the following Python code snippet?

```
print('abcefd'.replace('cd', '12'))
```

a) ab1ef2
b) abcefd
c) ab1efd
d) ab12ed2

Answer: b
Explanation: The .replace(‘cd’, ’12’) method attempts to find the substring ‘cd’ in ‘abcefd’. Since ‘cd’ does not exist in the string, no replacement is made, and the original string ‘abcefd’ is returned unchanged.

6. What will be the output of the following Python code snippet?

```
print('xyyxyyxyxyxxy'.replace('xy', '12', 0))
```

a) xyyxyyxyxyxxy
b) 12y12y1212x12
c) 12yxyyxyxyxxy
d) xyyxyyxyxyx12

Answer: a
Explanation: The .replace(old, new, count) method replaces at most count occurrences of old with new. If the count is 0, it means replace zero occurrences, so no replacements are made. In this case, ‘xyyxyyxyxyxxy’.replace(‘xy’, ’12’, 0) results in the original string being returned unchanged: ‘xyyxyyxyxyxxy’.

7. What will be the output of the following Python code snippet?

```
print('xyyxyyxyxyxxy'.replace('xy', '12', 100))
```

a) xyyxyyxyxyxxy
b) 12y12y1212x12
c) none of the mentioned
d) error

Answer: b
Explanation: The .replace(‘xy’, ’12’, 100) method replaces up to 100 occurrences of ‘xy’ with ’12’. Since there are fewer than 100 matches in the string ‘xyyxyyxyxyxxy’, all occurrences of ‘xy’ are replaced.
Here’s how it works step-by-step:

- ‘xyy’ → ’12y’
- ‘xyy’ → ’12y’
- next ‘xy’ → ’12’
- next ‘xy’ → ’12’
- ‘xxy’ → only one ‘xy’ is replaced → ‘x12’

Final result: ’12y12y1212x12’

8. What will be the output of the following Python code snippet?

```
print('abcdefcdghcd'.split('cd'))
```

a) [‘ab’, ‘ef’, ‘gh’]
b) [‘ab’, ‘ef’, ‘gh’, ”]
c) (‘ab’, ‘ef’, ‘gh’)
d) (‘ab’, ‘ef’, ‘gh’, ”)

Answer: b
Explanation: The .split(‘cd’) method divides the string ‘abcdefcdghcd’ at each occurrence of ‘cd’. This results in four parts: ‘ab’, ‘ef’, ‘gh’, and an empty string ” at the end because the string ends with ‘cd’. Therefore, the output is [‘ab’, ‘ef’, ‘gh’, ”].

9. What will be the output of the following Python code snippet?

```
print('abcdefcdghcd'.split('cd', 0))
```

a) [‘abcdefcdghcd’]
b) ‘abcdefcdghcd’
c) error
d) none of the mentioned

Answer: a
Explanation: The .split(‘cd’, 0) method tells Python to split the string at 0 occurrences of the substring ‘cd’. Since the count is zero, no splitting happens, and the entire original string is returned as a single-element list: [‘abcdefcdghcd’].

10. What will be the output of the following Python code snippet?

```
print('abcdefcdghcd'.split('cd', -1))
```

a) [‘ab’, ‘ef’, ‘gh’]
b) [‘ab’, ‘ef’, ‘gh’, ”]
c) (‘ab’, ‘ef’, ‘gh’)
d) (‘ab’, ‘ef’, ‘gh’, ”)

Answer: b
Explanation: When .split(‘cd’, -1) is called with a negative maxsplit, it behaves the same as if maxsplit was not specified at all. This means the string ‘abcdefcdghcd’ is split at every occurrence of ‘cd’, resulting in the list [‘ab’, ‘ef’, ‘gh’, ”]. The last element is an empty string because the string ends with ‘cd’.

1. What will be the output of the following Python code snippet?

```
print('abcdefcdghcd'.split('cd', 2))
```

a) [‘ab’, ‘ef’, ‘ghcd’]
b) [‘ab’, ‘efcdghcd’]
c) [‘abcdef’, ‘ghcd’]
d) none of the mentioned

Answer: a
Explanation: The .split() method in Python is used to divide a string into a list based on a specified separator. When a second argument maxsplit is provided, it limits the number of splits that will be performed. In the code print(‘abcdefcdghcd’.split(‘cd’, 2)), the string is split at the substring ‘cd’ with a maximum of 2 splits. This results in three parts: ‘ab’, ‘ef’, and ‘ghcd’. The method stops after two splits, even though the separator appears again, so the output is [‘ab’, ‘ef’, ‘ghcd’].

2. What will be the output of the following Python code snippet?

```
print('ab\ncd\nef'.splitlines())
```

a) [‘ab’, ‘cd’, ‘ef’]
b) [‘ab\n’, ‘cd\n’, ‘ef\n’]
c) [‘ab\n’, ‘cd\n’, ‘ef’]
d) [‘ab’, ‘cd’, ‘ef\n’]

Answer: a
Explanation: The .splitlines() method in Python is used to split a string at line boundaries, such as newline characters (\n), and returns a list of lines. By default, it removes the line break characters in the resulting list. In the code snippet print(‘ab\ncd\nef’.splitlines()), the string contains two newline characters, which split the string into three parts: ‘ab’, ‘cd’, and ‘ef’. Since the newline characters are not retained, the output is [‘ab’, ‘cd’, ‘ef’].

3. What will be the output of the following Python code snippet?

```
print('Ab!2'.swapcase())
```

a) AB!@
b) ab12
c) aB!2
d) aB1@

Answer: c
Explanation: The .swapcase() method switches uppercase letters to lowercase and lowercase letters to uppercase in the given string, while leaving non-letter characters unchanged. In ‘Ab!2’, the uppercase ‘A’ becomes lowercase ‘a’, the lowercase ‘b’ becomes uppercase ‘B’, and the characters ‘!’ and ‘2’ remain the same. Thus, the output is ‘aB!2’.

4. What will be the output of the following Python code snippet?

```
print('ab cd ef'.title())
```

a) Ab cd ef
b) Ab cd eF
c) Ab Cd Ef
d) None of the mentioned

Answer: c
Explanation: The .title() method capitalizes the first letter of every word in the string. Here, each word—'ab', 'cd', and 'ef'—has its first letter converted to uppercase, resulting in 'Ab Cd Ef'.

5. What will be the output of the following Python code snippet?

```
print('ab cd-ef'.title())
```

a) Ab cd-ef
b) Ab Cd-ef
c) Ab Cd-Ef
d) None of the mentioned

Answer: c
Explanation: The .title() method capitalizes the first letter of each word, where words are separated by whitespace and special characters like -. In the string 'ab cd-ef', the words are 'ab', 'cd', and 'ef'. Because the hyphen – is treated as a separator, 'ef' is also capitalized, resulting in 'Ab Cd-Ef'.

6. What will be the output of the following Python code snippet?

```
print('abcd'.translate('a'.maketrans('abc', 'bcd')))
```

a) bcde
b) abcd
c) error
d) bcdd

Answer: d
Explanation: The maketrans('abc', 'bcd') creates a translation table that

maps ‘a’→’b’, ‘b’→’c’, and ‘c’→’d’. When ‘abcd’.translate(…) is called, each character is replaced according to this table:

- ‘a’ becomes ‘b’
- ‘b’ becomes ‘c’
- ‘c’ becomes ‘d’
- ‘d’ remains ‘d’ (since it’s not in the translation table)

Hence, the final output is ‘bcdd’.

7. What will be the output of the following Python code snippet?

```
print('abcd'.translate({97: 98, 98: 99, 99: 100}))
```

a) bcde
b) abcd
c) error
d) none of the mentioned

Answer: d
Explanation: The translate() method takes a dictionary mapping Unicode ordinal values of characters to their replacements. Here, the dictionary {97: 98, 98: 99, 99: 100} maps:

- 97 (which is ‘a’) → 98 (‘b’)
- 98 (‘b’) → 99 (‘c’)
- 99 (‘c’) → 100 (‘d’)

The character ‘d’ (Unicode 100) is not mapped, so it remains unchanged. Therefore, ‘abcd’.translate(…) outputs ‘bcde’.

8. What will be the output of the following Python code snippet?

```
print('abcd'.translate({'a': '1', 'b': '2', 'c': '3', 'd': '4'}))
```

a) abcd
b) 1234
c) error
d) none of the mentioned

Answer: a
Explanation: The .translate() method requires a translation table with integer Unicode code points as keys. In the given code, the dictionary uses string characters instead of integers, so no translation occurs, and the original string ‘abcd’ is returned. To correctly perform character replacement, str.maketrans() should be used.

9. What will be the output of the following Python code snippet?

```
print('ab'.zfill(5))
```

a) 000ab
b) 00ab0
c) 0ab00
d) ab000

Answer: a
Explanation: The .zfill(5) method pads the string on the left with zeros until its total length is 5. Since ‘ab’ has length 2, three zeros are added to the left, resulting in ‘000ab’. This method is commonly used for formatting numbers with leading zeros.

10. What will be the output of the following Python code snippet?

```
print('+99'.zfill(5))
```

a) 00+99
b) 00099
c) +0099
d) +++99

Answer: c
Explanation: The .zfill(width) method pads the string with zeros on the left until it reaches the specified width. If the string starts with a plus or minus sign, zeros are inserted after the sign but before the rest of the

characters. Here, ‘+99’.zfill(5) adds two zeros after the + to make the total length 5, resulting in ‘+0099’.

1. Which of the following commands will create a list?

a) list1 = list()
b) list1 = []
c) list1 = list([1, 2, 3])
d) all of the mentioned

Answer: d
Explanation: All the given commands create a list in Python but in slightly different ways. The statement list1 = list() initializes an empty list using the list() constructor. Similarly, list1 = [] creates an empty list using list literal syntax, which is more concise. The command list1 = list([1, 2, 3]) creates a new list by copying the elements from the existing iterable [1, 2, 3]. Therefore, all options correctly create a list.

2. What is the output when we execute list(“hello”)?

a) [‘h’, ‘e’, ‘l’, ‘l’, ‘o’]
b) [‘hello’]
c) [‘llo’]
d) [‘olleh’]

Answer: a
Explanation: Using list(“hello”) converts the string into a list of its individual characters, so each letter becomes a separate element in the list. Hence, the output is [‘h’, ‘e’, ‘l’, ‘l’, ‘o’].

3. Suppose listExample is [‘h’,’e’,’l’,’l’,’o’], what is len(listExample)?

a) 5
b) 4
c) None

d) Error

Answer: a
Explanation: The list [‘h’, ‘e’, ‘l’, ‘l’, ‘o’] contains 5 elements, so using len(listExample) returns 5, which is the number of items in the list.

4. Suppose list1 is [2445,133,12454,123], what is max(list1)?
a) 2445
b) 133
c) 12454
d) 123

Answer: c
Explanation: The max() function returns the largest element in the list. Among the elements [2445, 133, 12454, 123], 12454 is the maximum value.
5. Suppose list1 is [3, 5, 25, 1, 3], what is min(list1)?
a) 3
b) 5
c) 25
d) 1

Answer: d
Explanation: The min() function returns the smallest element in the list. For the list [3, 5, 25, 1, 3], the smallest number is 1.
6. Suppose list1 is [1, 5, 9], what is sum(list1)?
a) 1
b) 9
c) 15

d) Error

Answer: c
Explanation: The sum() function adds all elements in the list. For [1, 5, 9], the sum is 1 + 5 + 9 = 15.

7. To shuffle the list(say list1) what function do we use?

a) list1.shuffle()
b) shuffle(list1)
c) random.shuffle(list1)
d) random.shuffleList(list1)

Answer: c
Explanation: The shuffle() function is part of the random module, so you need to import random first and then call random.shuffle(list1) to randomly reorder the elements of the list.

8. Suppose list1 is [4, 2, 2, 4, 5, 2, 1, 0], Which of the following is correct syntax for slicing operation?

a) print(list1[2:])
b) print(list1[:2])
c) print(list1[:-2])
d) all of the mentioned

Answer: d
Explanation: All these slicing operations are valid on lists:

- list1[2:] prints elements from index 2 to the end.
- list1[:2] prints elements from the start up to (but not including) index 2.
- list1[:-2] prints elements from the start up to (but not including) the last two elements.

Slicing works the same way for lists as it does for strings.

9. Suppose list1 is [2, 33, 222, 14, 25], What is list1[-1]?

a) Error
b) None
c) 25
d) 2

Answer: c
Explanation: In Python, negative indexing starts from the end of the list, where -1 refers to the last element. So, list1[-1] returns 25, which is the last item in the list [2, 33, 222, 14, 25].

10. Suppose list1 is [2, 33, 222, 14, 25], What is list1[:-1]?

a) [2, 33, 222, 14]
b) Error
c) 25
d) [25, 14, 222, 33, 2]

Answer: a
Explanation: list1[:-1] slices the list from the beginning up to (but not including) the last element. So it returns all elements except the last one, which gives [2, 33, 222, 14].

1. What will be the output of the following Python code?

```
names = ['Amir', 'Bear', 'Charlton', 'Daman']
print(names[-1][-1])
```

a) A
b) Daman
c) Error
d) n

Answer: d
Explanation: The output of the given Python code is n. This happens

because names[-1] accesses the last element in the list, which is the string ‘Daman’. Then, by adding another [-1], the code accesses the last character of that string, which is ‘n’. So, the final printed result is the character ‘n’.

2. What will be the output of the following Python code?

```
names1 = ['Amir', 'Bear', 'Charlton', 'Daman']
names2 = names1
names3 = names1[:]

names2[0] = 'Alice'
names3[1] = 'Bob'

sum = 0
for ls in (names1, names2, names3):
    if ls[0] == 'Alice':
        sum += 1
    if ls[1] == 'Bob':
        sum += 10

print (sum)
```

a) 11
b) 12
c) 21
d) 22

Answer: b
Explanation: In this code, names2 is just another reference to the same list as names1, so changes to names2 also affect names1. However, names3 is a copy of names1 and can be modified independently. As a

result, the sum counts the changes in both names1 and names2 for ‘Alice’ and in names3 for ‘Bob’, totaling 12.

3. Suppose list1 is [1, 3, 2], What is list1 * 2?

a) [2, 6, 4]
b) [1, 3, 2, 1, 3]
c) [1, 3, 2, 1, 3, 2]
d) [1, 3, 2, 3, 2, 1]

Answer: c
Explanation: When you multiply a list by an integer in Python, it repeats the list elements that many times. So, for the list [1, 3, 2], multiplying it by 2 will concatenate the list with itself, resulting in [1, 3, 2, 1, 3, 2].

4. Suppose list1 = [0.5 * x for x in range(0, 4)], list1 is:

a) [0, 1, 2, 3]
b) [0, 1, 2, 3, 4]
c) [0.0, 0.5, 1.0, 1.5]
d) [0.0, 0.5, 1.0, 1.5, 2.0]

Answer: c
Explanation: The list comprehension multiplies each integer x in range(0, 4) by 0.5, producing [0*0.5, 1*0.5, 2*0.5, 3*0.5] which equals [0.0, 0.5, 1.0, 1.5].

5. What will be the output of the following Python code?

```
list1 = [11, 2, 23]
list2 = [11, 2, 2]
print(list1 < list2)
```

a) True
b) False
c) Error

d) None

Answer: b
Explanation: In Python, when comparing two lists using the < operator, elements are compared lexicographically, just like strings. The comparison proceeds element by element until a difference is found. In the given example, [11, 2, 23] < [11, 2, 2] evaluates to False because 23 is greater than 2.

6. To add a new element to a list we use which command?

a) list1.add(5)
b) list1.append(5)
c) list1.addLast(5)
d) list1.addEnd(5)

Answer: b
Explanation: To add a new element to a list in Python, we use the append() method. This method adds the specified element to the end of the list. So, list1.append(5) correctly adds 5 to list1.

7. To insert 5 to the third position in list1, we use which command?

a) list1.insert(3, 5)
b) list1.insert(2, 5)
c) list1.add(3, 5)
d) list1.append(3, 5)

Answer: b
Explanation: To insert an element at a specific position in a Python list, we use the insert() method, which takes two arguments: the index and the element. Python uses 0-based indexing, so the third position corresponds to index 2. Hence, the correct command is: list1.insert(2, 5).

8. To remove string “hello” from list1, we use which command?
a) list1.remove(“hello”)
b) list1.remove(hello)
c) list1.removeAll(“hello”)
d) list1.removeOne(“hello”)

Answer: a
Explanation: The remove() method in Python is used to remove the first occurrence of a specified value from a list. In this case, “hello” is a string, so you need to pass it as a string literal (with quotes). If “hello” exists in list1, list1.remove(“hello”) will remove it.
9. Suppose list1 is [3, 4, 5, 20, 5], what is list1.index(5)?
a) 0
b) 1
c) 4
d) 2

Answer: d
Explanation: The index() method returns the index of the first occurrence of the specified value.
In the list list1 = [3, 4, 5, 20, 5], the value 5 first appears at index 2 (0-based indexing), so list1.index(5) returns 2.
10. Suppose list1 is [3, 4, 5, 20, 5, 25, 1, 3], what is list1.count(5)?
a) 0
b) 4
c) 1
d) 2

Answer: d
Explanation: The count() method returns the number of times a specified

value appears in the list. In list1 = [3, 4, 5, 20, 5, 25, 1, 3], the value 5 appears twice, so list1.count(5) returns 2.

1. Suppose list1 is [3, 4, 5, 20, 5, 25, 1, 3], what is list1 after list1.reverse()?

a) [3, 4, 5, 20, 5, 25, 1, 3]
b) [1, 3, 3, 4, 5, 5, 20, 25]
c) [25, 20, 5, 5, 4, 3, 3, 1]
d) [3, 1, 25, 5, 20, 5, 4, 3]

Answer: d
Explanation: The reverse() method in Python modifies the original list by reversing the order of its elements. For example, if list1 = [3, 4, 5, 20, 5, 25, 1, 3], then after list1.reverse(), the list becomes [3, 1, 25, 5, 20, 5, 4, 3]. This operation happens in-place, meaning no new list is created.

2. Suppose listExample is [3, 4, 5, 20, 5, 25, 1, 3], what is list1 after listExample.extend([34, 5])?

a) [3, 4, 5, 20, 5, 25, 1, 3, 34, 5]
b) [1, 3, 3, 4, 5, 5, 20, 25, 34, 5]
c) [25, 20, 5, 5, 4, 3, 3, 1, 34, 5]
d) [1, 3, 4, 5, 20, 5, 25, 3, 34, 5]

Answer: a
Explanation: The extend() method in Python adds each element from the iterable (in this case, [34, 5]) to the end of the list. Given listExample = [3, 4, 5, 20, 5, 25, 1, 3], after calling listExample.extend([34, 5]), the updated list becomes [3, 4, 5, 20, 5, 25, 1, 3, 34, 5].

3. Suppose listExample is [3, 4, 5, 20, 5, 25, 1, 3], what is list1 after listExample.pop(1)?

a) [3, 4, 5, 20, 5, 25, 1, 3]
b) [1, 3, 3, 4, 5, 5, 20, 25]

c) [3, 5, 20, 5, 25, 1, 3]
d) [1, 3, 4, 5, 20, 5, 25]

Answer: c
Explanation: The pop() method removes and returns the element at the specified index. In the list listExample = [3, 4, 5, 20, 5, 25, 1, 3], calling listExample.pop(1) removes the element at index 1, which is 4. The updated list becomes [3, 5, 20, 5, 25, 1, 3].

4. Suppose listExample is [3, 4, 5, 20, 5, 25, 1, 3], what is list1 after listExample.pop()?
a) [3, 4, 5, 20, 5, 25, 1]
b) [1, 3, 3, 4, 5, 5, 20, 25]
c) [3, 5, 20, 5, 25, 1, 3]
d) [1, 3, 4, 5, 20, 5, 25]

Answer: a
Explanation: The pop() method in Python removes and returns the last item from a list when no index is specified. In the given list [3, 4, 5, 20, 5, 25, 1, 3], calling pop() removes the last element 3, resulting in [3, 4, 5, 20, 5, 25, 1].
5. What will be the output of the following Python code?

```
print("Welcome to Python".split())
```

a) [‘Welcome’, ‘to’, ‘Python’]
b) (“Welcome”, “to”, “Python”)
c) {“Welcome”, “to”, “Python”}
d) “Welcome”, “to”, “Python”

Answer: a
Explanation: The split() method in Python divides a string into a list using

whitespace as the default delimiter. For example, “Welcome to Python”.split() returns [‘Welcome’, ‘to’, ‘Python’]. This is useful for breaking up sentences into words.

6. What will be the output of the following Python code?

```
print(list("a#b#c#d".split('#')))
```

a) [‘a’, ‘b’, ‘c’, ‘d’]
b) [‘a b c d’]
c) [‘a#b#c#d’]
d) [‘abcd’]

Answer: a
Explanation: The split(‘#’) method splits the string “a#b#c#d” at every # and returns a list of substrings: [‘a’, ‘b’, ‘c’, ‘d’]. Wrapping it with list() doesn’t change the structure since split() already returns a list. Therefore, the final output is [‘a’, ‘b’, ‘c’, ‘d’].

7. What will be the output of the following Python code?

```
myList = [1, 5, 5, 5, 5, 1]
max = myList[0]
indexOfMax = 0
for i in range(1, len(myList)):
    if myList[i] > max:
        max = myList[i]
        indexOfMax = i

print(indexOfMax)
```

a) 1
b) 2
c) 3
d) 4

Answer: a
Explanation: The code finds the index of the first occurrence of the maximum value in the list. Since the maximum value 5 appears first at index 1, the variable indexOfMax is set to 1. Therefore, the output of the code is 1.

8. What will be the output of the following Python code?

```
myList = [1, 2, 3, 4, 5, 6]
for i in range(1, 6):
    myList[i - 1] = myList[i]

for i in range(0, 6):
    print(myList[i], end = " ")
```

a) 2 3 4 5 6 1
b) 6 1 2 3 4 5
c) 2 3 4 5 6 6
d) 1 1 2 3 4 5

Answer: c
Explanation: The code shifts all elements of a list by one position towards the beginning, except the first element which is replaced by the second. The last element is duplicated since there is no next element to assign its value to. The output is “2 3 4 5 6 6”.

9. What will be the output of the following Python code?

```
list1 = [1, 3]
list2 = list1
list1[0] = 4
print(list2)
```

a) [1, 3]
b) [4, 3]
c) [1, 4]

d) [1, 3, 4]

Answer: b
Explanation: In the code, list2 = list1 creates a reference to the same list in memory. So when list1[0] is changed to 4, list2 also reflects that change. The output is [4, 3].

10. What will be the output of the following Python code?

```
def f(values):
   values[0] = 44

v = [1, 2, 3]
f(v)
print(v)
```

a) [1, 44]
b) [1, 2, 3, 44]
c) [44, 2, 3]
d) [1, 2, 3]

Answer: c
Explanation: In Python, lists are mutable and passed by reference to functions. The function f(values) modifies the first element of the list to 44. So, the original list v becomes [44, 2, 3] after the function call.

1. What will be the output of the following Python code?

```
def f(i, values = []):
   values.append(i)
   return values

f(1)
f(2)
v = f(3)
```

```
print(v)
```
a) [1] [2] [3]
b) [1] [1, 2] [1, 2, 3]
c) [1, 2, 3]
d) 1 2 3

Answer: c
Explanation: In Python, default arguments are evaluated only once when the function is defined. Since the default values is a mutable list, it retains changes across function calls. Thus, each call to f() appends to the same list, resulting in [1, 2, 3].

2. What will be the output of the following Python code?

```
names1 = ['Amir', 'Bala', 'Chales']

if 'amir' in names1:
   print(1)
else:
   print(2)
```
a) None
b) 1
c) 2
d) Error

Answer: c
Explanation: In the given code, the string ‘amir’ is checked for membership in the list [‘Amir’, ‘Bala’, ‘Chales’]. Since Python is case-sensitive, ‘amir’ (lowercase ‘a’) does not match ‘Amir’ (uppercase ‘A’), so the condition fails. Hence, the output is 2.

3. What will be the output of the following Python code?

```
names1 = ['Amir', 'Bala', 'Charlie']
names2 = [name.lower() for name in names1]

print(names2[2][0])
```

a) None
b) a
c) b
d) c

Answer: d
Explanation: The code uses list comprehension to convert each name in names1 to lowercase, resulting in [‘amir’, ‘bala’, ‘charlie’]. names2[2] refers to ‘charlie’, and names2[2][0] gives the first character ‘c’. Therefore, the output is c.

4. What will be the output of the following Python code?

```
numbers = [1, 2, 3, 4]

numbers.append([5,6,7,8])

print(len(numbers))
```

a) 4
b) 5
c) 8
d) 12

Answer: b
Explanation: The append() method adds its argument as a single element at the end of the list. Here, [5, 6, 7, 8] is appended as one sublist, not individual elements. So the list becomes [1, 2, 3, 4, [5, 6, 7, 8]], which has 5 elements.

5. To which of the following the “in” operator can be used to check if an item is in it?

a) Lists
b) Dictionary
c) Set
d) All of the mentioned

Answer: d
Explanation: The in operator is used to check for membership and works with lists, dictionaries, and sets. In lists and sets, it checks for presence of an element; in dictionaries, it checks for presence of a key. Therefore, it applies to all the mentioned data structures.

6. What will be the output of the following Python code?

```
list1 = [1, 2, 3, 4]
list2 = [5, 6, 7, 8]

print(len(list1 + list2))
```

a) 2
b) 4
c) 5
d) 8

Answer: d
Explanation: The + operator concatenates two lists by combining all their elements into a new list. So list1 + list2 results in [1, 2, 3, 4, 5, 6, 7, 8], which has 8 elements. Therefore, len(list1 + list2) returns 8.

7. What will be the output of the following Python code?

```
def addItem(listParam):
   listParam += [1]
```

```
mylist = [1, 2, 3, 4]
addItem(mylist)
print(len(mylist))
```

a) 1
b) 4
c) 5
d) 8

Answer: c
Explanation: The function addItem uses += [1] to modify the list passed to it. Since lists are mutable and passed by reference, mylist is modified directly. After appending 1, it becomes [1, 2, 3, 4, 1], so its length is 5.

8. What will be the output of the following Python code?

```
def increment_items(L, increment):
    i = 0
    while i < len(L):
        L[i] = L[i] + increment
        i = i + 1

values = [1, 2, 3]
print(increment_items(values, 2))
print(values)
```

a)
None
[3, 4, 5]
b)
None
[1, 2, 3]
c)
[3, 4, 5]

[1, 2, 3]

d)

[3, 4, 5]

None

Answer: a

Explanation: The function increment_items modifies the list in-place by adding the increment value to each element. However, since it doesn't return anything, the first print() outputs None. The second print() shows the updated list [3, 4, 5].

9. What will be the output of the following Python code?

```
def example(L):
    ''' (list) -> list
    '''
    i = 0
    result = []
    while i < len(L):
        result.append(L[i])
        i = i + 3
    return result
```

a) Return a list containing every third item from L starting at index 0

b) Return an empty list

c) Return a list containing every third index from L starting at index 0

d) Return a list containing the items from L starting from index 0, omitting every third item

Answer: a

Explanation: The function iterates over the list L, starting at index 0, and

appends every third element (i = i + 3) to result. This means it collects items at indices 0, 3, 6, and so on, effectively returning every third item from the list starting at the first element.

10. What will be the output of the following Python code?

```
veggies = ['carrot', 'broccoli', 'potato', 'asparagus']
veggies.insert(veggies.index('broccoli'), 'celery')
print(veggies)
```

a) [‘carrot’, ‘celery’, ‘broccoli’, ‘potato’, ‘asparagus’]
b) [‘carrot’, ‘celery’, ‘potato’, ‘asparagus’]
c) [‘carrot’, ‘broccoli’, ‘celery’, ‘potato’, ‘asparagus’]
d) [‘celery’, ‘carrot’, ‘broccoli’, ‘potato’, ‘asparagus’]

Answer: a
Explanation: The insert() method inserts ‘celery’ at the index where ‘broccoli’ is found, which is position 1. This pushes ‘broccoli’ and the following elements one position to the right, resulting in the list: [‘carrot’, ‘celery’, ‘broccoli’, ‘potato’, ‘asparagus’].

1. What will be the output of the following Python code?

```
m = [[x, x + 1, x + 2] for x in range(0, 3)]
print(m)
```

a) [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
b) [[0, 1, 2], [1, 2, 3], [2, 3, 4]]
c) [1, 2, 3, 4, 5, 6, 7, 8, 9]
d) [0, 1, 2, 1, 2, 3, 2, 3, 4]

Answer: b
Explanation: The list comprehension creates a list of lists, where for each x in range(0, 3), it creates [x, x+1, x+2]. This results in [[0, 1, 2], [1, 2, 3], [2, 3, 4]].

2. How many elements are in m?

```
m = [[x, y] for x in range(0, 4) for y in range(0, 4)]
print(m)
```

a) 8
b) 12
c) 16
d) 32

Answer: c
Explanation: The nested list comprehension creates a pair [x, y] for every combination of x and y in range(0, 4), which has 4 values each. So total elements = 4 × 4 = 16.

3. What will be the output of the following Python code?

```
values = [[3, 4, 5, 1], [33, 6, 1, 2]]

v = values[0][0]
for row in range(0, len(values)):
    for column in range(0, len(values[row])):
        if v < values[row][column]:
            v = values[row][column]

print(v)
```

a) 3
b) 5
c) 6
d) 33

Answer: d
Explanation: The code iterates through all elements in the 2D list values,

updating v whenever it finds a larger number. The largest value in the nested lists is 33, so the output is 33.

4. What will be the output of the following Python code?

```
values = [[3, 4, 5, 1], [33, 6, 1, 2]]

v = values[0][0]
for lst in values:
    for element in lst:
        if v > element:
            v = element

print(v)
```

a) 1
b) 3
c) 5
d) 6

Answer: a
Explanation: The code goes through all elements in the nested list values and updates v whenever it finds a smaller element (if v > element). Starting with v = 3, it finds smaller values 1 and updates v accordingly. The smallest element in the lists is 1, so the output is 1.

5. What will be the output of the following Python code?

```
values = [[3, 4, 5, 1 ], [33, 6, 1, 2]]

for row in values:
    row.sort()
    for element in row:
        print(element, end = " ")
    print()
```

a)
3 4 5 1
1 2 6 33
b)
3 4 5 1
33 6 1 2
c)
1 3 4 5
1 2 6 33
d) 1 3 4 5

Answer: c
Explanation: The code sorts each sublist inside the main list, so the first list becomes [1, 3, 4, 5] and the second becomes [1, 2, 6, 33]. It then prints each sorted list on a separate line, resulting in two rows of sorted numbers.

6. What will be the output of the following Python code?

```
matrix = [[1, 2, 3, 4],
      [4, 5, 6, 7],
      [8, 9, 10, 11],
      [12, 13, 14, 15]]

for i in range(0, 4):
   print(matrix[i][1], end = " ")
```

a) 1 2 3 4
b) 4 5 6 7
c) 1 3 8 12
d) 2 5 9 13

Answer: d
Explanation: The code prints the element at index 1 (the second element) from each of the 4 rows in matrix. These elements are 2, 5, 9, and 13, which are printed in order separated by spaces.

7. What will be the output of the following Python code?

```
def m(list):
   v = list[0]
   for e in list:
     if v < e: v = e
   return v

values = [[3, 4, 5, 1], [33, 6, 1, 2]]

for row in values:
   print(m(row), end = " ")
```

a) 3 33
b) 1 1
c) 5 6
d) 5 33

Answer: d
Explanation: The function m returns the maximum value in a given list by comparing each element. For the two sublists, it finds 5 as the max in [3, 4, 5, 1] and 33 as the max in [33, 6, 1, 2]. These are printed with a space separating them.

8. What will be the output of the following Python code?

```
data = [[[1, 2], [3, 4]], [[5, 6], [7, 8]]]

print(data[1][0][0])
```

a) 1
b) 2
c) 4
d) 5

Answer: d
Explanation: data[1] accesses the second element of the outer list: [[5, 6], [7, 8]]. Then [0] accesses the first sublist [5, 6], and [0] again accesses the first element of that sublist, which is 5.

9. What will be the output of the following Python code?

```
data = [[[1, 2], [3, 4]], [[5, 6], [7, 8]]]

def ttt(m):
    v = m[0][0]

    for row in m:
        for element in row:
            if v < element: v = element

    return v

print(ttt(data[0]))
```

a) 1
b) 2
c) 4
d) 5

Answer: c
Explanation: The function ttt finds the maximum element in the 2D list

data[0], which is [[1, 2], [3, 4]]. The largest number here is 4, so the output is 4.

10. What will be the output of the following Python code?

```
points = [[1, 2], [3, 1.5], [0.5, 0.5]]
points.sort()
print(points)
```

a) [[1, 2], [3, 1.5], [0.5, 0.5]]
b) [[3, 1.5], [1, 2], [0.5, 0.5]]
c) [[0.5, 0.5], [1, 2], [3, 1.5]]
d) [[0.5, 0.5], [3, 1.5], [1, 2]]

Answer: c

Explanation: The sort() method sorts lists of lists by comparing their elements lexicographically, starting from the first element of each sublist. Here, the points are sorted by their first coordinate, resulting in the order [[0.5, 0.5], [1, 2], [3, 1.5]].

1. What will be the output of the following Python code?

```
a=[10,23,56,[78]]
b=list(a)
a[3][0]=95
a[1]=34
print(b)
```

a) [10, 34, 56, [95]]
b) [10, 23, 56, [78]]
c) [10, 23, 56, [95]]
d) [10, 34, 56, [78]]

Answer: c

Explanation: The code creates a shallow copy of list a into b, meaning the outer list is copied, but inner objects like the sublist are shared between

both. When a[3][0] is modified, the change reflects in b as well, but changing a[1] does not affect b[1] since integers are immutable and copied by value. Hence, the output is [10, 23, 56, [95]].

2. What will be the output of the following Python code?

```
print(list(zip((1,2,3),('a'),('xxx','yyy'))))
print(list(zip((2,4),('b','c'),('yy','xx'))))
```

a)
[(1,2,3),(‘a’),(‘xxx’,’yyy’)]
[(2,4),(‘b’,’c’),(‘yy’,’xx’)]
b)
[(1, 'a', 'xxx'),(2,’ ‘,’yyy’),(3,’ ‘,’ ‘)]
[(2, 'b', 'yy'), (4, 'c', 'xx')]
c) Syntax error
d)
[(1, 'a', 'xxx')]
[(2, 'b', 'yy'), (4, 'c', 'xx')]

Answer: d
Explanation: The zip() function pairs elements from multiple iterables and stops when the shortest iterable is exhausted. In the first case, ‘a’ is treated as a one-character string, so only one tuple is formed. The second zip() processes all elements since all iterables are of equal length, resulting in two tuples.

3. What will be the output of the following Python code?

```
import copy
a=[10,23,56,[78]]
b=copy.deepcopy(a)
```

```
a[3][0]=95
a[1]=34
print(b)
```

a) [10, 34, 56, [95]]
b) [10, 23, 56, [78]]
c) [10, 23, 56, [95]]
d) [10, 34, 56, [78]]

Answer: b
Explanation: Using copy.deepcopy(a) creates a deep copy, meaning all nested objects (like the sublist [78]) are independently copied. So, any changes made to a or its sublists (like changing a[3][0] to 95 or a[1] to 34) do not affect b. Hence, b remains unchanged as [10, 23, 56, [78]].

4. What will be the output of the following Python code?

```
s="a@b@c@d"
a=list(s.partition("@"))
print(a)
b=list(s.split("@",3))
print(b)
```

a)
[‘a’, ’b’, ’c’, ’d’]
[‘a’, ’b’, ’c’, ’d’]
b)
[‘a’,’@’,’b’,’@’,’c’,’@’,’d’]
[‘a’,’b’,’c’,’d’]
c)
['a', '@', 'b@c@d']
['a', 'b', 'c', 'd']
d)
[‘a’,’@’,’b@c@d’]

[‘a’,’@’,’b’,’@’,’c’,’@’,’d’]

Answer: c
Explanation: The partition(“@”) method splits the string only at the first occurrence of “@” and includes the separator in the result, returning [‘a’, ‘@’, ‘b@c@d’]. The split(“@”, 3) method splits the string into at most 4 parts using “@” as the delimiter, resulting in [‘a’, ‘b’, ‘c’, ‘d’].

5. What will be the output of the following Python code?

```
a=[1,2,3,4]
b=[sum(a[0:x+1]) for x in range(0,len(a))]
print(b)
```

a) 10
b) [1, 3, 5, 7]
c) 4
d) [1, 3, 6, 10]

Answer: d
Explanation: The list comprehension calculates the cumulative sum of elements in list a. For each index x, it sums a[0] to a[x]. So it produces [1, (1+2), (1+2+3), (1+2+3+4)] which results in [1, 3, 6, 10].

6. What will be the output of the following Python code?

```
a="hello"
b=list((x.upper(),len(x)) for x in a)
print(b)
```

a) [(‘H’, 1), (‘E’, 1), (‘L’, 1), (‘L’, 1), (‘O’, 1)]
b) [(‘HELLO’, 5)]
c) [(‘H’, 5), (‘E’, 5), (‘L’, 5), (‘L’, 5), (‘O’, 5)]

d) Syntax error

Answer: a
Explanation: The string “hello” is iterable, so the generator expression (x.upper(), len(x)) for x in a processes each character. x.upper() converts each character to uppercase, and len(x) is 1 because x is a single character. The result is a list of tuples: [(‘H’, 1), (‘E’, 1), (‘L’, 1), (‘L’, 1), (‘O’, 1)].

7. What will be the output of the following Python code?

```
a = [2, 4, 6, 8]
b = [sum(a[0:x+1]) for x in range(0, len(a))]
print(b)
```

a) 10
b) [2, 3, 6, 7]
c) 4
d) [2, 6, 12, 20]

Answer: d
Explanation: The code computes the cumulative sum of elements in the list a = [2, 4, 6, 8] using list comprehension. For each index x, it calculates the sum of elements from the start of the list up to index x, resulting in [2, 6, 12, 20].

8. What will be the output of the following Python code?

```
a=[[]]*3
a[1].append(7)
print(a)
```

a) Syntax error
b) [[7], [7], [7]]
c) [[7], [], []]

d) [[],7, [], []]

Answer: b
Explanation: The expression [[]]*3 creates a list with three references to the same empty list. When you append 7 to one of them (e.g., a[1]), it reflects in all because they all point to the same object. Hence, the output is [[7], [7], [7]].

9. What will be the output of the following Python code?

```
b=[2,3,4,5]
a=list(filter(lambda x:x%2,b))
print(a)
```

a) [2, 4]
b) [ ]
c) [3, 5]
d) Invalid arguments for filter function

Answer: c
Explanation: The filter() function applies the lambda function lambda x: x%2 to each element of list b. In Python, x%2 returns 1 for odd numbers and 0 for even ones. Since filter only keeps values where the result is truthy (non-zero), only the odd numbers [3, 5] are included in the output.

10. What will be the output of the following Python code?

```
lst=[3,4,6,1,2]
lst[1:2]=[7,8]
print(lst)
```

a) [3, 7, 8, 6, 1, 2]
b) Syntax error
c) [3,[7,8],6,1,2]
d) [3, 4, 6, 7, 8]

Answer: a
Explanation: The slice assignment lst[1:2] = [7, 8] replaces the elements from index 1 up to (but not including) index 2 with the new list [7, 8]. This effectively inserts 7 and 8 in place of the single element 4, resulting in [3, 7, 8, 6, 1, 2].

1. What will be the output of the following Python code?

```
a=[1,2,3]
b=a.append(4)
print(a)
print(b)
```

a)
[1, 2, 3, 4]
[1, 2, 3, 4]

b)
[1, 2, 3, 4]
None
c) Syntax error
d)
[1, 2, 3]
[1, 2, 3, 4]

Answer: b
Explanation: The append() method modifies the list in place and does not return a new list, so b = a.append(4) assigns None to b. However, a is updated to [1, 2, 3, 4], which is why print(a) shows the updated list while print(b) outputs None.

2. What will be the output of the following Python code?

```
a=[14,52,7]
b=a.copy()
print(b is a)
```

a) True
b) False

Answer: b
Explanation: The copy() method creates a shallow copy of the list, meaning b is a new list object with the same elements as a. Since b and a are different objects, b is a evaluates to False. Changes to one won't affect the other's identity.

3. What will be the output of the following Python code?

```
a=[13,56,17]
a.append([87])
a.extend([45,67])
print(a)
```

a) [13, 56, 17, [87], 45, 67]
b) [13, 56, 17, 87, 45, 67]
c) [13, 56, 17, 87,[45, 67]]
d) [13, 56, 17, [87], [45, 67]]

Answer: a
Explanation: The append([87]) adds the entire list [87] as a single element at the end of a, while extend([45, 67]) adds each element individually to the list. Hence, the final list is [13, 56, 17, [87], 45, 67].

4. What is the output of the following piece of code?

```
a=list((45,)*4)
print((45)*4)
print(a)
```

a)

180
[(45), (45), (45),(45)]
b)
(45, 45, 45, 45)
[45, 45, 45, 45]
c)
180
[45, 45, 45, 45]
d) Syntax error

Answer: c
Explanation: The expression (45)*4 multiplies the integer 45 by 4, resulting in 180. The expression list((45,)*4) creates a tuple (45,) repeated 4 times, then converts it into a list [45, 45, 45, 45]. Note the comma makes (45,) a tuple, while (45) is just an integer in parentheses.

5. What will be the output of the following Python code?

```
lst=[[1,2],[3,4]]
print(sum(lst,[]))
```

a) [[3], [7]]
b) [1, 2, 3, 4]
c) Error
d) [10]

Answer: b
Explanation: Using sum(lst, []) adds up the sublists starting from an empty list, effectively concatenating them into a single flat list [1, 2, 3, 4]. This is a common Python trick to flatten a list of lists by one level.

6. What will be the output of the following Python code?

```
word1="Apple"
word2="Apple"
```

```
list1=[1,2,3]
list2=[1,2,3]
print(word1 is word2)
print(list1 is list2)
```

a)
True
True
b)
False
True
c)
False
False
d)
True
False

Answer: d
Explanation: String literals like “Apple” are interned by Python, so word1 is word2 returns True because both refer to the same object. However, list1 and list2 are two separate list objects with the same content, so list1 is list2 returns False as they are not the same object in memory.

7. What will be the output of the following Python code?

```
def unpack(a,b,c,d):
    print(a+d)
x = [1,2,3,4]
unpack(*x)
```

a) Error
b) [1, 4]
c) [5]
d) 5

Answer: d
Explanation: The *x syntax unpacks the list [1, 2, 3, 4] into individual arguments a=1, b=2, c=3, d=4. Then, the function prints a + d, which is 1 + 4 = 5.

8. What will be the output of the following Python code?

```
places = ['Bangalore', 'Mumbai', 'Delhi']
places1 = places
places2 = places[:]
places1[1]="Pune"
places2[2]="Hyderabad"
print(places)
```

a) [‘Bangalore’, ‘Pune’, ‘Hyderabad’]
b) [‘Bangalore’, ‘Pune’, ‘Delhi’]
c) [‘Bangalore’, ‘Mumbai’, ‘Delhi’]
d) [‘Bangalore’, ‘Mumbai’, ‘Hyderabad’]

Answer: b
Explanation: Since places1 is just a reference to the original places list, changing places1[1] also changes places[1] to ‘Pune’. On the other hand, places2 is a copy of places, so modifying places2[2] to ‘Hyderabad’ does not affect the original places list.

9. What will be the output of the following Python code?

```
x=[[1],[2]]
print(" ".join(list(map(str,x))))
```

a) [1] [2]
b) [49] [50]
c) Syntax error
d) [[1]] [[2]]

Answer: a
Explanation: The code converts each sublist in x to a string, resulting in “[1]” and “[2]”. Then, it joins these string representations with a space between them. So, the output prints the two sublists as [1] [2].

10. What will be the output of the following Python code?

```
a=165
b=sum(list(map(int,str(a))))
print(b)
```

a) 561
b) 5
c) 12
d) Syntax error

Answer: c
Explanation: The code converts the integer a into a string, then maps each character back to an integer, creating a list of digits [1, 6, 5]. The sum function adds these digits together, resulting in 12. Hence, the output is 12.

11. What will be the output of the following Python code?

```
a= [1, 2, 3, 4, 5]
for i in range(1, 5):
    a[i-1] = a[i]
for i in range(0, 5):
    print(a[i],end = " ")
```

a) 5 5 1 2 3
b) 5 1 2 3 4
c) 2 3 4 5 1
d) 2 3 4 5 5

Answer: d
Explanation: The first loop shifts each element from index 1 to 4 to the previous index (i.e., a[0] = a[1], a[1] = a[2], and so on). This changes the list step by step to [2, 3, 4, 5, 5]. The second loop prints all elements in the updated list, resulting in 2 3 4 5 5.
12. What will be the output of the following Python code?

```
def change(var, lst):
    var = 1
    lst[0] = 44
k = 3
a = [1, 2, 3]
change(k, a)
print(k)
print(a)
```

a)
3
[44, 2, 3]
b)
1
[1, 2, 3]
c)
3
[1, 2, 3]
d)
1

[44, 2, 3]

Answer: a
Explanation: In Python, integers are immutable, so assigning a new value to var inside the function doesn't affect the original variable k. However, lists are mutable, so modifying lst[0] directly changes the original list a, resulting in [44, 2, 3].

13. What will be the output of the following Python code?

```
a = [1, 5, 7, 9, 9, 1]
b=a[0]
x= 0
for x in range(1, len(a)):
    if a[x] > b:
        b = a[x]
        b= x
print(b)
```

a) 5
b) 3
c) 4
d) 0

Answer: c
Explanation: The code compares each element in the list and updates b with the index whenever a new maximum is found. Since b is overwritten with the index during each update, the final value represents the index of the last occurrence of the maximum element, which is at index 4.

14. What will be the output of the following Python code?

```
a=["Apple","Ball","Cobra"]
```

```
a.sort(key=len)
print(a)
```
a) [‘Apple’, ‘Ball’, ‘Cobra’]
b) [‘Ball’, ‘Apple’, ‘Cobra’]
c) [‘Cobra’, ‘Apple’, ‘Ball’]
d) Invalid syntax for sort()

Answer: b
Explanation: The sort() function with key=len sorts the list based on the length of each string. Since “Ball” has 4 letters and both “Apple” and “Cobra” have 5, “Ball” comes first, followed by “Apple” and “Cobra” in their original order.

15. What will be the output of the following Python code?

```
num = ['One', 'Two', 'Three']
for i, x in enumerate(num):
   print('{}: {}'.format(i, x),end=" ")
```
a) 1: 2: 3:
b) Exception is thrown
c) One Two Three
d) 0: One 1: Two 2: Three

Answer: d
Explanation: The enumerate() function returns both the index and value from the list, starting with index 0 by default. Each pair is printed in the format index: value on the same line with spaces in between, resulting in 0: One 1: Two 2: Three.

1. What will be the output of the following Python code snippet?

```
my_string = "hello world"
k = [print(i) for i in my_string if i not in "aeiou"]
```

a) prints all the vowels in my_string
b) prints all the consonants in my_string
c) prints all characters of my_string that aren't vowels
d) prints only on executing print(k)

Answer: c
Explanation: The list comprehension filters out vowels and prints only those characters in my_string that are not vowels. Since print(i) is inside the comprehension, it gets executed immediately for each non-vowel character.

2. What is the output of print(k) in the following Python code snippet?

```
k = [print(i) for i in my_string if i not in "aeiou"]
print(k)
```

a) all characters of my_string that aren't vowels
b) a list of Nones
c) list of Trues
d) list of Falses

Answer: b
Explanation: The print(i) inside the list comprehension prints each non-vowel character immediately but returns None. Therefore, k becomes a list of None values, and printing k displays that list of Nones.

3. What will be the output of the following Python code snippet?

```
my_string = "hello world"
k = [(i.upper(), len(i)) for i in my_string]
print(k)
```

a) [('HELLO', 5), ('WORLD', 5)]
b) [('H', 1), ('E', 1), ('L', 1), ('L', 1), ('O', 1), (' ', 1), ('W', 1), ('O', 1), ('R', 1), ('L', 1), ('D', 1)]

c) [(‘HELLO WORLD’, 11)]
d) none of the mentioned

Answer: b
Explanation: The code iterates over each character in the string “hello world” and creates a list of tuples where each tuple contains the uppercase version of the character and its length (which is always 1). Since iteration is character-wise, the result includes individual letters and the space character. Thus, the output is a list of tuples like (‘H’, 1), (‘E’, 1), …, (‘D’, 1).

4. Which of the following is the correct expansion of list_1 = [expr(i) for i in list_0 if func(i)]?

a)

```
list_1 = []
for i in list_0:
    if func(i):
        list_1.append(i)
```

b)

```
for i in list_0:
    if func(i):
        list_1.append(expr(i))
```

c)

```
list_1 = []
for i in list_0:
    if func(i):
        list_1.append(expr(i))
```

d) none of the mentioned

Answer: c
Explanation: The list comprehension builds a new list by evaluating

expr(i) only for those elements of list_0 that satisfy func(i). This is correctly expanded by first initializing an empty list, looping through list_0, checking the condition, and appending the transformed result.

5. What will be the output of the following Python code snippet?

```
x = [i**+1 for i in range(3)]; print(x);
```

a) [0, 1, 2]
b) [1, 2, 5]
c) error, **+ is not a valid operator
d) error, ';' is not allowed

Answer: a
Explanation: The expression i**+1 is evaluated as i raised to the power of +1, which is effectively i. Therefore, the list comprehension returns [0, 1, 2], and the semicolon used is syntactically valid in Python.

6. What will be the output of the following Python code snippet?

```
print([i.lower() for i in "HELLO"])
```

a) ['h', 'e', 'l', 'l', 'o']
b) 'hello'
c) ['hello']
d) hello

Answer: a
Explanation: The code loops through each character in the string "HELLO" and converts each to lowercase using .lower(). The list comprehension collects these lowercase letters into a list and prints it.

7. What will be the output of the following Python code snippet?

```
print([i+j for i in "abc" for j in "def"])
```

a) ['da', 'ea', 'fa', 'db', 'eb', 'fb', 'dc', 'ec', 'fc']
b) [['ad', 'bd', 'cd'], ['ae', 'be', 'ce'], ['af', 'bf', 'cf']]

c) [[‘da’, ‘db’, ‘dc’], [‘ea’, ‘eb’, ‘ec’], [‘fa’, ‘fb’, ‘fc’]]
d) [‘ad’, ‘ae’, ‘af’, ‘bd’, ‘be’, ‘bf’, ‘cd’, ‘ce’, ‘cf’]

Answer: d
Explanation: The nested list comprehension combines each character i from “abc” with each character j from “def”, producing all possible concatenated pairs in the order of outer i and inner j.

8. What will be the output of the following Python code snippet?

```
print([[i+j for i in "abc"] for j in "def"])
```

a) [‘da’, ‘ea’, ‘fa’, ‘db’, ‘eb’, ‘fb’, ‘dc’, ‘ec’, ‘fc’]
b) [[‘ad’, ‘bd’, ‘cd’], [‘ae’, ‘be’, ‘ce’], [‘af’, ‘bf’, ‘cf’]]
c) [[‘da’, ‘db’, ‘dc’], [‘ea’, ‘eb’, ‘ec’], [‘fa’, ‘fb’, ‘fc’]]
d) [‘ad’, ‘ae’, ‘af’, ‘bd’, ‘be’, ‘bf’, ‘cd’, ‘ce’, ‘cf’]

Answer: b
Explanation: The expression uses a nested list comprehension. For each character j in “def”, it iterates over “abc” with i, concatenating i + j. This forms inner lists like [‘ad’, ‘bd’, ‘cd’], [‘ae’, ‘be’, ‘ce’], and [‘af’, ‘bf’, ‘cf’], which are collected into a single outer list.

9. What will be the output of the following Python code snippet?

```
print([if i%2==0: i; else: i+1; for i in range(4)])
```

a) [0, 2, 2, 4]
b) [1, 1, 3, 3]
c) error
d) none of the mentioned

Answer: c
Explanation: The code uses an invalid syntax for conditional logic within a list comprehension. Python requires the ternary operator format (x if condition else y), not an if-else block with colons and semicolons.

10. Which of the following is the same as list(map(lambda x: x-1, [1, 2,** 3]))?**
a) [x**-1 for x in [(1, 2, 3)]]
b) [1/x for x in [(1, 2, 3)]]
c) [1/x for x in (1, 2, 3)]
d) error

Answer: c
Explanation: The expression x**-1 is equivalent to 1/x. The map() function applies the lambda to each element of the list [1, 2, 3], just like the list comprehension in option c, which also iterates over a tuple (1, 2, **3) and evaluates 1/x for each element. Thus, both produce the same result: [1.0, 0.5, 0.333…].**
1. What will be the output of the following Python code?

```
l=[1,2,3,4,5]
print([x&1 for x in l])
```

a) [1, 1, 1, 1, 1]
b) [1, 0, 1, 0, 1]
c) [1, 0, 0, 0, 0]
d) [0, 1, 0, 1, 0]

Answer: b
Explanation: The bitwise AND operator & checks whether each number is odd by evaluating x & 1. If the number is odd, it returns 1; otherwise, it returns 0. This results in the list [1, 0, 1, 0, 1].

2. What will be the output of the following Python code?

```
l1=[1,2,3]
l2=[4,5,6]
print([x*y for x in l1 for y in l2])
```

a) [4, 8, 12, 5, 10, 15, 6, 12, 18]
b) [4, 10, 18]
c) [4, 5, 6, 8, 10, 12, 12, 15, 18]
d) [18, 12, 6, 15, 10, 5, 12, 8, 4]

Answer: c
Explanation: The given list comprehension multiplies each element from list l1 with every element from list l2, generating a Cartesian product of their multiplication. It first takes x from l1 and for each x, iterates through all y in l2, calculating x*y. This results in the list: [4, 5, 6, 8, 10, 12, 12, 15, 18].

3. Write the list comprehension to pick out only negative integers from a given list 'l'.

a) [x<0 in l]
b) [x for x<0 in l]
c) [x in l for x<0]
d) [x for x in l if x<0]

Answer: d
Explanation: The correct list comprehension to extract only negative integers from a list l is [x for x in l if x < 0]. This statement iterates over each element x in the list l and includes it in the new list only if x is less than zero. For example, given l = [-65, 2, 7, -99, -4, 3], this comprehension will produce [-65, -99, -4].

4. What will be the output of the following Python code?

```
s=["pune", "mumbai", "delhi"]
print([(w.upper(), len(w)) for w in s])
```

a) Error
b) ['PUNE', 4, 'MUMBAI', 6, 'DELHI', 5]
c) [PUNE, 4, MUMBAI, 6, DELHI, 5]

d) [(‘PUNE’, 4), (‘MUMBAI’, 6), (‘DELHI’, 5)]

Answer: d
Explanation: The output of the code is a list of tuples where each tuple contains a word from the list in uppercase and its length. This happens because the list comprehension iterates over each word w in the list s, converts it to uppercase using w.upper(), and pairs it with len(w) in a tuple. Thus, the output is: [(‘PUNE’, 4), (‘MUMBAI’, 6), (‘DELHI’, 5)].

5. What will be the output of the following Python code?

```
l1=[2,4,6]
l2=[-2,-4,-6]
for i in zip(l1, l2):
        print(i)
```

a)
2, -2
4, -4
6, -6

b) [(2, -2), (4, -4), (6, -6)]

c)
(2, -2)
(4, -4)
(6, -6)

d) [-4, -16, -36]

Answer: c
Explanation: The output of the code shown will be:
(2, -2)
(4, -4)
(6, -6)

The zip function pairs elements from l1 and l2 into tuples. The for loop then prints each tuple on a new line, resulting in the tuple format with parentheses shown in the output.

6. What will be the output of the following Python code?

```
l1=[10, 20, 30]
l2=[-10, -20, -30]
l3=[x+y for x, y in zip(l1, l2)]
print(l3)
```

a) Error
b) 0
c) [-20, -60, -80]
d) [0, 0, 0]

Answer: d
Explanation: The zip function pairs elements from l1 and l2. The list comprehension sums each pair: 10 + (-10) = 0, 20 + (-20) = 0, and 30 + (-30) = 0. Hence, the resulting list is [0, 0, 0].

7. Write a list comprehension for number and its cube for l=[1, 2, 3, 4, 5, 6, 7, 8, 9].

a) [x**3 for x in l]
b) [x^3 for x in l]
c) [x**3 in l]
d) [x^3 in l]

Answer: a
Explanation: The list comprehension [x**3 for x in l] computes the cube of each number in the list l. The ** operator is used for exponentiation in Python.

8. What will be the output of the following Python code?

```
l=[[1 ,2, 3], [4, 5, 6], [7, 8, 9]]
print([[row[i] for row in l] for i in range(3)])
```

a) Error
b) [[1, 4, 7], [2, 5, 8], [3, 6, 9]]
c)
  1 4 7
  2 5 8
  3 6 9
d)
  (1 4 7)
  (2 5 8)
  (3 6 9)

Answer: b
Explanation: This code is effectively transposing the matrix l. It takes the elements at index i from each row and creates a new list for each i in the range 0 to 2 (since there are 3 columns). So, it converts rows into columns, resulting in [[1, 4, 7], [2, 5, 8], [3, 6, 9]].

9. What will be the output of the following Python code?

```
import math
print([str(round(math.pi)) for i in range (1, 6)])
```

a) [‘3’, ‘3’, ‘3’, ‘3’, ‘3’, ‘3’]
b) [‘3.1’, ‘3.14’, ‘3.142’, ‘3.1416’, ‘3.14159’, ‘3.141582’]
c) [‘3’, ‘3’, ‘3’, ‘3’, ‘3’]
d) [‘3.1’, ‘3.14’, ‘3.142’, ‘3.1416’, ‘3.14159’]

Answer: c
Explanation: The code rounds math.pi (which is approximately 3.14159) to the nearest integer using round(math.pi), which gives 3. This value is converted to a string and repeated 5 times in a list. So, the output is: [‘3’, ‘3’, ‘3’, ‘3’, ‘3’].

10. What will be the output of the following Python code?

```
l1=[1,2,3]
l2=[4,5,6]
l3=[7,8,9]
for x, y, z in zip(l1, l2, l3):
        print(x, y, z)
```

a)
1 4 7
2 5 8
3 6 9
b)
(1 4 7)
(2 5 8)
(3 6 9)
c) [(1, 4, 7), (2, 5, 8), (3, 6, 9)]
d) Error

Answer: a
Explanation: The zip() function combines l1, l2, and l3 element-wise into tuples like (1,4,7), (2,5,8), and (3,6,9). The for loop unpacks each tuple into x, y, z and prints them. Hence, the output is:
1 4 7
2 5 8
3 6 9

1. Read the information given below carefully and write a list comprehension such that the output is: [‘e’, ‘o’]

```
w="hello"
v=('a', 'e', 'i', 'o', 'u')
```

a) [x for w in v if x in v]
b) [x for x in w if x in v]
c) [x for x in v if w in v]
d) [x for v in w for x in w]

Answer: b
Explanation: The list comprehension [x for x in w if x in v] filters characters from the string w = “hello” that are also in the tuple v (which contains vowels). It returns only the vowels ‘e’ and ‘o’, producing the output: [‘e’, ‘o’].

2. What will be the output of the following Python code?

```
print([ord(ch) for ch in 'abc'])
```

a) [97, 98, 99]
b) [‘97’, ‘98’, ‘99’]
c) [65, 66, 67]
d) Error

Answer: a
Explanation: The ord() function returns the ASCII value of a character. The list comprehension [ord(ch) for ch in ‘abc’] gives the ASCII values of ‘a’, ‘b’, and ‘c’, resulting in: [97, 98, 99].

3. What will be the output of the following Python code?

```
t=32.00
print([round((x-32)*5/9) for x in t])
```

a) [0]
b) 0
c) [0.00]
d) Error

Answer: d
Explanation: The variable t = 32.00 is a float, and trying to iterate over it with for x in t will raise a TypeError because float objects are not iterable. Therefore, the code results in an error.

4. Write a list comprehension for producing a list of numbers between 1 and 1000 that are divisible by 3.
a) [x in range(1, 1000) if x%3==0]
b) [x for x in range(1, 1001) if x % 3 == 0]
c) [x%3 for x in range(1, 1000)]
d) [x%3=0 for x in range(1, 1000)]

Answer: b
Explanation: The list comprehension [x for x in range(1, 1001) if x % 3 == 0] correctly generates all numbers from 1 to 1000 (inclusive) that are divisible by 3. It uses a for loop inside the comprehension and filters values using the condition x % 3 == 0.
5. Write a list comprehension equivalent for the Python code shown below.

```
for i in range(1, 101):
        if int(i*0.5)==i*0.5:
                print(i)
```

a) [i for i in range(1, 100) if int(i*0.5)==(i*0.5)]
b) [i for i in range(1, 101) if int(i*0.5)==(i*0.5)]
c) [i for i in range(1, 101) if int(i*0.5)=(i*0.5)]

d) [i for i in range(1, 100) if int(i*0.5)=(i*0.5)]

Answer: b
Explanation: The list comprehension [i for i in range(1, 101) if int(i*0.5) == (i*0.5)] iterates over numbers from 1 to 100, and includes only those numbers where multiplying by 0.5 results in an integer (i.e., even numbers). This matches the behavior of the original for-loop with the conditional print.

6. What is the list comprehension equivalent for: list(map(lambda x:x-1, [1, 2, 3]))?**

a) [1|x for x in [1, 2, 3]]
b) [-1**x for x in [1, 2, 3]]
c) [x**-1 for x in [1, 2, 3]]
d) [x^-1 for x in range(4)]

Answer: c
Explanation: The list comprehension [x**-1 for x in [1, 2, 3]] computes the reciprocal of each element in the list, just like the map function with a lambda. Raising a number to the power of -1 means taking its inverse, so the output for both methods is the same: [1.0, 0.5, 0.3333…].

7. Write a list comprehension to produce the list: [1, 2, 4, 8, 16……212].

a) [(2**x) for x in range(0, 13)]
b) [(x**2) for x in range(1, 13)]
c) [(2**x) for x in range(1, 13)]
d) [(x**2) for x in range(0, 13)]

Answer: a
Explanation: The list comprehension [2**x for x in range(0, 13)] generates powers of 2 starting from $2^0 = 1$ up to $2^{12} = 4096$. It follows exponential growth using base 2.

8. What is the list comprehension equivalent for?

{x : x is a whole number less than 20, x is even} (including zero)

a) [x for x in range(1, 20) if (x%2==0)]

b) [x for x in range(0, 20) if (x//2==0)]

c) [x for x in range(1, 20) if (x//2==0)]

d) [x for x in range(0, 20) if (x%2==0)]

Answer: d

Explanation: This comprehension generates all whole numbers from 0 up to (but not including) 20 and includes only those that are even (i.e., x % 2 == 0). It matches the requirement of even whole numbers less than 20, including zero.

9. What will be the output of the following Python list comprehension?

```
print([j for i in range(2,8) for j in range(i*2, 50, i)])
```

a) A list of prime numbers up to 50

b) A list of numbers divisible by 2, up to 50

c) A list of non prime numbers, up to 50

d) Error

Answer: c

Explanation: The comprehension iterates over i from 2 to 7, and for each i, it generates multiples of i starting from i*2 up to 50 with a step of i. This effectively lists out composite numbers (non-primes) that have factors between 2 and 7, covering most non-prime numbers under 50.

10. What will be the output of the following Python code?

```
l=["good", "oh!", "excellent!", "#450"]
print([n for n in l if n.isalpha() or n.isdigit()])
```

a) [‘good’, ‘oh’, ‘excellent’, ‘450’]

b) [‘good’]

c) [‘good’, ‘#450’]

d) [‘oh!’, ‘excellent!’, ‘#450’]

Answer: b
Explanation: The list comprehension filters elements that are either fully alphabetic or fully numeric. Only “good” is purely alphabetic; others contain special characters or symbols, so they are excluded.

1. Which of the following matrices will throw an error in Python?

a)

A = [[1, 2, 3],
[4, 5, 6],
[7, 8, 9]]

b)

B = [[3, 3, 3]
[4, 4, 4]
[5, 5, 5]]

c)

C = [(1, 2, 4),
(5, 6, 7),
(8, 9, 10)]

d)

D = [2, 3, 4,
3, 3, 3,
4, 5, 6]

Answer: b
Explanation: In matrix B will result in an error because in the absence of a comma at the end of each row, it behaves like three separate lists. The error thrown states that the list integers must be integers or slices, not tuples.

2. What will be the output of the following Python code?

```
A = [[1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]]
print(A[1])
```

a) [4, 5, 6]
b) [3, 6, 9]
c) [1, 4, 7]
d) [1, 2, 3]

Answer: a
Explanation: In the given 2D list A, A[1] refers to the second row (indexing starts from 0). So, A[1] retrieves the list [4, 5, 6], which is the second row of the matrix.

3. Which of the following Python statements will result in the output: 6?

```
A = [[1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]]
```

a) A[2][3]
b) A[2][1]
c) A[1][2]
d) A[3][2]

Answer: c
Explanation: In the 2D list A, the second row (index 1) is [4, 5, 6]. The third element in this row (index 2) is 6. Therefore, A[1][2] results in the output 6.

4. What will be the output of the following Python code?

```
A = [[1, 2, 3],
     [4, 5, 6],
     [7, 8, 9]]
print([A[row][1] for row in (0, 1, 2)])
```

a) [7, 8, 9]
b) [4, 5, 6]
c) [2, 5, 8]
d) [1, 4, 7]

Answer: c
Explanation: The list comprehension [A[row][1] for row in (0, 1, 2)] iterates over each row and selects the element at index 1 (the second element) from each row. The second element in each row is 2, 5, and 8, producing the output [2, 5, 8].

5. What will be the output of the following Python code?

```
A = [[1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]]
print([A[i][i] for i in range(len(A))])
```

a) [1, 5, 9]
b) [3, 5, 7]
c) [4, 5, 6]
d) [2, 5, 8]

Answer: a
Explanation: The list comprehension [A[i][i] for i in range(len(A))] selects elements where the row index and column index are the same, effectively pulling out the diagonal of the matrix. This results in the elements A[0][0], A[1][1], and A[2][2], which are 1, 5, and 9, respectively.

6. What will be the output of the following Python code?

```
l=[[1, 2, 3], [4, 5, 6]]
for i in range(len(l)):
        for j in range(len(l[i])):
                l[i][j]+=10
print(l)
```

a) No output
b) Error
c) [[1, 2, 3], [4, 5, 6]]
d) [[11, 12, 13], [14, 15, 16]]

Answer: d
Explanation: The code iterates through each sublist in l, and within each sublist, it adds 10 to each element. After the loop, the updated list is [[11, 12, 13], [14, 15, 16]].

7. What will be the output of the following Python code?

```
A = [[1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]]

print([[col + 10 for col in row] for row in A])
```

a) [[11, 12, 13], [14, 15, 16], [17, 18, 19]]
b) Error
c) [11, 12, 13], [14, 15, 16], [17, 18, 19]
d) [11, 12, 13, 14, 15, 16, 17, 18, 19]

Answer: a
Explanation: The list comprehension [[col + 10 for col in row] for row in A]

iterates over each row in matrix A and adds 10 to each element in that row. The resulting matrix is [[11, 12, 13], [14, 15, 16], [17, 18, 19]].

8. What will be the output of the following Python code?

```
A = [[1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]]
print([A[i][len(A)-1-i] for i in range(len(A))])
```

a) [1, 5, 9]
b) [4, 5, 6]
c) [3, 5, 7]
d) [2, 5, 8]

Answer: c
Explanation: The expression [A[i][len(A)-1-i] for i in range(len(A))] selects elements from the anti-diagonal of the square matrix A. It fetches A[0][2], A[1][1], and A[2][0], which are 3, 5, and 7, resulting in [3, 5, 7].

9. What will be the output of the following Python code?

```
A = [[1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]]
B = [[3, 3, 3],
    [4, 4, 4],
    [5, 5, 5]]
print([B[row][col]*A[row][col] for row in range(3) for col in range(3)])
```

a) [3, 6, 9, 16, 20, 24, 35, 40, 45]
b) Error
c) [0, 30, 60, 120, 160, 200, 300, 350, 400]
d) 0

Answer: a

Explanation: The list comprehension [B[row][col]*A[row][col] for row in range(3) for col in range(3)] performs element-wise multiplication of two 3×3 matrices A and B, row by row. The results are flattened into a single list:

[1×3, 2×3, 3×3, 4×4, 5×4, 6×4, 7×5, 8×5, 9×5] → [3, 6, 9, 16, 20, 24, 35, 40, 45].

10. What will be the output of the following Python code?

```
r = [11, 12, 13, 14, 15, 16, 17, 18, 19]
A = [[0, 10, 20],
        [30, 40, 50],
        [60, 70, 80]]
for row in A:
    for col in row:
        r.append(col+10)
print(r)
```

a) [11, 12, 13, 14, 15, 16, 17, 18, 19, 10, 20, 30, 40, 50, 60, 70, 80, 90]

b) [10, 20, 30, 40, 50, 60, 70, 80, 90]

c) [11, 12, 13, 14, 15, 16, 17, 18, 19]

d) [0, 10, 20, 30, 40, 50, 60, 70, 80]

11. What will be the output of the following Python code?

```
A = [[1, 2, 3],
   [4, 5, 6],
   [7, 8, 9]]
B = [[3, 3, 3],
   [4, 4, 4],
   [5, 5, 5]]
print([[col1 * col2 for (col1, col2) in zip(row1, row2)] for (row1, row2) in zip(A, B)])
```

a) [0, 30, 60, 120, 160, 200, 300, 350, 400]
b) [[3, 6, 9], [16, 20, 24], [35, 40, 45]]
c) No output
d) Error

Answer: b
Explanation: The code multiplies corresponding elements of matrices A and B using nested list comprehensions with zip(). Each row from A and B is paired, and their elements are multiplied element-wise. The result is a new matrix with the same dimensions.

12. What will be the output of the following Python code?

```
A = [[1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]]
B = [[3, 3, 3],
    [4, 4, 4],
    [5, 5, 5]]
print(zip(A, B))
```

a) Address of the zip object
b) Address of the matrices A and B
c) No output
d) [3, 6, 9, 16, 20, 24, 35, 40, 45]

Answer: a
Explanation: The function zip(A, B) returns a zip object, which is an iterator of paired rows from A and B. Since it's not converted to a list or looped over, printing it will show its memory address.
list(zip(A, B))
[([1, 2, 3], [3, 3, 3]), ([4, 5, 6], [4, 4, 4]), ([7, 8, 9], [5, 5, 5])]

1. Which of the following is a Python tuple?

a) [1, 2, 3]
b) (1, 2, 3)
c) {1, 2, 3}
d) {}

Answer: b
Explanation: A tuple in Python is an immutable sequence type and is defined using round brackets ( ). For example, (1, 2, 3) is a tuple.

2. Suppose t = (1, 2, 4, 3), which of the following is incorrect?

a) print(t[3])
b) t[3] = 45
c) print(max(t))
d) print(len(t))

Answer: b
Explanation: Tuples are immutable, meaning their elements can't be changed after creation. So, t[3] = 45 is invalid and will raise a TypeError.

3. What will be the output of the following Python code?

```
t=(1,2,4,3)
print(t[1:3])
```

a) (1, 2)
b) (1, 2, 4)
c) (2, 4)
d) (2, 4, 3)

Answer: c
Explanation: t[1:3] extracts elements from index 1 up to (but not

including) index 3. So it returns (2, 4) from the tuple (1, 2, 4, 3). Slicing in tuples works like in lists or strings.

4. What will be the output of the following Python code?

```
t=(1,2,4,3)
print(t[1:-1])
```

a) (1, 2)
b) (1, 2, 4)
c) (2, 4)
d) (2, 4, 3)

Answer: c
Explanation: t[1:-1] slices the tuple from index 1 up to (but not including) the last element. So it returns (2, 4) from (1, 2, 4, 3). Negative indices count from the end, just like in lists or strings.

5. What will be the output of the following Python code?

```
t = (1, 2, 4, 3, 8, 9)
print([t[i] for i in range(0, len(t), 2)])
```

a) [2, 3, 9]
b) [1, 2, 4, 3, 8, 9]
c) [1, 4, 8]
d) (1, 4, 8)

Answer: c
Explanation: The list comprehension picks every second element from the tuple starting at index 0. This gives elements at indices 0, 2, and 4, resulting in [1, 4, 8].

6. What will be the output of the following Python code?

```
d = {"john":40, "peter":45}
print(d["john"])
```

a) 40
b) 45
c) “john”
d) “peter”

Answer: a
Explanation: The dictionary d stores key-value pairs. Accessing d[“john”] looks up the key “john” and returns its value, which is 40.

7. What will be the output of the following Python code?

```
t = (1, 2)
print(2 * t)
```

a) (1, 2, 1, 2)
b) [1, 2, 1, 2]
c) (1, 1, 2, 2)
d) [1, 1, 2, 2]

Answer: a
Explanation: Multiplying a tuple by an integer n replicates its elements n times in sequence. So 2 * (1, 2) results in (1, 2, 1, 2).

8. What will be the output of the following Python code?

```
t1 = (1, 2, 4, 3)
t2 = (1, 2, 3, 4)
print(t1 < t2)
```

a) True
b) False
c) Error
d) None

Answer: b
Explanation: Tuples are compared element by element. Since the third

element in t1 (which is 4) is greater than that in t2 (which is 3), the comparison t1 < t2 evaluates to False.

9. What will be the output of the following Python code?

```
my_tuple = (1, 2, 3, 4)
my_tuple.append( (5, 6, 7) )
print len(my_tuple)
```

a) 1
b) 2
c) 5
d) Error

Answer: d
Explanation: Tuples are immutable and do not have an append() method. So trying to call my_tuple.append(…) will raise an AttributeError.

10. What will be the output of the following Python code?

```
numberGames = {}
numberGames[(1,2,4)] = 8
numberGames[(4,2,1)] = 10
numberGames[(1,2)] = 12
sum = 0
for k in numberGames:
   sum += numberGames[k]
print(len(numberGames) + sum)
```

a) 30
b) 24
c) 33
d) 12

Answer: c
Explanation: The dictionary has 3 unique tuple keys with values 8, 10, and **12. Their sum is 30. Adding the number of keys (3) gives the final result: 30 + 3 = 33.**
1. What is the data type of (1)?
a) Tuple
b) Integer
c) List
d) Both tuple and integer

Answer: b
Explanation: The data type of (1) is integer because the parentheses alone don't make it a tuple—they act as grouping. To create a tuple with one element, you must include a comma: (1,).
2. If a=(1,2,3,4), a[1:-1] is _____
a) Error, tuple slicing doesn't exist
b) [2,3]
c) (2,3,4)
d) (2,3)

Answer: d
Explanation: Tuple slicing works just like list slicing. For a = (1, 2, 3, 4), a[1:-1] extracts elements from index 1 up to (but not including) the last element, which gives (2, 3).
3. What will be the output of the following Python code?

```
a=(1,2,(4,5))
b=(1,2,(3,4))
print(a<b)
```

a) False
b) True
c) Error, < operator is not valid for tuples
d) Error, < operator is valid for tuples but not if there are sub-tuples

Answer: a
Explanation: Python compares tuples element by element from left to right. Here, the first two elements of a and b are equal, so it compares the sub-tuples (4, 5) and (3, 4). Since 4 is greater than 3, a < b is False.

4. What will be the output of the following Python code?

```
a=("Check")*3
print(a)
```

a) ('Check','Check','Check')
b) * Operator not valid for tuples
c) ('CheckCheckCheck')
d) Syntax error

Answer: c
Explanation: ("Check") without a comma is just a string in parentheses, not a tuple. Multiplying the string by 3 repeats it three times, resulting in "CheckCheckCheck".

5. What will be the output of the following Python code?

```
a=(1,2,3,4)
print(del(a[2]))
```

a) Now, a=(1,2,4)
b) Now, a=(1,3,4)
c) Now a=(3,4)
d) Error as tuple is immutable

Answer: d
Explanation: Tuples are immutable, so you cannot delete an element using del(a[2]). This operation raises a TypeError.

6. What will be the output of the following Python code?

```
a=(2,3,4)
print(sum(a,3))
```

a) Too many arguments for sum() method
b) The method sum() doesn't exist for tuples
c) 12
d) 9

Answer: c
Explanation: The sum() function takes an iterable (here, the tuple a) and an optional start value (here, 3). It adds all elements of a and then adds the start value. So, sum(a, 3) = 2 + 3 + 4 + 3 = 12.

7. Is the following Python code valid?

```
a=(1,2,3,4)
del a
```

a) No because tuple is immutable
b) Yes, first element in the tuple is deleted
c) Yes, the entire tuple is deleted
d) No, invalid syntax for del method

Answer: c
Explanation: del a deletes the entire variable a, not just elements inside it. Even though tuples are immutable, deleting the whole variable is allowed.

8. What type of data is: a=[(1,1),(2,4),(3,9)]?

a) Array of tuples
b) List of tuples

c) Tuples of lists
d) Invalid type

Answer: b
Explanation: The variable a is a list where each item is a tuple — for example, (1, 1), (2, 4), (3, 9). Hence, the data type is a list of tuples.

9. What will be the output of the following Python code?

```
a=(0,1,2,3,4)
b=slice(0,2)
print(a[b])
```

a) Invalid syntax for slicing
b) [0,2]
c) (0,1)
d) (0,2)

Answer: c
Explanation: slice(0, 2) creates a slice object equivalent to [0:2]. So, a[b] returns elements from index 0 up to (but not including) 2, which is (0, 1).

10. Is the following Python code valid?

```
a=(1,2,3)
b=('A','B','C')
c=tuple(zip(a,b))
print(c)
```

a) Yes, c will be ((1, ‘A’), (2, ‘B’), (3, ‘C’))
b) Yes, c will be ((1,2,3),(‘A’,’B’,’C’))
c) No because tuples are immutable
d) No because the syntax for zip function isn’t valid

Answer: a
Explanation: The zip(a, b) pairs elements from a and b one by one,

creating tuples like (1, ‘A’), (2, ‘B’), (3, ‘C’). Wrapping it with tuple() converts the zipped object into a tuple of these pairs.

1. Is the following Python code valid?

```
a,b,c=1,2,3
print (a,b,c)
```

a) Yes, [1,2,3] is printed
b) No, invalid syntax
c) Yes, (1,2,3) is printed
d) 1 is printed

Answer: c
Explanation: In Python, multiple assignment like a, b, c = 1, 2, 3 assigns values to variables a, b, and c simultaneously. When you just type a, b, c in the shell, it displays a tuple (1, 2, 3).

2. What will be the output of the following Python code?

```
a = ('check',)
n = 2
for i in range(int(n)):
    a = (a,)
    print(a)
```

a) Error, tuples are immutable
b)
(('check',),)
((('check',),),)
c) ((‘check’,)’check’,)
d)
(('check',)’check’,)
((('check',)’check’,)’check’,)

Answer: b
Explanation: The code initializes a as a tuple with one element ‘check’. In each loop iteration, a is rewrapped in another tuple, adding one level of nesting. The output is two lines: ((‘check’,),) and (((‘check’,),),).

3. Is the following Python code valid?

```
a,b=1,2,3
```

a) Yes, this is an example of tuple unpacking. a=1 and b=2
b) Yes, this is an example of tuple unpacking. a=(1,2) and b=3
c) No, too many values to unpack
d) Yes, this is an example of tuple unpacking. a=1 and b=(2,3)

Answer: c
Explanation: Unpacking requires the number of variables on the left to match the number of values on the right. Here, there are 2 variables (a, b) but 3 values (1, 2, 3). This mismatch causes Python to raise a “too many values to unpack” error.

4. What will be the output of the following Python code?

```
a=(1,2)
b=(3,4)
c=a+b
print(c)
```

a) (4, 6)
b) (1, 2, 3, 4)
c) Error as tuples are immutable
d) None

Answer: b
Explanation: The + operator concatenates two tuples by joining their

elements in order. Since a is (1, 2) and b is (3, 4), a + b creates a new tuple (1, 2, 3, 4). Tuples remain unchanged because they are immutable, but concatenation produces a new tuple.

5. What will be the output of the following Python code?

```
a,b=6,7
a,b=b,a
print(a,b)
```

a) 6, 7
b) Invalid syntax
c) 7, 6
d) Nothing is printed

Answer: c
Explanation: The statement a, b = b, a swaps the values of a and b using tuple unpacking. Initially, a is 6 and b is 7, but after swapping, a becomes 7 and b becomes 6. This is a common Pythonic way to swap variables without a temporary variable.

6. What will be the output of the following Python code?

```
import collections
a=collections.namedtuple('a',['i','j'])
obj=a(i=4,j=7)
print(obj)
```

a) a(i=4, j=7)
b) obj(i=4, j=7)
c) (4,7)
d) An exception is thrown

Answer: a
Explanation: The namedtuple creates a custom tuple class named ‘a’ with

fields ‘i’ and ‘j’. When you create an object obj = a(i=4, j=7), it stores the values as a tuple with named fields. Printing obj shows a(i=4, j=7).

7. Tuples can’t be made keys of a dictionary.

a) True

b) False

Answer: b

Explanation: Tuples can be used as dictionary keys because they are immutable. However, this only works if all elements inside the tuple are also immutable (e.g., numbers, strings). Mutable types like lists inside a tuple make it unhashable and unusable as a key.

8. Is the following Python code valid?

```
a=2,3,4,5
print (a)
```

a) Yes, 2 is printed

b) Yes, [2,3,4,5] is printed

c) No, too many values to unpack

d) Yes, (2, 3, 4, 5) is printed

Answer: d

Explanation: In Python, assigning values like a=2,3,4,5 without parentheses creates a tuple. So a becomes the tuple (2, 3, 4, 5) and printing a displays it in that format. Parentheses are optional when defining a tuple with multiple items.

9. What will be the output of the following Python code?

```
a=(2,3,1,5)
a.sort()
print(a)
```

a) (1,2,3,5)

b) (2,3,1,5)

c) None
d) Error, tuple has no attribute sort

Answer: d
Explanation: Tuples are immutable in Python, meaning you can't change their contents. The .sort() method is only available for mutable sequences like lists, not for tuples. Hence, calling a.sort() raises an AttributeError.

10. Is the following Python code valid?

```
a=(1,2,3)
print(b=a.update(4,))
```

a) Yes, a=(1,2,3,4) and b=(1,2,3,4)
b) Yes, a=(1,2,3) and b=(1,2,3,4)
c) No because tuples are immutable
d) No because wrong syntax for update() method

Answer: c
Explanation: Tuples in Python cannot be modified after creation, so there's no update() method for tuples. Attempting to call a.update(4,) will raise an AttributeError. Tuples are immutable, meaning their contents cannot be changed or added to.

11. What will be the output of the following Python code?

```
a=[(2,4),(1,2),(3,9)]
a.sort()
print(a)
```

a) [(1, 2), (2, 4), (3, 9)]
b) [(2, 4),(1, 2),(3, 9)]
c) Error because tuples are immutable
d) Error, tuple has no sort attribute

Answer: a
Explanation: The list a contains tuples, and calling a.sort() sorts the list based on the first element of each tuple (and the second if needed). Since lists are mutable, sorting works fine, even if the elements are immutable like tuples.

1. Which of these about a set is not true?

a) Mutable data type
b) Does not allow duplicate values
c) Data type with unordered values
d) Immutable data type

Answer: d
Explanation: A set in Python is a mutable data type, meaning its elements can be added or removed after creation. It does not allow duplicate values and stores elements in an unordered manner. Therefore, the statement that a set is an immutable data type is incorrect.

2. Which of the following is not the correct syntax for creating a set?

a) set([[1,2],[3,4]])
b) set([1,2,2,3,4])
c) set((1,2,3,4))
d) {1,2,3,4}

Answer: a
Explanation: While sets require an iterable as input, all elements inside the iterable must also be hashable. Lists like [1,2] and [3,4] are unhashable, so set([[1,2],[3,4]]) will raise a TypeError. The other options use valid and hashable elements like integers or tuples.

3. What will be the output of the following Python code?

```
nums = set([1,1,2,3,3,3,4,4])
```

```
print(len(nums))
```
a) 7
b) Error, invalid syntax for formation of set
c) 4
d) 8

Answer: c
Explanation: In the code nums = set([1,1,2,3,3,3,4,4]), the list contains duplicates, but when it's converted to a set, all duplicates are removed. The resulting set is {1, 2, 3, 4}, which has 4 unique elements. Thus, len(nums) returns 4.

4. What will be the output of the following Python code?

```
a = [5,5,6,7,7,7]
b = set(a)
def test(lst):
    if lst in b:
        return 1
    else:
        return 0
for i in  filter(test, a):
    print(i,end=" ")
```
a) 5 5 6
b) 5 6 7
c) 5 5 6 7 7 7
d) 5 6 7 7 7

Answer: c
Explanation: The test function checks whether each element of list a exists in set b. Since all elements of a are present in b, the filter includes every item in a. Therefore, the output is the full list: 5 5 6 7 7 7.

5. Which of the following statements is used to create an empty set?

a) { }

b) set()

c) [ ]

d) ( )

Answer: b

Explanation: In Python, using {} creates an empty dictionary, not a set. To create an empty set, you must use the set() constructor. Options like [] and () create empty list and tuple respectively, not sets. Therefore, only set() correctly creates an empty set.

6. What will be the output of the following Python code?

```
a={5,4}
b={1,2,4,5}
print(a<b)
```

a) {1,2}

b) True

c) False

d) Invalid operation

Answer: b

Explanation: In Python, the < operator checks whether one set is a proper subset of another. In this case, a = {5, 4} and b = {1, 2, 4, 5}. All elements of set a are present in set b, and a is not equal to b, so a < b evaluates to True.

7. If a={5,6,7,8}, which of the following statements is false?

a) print(len(a))

b) print(min(a))

c) a.remove(5)

d) a[2]=45

Answer: d
Explanation: Sets in Python are unordered collections, so their elements cannot be accessed or modified using indexing like a[2]. Therefore, a[2] = 45 is invalid and raises a TypeError, making it the false statement.

8. If a={5,6,7}, what happens when a.add(5) is executed?
a) a={5,5,6,7}
b) a={5,6,7}
c) Error as there is no add function for set data type
d) Error as 5 already exists in the set

Answer: b
Explanation: In a set, duplicate elements are automatically ignored. So, adding 5 again does nothing since it already exists in the set. The set remains unchanged as {5, 6, 7}.
9. What will be the output of the following Python code?

```
a={4,5,6}
b={2,8,6}
print(a+b)
```

a) {4,5,6,2,8}
b) {4,5,6,2,8,6}
c) Error as unsupported operand type for sets
d) Error as the duplicate item 6 is present in both sets

Answer: c
Explanation: The + operator is not supported between sets in Python. To combine two sets, you should use set methods like union() or the | operator. Using a + b results in a TypeError.

10. What will be the output of the following Python code?

```
a={4,5,6}
b={2,8,6}
print(a-b)
```

a) {4, 5}
b) {6}
c) Error as unsupported operand type for set data type
d) Error as the duplicate item 6 is present in both sets

Answer: a
Explanation: The – operator performs set difference. It returns elements that are in set a but not in set b. Since 6 is present in both sets, it is excluded from the result, leaving {4, 5}.

11. What will be the output of the following Python code?

```
a={5,6,7,8}
b={7,8,10,11}
print(a^b)
```

a) {5, 6, 7, 8, 10, 11}
b) {7, 8}
c) Error as unsupported operand type of set data type
d) {5, 6, 10, 11}

Answer: d
Explanation: The ^ operator performs a symmetric difference, which returns elements that are in either a or b, but not in both. Since 7 and 8 are common in both sets, they are excluded. The result is {5, 6, 10, 11}.

12. What will be the output of the following Python code?

```
s={5,6}
print(s*3)
```

a) Error as unsupported operand type for set data type
b) {5,6,5,6,5,6}
c) {5,6}
d) Error as multiplication creates duplicate elements which isn't allowed

Answer: a
Explanation: The * operator is not defined for sets in Python. Sets do not support repetition through multiplication, so attempting s * 3 results in a TypeError.

13. What will be the output of the following Python code?

```
a={5,6,7,8}
b={7,5,6,8}
print(a==b)
```

a) True
b) False

Answer: a
Explanation: Sets in Python are unordered collections, so the order of elements doesn't matter during comparison. Since both sets contain the same elements, a == b evaluates to True.

14. What will be the output of the following Python code?

```
a={3,4,5}
b={5,6,7}
print(a|b)
```

a) Invalid operation
b) {3, 4, 5, 6, 7}
c) {5}
d) {3, 4, 6, 7}

Answer: b
Explanation: The | operator performs a union of sets, which combines all unique elements from both sets. So, a | b results in {3, 4, 5, 6, 7}.

15. Is the following Python code valid?

```
a={3,4,{7,5}}
print(a[2][0])
```

a) Yes, 7 is printed
b) Error, elements of a set can't be printed
c) Error, subsets aren't allowed
d) Yes, {7,5} is printed

Answer: c
Explanation: In Python, set elements must be immutable (like integers, strings, tuples). Since sets themselves are mutable, trying to include a set {7, 5} as an element inside another set causes a TypeError. Therefore, subsets like {7, 5} inside a set are not allowed.

1. Which of these about a frozenset is not true?

a) Mutable data type
b) Allows duplicate values
c) Data type with unordered values
d) Immutable data type

Answer: a
Explanation: A frozenset is an immutable version of a set, meaning its elements cannot be changed after creation. Hence, calling it a mutable data type is not true. It does allow only unique elements and stores them in an unordered fashion.

2. What is the syntax of the following Python code?

```
a=frozenset(set([5,6,7]))
```

```
print(a)
```
a) {5, 6, 7}
b) frozenset({5, 6, 7})
c) Error, not possible to convert set into frozenset
d) Syntax error

Answer: b
Explanation: The code correctly converts a list to a set, and then to a frozenset. When printed, Python shows it as frozenset({5, 6, 7}), which indicates the data type and its contents.

3. Is the following Python code valid?

```
a = frozenset([5, 6, 7])
print(a)
a.add(5)
```
a) Yes, now a is {5,5,6,7}
b) No, frozen set is immutable
c) No, invalid syntax for add method
d) Yes, now a is {5,6,7}

Answer: b
Explanation: A frozenset is immutable, which means no elements can be added or removed after its creation. Calling a.add(5) will raise an AttributeError because frozenset objects do not support the .add() method.

4. Set members must not be hashable.

a) True
b) False

Answer: b
Explanation: Set members must be hashable, not the opposite. This means only immutable types like integers, strings, and tuples (with immutable elements) can be added to a set. Mutable types like lists or other sets cannot be elements of a set.

5. What will be the output of the following Python code?

```
a={3,4,5}
a.update([1,2,3])
print(a)
```

a) Error, no method called update for set data type
b) {1, 2, 3, 4, 5}
c) Error, list can't be added to set
d) Error, duplicate item present in list

Answer: b
Explanation: The update() method adds elements from an iterable (like a list) to the set. Duplicate values are ignored, and the final set includes all unique elements: {1, 2, 3, 4, 5}.

6. What will be the output of the following Python code?

```
a={1,2,3}
a.intersection_update({2,3,4,5})
print(a)
```

a) {2, 3}
b) Error, duplicate item present in list
c) Error, no method called intersection_update for set data type
d) {1, 4, 5}

Answer: a
Explanation: The intersection_update() method updates the set by

keeping only the elements that are also in the specified set. So, a becomes {2, 3} after removing elements not common to both sets.

7. What will be the output of the following Python code?

```
a={1,2,3}
b=a
b.remove(3)
print(a)
```

a) {1, 2, 3}
b) Error, copying of sets isn't allowed
c) {1, 2}
d) Error, invalid syntax for remove

Answer: c
Explanation: In Python, b = a makes b refer to the same set object as a (not a copy). So any modification done through b reflects in a. Removing 3 from b also removes it from a, resulting in {1, 2}.

8. What will be the output of the following Python code?

```
a={1,2,3}
b=a.copy()
b.add(4)
print(a)
```

a) {1, 2, 3}
b) Error, invalid syntax for add
c) {1, 2, 3, 4}
d) Error, copying of sets isn't allowed

Answer: a
Explanation: The copy() method creates a shallow copy of the set a, so b is an independent set. Adding 4 to b does not affect a. Therefore, printing a gives {1, 2, 3}.

9. What will be the output of the following Python code?

```
a={1,2,3}
b=a.add(4)
print(b)
```

a) 0
b) {1,2,3,4}
c) {1,2,3}
d) None

Answer: d
Explanation: In Python, the add() method adds an element to a set in-place and does not return anything. So when you write b = a.add(4), the value of b becomes None because add() returns None. The set a is updated to {1, 2, 3, 4}, but printing b outputs None.

10. What will be the output of the following Python code?

```
a={1,2,3}
b=frozenset([3,4,5])
print(a-b)
```

a) {1, 2}
b) Error as difference between a set and frozenset can’t be found out
c) Error as unsupported operand type for set data type
d) frozenset({1, 2})

Answer: a
Explanation: In Python, you can perform set operations like difference (-) between a set and a frozenset because both are iterable and support set operations. Here, a = {1, 2, 3} and b = frozenset([3, 4, 5]). The difference a – b removes elements in b from a, resulting in {1, 2}. The result is still a regular set, not a frozenset.

11. What will be the output of the following Python code?

```
a={5,6,7}
print(sum(a,5))
```

a) 5
b) 23
c) 18
d) Invalid syntax for sum method, too many arguments

Answer: b
Explanation: The sum() function in Python takes two arguments: an iterable and an optional starting value. In the code sum(a, 5), the set a = {5, 6, 7} is summed with a starting value of 5. So it calculates 5 (start) + 5 + 6 + 7 = 23. Therefore, the output is 23.

12. What will be the output of the following Python code?

```
a={1,2,3}
print({x*2 for x in a|{4,5}})
```

a) {2, 4, 6}
b) Error, set comprehensions aren't allowed
c) {2, 4, 6, 8, 10}
d) {8, 10}

Answer: c
Explanation: The union operator | combines sets a and {4, 5} resulting in {1, 2, 3, 4, 5}. The set comprehension then multiplies each element by 2, producing {2, 4, 6, 8, 10}.

13. What will be the output of the following Python code?

```
a={5,6,7,8}
b={7,8,9,10}
print(len(a+b))
```

a) 8
b) Error, unsupported operand '+' for sets

c) 6
d) Nothing is displayed

Answer: b
Explanation: In Python, the + operator is not supported for sets. You cannot directly add two sets using a + b; doing so will raise a TypeError. To combine sets, you should use set operations like a | b (union) or a.union(b). Therefore, the code results in an error.

14. What will be the output of the following Python code?

```
a={1,2,3}
b={1,2,3}
c=a.issubset(b)
print(c)
```

a) True
b) Error, no method called issubset() exists
c) Syntax error for issubset() method
d) False

Answer: a
Explanation: The issubset() method checks whether all elements of set a are present in set b. Here, both a and b are {1, 2, 3}, so a.issubset(b) returns True, since every element in a is also in b. The method is valid and correctly used.

15. Is the following Python code valid?

```
a={1,2,3}
b={1,2,3,4}
c=a.issuperset(b)
print(c)
```

a) False
b) True

c) Syntax error for issuperset() method
d) Error, no method called issuperset() exists

Answer: a
Explanation: The issuperset() method checks whether all elements of another set are contained in the calling set. In this case, a = {1, 2, 3} and b = {1, 2, 3, 4}. Since a does not contain all elements of b (it lacks 4), a.issuperset(b) returns False. The method is valid and correctly used.

1. What will be the output of the following Python code?

```
s=set()
print(type(s))
```

a) <'set'>
b) <class 'set'>
c) set
d) class set

Answer: b
Explanation: When you create an empty set using s = set() and then print type(s), Python outputs the type of the object. Since s is a set, the output is <class 'set'>, which indicates the data type of the variable.

2. The following Python code results in an error.

```
s={2, 3, 4, [5, 6]}
print(s)
```

a) True
b) False

Answer: a
Explanation: In Python, sets can only contain hashable (immutable) elements. Lists like [5, 6] are mutable and unhashable, so trying to

include a list inside a set (s = {2, 3, 4, [5, 6]}) will raise a TypeError. Therefore, the statement is True — the code results in an error.

3. Set makes use of _____

Dictionary makes use of _____

a) keys, keys

b) key values, keys

c) keys, key values

d) key values, key values

Answer: c

Explanation: A set in Python is implemented using a hash table and only stores keys (i.e., unique values without associated data). A dictionary also uses a hash table but stores both keys and their associated values (key-value pairs).

4. Which of the following lines of code will result in an error?

a) s={abs}

b) s={4, 'abc', (1,2)}

c) s={2, 2.2, 3, 'xyz'}

d) s={san}

Answer: d

Explanation: Here,

- abs is a built-in function and is hashable, so s = {abs} is valid.
- The set {4, 'abc', (1, 2)} contains immutable and hashable elements, so it's valid.
- The set {2, 2.2, 3, 'xyz'} contains hashable types, so it's valid.
- san is undefined (not quoted or declared), so s = {san} raises a NameError because san is not defined.

5. What will be the output of the following Python code?

```
s={2, 5, 6, 6, 7}
```

```
print(s)
```
a) {2, 5, 7}
b) {2, 5, 6, 7}
c) {2, 5, 6, 6, 7}
d) Error

Answer: b
Explanation: Sets automatically remove duplicate elements. In the set s = {2, 5, 6, 6, 7}, the duplicate 6 will be removed, so printing s outputs {2, 5, 6, 7}.

6. Input order is preserved in sets.

a) True
b) False

Answer: b
Explanation: Sets in Python are unordered collections, meaning they do not preserve the order of elements. When you create or print a set, the order of elements can appear arbitrary and may vary. Only from Python 3.7+, dictionaries preserve insertion order, but sets do not.

7. Write a list comprehension for number and its cube for:

```
l=[1, 2, 3, 4, 5, 6, 7, 8, 9]
```
a) [x**3 for x in l]
b) [x^3 for x in l]
c) [x**3 in l]
d) [x^3 in l]

Answer: a
Explanation: In Python, ** is the exponentiation operator used to calculate the cube of a number (x**3). The list comprehension [x**3 for x in l] generates a new list with each element being the cube of the

corresponding element in l. The ^ operator, used in options b and d, is a bitwise XOR, not exponentiation, so it's incorrect here.

8. What will be the output of the following Python code?

```
s={1, 2, 3}
s.update(4)
print(s)
```

a) {1, 2, 3, 4}
b) {1, 2, 4, 3}
c) {4, 1, 2, 3}
d) Error

Answer: d
Explanation: The update() method expects an iterable as an argument (like a list, set, or tuple) to add multiple elements to the set. Passing an integer 4 directly is not iterable, so s.update(4) will raise a TypeError. To add a single element using update(), you need to pass it as an iterable, for example: s.update([4]).

9. Which of the following functions cannot be used on heterogeneous sets?

a) pop
b) remove
c) update
d) sum

Answer: d
Explanation: Functions like pop(), remove(), and update() can be used on sets regardless of whether the elements are heterogeneous (mixed data types) or homogeneous. However, sum() requires all elements to be numeric because it adds them up. If the set contains mixed types (like

strings and numbers), sum() will raise a TypeError. Therefore, sum() cannot be used on heterogeneous sets.

10. What will be the output of the following Python code?

```
s={4>3, 0, 3-3}
print(all(s))
print(any(s))
```

a)
True
False
b)
False
True
c)
True
True
d)
False
False

Answer: b
Explanation: The set contains the values True, 0, and 0. The all() function returns False because not all elements are true—there are zeros which are considered false. The any() function returns True since at least one element (True) is True.

1. Which of the following functions will return the symmetric difference between two sets, x and y?

a) x | y
b) x ^ y
c) x & y

d) x – y

Answer: b
Explanation: The function x ^ y returns the symmetric difference between the two sets x and y. This is basically an XOR operation being performed on the two sets.

2. What will be the output of the following Python code snippet?

```
z=set('abc$de')
print('a' in z)
```

a) True
b) False
c) No output
d) Error

Answer: a
Explanation: The code shown above is used to check whether a particular item is a part of a given set or not. Since ‘a’ is a part of the set z, the output is true. Note that this code would result in an error in the absence of the quotes.

3. What will be the output of the following Python code snippet?

```
z=set('abc')
z.add('san')
z.update(set(['p', 'q']))
print(z)
```

a) {‘abc’, ‘p’, ‘q’, ‘san’}
b) {‘a’, ‘b’, ‘c’, [‘p’, ‘q’], ‘san}
c) {‘a’, ‘c’, ‘c’, ‘p’, ‘q’, ‘s’, ‘a’, ‘n’}
d) {‘c’, ‘a’, ‘san’, ‘q’, ‘p’, ‘b’}

Answer: d
Explanation: The code shown first adds the element ‘san’ to the set z. The set z is then updated and two more elements, namely, ‘p’ and ‘q’ are added to it. Hence the output is: {‘a’, ‘b’, ‘c’, ‘p’, ‘q’, ‘san’}

4. What will be the output of the following Python code snippet?

```
s=set([1, 2, 3])
print(s.union([4, 5]))
print(s|([4, 5]))
```

a)
  {1, 2, 3, 4, 5}
  {1, 2, 3, 4, 5}
b)
  Error
  {1, 2, 3, 4, 5}
c)
  {1, 2, 3, 4, 5}
  Error
d)
  Error
  Error

Answer: c
Explanation: In this Python snippet, the set s is created with values {1, 2, 3}. The method s.union([4, 5]) successfully returns a new set {1, 2, 3, 4, 5} because the union() method can take any iterable, including a list. However, the next line attempts to use the set union operator | between a set and a list, which is not supported in Python and causes a TypeError. So, the first print statement displays the correct union, while the second one raises an error.

5. What will be the output of the following Python code snippet?

```
for x in set('pqr'):
        print(x*2)
```

a)
pp
qq
rr
b)
pqr
pqr
c) ppqqrr
d) pqrpqr

Answer: a
Explanation: The expression set('pqr') creates a set with the characters 'p', 'q', and 'r', but in no guaranteed order because sets are unordered. The for loop iterates through each element in the set, and print(x*2) prints each character twice on a new line. Therefore, the output will be:
pp
qq
rr

6. What will be the output of the following Python code snippet?

```
print({a**2 for a in range(4)})
```

a) {1, 4, 9, 16}
b) {0, 1, 4, 9, 16}
c) Error
d) {0, 1, 4, 9}

Answer: d

Explanation: The code uses set comprehension to compute squares of numbers in range(4), i.e., 0 to 3. It calculates 0**2, 1**2, 2**2, and 3**2, resulting in the set {0, 1, 4, 9}. Since sets are unordered and contain unique values, the output is {0, 1, 4, 9}.

7. What will be the output of the following Python function?

```
print({x for x in 'abc'})
print({x*3 for x in 'abc'})
```

a)

{abc}
aaa
bbb
ccc

b)

abc
abc abc abc

c)

{‘a’, ‘b’, ‘c’}
{‘aaa’, ‘bbb’, ‘ccc’}

d)

{‘a’, ‘b’, ‘c’}
abc
abc
abc

Answer: c

Explanation: The expression {x for x in ‘abc’} is a set comprehension that creates a set of individual characters from the string ‘abc’, resulting in {‘a’, ‘b’, ‘c’} (order may vary due to the unordered nature of sets). The expression {x*3 for x in ‘abc’} multiplies each character by 3, producing

‘aaa’, ‘bbb’, and ‘ccc’, and collects them into a set: {‘aaa’, ‘bbb’, ‘ccc’}.
So the output will be:
{‘a’, ‘b’, ‘c’}
{‘aaa’, ‘bbb’, ‘ccc’}

8. The output of the following code is: class<’set’>.

```
type({})
```

a) True
b) False

Answer: b
Explanation: The expression type({}) does not return <class ‘set’> it actually returns because {} by default creates an empty dictionary, not a set. To create an empty set, you must use set(). Hence, the output is not <class ‘set’>, and the statement is False.

9. What will be the output of the following Python code snippet?

```
a=[1, 4, 3, 5, 2]
b=[3, 1, 5, 2, 4]
print(a==b)
print(set(a)==set(b))
```

a)
True
False
b)
False
False
c)
False
True

d)
True
True

Answer: c
Explanation: Here,

- a == b compares the order and elements of both lists. Since a and b have the same elements but in a different order, this comparison returns False.
- set(a) == set(b) compares the set of elements, which ignores order and duplicates. Since both a and b contain the same elements, their sets are equal, so this returns True.
  Thus, the output is:
  False
  True

10. What will be the output of the following Python code snippet?

```
l=[1, 2, 4, 5, 2, 'xy', 4]
print(set(l))
print(l)
```

a)
{1, 2, 4, 5, 2, ‘xy’, 4}
[1, 2, 4, 5, 2, ‘xy’, 4]
b)
{1, 2, 4, 5, ‘xy’}
[1, 2, 4, 5, 2, ‘xy’, 4]
c)
{1, 5, ‘xy’}
[1, 5, ‘xy’]

d)
{1, 2, 4, 5, ‘xy’}
[1, 2, 4, 5, ‘xy’]

Answer: b
Explanation: The set(l) function removes all duplicate elements from the list l and returns a set, which is an unordered collection of unique items. So set(l) gives {1, 2, 4, 5, ‘xy’}. However, print(l) displays the original list, which remains unchanged as [1, 2, 4, 5, 2, ‘xy’, 4].
So the output will be something like:
{1, 2, 4, 5, 'xy'}
[1, 2, 4, 5, 2, 'xy', 4]
(Note: Set element order may vary when printed.)

1. What will be the output of the following Python code?

```
s1={3, 4}
s2={1, 2}
s3=set()
i=0
j=0
for i in s1:
    for j in s2:
        s3.add((i,j))
        i+=1
        j+=1
print(s3)
```

a) {(3, 4), (1, 2)}
b) Error
c) {(4, 2), (3, 1), (4, 1), (5, 2)}
d) {(3, 1), (4, 2)}

Answer: c
Explanation: The code shown above finds the Cartesian product of the two sets, s1 and s2. The Cartesian product of these two sets is stored in a third set, that is, s3. Hence the output of this code is: {(4, 2), (3, 1), (4, 1), (5, 2)}.

2. The _____ function removes the first element of a set and the last element of a list.

a) remove
b) pop
c) discard
d) dispose

Answer: b
Explanation: The pop() function removes and returns an arbitrary element from a set (since sets are unordered) and removes the last element from a list by default. It behaves differently based on the data type.

3. The difference between the functions discard and remove is that:

a) Discard removes the last element of the set whereas remove removes the first element of the set
b) Discard throws an error if the specified element is not present in the set whereas remove does not throw an error in case of absence of the specified element
c) Remove removes the last element of the set whereas discard removes the first element of the set
d) Remove throws an error if the specified element is not present in the set whereas discard does not throw an error in case of absence of the specified element

Answer: d
Explanation: The difference between remove and discard in sets is that remove will raise an error if the element to be removed is not present, while discard will not raise any error and simply do nothing if the element is missing. This makes discard safer to use when you're unsure if the element exists in the set.

4. What will be the output of the following Python code?

```
s1={1, 2, 3}
s2={3, 4, 5, 6}
print(s1.difference(s2))
print(s2.difference(s1))
```

a)

{1, 2}
{4, 5, 6}
b)
{1, 2}
{1, 2}
c)
{4, 5, 6}
{1, 2}
d)
{4, 5, 6}
{4, 5, 6}

Answer: a
Explanation: The function s1.difference(s2) returns a set containing the elements which are present in the set s1 but not in the set s2. Similarly, the function s2.difference(s1) returns a set containing elements which

are present in the set s2 but not in the set s1. Hence the output of the code shown above will be:

{1, 2}
{4, 5, 6}
.

5. What will be the output of the following Python code?

```
s1={1, 2, 3}
s2={4, 5, 6}
print(s1.isdisjoint(s2))
print(s2.isdisjoint(s1))
```

a)
True
False
b)
False
True
c)
True
True
d)
False
False

Answer: c
Explanation: The isdisjoint() method returns True if two sets have no elements in common. Since s1 = {1, 2, 3} and s2 = {4, 5, 6} share no elements, both s1.isdisjoint(s2) and s2.isdisjoint(s1) return True.

6. If we have two sets, s1 and s2, and we want to check if all the elements of s1 are present in s2 or not, we can use the function:

a) s2.issubset(s1)
b) s2.issuperset(s1)
c) s1.issuperset(s2)
d) s1.isset(s2)

Answer: b
Explanation: To check if all elements of s1 are present in s2, you want to verify if s2 is a superset of s1. The issuperset() method returns True if the set on which it's called contains all elements of the specified set. So, s2.issuperset(s1) checks exactly that.

7. What will be the output of the following Python code?

```
s1={1, 2, 3, 8}
s2={3, 4, 5, 6}
print(s1|s2)
print(s1.union(s2))
```

a)
{3}
{1, 2, 3, 4, 5, 6, 8}
b)
{1, 2, 4, 5, 6, 8}
{1, 2, 4, 5, 6, 8}
c)
{3}
{3}
d)

{1, 2, 3, 4, 5, 6, 8}
{1, 2, 3, 4, 5, 6, 8}

Answer: d
Explanation: The function s1|s2 as well as the function s1.union(s2) returns a union of the two sets s1 and s2. Hence the output of both of these functions is: {1, 2, 3, 4, 5, 6, 8}.

8. What will be the output of the following Python code?

```
a=set('abc')
b=set('def')
b.intersection_update(a)
print(a)
print(b)
```

a)
set()
(‘e’, ‘d’, ‘f’}
b)
{}
{}
c)
{‘b’, ‘c’, ‘a’}
set()
d)
set()
set()

Answer: c
Explanation: The function b.intersection_update(a) puts those elements

in the set b which are common to both the sets a and b. The set a remains as it is. Since there are no common elements between the sets a and b, the output is:
{‘b’, ‘c’, ‘a’}
set()

9. What will be the output of the following Python code?

```
s1= {1, 2, 3}
print(s1.issubset(s1))
```

a) True
b) Error
c) No output
d) False

Answer: a
Explanation: The method issubset() checks if all elements of a set are contained in another set. Since a set is always a subset of itself, s1.issubset(s1) returns True.

10. What will be the output of the following Python code?

```
x=set('abcde')
y=set('xyzbd')
x.difference_update(y)
print(x)
print(y)
```

a)
{‘a’, ‘b’, ‘c’, ‘d’, ‘e’}
{‘x’, ‘y’, ‘z’}
b)
{‘a’, ‘c’, ‘e’}

{‘x’, ‘y’, ‘z’, ‘b’, ‘d’}
c)
{‘b’, ‘d’}
{‘b’, ‘d’}
d)
{‘a’, ‘c’, ‘e’}
{‘x’, ‘y’, ‘z’}

Answer: b
Explanation: The function x.difference_update(y) removes all the elements of the set y from the set x. Hence the output of the code is:
{‘a’, ‘c’, ‘e’}
{‘x’, ‘y’, ‘z’, ‘b’, ‘d’}
.

1. Which of the following statements create a dictionary?

a) d = {}
b) d = {“john”:40, “peter”:45}
c) d = {40:”john”, 45:”peter”}
d) All of the mentioned

Answer: d
Explanation:

- d = {} creates an empty dictionary.
- d = {“john”: 40, “peter”: 45} creates a dictionary with string keys and integer values.
- d = {40: “john”, 45: “peter”} creates a dictionary with integer keys and string values.

All of these are valid ways to create dictionaries in Python.

2. What will be the output of the following Python code snippet?

```
d = {"john":40, "peter":45}
print(d)
```
a) “john”, 40, “peter”, 45
b) {‘john’: 40, ‘peter’: 45}
c) 40 and 45
d) d = (40:”john”, 45:”peter”)

Answer: b
Explanation: Dictionaries in Python are displayed as key-value pairs inside curly braces. The keys are associated with their values using a colon (:). Hence, printing the dictionary shows `{‘john’: 40, ‘peter’: 45}`.

3. What will be the output of the following Python code snippet?

```
d = {"john":40, "peter":45}
print("john" in d)
```
a) True
b) False
c) None
d) Error

Answer: a
Explanation: The “in” operator checks if the specified key (“john”) exists in the dictionary d. Since “john” is a key, it returns True.

4. What will be the output of the following Python code snippet?

```
d1 = {"john":40, "peter":45}
d2 = {"john":466, "peter":45}
print(d1 == d2)
```
a) True
b) False
c) None

d) Error

Answer: b
Explanation: Two dictionaries are equal only if they have the same keys and corresponding values. Since d1 has “john”: 40 and d2 has “john”: 466, the dictionaries are not equal.

5. What will be the output of the following Python code snippet?

```
d1 = {"john":40, "peter":45}
d2 = {"john":466, "peter":45}
print(d1 > d2)
```

a) True
b) False
c) Error
d) None

Answer: c
Explanation: Python does not support comparison operators like > between dictionaries. Trying to do d1 > d2 raises a TypeError.

6. What will be the output of the following Python code snippet?

```
d = {"john":40, "peter":45}
print(d["john"])
```

a) 40
b) 45
c) “john”
d) “peter”

Answer: a
Explanation: Accessing a dictionary with a key like d[“john”] retrieves the value associated with that key. Since “john” maps to 40 in the dictionary, the output is 40.

7. Suppose d = {“john”:40, “peter”:45}, to delete the entry for “john” what command do we use?

a) d.delete(“john”:40)
b) d.delete(“john”)
c) del d[“john”]
d) del d(“john”:40)

Answer: c
Explanation: To delete a key-value pair from a dictionary, you use the `del` statement followed by the dictionary and key inside square brackets. So, `del d[“john”]` removes the entry with key `”john”`.

8. Suppose d = {“john”:40, “peter”:45}. To obtain the number of entries in dictionary which command do we use?

a) d.size()
b) len(d)
c) size(d)
d) d.len()

Answer: b
Explanation: The built-in `len()` function returns the number of key-value pairs in a dictionary. So, `len(d)` returns `2` for this dictionary.

9. What will be the output of the following Python code snippet?

```
d = {"john":40, "peter":45}
print(list(d.keys()))
```

a) [“john”, “peter”]
b) [“john”:40, “peter”:45]
c) (“john”, “peter”)
d) (“john”:40, “peter”:45)

Answer: a
Explanation: `d.keys()` returns a view object containing the keys of the dictionary. Using `list()` converts this view into a list of keys, so the output is `[“john”, “peter”]`.

10. Suppose d = {“john”:40, “peter”:45}, what happens when we try to retrieve a value using the expression d[“susan”]?

a) Since “susan” is not a value in the set, Python raises a KeyError exception
b) It is executed fine and no exception is raised, and it returns None
c) Since “susan” is not a key in the set, Python raises a KeyError exception
d) Since “susan” is not a key in the set, Python raises a syntax error

Answer: c
Explanation: When you try to access a dictionary key that does not exist using d[“susan”], Python raises a KeyError because “susan” is not found in the dictionary keys.

1. Which of these about a dictionary is false?

a) The values of a dictionary can be accessed using keys
b) The keys of a dictionary can be accessed using values
c) Dictionaries aren’t ordered
d) Dictionaries are mutable

Answer: b
Explanation: In Python, you can access values by using their keys (e.g., d[key]), but the reverse is not directly possible—you cannot access keys using values unless you explicitly search through the dictionary. This makes option “The keys of a dictionary can be accessed using values” false.

2. Which of the following is not a declaration of the dictionary?

a) {1: ‘A’, 2: ‘B’}

b) dict([[1,”A”],[2,”B”]])
c) {1,”A”,2”B”}
d) { }

Answer: c
Explanation: {1, “A”, 2, “B”} is not a valid dictionary declaration—it resembles a set with multiple elements, not key-value pairs. A dictionary must contain key-value pairs separated by colons. The other options are valid ways to define dictionaries.

3. What will be the output of the following Python code snippet?

```
a={1:"A",2:"B",3:"C"}
for i,j in a.items():
    print(i,j,end=" ")
```

a) 1 A 2 B 3 C
b) 1 2 3
c) A B C
d) 1:”A” 2:”B” 3:”C”

Answer: a
Explanation: The items() method returns key-value pairs from the dictionary. The for loop unpacks each pair into i (key) and j (value), and prints them with a space between, all on the same line due to end=” “.

4. What will be the output of the following Python code snippet?

```
a={1:"A",2:"B",3:"C"}
print(a.get(1,4))
```

a) 1
b) A
c) 4

d) Invalid syntax for get method

Answer: b
Explanation: The get() method is used to retrieve the value for a given key. In a.get(1, 4), the key 1 exists in the dictionary, so it returns its value, which is “A”. The second argument 4 is ignored because the key is found.

5. What will be the output of the following Python code snippet?

```
a={1:"A",2:"B",3:"C"}
print(a.get(5,4))
```

a) Error, invalid syntax
b) A
c) 5
d) 4

Answer: d
Explanation: The get() method returns the value for the specified key if it exists; otherwise, it returns the default value provided. In this case, key 5 is not in the dictionary, so a.get(5, 4) returns the default value 4.

6. What will be the output of the following Python code snippet?

```
a={1:"A",2:"B",3:"C"}
print(a.setdefault(3))
```

a) {1: ‘A’, 2: ‘B’, 3: ‘C’}
b) C
c) {1: 3, 2: 3, 3: 3}
d) No method called setdefault() exists for dictionary

Answer: b
Explanation: The setdefault() method returns the value of the given key if it exists. If the key doesn’t exist, it adds the key with the specified default

value. In this case, key 3 already exists in the dictionary with the value “C”, so a.setdefault(3) simply returns “C”.

7. What will be the output of the following Python code snippet?

```
a={1:"A",2:"B",3:"C"}
a.setdefault(4,"D")
print(a)
```

a) {1: ‘A’, 2: ‘B’, 3: ‘C’, 4: ‘D’}
b) None
c) Error
d) [1,3,6,10]

Answer: a
Explanation: The setdefault() method adds a key with a specified default value if the key is not already in the dictionary. Here, key 4 is not present in a, so a.setdefault(4, “D”) adds the key-value pair 4: “D” to the dictionary. The updated dictionary is {1: ‘A’, 2: ‘B’, 3: ‘C’, 4: ‘D’}.

8. What will be the output of the following Python code?

```
a={1:"A",2:"B",3:"C"}
b={4:"D",5:"E"}
a.update(b)
print(a)
```

a) {1: ‘A’, 2: ‘B’, 3: ‘C’}
b) Method update() doesn’t exist for dictionaries
c) {1: ‘A’, 2: ‘B’, 3: ‘C’, 4: ‘D’, 5: ‘E’}
d) {4: ‘D’, 5: ‘E’}

Answer: c
Explanation: The update() method merges one dictionary into another by adding key-value pairs. Here, a.update(b) adds all items from dictionary b into a, resulting in: {1: ‘A’, 2: ‘B’, 3: ‘C’, 4: ‘D’, 5: ‘E’}.

9. What will be the output of the following Python code?

```
a={1:"A",2:"B",3:"C"}
b=a.copy()
b[2]="D"
print(a)
```

a) Error, copy() method doesn't exist for dictionaries
b) {1: 'A', 2: 'B', 3: 'C'}
c) {1: 'A', 2: 'D', 3: 'C'}
d) "None" is printed

Answer: b
Explanation: The copy() method creates a shallow copy of the dictionary. When b[2] is changed to "D", it only affects the copy b and not the original dictionary a. So, printing a shows the original unchanged dictionary {1: 'A', 2: 'B', 3: 'C'}.

10. What will be the output of the following Python code?

```
a={1:"A",2:"B",3:"C"}
a.clear()
print(a)
```

a) None
b) { None:None, None:None, None:None}
c) {1:None, 2:None, 3:None}
d) { }

Answer: d
Explanation: The clear() method empties the dictionary by removing all key-value pairs. After calling a.clear(), the dictionary a becomes empty, so printing it shows {}.

11. Which of the following isn't true about dictionary keys?

a) More than one key isn't allowed

b) Keys must be immutable
c) Keys must be integers
d) When duplicate keys encountered, the last assignment wins

Answer: c
Explanation: Dictionary keys in Python must be immutable but can be of various types like strings, numbers, or tuples — they do not have to be integers specifically. So, saying “Keys must be integers” is false. The other statements are true: duplicate keys overwrite previous values, keys must be immutable, and keys must be unique.

12. What will be the output of the following Python code?

```
a={1:5,2:3,3:4}
a.pop(3)
print(a)
```

a) {1: 5}
b) {1: 5, 2: 3}
c) Error, syntax error for pop() method
d) {1: 5, 3: 4}

Answer: b
Explanation: The pop(3) method removes the key 3 and its associated value from the dictionary a. After removal, the dictionary contains only the pairs for keys 1 and 2. So, the output is {1: 5, 2: 3}.

13. What will be the output of the following Python code?

```
a={1:5,2:3,3:4}
print(a.pop(4,9))
```

a) 9
b) 3
c) Too many arguments for pop() method

d) 4

Answer: a
Explanation: The pop() method removes the specified key and returns its value. If the key doesn't exist, it returns the default value provided as the second argument. Here, key 4 is not in the dictionary, so a.pop(4, 9) returns 9.

14. What will be the output of the following Python code?

```
a={1:"A",2:"B",3:"C"}
for i in a:
    print(i,end=" ")
```

a) 1 2 3
b) 'A' 'B' 'C'
c) 1 'A' 2 'B' 3 'C'
d) Error, it should be: for i in a.items():

Answer: a
Explanation: When you iterate over a dictionary using `for i in a`, it iterates over the **keys** by default. So the output is the keys: `1 2 3`.

15. What will be the output of the following Python code?

```
a={1:"A",2:"B",3:"C"}
print(a.items())
```

a) Syntax error
b) dict_items([('A'), ('B'), ('C')])
c) dict_items([(1,2,3)])
d) dict_items([(1, 'A'), (2, 'B'), (3, 'C')])

Answer: d
Explanation: The `a.items()` method returns a view object containing the dictionary's key-value pairs as tuples. So the output is `dict_items([(1, 'A'), (2, 'B'), (3, 'C')])`.

1. Which of the statements about dictionary values if false?

a) More than one key can have the same value
b) The values of the dictionary can be accessed as dict[key]
c) Values of a dictionary must be unique
d) Values of a dictionary can be a mixture of letters and numbers

Answer: c
Explanation: Dictionary values do not have to be unique — multiple keys can have the same value. Only keys must be unique. So, the statement "Values of a dictionary must be unique" is false.

2. What will be the output of the following Python code snippet?

```
a = {1: "A", 2: "B", 3: "C"}
del a
print(a)
```

a) method del doesn't exist for the dictionary
b) del deletes the values in the dictionary
c) del deletes the entire dictionary
d) del deletes the keys in the dictionary

Answer: c
Explanation: The del statement deletes the entire dictionary object from memory. After del a, the variable a no longer exists. Attempting to access it afterward would raise a NameError.

3. If a is a dictionary with some key-value pairs, what does a.popitem() do?

a) Removes an arbitrary element
b) Removes all the key-value pairs
c) Removes the key-value pair for the key given as an argument
d) Invalid method for dictionary

Answer: a
Explanation: The a.popitem() method removes and returns an arbitrary key-value pair from the dictionary. In Python 3.7 and later, it removes the last inserted item due to insertion-order preservation.

4. What will be the output of the following Python code snippet?

```
total={}
def insert(items):
   if items in total:
      total[items] += 1
   else:
      total[items] = 1
insert('Apple')
insert('Ball')
insert('Apple')
print (len(total))
```

a) 3
b) 1
c) 2
d) 0

Answer: c
Explanation:The insert() function keeps track of how many times an item is inserted using a dictionary named total.

- ‘Apple’ is inserted first → total = {‘Apple’: 1}
- ‘Ball’ is inserted → total = {‘Apple’: 1, ‘Ball’: 1}

- ‘Apple’ is inserted again → count updated → total = {‘Apple’: 2, ‘Ball’: 1}

The length of the dictionary (len(total)) is 2, since there are two unique keys: ‘Apple’ and ‘Ball’.

5. What will be the output of the following Python code snippet?

```
a = {}
a[1] = 1
a['1'] = 2
a[1]=a[1]+1
count = 0
for i in a:
   count += a[i]
print(count)
```

a) 1
b) 2
c) 4
d) Error, the keys can’t be a mixture of letters and numbers

Answer: c
Explanation: In the given code, a dictionary is created with both an integer key 1 and a string key ‘1’, which are treated as separate keys in Python. The values for both keys are updated and then summed using a loop. The final output is 4, since both keys have the value 2.

6. What will be the output of the following Python code snippet?

```
numbers = {}
letters = {}
comb = {}
numbers[1] = 56
numbers[3] = 7
letters[4] = 'B'
```

```
comb['Numbers'] = numbers
comb['Letters'] = letters
print(comb)
```

a) Error, dictionary in a dictionary can’t exist
b) ‘Numbers’: {1: 56, 3: 7}
c) {‘Numbers’: {1: 56}, ‘Letters’: {4: ‘B’}}
d) {‘Numbers’: {1: 56, 3: 7}, ‘Letters’: {4: ‘B’}}

Answer: d
Explanation: In this code, two dictionaries numbers and letters are created and then assigned as values to keys ‘Numbers’ and ‘Letters’ in another dictionary comb. Python supports nested dictionaries, so the output will be: {‘Numbers’: {1: 56, 3: 7}, ‘Letters’: {4: ‘B’}}.

7. What will be the output of the following Python code snippet?

```
test = {1:'A', 2:'B', 3:'C'}
test = {}
print(len(test))
```

a) 0
b) None
c) 3
d) An exception is thrown

Answer: a
Explanation: The dictionary test is first initialized with three key-value pairs but is then reassigned to an empty dictionary using test = {}. So, len(test) returns 0.

8. What will be the output of the following Python code snippet?

```
test = {1:'A', 2:'B', 3:'C'}
del test[1]
test[1] = 'D'
```

```
del test[2]
print(len(test))
```
a) 0
b) 2
c) Error as the key-value pair of 1:’A’ is already deleted
d) 1

Answer: b
Explanation: The dictionary starts with 3 items. Key 1 is deleted and then re-added with a new value. Key 2 is deleted, leaving only two keys: 1 and 3. So, len(test) returns 2.

9. What will be the output of the following Python code snippet?

```
a = {}
a[1] = 1
a['1'] = 2
a[1.0]=4
count = 0
for i in a:
   count += a[i]
print(count)
```
a) An exception is thrown
b) 3
c) 6
d) 2

Answer: c
Explanation: In Python, 1 (int) and 1.0 (float) are considered the same dictionary key. So when a[1.0] = 4 is assigned, it overwrites a[1] = 1. The dictionary becomes: {1: 4, ‘1’: 2}
The loop adds 4 + 2, so the final output is 6.

10. What will be the output of the following Python code snippet?

```
a={}
a['a']=1
a['b']=[2,3,4]
print(a)
```

a) Exception is thrown
b) {‘b’: [2], ‘a’: 1}
c) {‘b’: [2], ‘a’: [3]}
d) {‘b’: [2, 3, 4], ‘a’: 1}

Answer: d
Explanation: The dictionary a is assigned two key-value pairs: ‘a’ maps to 1 and ‘b’ maps to the list [2, 3, 4]. Printing a outputs the dictionary with both these entries exactly as assigned, so the output is {‘b’: [2, 3, 4], ‘a’: 1}.

11. What will be the output of the following Python code snippet?

```
import collections
a=collections.Counter([1,1,2,3,3,4,4,4])
print(a)
```

a) {1,2,3,4}
b) Counter({4, 1, 3, 2})
c) Counter({4: 3, 1: 2, 3: 2, 2: 1})
d) {4: 3, 1: 2, 3: 2, 2: 1}

Answer: c
Explanation: The code uses collections.Counter to count the occurrences of each element in the list [1,1,2,3,3,4,4,4]. It returns a Counter object showing how many times each number appears: 4 appears 3 times, 1 and 3 appear twice each, and 2 appears once. Therefore, the output is Counter({4: 3, 1: 2, 3: 2, 2: 1}).

12. What will be the output of the following Python code snippet?

```
import collections
b=collections.Counter([2,2,3,4,4,4])
print(b.most_common(1))
```

a) Counter({4: 3, 2: 2, 3: 1})
b) {3:1}
c) {4:3}
d) [(4, 3)]

Answer: d
Explanation: The code uses collections.Counter to count the occurrences of each element in the list [2,2,3,4,4,4]. The most_common(1) method returns a list with the single most common element and its count as a tuple. Since 4 appears 3 times (the most), the output is [(4, 3)].

13. What will be the output of the following Python code snippet?

```
import collections
b=collections.Counter([2,2,3,4,4,4])
print(b.most_common(0))
```

a) Counter({4: 3, 2: 2, 3: 1})
b) {3:1}
c) {4:3}
d) []

Answer: d
Explanation: The most_common(0) method returns an empty list because it asks for zero most common elements. So, the output is [].

14. What will be the output of the following Python code snippet?

```
import collections
a=collections.Counter([2,2,3,3,3,4])
b=collections.Counter([2,2,3,4,4])
```

```
print(a|b)
```
a) Counter({3: 3, 2: 2, 4: 2})
b) Counter({2: 2, 3: 1, 4: 1})
c) Counter({3: 2})
d) Counter({4: 1})

Answer: a
Explanation: The | operator on Counter objects returns the element-wise maximum of counts from both counters. For each key, it takes the higher count between the two counters.

- For 2, counts are 2 in both a and b, so max is 2.
- For 3, counts are 3 in a and 1 in b, so max is 3.
- For 4, counts are 1 in a and 2 in b, so max is 2.

Hence, the result is Counter({3: 3, 2: 2, 4: 2}).

15. What will be the output of the following Python code snippet?

```
import collections
a=collections.Counter([3,3,4,5])
b=collections.Counter([3,4,4,5,5,5])
print(a&b)
```
a) Counter({3: 12, 4: 1, 5: 1})
b) Counter({3: 1, 4: 1, 5: 1})
c) Counter({4: 2})
d) Counter({5: 1})

Answer: b
Explanation: The & operator on Counter objects returns the intersection, which is the minimum count for each element present in both counters.

- For 3, counts are 2 in a and 1 in b, so min is 1.
- For 4, counts are 1 in a and 2 in b, so min is 1.
- For 5, counts are 1 in a and 3 in b, so min is 1.

So the result is Counter({3: 1, 4: 1, 5: 1}).

1. The following Python code is invalid.

```
class demo(dict):
  def __test__(self,key):
    return []
a = demo()
a['test'] = 7
print(a)
```

a) True
b) False

Answer: b
Explanation: The code runs successfully and prints {‘test’: 7}, so it is valid Python code. It defines a class inheriting from dict, adds an unused method __test__, sets a key ‘test’, and prints the dictionary. There is no syntax or runtime error. Therefore, the code is not invalid, and the correct answer is False.

2. What will be the output of the following Python code?

```
count={}
count[(1,2,4)] = 5
count[(4,2,1)] = 7
count[(1,2)] = 6
count[(4,2,1)] = 2
tot = 0
for i in count:
   tot=tot+count[i]
print(len(count)+tot)
```

a) 25
b) 17

c) 16
d) Tuples can’t be made keys of a dictionary

Answer: c
Explanation: The dictionary uses tuples as keys, which is valid since tuples are immutable. The key (4,2,1) is assigned twice, so the second value (2) overwrites the first (7). The final dictionary has 3 keys, and the sum of the values is 5 + 2 + 6 = 13. Thus, the output is 3 + 13 = 16, so the correct answer is 16.

3. What will be the output of the following Python code?

```
a={}
a[2]=1
a[1]=[2,3,4]
print(a[1][1])
```

a) [2,3,4]
b) 3
c) 2
d) An exception is thrown

Answer: b
Explanation: The dictionary a stores an integer under key 2 and a list under key 1. The expression a[1][1] first retrieves the list [2, 3, 4], then accesses its second element, which is 3. Hence, the output is 3.

4. What will be the output of the following Python code?

```
a={'B':5,'A':9,'C':7}
print(sorted(a))
```

a) [‘A’, ’B’, ’C’]
b) [‘B’, ’C’, ’A’]
c) [5, 7, 9]

d) [9, 5, 7]

Answer: a
Explanation: The sorted() function sorts the dictionary keys in ascending (alphabetical) order. Since the keys are ‘B’, ‘A’, and ‘C’, sorting them gives [‘A’, ‘B’, ‘C’].

5. What will be the output of the following Python code?

```
a={i: i*i for i in range(6)}
print(a)
```

a) Dictionary comprehension doesn’t exist
b) {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6:36}
c) {0: 0, 1: 1, 4: 4, 9: 9, 16: 16, 25: 25}
d) {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

Answer: d
Explanation: This is a dictionary comprehension that generates a dictionary where each key i is mapped to its square i*i for i in range(6), i.e., 0 to 5. So the output is: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

6. What will be the output of the following Python code?

```
a={}
print(a.fromkeys([1,2,3],"check"))
```

a) Syntax error
b) {1: ‘check’, 2: ‘check’, 3: ‘check’}
c) “check”
d) {1:None,2:None,3:None}

Answer: b
Explanation: The fromkeys() method creates a new dictionary with the specified keys and assigns each key the same value. Here, keys [1, 2, 3]

are all given the value “check”. So the output is {1: ‘check’, 2: ‘check’, 3: ‘check’}.

7. What will be the output of the following Python code?

```
b={}
print(all(b))
```

a) { }
b) False
c) True
d) An exception is thrown

Answer: c
Explanation: The all() function returns True when called on an empty iterable, as there are no false elements to contradict it. Since b is an empty dictionary, all(b) returns True. Hence, the output is True.

8. If b is a dictionary, what does any(b) do?

a) Returns True if any key of the dictionary is true
b) Returns False if any key of the dictionary is false
c) Returns True if all keys of the dictionary are true
d) Method any() doesn’t exist for dictionary

Answer: a
Explanation: The any() function returns True if any element of the iterable is True. When used on a dictionary, it checks the keys (not the values). So any(b) returns True if at least one key is truthy (e.g., not 0, not False, not empty string).
Example:

```
b = {0: 'x', 1: 'y'}
any(b) → True  (because key 1 is truthy)
```

9. What will be the output of the following Python code?

```
a={"a":1,"b":2,"c":3}
b=dict(zip(a.values(),a.keys()))
print(b)
```

a) {‘a’: 1, ‘b’: 2, ‘c’: 3}
b) An exception is thrown
c) {‘a’: ‘b’: ‘c’: }
d) {1: ‘a’, 2: ‘b’, 3: ‘c’}

Answer: d
Explanation: The zip(a.values(), a.keys()) pairs each value with its corresponding key, effectively reversing the dictionary’s key-value order. Using dict() on this zipped object creates a new dictionary with values as keys and keys as values. So, the output is {1: ‘a’, 2: ‘b’, 3: ‘c’}.

10. What will be the output of the following Python code?

```
a={i: 'A' + str(i) for i in range(5)}
print(a)
```

a) An exception is thrown
b) {0: ‘A0’, 1: ‘A1’, 2: ‘A2’, 3: ‘A3’, 4: ‘A4’}
c) {0: ‘A’, 1: ‘A’, 2: ‘A’, 3: ‘A’, 4: ‘A’}
d) {0: ‘0’, 1: ‘1’, 2: ‘2’, 3: ‘3’, 4: ‘4’}

Answer: b
Explanation: The dictionary comprehension creates keys from 0 to 4 and assigns each key a value formed by concatenating ‘A’ with the string of the key. This results in {0: ‘A0’, 1: ‘A1’, 2: ‘A2’, 3: ‘A3’, 4: ‘A4’}.

11. What will be the output of the following Python code?

```
a=dict()
```

```
print(a[1])
```
a) An exception is thrown since the dictionary is empty
b) ‘ ‘
c) 1
d) 0

Answer: a
Explanation: The dictionary a is empty and does not contain the key 1. Attempting to access a non-existent key using square brackets (a[1]) raises a KeyError in Python. To avoid this, the get() method can be used instead, which returns None or a default value if the key is missing.

12. What will be the output of the following Python code?

```
import collections
a=dict()
a=collections.defaultdict(int)
print(a[1])
```
a) 1
b) 0
c) An exception is thrown
d) ‘ ‘

Answer: b
Explanation: The code uses collections.defaultdict with int as the default factory function. When a[1] is accessed and the key 1 doesn’t exist, defaultdict automatically creates it with a default value of int(), which is 0. So the output is 0, and no exception is raised.

13. What will be the output of the following Python code?

```
import collections
a=dict()
a=collections.defaultdict(str)
```

```
print(a['A'])
```
a) An exception is thrown since the dictionary is empty
b) ‘ ‘
c) ‘A’
d) 0

Answer: b
Explanation: The collections.defaultdict(str) creates a dictionary that returns a default value when a missing key is accessed. Since the default factory is str, which returns an empty string (”), accessing a missing key like ‘A’ doesn’t raise an error—it simply returns ”. Therefore, the output is an empty string.

14. What will be the output of the following Python code?

```
import collections
b=dict()
b=collections.defaultdict(lambda: 7)
print(b[4])
```
a) 4
b) 0
c) An exception is thrown
d) 7

Answer: d
Explanation: The defaultdict is initialized with a lambda function that returns 7. When the key 4 is accessed and not found, the lambda function is called to provide a default value. So, b[4] returns 7 without raising an error.

15. What will be the output of the following Python code?

```
import collections
a=collections.OrderedDict((str(x),x) for x in range(3))
```

```
print(a)
```
a) {‘2’:2, ‘0’:0, ‘1’:1}
b) OrderedDict([(‘0’, 0), (‘1’, 1), (‘2’, 2)])
c) An exception is thrown
d) ‘ ‘

Answer: b
Explanation: The OrderedDict stores items in the order they are added. The expression creates key-value pairs from 0 to 2 with keys as strings. So, the output maintains insertion order and prints: OrderedDict([(‘0’, 0), (‘1’, 1), (‘2’, 2)]).

1. Which of the following functions is a built-in function in python?
a) seed()
b) sqrt()
c) factorial()
d) print()

Answer: d
Explanation: The function seed is a function which is present in the random module. The functions sqrt and factorial are a part of the math module. The print function is a built-in function which prints a value directly to the system output.

2. What will be the output of the following Python expression?
```
print(round(4.576))
```
a) 4.5
b) 5
c) 4
d) 4.6

Answer: b
Explanation: The round() function rounds the number to the nearest integer by default. Since 4.576 is closer to 5 than 4, round(4.576) returns 5. Therefore, the output is 5.

3. The function pow(x,y,z) is evaluated as:
a) (x**y)**z
b) (x**y) / z
c) (x**y) % z
d) (x**y)*z

Answer: c
Explanation: The built-in pow() function with three arguments computes (x ** y) % z. With two arguments, it simply calculates x ** y. So, for three arguments, the result is the modular exponentiation (x**y) % z. Therefore, the correct answer is (x**y) % z.
4. What will be the output of the following Python function?

```
print(all([2,4,0,6]))
```

a) Error
b) True
c) False
d) 0

Answer: c
Explanation: The all() function returns True only if all elements in the iterable are truthy. In the list [2, 4, 0, 6], the value 0 is considered falsy in Python. Since not all elements are truthy, all() returns False.
5. What will be the output of the following Python expression?

```
print(round(4.5676,2))
```

a) 4.5
b) 4.6
c) 4.57
d) 4.56

Answer: c
Explanation: The round() function rounds a number to the specified number of decimal places. In round(4.5676, 2), the number is rounded to two decimal places, resulting in 4.57. So, the correct output is 4.57.

6. What will be the output of the following Python function?

```
print(any([2>8, 4>2, 1>2]))
```

a) Error
b) True
c) False
d) 4>2

Answer: b
Explanation: The any() function returns True if at least one element in the iterable is truthy. In the list [2>8, 4>2, 1>2], only 4>2 is True, while the others are False. Since there's at least one True value, any() returns True.

7. What will be the output of the following Python function?

```
import math
abs(math.sqrt(25))
```

a) Error
b) -5
c) 5
d) 5.0

Answer: d
Explanation: The expression math.sqrt(25) returns 5.0, the square root of

25 as a float. The abs() function returns the absolute value, which doesn't change anything here since 5.0 is already positive. So the final output is 5.0.

8. What will be the output of the following Python function?

```
print(sum([1,2,3]))
print(sum(2,4,6))
```

a) 6, Error
b) 12, Error
c) 12, 6
d) Error, Error

Answer: a
Explanation: The first sum() call works correctly because it sums the elements of the list [1, 2, 3] resulting in 6. The second call causes an error because sum() expects an iterable as the first argument, but 2 is an integer, not an iterable, and the function doesn't accept three separate positional arguments. This leads to a TypeError.

9. What will be the output of the following Python function?

```
print(all(3,0,4.2))
```

a) True
b) False
c) Error
d) 0

Answer: c
Explanation: The all() function expects a single iterable (like a list or tuple), not multiple separate arguments. Calling all(3, 0, 4.2) raises a TypeError because you're passing multiple non-iterable arguments.

10. What will be the output of the following Python function?

```
print(min(max(False,-3,-4), 2,7))
```

a) 2
b) False
c) -3
d) -4

Answer: b
Explanation: The max(False, -3, -4) evaluates to 0 because False is treated as 0, and 0 is greater than -3 and -4. Then, min(0, 2, 7) returns 0. Since 0 is equivalent to False, the output is False.

1. What will be the output of the following Python functions?

```
print(chr(97))
print(chr('97'))
```

a)
  a
  Error

b)
  'a'
   a

c)
  Error
  a

d)
  Error
  Error

Answer: a
Explanation: The chr() function expects an integer representing a Unicode code point. chr(97) correctly returns 'a' since 97 is the Unicode code for

‘a’. However, chr(’97’) raises a TypeError because the argument ’97’ is a string, not an integer.

2. What will be the output of the following Python function?

```
print(complex(1+2j))
```

a) Error
b) 1
c) 2j
d) 1+2j

Answer: d
Explanation: The complex() function converts the argument into a complex number. Since 1+2j is already a complex number, complex(1+2j) simply returns 1+2j without any changes.

3. What is the output of the function complex()?

a) 0j
b) 0+0j
c) 0
d) Error

Answer: a
Explanation: When called without any arguments, the complex() function returns a complex number with both real and imaginary parts as zero, which is represented as 0j in Python. Note that 0j is the canonical way Python shows a zero-valued complex number with an imaginary part.

4. The function divmod(a,b), where both ‘a’ and ‘b’ are integers is evaluated as:

a) (a%b, a//b)
b) (a//b, a%b)
c) (a//b, a*b)
d) (a/b, a%b)

Answer: b
Explanation: The divmod(a, b) function returns a tuple containing the quotient and the remainder when a is divided by b. Specifically, it evaluates as (a // b, a % b). The first element is integer division, and the second is the modulus.

5. What will be the output of the following Python function?

```
print(divmod(10.5,5))
print(divmod(2.4,1.2))
```

a)
(2.00, 0.50)
(2.00, 0.00)
b)
(2, 0.5)
(2, 0)
c)
(2.0, 0.5)
(2.0, 0.0)
d)
(2, 0.5)
(2)

Answer: c
Explanation: When divmod() is used with float arguments, it returns a tuple of two floats: the quotient (using floor division) and the remainder. For divmod(10.5, 5), 10.5 // 5 is 2.0, and the remainder is 0.5. Similarly,

divmod(2.4, 1.2) gives 2.0 and 0.0. Therefore, the output is (2.0, 0.5) and (2.0, 0.0).

6. The function complex('2-3j') is valid but the function complex('2 – 3j') is invalid.

a) True
b) False

Answer: a
Explanation: The function complex('2-3j') is valid because the string is in the correct format for Python to interpret as a complex number. However, complex('2 – 3j') is invalid due to the presence of an en dash (–, Unicode U+2013) instead of a regular minus sign (-, ASCII 0x2D). Python does not recognize the en dash as a valid operator in numeric expressions, leading to a ValueError.

7. What will be the output of the following Python function?

```
print(list(enumerate([2, 3])))
```

a) Error
b) [(1, 2), (2, 3)]
c) [(0, 2), (1, 3)]
d) [(2, 3)]

Answer: c
Explanation: The enumerate() function in Python adds a counter to an iterable and returns it as an enumerate object. When converted to a list, each item is a tuple of the form (index, value). So, list(enumerate([2, 3])) results in [(0, 2), (1, 3)], where 0 and 1 are the indices of the elements 2 and 3 respectively.

8. What will be the output of the following Python functions?

```
x=3
print(eval('x^2'))
```

a) Error
b) 1
c) 9
d) 6

Answer: b
Explanation: The expression eval('x^2') evaluates the string 'x^2' as Python code. In Python, the ^ operator is a bitwise XOR, not exponentiation.
Given x = 3, the expression 3 ^ 2 (i.e., 3 XOR 2 in binary) equals 1, because:

- 3 in binary is 011
- 2 in binary is 010

XOR of 011 and 010 is 001, which is 1 in decimal. Hence, the output is 1.

9. What will be the output of the following Python functions?

```
print(float('1e-003'))
print(float('2e+003'))
```

a)
3.00
300
b)
0.001
2000.0
c)
0.001
200
d)
Error

2003

Answer: b
Explanation: The float() function in Python can parse strings representing numbers in scientific notation.

- ‘1e-003’ means $1\times10^{-3} = 0.001$.
- ‘2e+003’ means $2\times10^{3} = 2000.0$

So the output will be:
0.001
2000.0

10. Which of the following functions does not necessarily accept only iterables as arguments?
a) enumerate()
b) all()
c) chr()
d) max()

Answer: c
Explanation: The chr() function takes a single integer (representing a Unicode code point) as its argument, not an iterable. In contrast, enumerate(), all(), and max() typically operate on iterables like lists, tuples, or strings. Therefore, chr() is the one that does not necessarily accept an iterable as an argument.
1. Which of the following functions accepts only integers as arguments?
a) ord()
b) min()
c) chr()

d) any()

Answer: c
Explanation: The chr() function accepts only integers as arguments and returns the corresponding Unicode character. For example, chr(97) returns ‘a’. In contrast, ord() takes a string of a single Unicode character, min() works with iterables, and any() also takes an iterable.

2. Suppose there is a list such that: l=[2,3,4]. If we want to print this list in reverse order, which of the following methods should be used?

a) reverse(l)
b) list(reverse[(l)])
c) reversed(l)
d) list(reversed(l))

Answer: d
Explanation: The correct way to reverse and print a list in Python is to use list(reversed(l)). The reversed() function returns an iterator that yields the elements of the list in reverse order. Wrapping it with list() converts the iterator into a list. So, for l = [2, 3, 4], list(reversed(l)) will output [4, 3, 2].

3. What will be the output of the following Python function?

```
print(float('   -12345\n'))
```

(Note that the number of blank spaces before the number is 5)
a)   -12345.0 (5 blank spaces before the number)
b) -12345.0
c) Error
d) -12345.000000000…. (infinite decimal places)

Answer: b
Explanation: The float() function in Python automatically strips leading

and trailing whitespace (including spaces and newline characters) from the input string before converting it to a floating-point number. So, float(‘ -12345\n’) correctly returns -12345.0 without raising an error or preserving whitespace.

4. What will be the output of the following Python function?

```
print(ord(‘A’))
print(ord(65))
```

a)
A
65
b)
65
Error
c)
A
Error
d)
Error
Error

Answer: b
Explanation: Here,

- ord(‘A’) returns the ASCII (Unicode) integer value of the character ‘A’, which is 65.
- ord(65) will cause a TypeError because ord() expects a single character string, not an integer.

So the output will be: 65 Error

5. What will be the output of the following Python function?

```
print(float('-infinity'))
print(float('inf'))
```

a)
–inf
inf
b)
–infinity
inf
c)
Error
Error
d)
Error
Junk value

Answer: a
Explanation: The float() function in Python can convert special string representations of floating-point numbers such as ‘inf’ for positive infinity and ‘-infinity’ (or ‘-inf’) for negative infinity. Thus, float(‘-infinity’) returns negative infinity (-inf) and float(‘inf’) returns positive infinity (inf).

6. Which of the following functions will not result in an error when no arguments are passed to it?
a) min()
b) divmod()
c) all()
d) float()

Answer: d
Explanation: The built-in functions min(), max(), divmod(), ord(), any(), all() etc throw an error when no arguments are passed to them. However there are some built-in functions like float(), complex() etc which do not throw an error when no arguments are passed to them. The output of float() is 0.0.

7. What will be the output of the following Python function?

```
print(hex(15))
```

a) f
b) 0xF
c) 0Xf
d) 0xf

Answer: d
Explanation: The hex() function converts an integer to its hexadecimal string representation using lowercase letters and includes the 0x prefix. So, hex(15) returns '0xf'.

8. Which of the following functions does not throw an error?

a) ord()
b) ord(' ')
c) ord(")
d) ord("")

Answer: b
Explanation: The ord() function in Python returns the Unicode code point for a given single character. When you pass a blank space ' ' to ord(), it returns 32, which is the ASCII value for a space character, so it does not throw an error. However, calling ord() without any argument or passing an empty string "" causes an error because ord() requires exactly one

character as input. Therefore, only ord(‘ ‘) among the given options does not raise an error.

9. What will be the output of the following Python function?

```
print(len(["hello",2, 4, 6]))
```

a) 4
b) 3
c) Error
d) 6

Answer: a
Explanation: The len() function returns the number of elements in the list, regardless of their types. In this case, the list [“hello”, 2, 4, 6] contains four elements, so len() returns 4.

10. What will be the output of the following Python function?

```
print(oct(7))
print(oct('7'))
```

a)
Error
07
b)
0o7
7
c)
0o7
Error
d)
07

0o7

Answer: c
Explanation: The oct() function in Python converts an integer into its octal string representation. When you pass the integer 7 to oct(7), it returns the string ‘0o7’, which is the octal equivalent of the decimal number 7. However, if you try to pass a string like ‘7’ to the oct() function, it will result in a TypeError because oct() only accepts integers as valid input. Thus, the first call outputs 0o7, while the second call raises an error.

1. Which of the following is the use of function in python?

a) Functions are reusable pieces of programs
b) Functions don’t provide better modularity for your application
c) you can’t also create your own functions
d) All of the mentioned

Answer: a
Explanation: Functions in Python are reusable blocks of code that perform a specific task. They improve modularity, make code easier to read and maintain, and help avoid repetition. You can also define your own functions using the def keyword.

2. Which keyword is used for function?

a) Fun
b) Define
c) def
d) Function

Answer: c
Explanation: The def keyword is used in Python to define a function. It is followed by the function name and parentheses that may include parameters. For example: def my_function():.

3. What will be the output of the following Python code?

```
def sayHello():
    print('Hello World!')
sayHello()
sayHello()
```

a)
Hello World!
Hello World!
b)
'Hello World!'
'Hello World!'
c)
Hello
Hello
d) None of the mentioned

Answer: a
Explanation: The function sayHello() prints “Hello World!” each time it’s called. Since it is called twice, the output is printed twice on separate lines. Functions help avoid code repetition.

4. What will be the output of the following Python code?

```
def printMax(a, b):
    if a > b:
        print(a, 'is maximum')
    elif a == b:
        print(a, 'is equal to', b)
    else:
        print(b, 'is maximum')
printMax(3, 4)
```

a) 3
b) 4
c) 4 is maximum
d) None of the mentioned

Answer: c
Explanation: The function printMax(a, b) compares two numbers and prints which one is greater or if they are equal. In the call printMax(3, 4), since 4 is greater than 3, the else block is executed and it prints 4 is maximum.

5. What will be the output of the following Python code?

```
x = 50
def func(x):
    print('x is', x)
    x = 2
    print('Changed local x to', x)
func(x)
print('x is now', x)
```

a)
x is 50
Changed local x to 2
x is now 50
b)
x is 50
Changed local x to 2
x is now 2
c)
x is 50
Changed local x to 2
x is now 100

d) None of the mentioned

Answer: a
Explanation: The function func(x) takes a parameter x and prints its value. Inside the function, the value of x is reassigned to 2, which is a local change and does not affect the global variable x. So, when func(x) is called, it prints x is 50 followed by Changed local x to 2. Outside the function, the global x is still 50, so print('x is now', x) outputs x is now 50.

6. What will be the output of the following Python code?

```
x = 50
def func():
   global x
   print('x is', x)
   x = 2
   print('Changed global x to', x)
func()
print('Value of x is', x)
```

a)
x is 50
Changed global x to 2
Value of x is 50
b)
x is 50
Changed global x to 2
Value of x is 2
c)
x is 50
Changed global x to 50
Value of x is 50

d) None of the mentioned

Answer: b
Explanation: The global keyword is used to indicate that x inside the function refers to the global variable. So, when x = 2 is executed inside func(), it modifies the global x. Hence, after calling the function, the global variable x becomes 2.

7. What will be the output of the following Python code?

```
def say(message, times = 1):
   print(message * times)
say('Hello')
say('World', 5)
```

a)
Hello
WorldWorldWorldWorldWorld
b)
Hello
World 5
c)
Hello
World,World,World,World,World
d)
Hello
HelloHelloHelloHelloHello

Answer: a
Explanation: The function say() takes two parameters: message & times, with times defaulting to 1.

- say('Hello') prints "Hello" once because times defaults to 1.

- say('World', 5) prints "World" five times, concatenated together (i.e., WorldWorldWorldWorldWorld).

This demonstrates Python's default arguments feature, allowing functions to have optional parameters.

8. What will be the output of the following Python code?

```
def func(a, b=5, c=10):
   print('a is', a, 'and b is', b, 'and c is', c)

func(3, 7)
func(25, c = 24)
func(c = 50, a = 100)
```

a)
a is 7 and b is 3 and c is 10
a is 25 and b is 5 and c is 24
a is 5 and b is 100 and c is 50
b)
a is 3 and b is 7 and c is 10
a is 5 and b is 25 and c is 24
a is 50 and b is 100 and c is 5
c)
a is 3 and b is 7 and c is 10
a is 25 and b is 5 and c is 24
a is 100 and b is 5 and c is 50
d) None of the mentioned

Answer: c
Explanation: Here,

- func(3, 7) assigns a=3, b=7, and uses default c=10.

- func(25, c=24) assigns a=25 (positional), c=24 (keyword), and default b=5.
- func(c=50, a=100) uses keyword arguments, so a=100, c=50, and default b=5.

This shows how keyword arguments let you specify parameters out of order and selectively override defaults.

9. What will be the output of the following Python code?

```
def maximum(x, y):
    if x > y:
        return x
    elif x == y:
        return 'The numbers are equal'
    else:
        return y

print(maximum(2, 3))
```

a) 2
b) 3
c) The numbers are equal
d) None of the mentioned

Answer: b
Explanation: The function maximum(x, y) compares two numbers. Since 2 < 3, it goes to the else block and returns y, which is 3.

10. Which of the following is a feature of DocString?

a) Provide a convenient way of associating documentation with Python modules, functions, classes, and methods
b) All functions should have a docstring
c) Docstrings can be accessed by the __doc__ attribute on objects

d) All of the mentioned

Answer: d
Explanation: DocStrings provide built-in documentation for Python modules, functions, classes, and methods. They are accessed via the __doc__ attribute and it's recommended that all functions have them for better code clarity and maintainability. Hence, all the mentioned points are true.

1. Which are the advantages of functions in python?

a) Reducing duplication of code
b) Decomposing complex problems into simpler pieces
c) Improving clarity of the code
d) All of the mentioned

Answer: d
Explanation: Functions in Python offer several advantages:

- Reducing duplication of code: You can reuse functions instead of repeating code.
- Decomposing complex problems: Functions allow you to break down a big problem into smaller, manageable parts.
- Improving clarity: By using descriptive names and organizing code into functions, your code becomes easier to read and maintain.

So, all the mentioned options are correct.

2. What are the two main types of functions?

a) Custom function
b) Built-in function & User defined function
c) User function
d) System function

Answer: b
Explanation: Python functions are of two main types: built-in and user-defined. Built-in functions like print(), len(), and abs() are available by default. User-defined functions are created using the def keyword to organize code into reusable, modular blocks.

3. Where is function defined?
a) Module
b) Class
c) Another function
d) All of the mentioned

Answer: d
Explanation: Functions in Python can be defined in various places, including modules, classes, and even inside other functions. When defined in a module, the function is available throughout the module or can be imported elsewhere. Inside a class, functions are defined as methods, and within another function, they become nested or inner functions. Hence, all of the mentioned options are valid.
4. What is called when a function is defined inside a class?
a) Module
b) Class
c) Another function
d) Method

Answer: d
Explanation: When a function is defined inside a class, it is called a method. Methods are functions that are associated with a class and can operate on instances of that class, often using the self keyword to access or modify the object's attributes.

5. Which of the following is the use of id() function in python?

a) Id returns the identity of the object

b) Every object doesn't have a unique id

c) All of the mentioned

d) None of the mentioned

Answer: a

Explanation: The id() function in Python returns the identity of an object. This identity is a unique integer (or memory address) that remains constant for the object during its lifetime. Every object in Python has a unique id, which helps in comparing object references.

6. Which of the following refers to mathematical function?

a) sqrt

b) rhombus

c) add

d) rhombus

Answer: a

Explanation: The sqrt function refers to the square root, which is a mathematical operation. In Python, it is typically used from the math module as math.sqrt(). Other options like "rhombus" and "add" are not standard mathematical functions in Python.

7. What will be the output of the following Python code?

```
def cube(x):
    return x * x * x
x = cube(3)
print (x)
```

a) 9

b) 3

c) 27

d) 30

Answer: c
Explanation: The function cube(x) returns the cube of the input number x, calculated as x * x * x.
When called with cube(3), it computes 3 * 3 * 3 = 27, so the value of x becomes 27.
Thus, print(x) outputs 27.

8. What will be the output of the following Python code?

```
def C2F(c):
    return c * 9/5 + 32
print(C2F(100))
print(C2F(0))
```

a)
212.0
32.0
b)
314.0
24.0
c)
567.0
98.0
d) None of the mentioned

Answer: a
Explanation: This function converts Celsius temperatures to Fahrenheit using the formula (C * 9/5) + 32. When called with 100, it returns 212.0, and with 0, it returns 32.0, which are the correct Fahrenheit equivalents for these Celsius values.

9. What will be the output of the following Python code?

```
def power(x, y=2):
    r = 1
    for i in range(y):
        r = r * x
    return r
print (power(3))
print (power(3, 3))
```

a)
212
32
b)
9
27
c)
567
98
d) None of the mentioned

Answer: b
Explanation: This function calculates the power of a number x raised to y. The default value of y is 2, so if no second argument is provided, it squares x.

- power(3) returns $3^2 = 9$
- power(3, 3) returns $3^3 = 27$

So the output is:
9 27

10. What will be the output of the following Python code?

```
def sum(*args):
  '''Function returns the sum of all values'''
```

```
  r = 0
  for i in args:
    r += i
  return r
print(sum.__doc__)
print(sum(1, 2, 3))
print(sum(1, 2, 3, 4, 5))
```

a)
Function returns the sum of all values
6
15
b)
6
100
c)
123
12345
d) None of the mentioned

Answer: a
Explanation: The function sum(*args) uses *args to accept any number of arguments and returns their total sum.

- sum.__doc__ prints the docstring: “Function returns the sum of all values”
- sum(1, 2, 3) calculates 1 + 2 + 3 = 6
- sum(1, 2, 3, 4, 5) calculates 1 + 2 + 3 + 4 + 5 = 15

1. Python supports the creation of anonymous functions at runtime, using a construct called _____

a) lambda
b) pi

c) anonymous
d) none of the mentioned

Answer: a
Explanation: In Python, lambda functions are anonymous, meaning they don't have a name. They are defined using the lambda keyword and can take any number of arguments but only have one expression. Lambdas are useful for creating small, throwaway functions quickly without formally defining them using def.

2. What will be the output of the following Python code?

```
y = 6
z = lambda x: x * y
print (z(8))
```

a) 48
b) 14
c) 64
d) None of the mentioned

Answer: a
Explanation: The lambda function takes x as input and multiplies it by the variable y which is 6. When z(8) is called, it calculates 8 * 6, resulting in 48. Thus, the output is 48.

3. What will be the output of the following Python code?

```
lamb = lambda x: x ** 3
print(lamb(5))
```

a) 15
b) 555
c) 125

d) None of the mentioned

Answer: c
Explanation: The lambda function calculates the cube of the input x. So, when lamb(5) is called, it computes $5^3 = 125$. Hence, the output is 125.

4. Does Lambda contains return statements?

a) True
b) False

Answer: b
Explanation: In Python, lambda functions do not contain return statements. Unlike regular functions defined using def, lambda functions are anonymous and consist of a single expression. This expression is automatically returned when the lambda is called, making the use of the return keyword unnecessary and invalid within a lambda. For example, lambda x: x * 2 will return the result of x * 2 when invoked. This concise syntax is useful for small, throwaway functions.

5. Lambda is a statement.

a) True
b) False

Answer: b
Explanation: Lambda is not a statement; it is an expression in Python. This means it returns a function object and can be used wherever expressions are allowed, such as in function calls or assignments. Since it's an expression, it cannot include statements like return, pass, or print inside its body.

6. Lambda contains block of statements.

a) True

b) False

Answer: b
Explanation: A lambda in Python can only contain a single expression, not a block of statements. This expression is evaluated and returned. You cannot include multiple lines, control flow statements like if, for, or while, or use return within a lambda.

7. What will be the output of the following Python code?

```
def f(x, y, z): return x + y + z
print(f(2, 30, 400))
```

a) 432
b) 24000
c) 430
d) No output

Answer: a
Explanation: The function f(x, y, z) returns the sum of the three arguments. So, f(2, 30, 400) results in: 2 + 30 + 400 = 432.

8. What will be the output of the following Python code?

```
def writer():
        title = 'Sir'
        name = (lambda x:title + ' ' + x)
        return name

who = writer()
print(who('Arthur'))
```

a) Arthur Sir
b) Sir Arthur
c) Arthur

d) None of the mentioned

Answer: b
Explanation: In the code, the function writer() defines a local variable title with the value ‘Sir’. It then creates and returns a lambda function name that takes an argument x and returns the string title + ‘ ‘ + x. Since the lambda function closes over the title variable (a closure), when you call who(‘Arthur’), it returns ‘Sir Arthur’.

9. What will be the output of the following Python code?

```
L = [lambda x: x ** 2,
     lambda x: x ** 3,
     lambda x: x ** 4]

for f in L:
    print(f(3))
```

a)
27
81
343
b)
6
9
12
c)
9
27
81
d) None of the mentioned

Answer: c
Explanation: The list L contains three lambda functions that calculate the square, cube, and fourth power of a number.

- The first computes the square (x ** 2), so 3 ** 2 = 9.
- The second computes the cube (x ** 3), so 3 ** 3 = 27.
- The third computes the fourth power (x ** 4), so 3 ** 4 = 81.

The for loop calls each lambda function with argument 3, printing their respective results.

10. What will be the output of the following Python code?

```
min = (lambda x, y: x if x < y else y)
print(min(101*99, 102*98))
```

a) 9997
b) 9999
c) 9996
d) None of the mentioned

Answer: c
Explanation: The lambda function compares two numbers and returns the smaller one. Since 101 * 99 equals 9999 and 102 * 98 equals 9996, the function returns 9996, which is the smaller value. Hence, the output is 9996.

1. What is a variable defined outside a function referred to as?

a) A static variable
b) A global variable
c) A local variable
d) An automatic variable

Answer: b
Explanation: A variable defined outside of any function is known as a global variable. It can be accessed inside functions using the global

keyword if you want to modify its value; otherwise, it can only be read inside functions.

2. What is a variable defined inside a function referred to as?

a) A global variable
b) A volatile variable
c) A local variable
d) An automatic variable

Answer: c
Explanation: A variable defined inside a function is called a local variable. It is only accessible within the function where it is defined and is created when the function is called and destroyed when the function ends.

3. What will be the output of the following Python code?

```
i=0
def change(i):
  i=i+1
  return i
change(1)
print(i)
```

a) 1
b) Nothing is displayed
c) 0
d) An exception is thrown

Answer: c
Explanation: The variable i defined outside the function is a global variable, and the i inside the change() function is local to that function. When change(1) is called, it returns 2, but this return value is not stored

or used. The global i remains unchanged and retains its original value 0. Hence, print(i) outputs 0.

4. What will be the output of the following Python code?

```
def a(b):
    b = b + [5]

c = [1, 2, 3, 4]
a(c)
print(len(c))
```

a) 4
b) 5
c) 1
d) An exception is thrown

Answer: a
Explanation: The list c = [1, 2, 3, 4] is passed to function a(b), but inside the function, the operation b = b + [5] creates a new list and assigns it to the local variable b. It does not modify the original list c because the + operator creates a new object instead of modifying the list in-place. Therefore, the length of c remains 4.

5. What will be the output of the following Python code?

```
a=10
b=20
def change():
    global b
    a=45
    b=56
change()
```

```
print(a)
print(b)
```

a)
10
56
b)
45
56
c)
10
20
d) Syntax Error

Answer: a
Explanation: In the given code: a = 10 and b = 20 are defined in the global scope.
Inside the change() function:

- a = 45 is a local assignment, so it does not affect the global variable a.
- global b declares that the function will modify the global variable b, so b becomes 56 globally.

Therefore, print(a) outputs 10 (unchanged global value) and print(b) outputs 56 (modified global value).

6. What will be the output of the following Python code?

```
def change(i = 1, j = 2):
   i = i + j
   j = j + 1
   print(i, j)
change(j = 1, i = 2)
```

a) An exception is thrown because of conflicting values
b) 1 2
c) 3 3
d) 3 2

Answer: d
Explanation: The function change has default parameters i = 1 and j = 2. However, when calling change(j = 1, i = 2), we're explicitly passing i = 2 and j = 1, overriding the defaults.
Inside the function:

```
i = i + j  # i = 2 + 1 = 3
j = j + 1  # j = 1 + 1 = 2
```

So the output is: 3 2

7. What will be the output of the following Python code?

```
def change(one, *two):
  print(type(two))
change(1,2,3,4)
```

a) Integer
b) <class 'tuple'>
c) <class 'Dict'>
d) An exception is thrown

Answer: b
Explanation: When a function in Python is defined with a parameter preceded by an asterisk (*), such as *two, it gathers any extra positional arguments passed into the function into a tuple. In the given code, the function change(1, 2, 3, 4) is called with more than one argument. The first value 1 is assigned to the parameter one, while the rest—2, 3, and 4—are captured as a tuple in two. Therefore, the print(type(two)) statement outputs <class 'tuple'>, indicating that two is indeed a tuple.

8. If a function doesn’t have a return statement, which of the following does the function return?

a) int
b) null
c) None
d) An exception is thrown without the return statement

Answer: c
Explanation: If a function in Python does not include a return statement, it returns None by default. This indicates the absence of a return value but still allows the function to complete successfully.

9. What will be the output of the following Python code?

```
def display(b, n):
    while n > 0:
        print(b,end="")
        n=n-1
display('z',3)
```

a) zzz
b) zz
c) An exception is executed
d) Infinite loop

Answer: a
Explanation: The function display takes a character b and an integer n, and uses a while loop to print the character n times without newline (due to end=””). When display(‘z’, 3) is called, it prints ‘z’ three times, resulting in the output zzz.

10. What will be the output of the following Python code?

```
def find(a, **b):
  print(type(b))
```

```
find('letters',A='1',B='2')
```

a) <class ‘string’>
b) <class ‘tuple’>
c) <class ‘dict’>
d) An exception is thrown

Answer: c
Explanation: In Python, the **b syntax collects all additional keyword arguments into a dictionary. In the function find(‘letters’, A=’1’, B=’2’), a gets ‘letters’, and b becomes {‘A’: ‘1’, ‘B’: ‘2’}. Hence, type(b) returns <class ‘dict’>.

1. What is the type of each element in sys.argv?

a) set
b) list
c) tuple
d) string

Answer: d
Explanation: sys.argv is a list in Python that contains command-line arguments passed to the script. Each element in sys.argv is of type string, regardless of whether it looks like a number or not. The first element (sys.argv[0]) is the script name.

2. What is the length of sys.argv?

a) number of arguments
b) number of arguments + 1
c) number of arguments – 1
d) none of the mentioned

Answer: b
Explanation: sys.argv includes the name of the script as the first element

(sys.argv[0]), followed by the actual command-line arguments. So, if there are n command-line arguments, the length of sys.argv will be number of arguments + 1.

3. What will be the output of the following Python code?

```
def foo(k):
    k[0] = 1
q = [0]
foo(q)
print(q)
```

a) [0]
b) [1]
c) [1, 0]
d) [0, 1]

Answer: b
Explanation: Lists in Python are mutable, so when the list q is passed to the function foo, the change made (k[0] = 1) modifies the original list. Therefore, after the function call, q becomes [1].

4. How are keyword arguments specified in the function heading?

a) one-star followed by a valid identifier
b) one underscore followed by a valid identifier
c) two stars followed by a valid identifier
d) two underscores followed by a valid identifier

Answer: c
Explanation: In Python, keyword arguments (also known as **kwargs) are specified in the function heading using two asterisks (**) followed by a valid identifier. For example:

```
def func(**kwargs):
```

```
    print(kwargs)
```
This allows the function to accept an arbitrary number of keyword arguments as a dictionary.

5. How many keyword arguments can be passed to a function in a single function call?

a) zero
b) one
c) zero or more
d) one or more

Answer: c
Explanation: In Python, a function can accept zero or more keyword arguments during a single call. These keyword arguments are specified by explicitly naming the parameters (e.g., func(a=1, b=2)), and Python functions can handle any number of such arguments, including none.

6. What will be the output of the following Python code?

```
def foo(fname, val):
    print(fname(val))
foo(max, [1, 2, 3])
foo(min, [1, 2, 3])
```
a)
3
1
b)
1
3
c) error
d) none of the mentioned

Answer: a
Explanation: The function foo takes a function fname and a value val as arguments. It then applies fname to val and prints the result.

- foo(max, [1, 2, 3]) applies max() to the list, which returns 3.
- foo(min, [1, 2, 3]) applies min() to the list, which returns 1.

So, the output is:
3
1

7. What will be the output of the following Python code?

```
def foo():
    return total + 1
total = 0
print(foo())
```

a) 0
b) 1
c) error
d) none of the mentioned

Answer: b
Explanation: The function foo() returns total + 1. Even though total is not defined inside the function, it is defined in the global scope before the function is called. Python allows reading a global variable inside a function unless you try to assign to it. So total = 0 globally, and foo() returns 0 + 1 = 1.

8. What will be the output of the following Python code?

```
def foo():
    total += 1
    return total
total = 0
```

```
print(foo())
```
a) 0
b) 1
c) error
d) none of the mentioned

Answer: c
Explanation: The code results in an error because inside the function foo(), you are trying to modify the variable total without declaring it as global. Python treats total as a local variable, but it's being used before assignment, leading to an UnboundLocalError.

9. What will be the output of the following Python code?

```
def foo(x):
    x = ['def', 'abc']
    return id(x)
q = ['abc', 'def']
print(id(q) == foo(q))
```
a) True
b) False
c) None
d) Error

Answer: b
Explanation: The function foo(x) assigns a new list ['def', 'abc'] to x, so x now points to a different object in memory. Therefore, the id(x) inside foo() will differ from id(q) outside the function, and id(q) == foo(q) will return False.

10. What will be the output of the following Python code?

```
def foo(i, x=[]):
    x.append(i)
```

```
    return x
for i in range(3):
    print(foo(i))
```

a)
[0]
[1]
[2]
b)
[0]
[0, 1]
[0, 1, 2]
c)
[1]
[2]
[3]
d) None of the mentioned

Answer: b
Explanation: The default argument x=[] is evaluated only once when the function is defined, not each time it's called. So the same list is used and modified across all calls to foo(). As a result, each call appends to the same list, and the list grows with each iteration.
[0]
[0, 1]
[0, 1, 2]

1. What will be the output of the following Python code?

```
def foo(k):
    k = [1]
q = [0]
foo(q)
```

```
print(q)
```
a) [0]
b) [1]
c) [1, 0]
d) [0, 1]

Answer: a
Explanation: In this code, the function foo(k) assigns a new list [1] to the local variable k, which does not affect the original list q. Since the reference is reassigned inside the function, it doesn't modify the external variable. Therefore, print(q) still outputs the original list: [0].

2. How are variable length arguments specified in the function heading?
a) one star followed by a valid identifier
b) one underscore followed by a valid identifier
c) two stars followed by a valid identifier
d) two underscores followed by a valid identifier

Answer: a
Explanation: In Python, variable-length arguments are specified using a single asterisk (*) followed by a valid identifier in the function definition. This collects any extra positional arguments into a tuple. For example:

```
def func(*args):
    for arg in args:
        print(arg)
```

Here, *args can take any number of positional arguments.
3. Which module in the python standard library parses options received from the command line?
a) getopt
b) os

c) getarg
d) main

Answer: a
Explanation: The getopt module in Python's standard library is used to parse command-line options and arguments. It allows the script to accept flags and parameters (like -h or –help) similar to those in shell scripts. For example:

```
import getopt, sys
opts, args = getopt.getopt(sys.argv[1:], "h", ["help"])
```

This line parses short option -h and long option –help.

4. What is the type of sys.argv?

a) set
b) list
c) tuple
d) string

Answer: b
Explanation: In Python, sys.argv is a list in the sys module that contains the command-line arguments passed to a script. The first element (sys.argv[0]) is the name of the script itself, and the remaining elements are the arguments provided by the user. So the type of sys.argv is list.

5. What is the value stored in sys.argv[0]?

a) null
b) you cannot access it
c) the program's name
d) the first argument

Answer: c
Explanation: sys.argv[0] stores the name of the Python script being

executed. It’s the first element of the sys.argv list, which always contains the script name, followed by any command-line arguments provided.

6. How are default arguments specified in the function heading?

a) identifier followed by an equal to sign and the default value
b) identifier followed by the default value within backticks (“)
c) identifier followed by the default value within square brackets ([])
d) identifier

Answer: a
Explanation: Default arguments in Python are assigned using the = sign in the function definition. If a value is not provided during the function call, the default is used. This helps make function calls simpler and avoids errors from missing arguments.

7. How are required arguments specified in the function heading?

a) identifier followed by an equal to sign and the default value
b) identifier followed by the default value within backticks (“)
c) identifier followed by the default value within square brackets ([])
d) identifier

Answer: d
Explanation: Required arguments in a Python function are specified simply by using the parameter name (identifier) without assigning any default value. These arguments must be passed when the function is called. For example:

```
def greet(name):
    print("Hello", name)
```

Here, name is a required argument—omitting it in a function call would raise an error.

8. What will be the output of the following Python code?

```
def foo(x):
```

```
    x[0] = ['def']
    x[1] = ['abc']
    return id(x)
q = ['abc', 'def']
print(id(q) == foo(q))
```

a) True
b) False
c) None
d) Error

Answer: a
Explanation: The list q is passed by reference to the function foo, so both x and q refer to the same object in memory. The id() function returns the memory address of the object, which remains unchanged. Therefore, id(q) == foo(q) evaluates to True.

9. Where are the arguments received from the command line stored?

a) sys.argv
b) os.argv
c) argv
d) none of the mentioned

Answer: a
Explanation: In Python, command-line arguments are stored in sys.argv, which is a list provided by the sys module. The first element (sys.argv[0]) is the script name, and the subsequent elements are the command-line arguments passed to the script.

10. What will be the output of the following Python code?

```
def foo(i, x=[]):
    x.append(x.append(i))
    return x
```

```
for i in range(3):
    y = foo(i)
print(y)
```

a) [[[0]], [[[0]], [1]], [[[0]], [[[0]], [1]], [2]]]
b) [[0], [[0], 1], [[0], [[0], 1], 2]]
c) [0, None, 1, None, 2, None]
d) [[[0]], [[[0]], [1]], [[[0]], [[[0]], [1]], [2]]]

Answer: c
Explanation: In this code, x.append(i) adds i to the list, but since append() returns None, x.append(x.append(i)) ends up adding both i and None to the list. This happens in each iteration, resulting in the final list: [0, None, 1, None, 2, None].

1. What will be the output of the following Python code?

```
def f1():
    x=15
    print(x)
x=12
f1()
```

a) Error
b) 12
c) 15
d) 1512

Answer: c
Explanation: The function f1() defines a local variable x with the value 15 and prints it. Even though there is a global x with value 12, the function uses its local x, so the output is 15.

2. What will be the output of the following Python code?

```
def f1():
   x=100
   print(x)
x=+1
f1()
```

a) Error
b) 100
c) 101
d) 99

Answer: b
Explanation: In this code, x = +1 sets the global variable x to 1 (unary plus doesn't change the value). Inside the function f1(), a new local variable x is defined and assigned the value 100. The function then prints this local variable, so the output is 100. The global x has no effect on the output of f1().

3. What will be the output of the following Python code?

```
def san(x):
   print(x+1)
x=-2
x=4
san(12)
```

a) 13
b) 10
c) 2
d) 5

Answer: a
Explanation: The function san takes the argument x and prints x + 1.

When called as san(12), it prints 12 + 1 = 13. The global variables x = -2 and x = 4 are not used inside the function call. So the output is 13.

4. What will be the output of the following Python code?

```
def f1():
   global x
   x+=1
   print(x)
x=12
print("x")
```

a) Error
b) 13
c)
13
x
d) x

Answer: d
Explanation: The code first defines a function f1() that uses the global variable x and increments it, but f1() is never called. Then it sets x = 12 and prints the string “x” (because “x” is in quotes, it prints the literal letter x). So the output is just: x.

5. What will be the output of the following Python code?

```
def f1(x):
   global x
   x+=1
   print(x)
f1(15)
print("hello")
```

a) error
b) hello

c) 16
d)
16
hello

Answer: a
Explanation: This code throws an error because you're trying to declare a function parameter x as global, which is not allowed in Python. The global statement can only be used for variables defined at the module level, not for function parameters. Therefore, the line global x after x has already been defined as a parameter causes a SyntaxError.

6. What will be the output of the following Python code?

```
x=12
def f1(a,b=x):
   print(a,b)
x=15
f1(4)
```

a) Error
b) 12 4
c) 4 12
d) 4 15

Answer: c
Explanation: Default argument values are evaluated only once at the time the function is defined, not each time the function is called. Here, b gets the default value of x at the time of function definition, which was 12. Even though x is later changed to 15, the default value remains 12. So calling f1(4) prints 4 12.

7. What will be the output of the following Python code?

```
def f():
```

```
    global a
    print(a)
    a = "hello"
    print(a)
a = "world"
f()
print(a)
```

a)
    hello
    hello
    world

b)
    world
    hello
    hello

c)
    hello
    world
    world

d)
    world
    hello
    world

Answer: b

Explanation: The variable a is initially set to “world” in the global scope. Inside the function f(), global a allows access and modification of the global variable a. The first print(a) outputs “world”, then a is updated to “hello” and printed again. After the function call, since a has been modified globally, the final print(a) also outputs “hello”.

8. What will be the output of the following Python code?

```
def f1(a,b=[]):
    b.append(a)
    return b
print(f1(2,[3,4]))
```

a) [3, 2, 4]
b) [2, 3, 4]
c) Error
d) [3, 4, 2]

Answer: d
Explanation: In the function f1, a is added to the list b using b.append(a). The list [3, 4] is passed explicitly as an argument, so a=2 is appended to it, resulting in [3, 4, 2]. Hence, the output is: [3, 4, 2].

9. What will be the output of the following Python code?

```
def f(p, q, r):
    global s
    p = 10
    q = 20
    r = 30
    s = 40
    print(p,q,r,s)
p,q,r,s = 1,2,3,4
f(5,10,15)
```

a) 1 2 3 4
b) 5 10 15 4
c) 10 20 30 40

d) 5 10 15 40

Answer: c
Explanation: In the function f(p, q, r), the values of p, q, and r are reassigned locally to 10, 20, and 30 respectively. The variable s is declared global, so s = 40 updates the global variable. The print statement inside the function prints the local p, q, r, and global s, resulting in: 10 20 30 40.

10. What will be the output of the following Python code?

```
def f(x):
   print("outer")
   def f1(a):
      print("inner")
      print(a,x)
f(3)
f1(1)
```

a)
outer
error
b)
inner
error
c)
outer
inner
d) error

Answer: a
Explanation: The function f defines an inner function f1, but f1 is local to f and cannot be accessed outside it. When f(3) is called, it prints “outer” and defines f1, but since f1 is not returned or called inside f, it remains

inaccessible from the outside.
So when f1(1) is called outside f, Python throws a NameError because f1 is not defined in the global scope.

11. What will be the output of the following Python code?

```
x = 5
def f1():
   global x
   x = 4
def f2(a,b):
   global x
   return a+b+x
f1()
total = f2(1,2)
print(total)
```

a) Error
b) 7
c) 8
d) 15

Answer: b
Explanation: The function f1() sets the global variable x to 4. Then f2(1, 2) adds 1 + 2 + x (which is 4) and returns 7. Thus, the output is 7.

12. What will be the output of the following Python code?

```
x=100
def f1():
   global x
   x=90
def f2():
   global x
   x=80
```

```
print(x)
```
a) 100
b) 90
c) 80
d) Error

Answer: a
Explanation: The functions f1() and f2() are defined but not called, so the global variable x remains unchanged at its original value of 100. Therefore, print(x) outputs 100.

13. Read the following Python code carefully and point out the global variables?

```
y, z = 1, 2
def f():
   global x
   x = y+z
```
a) x
b) y and z
c) x, y and z
d) Neither x, nor y, nor z

Answer: c
Explanation: y and z are defined at the module level, so they are global variables. Inside the function f(), x is declared with the global keyword, meaning it refers to a variable in the global scope and will be created or modified there. Thus, all three variables x, y, and z are global.

1. Which of the following data structures is returned by the functions globals() and locals()?
a) list
b) set

c) dictionary
d) tuple

Answer: c
Explanation: Both globals() and locals() return dictionaries that represent the current global and local symbol tables, respectively. These dictionaries contain variable names as keys and their corresponding values. They are useful for introspection and dynamic access to variables.

2. What will be the output of the following Python code?

```
x=1
def cg():
        global x
        x=x+1
cg()
print(x)
```

a) 2
b) 1
c) 0
d) Error

Answer: a
Explanation: The function cg() uses the global keyword to modify the global variable x. It increments x by 1, so after calling cg(), the value of x becomes 2. Hence, the output is 2.

3. On assigning a value to a variable inside a function, it automatically becomes a global variable.

a) True
b) False

Answer: b
Explanation: Assigning a value to a variable inside a function creates a local variable by default. To modify a global variable inside a function, you must explicitly declare it as global. Otherwise, the variable remains local to the function.

4. What will be the output of the following Python code?

```
e="butter"
def f(a): print(a)+e
f("bitter")
```

a) error
b)
butter
error
c)
bitter
error
d) bitterbutter

Answer: c
Explanation: The function f(a) tries to execute print(a) + e, but print(a) returns None, and adding None + e raises a TypeError. However, print(a) executes before the error occurs, so “bitter” is printed, followed by an error due to the invalid addition.

5. What happens if a local variable exists with the same name as the global variable you want to access?

a) Error
b) The local variable is shadowed
c) Undefined behavior
d) The global variable is shadowed

Answer: d
Explanation: When a local variable has the same name as a global variable, the local variable shadows the global one within that function or scope. This means the global variable is not accessible in that local context unless explicitly accessed using the global keyword.

6. What will be the output of the following Python code?

```
a=10
globals()['a']=25
print(a)
```

a) 10
b) 25
c) Junk value
d) Error

Answer: b
Explanation: globals() returns a dictionary representing the current global symbol table. When you do globals()[‘a’] = 25, you are directly modifying the global variable a to 25. So, when you print a afterward, it reflects the updated value 25.

7. What will be the output of the following Python code?

```
def f(): x=4
x=1
f()
print(x)
```

a) Error
b) 4
c) Junk value
d) 1

Answer: d
Explanation: Inside the function f(), x = 4 creates a local variable x that exists only within the function scope. The global variable x remains unchanged with the value 1. Since f() doesn't modify the global x, when you print x outside the function, it prints 1.

8. _____ returns a dictionary of the module namespace.
_____ returns a dictionary of the current namespace.

a)
locals()
globals()
b)
locals()
locals()
c)
globals()
locals()
d)
globals()
globals()

Answer: c
Explanation: globals() returns a dictionary representing the current global symbol table (module namespace). locals() returns a dictionary representing the current local symbol table (current namespace inside a function or block).

1. Which is the most appropriate definition for recursion?

a) A function that calls itself
b) A function execution instance that calls another execution instance of the same function
c) A class method that calls another class method

d) An in-built method that is automatically called

Answer: b
Explanation: The most appropriate and precise definition of recursion is “a function execution instance that calls another execution instance of the same function.” This emphasizes not just the function referring to itself, but the actual runtime behavior where a new invocation of the same function is made during execution.

2. Only problems that are recursively defined can be solved using recursion.
a) True
b) False

Answer: b
Explanation: The statement is False. While recursion is a natural fit for problems that are recursively defined (like tree traversals, factorial, Fibonacci, etc.), any problem that can be broken down into smaller subproblems can potentially be solved using recursion. Many problems that aren’t inherently recursive can still be approached recursively, though they might be more efficiently handled using iteration.

3. Which of these is false about recursion?
a) Recursive function can be replaced by a non-recursive function
b) Recursive functions usually take more memory space than non-recursive function
c) Recursive functions run faster than non-recursive function
d) Recursion makes programs easier to understand

Answer: c
Explanation: Recursive functions are often slower and consume more memory than their non-recursive counterparts due to the overhead of

multiple function calls and maintaining the call stack. While recursion can make code easier to read and understand for problems like tree traversal or factorials, it is generally less efficient than an iterative solution.

4. Fill in the line of the following Python code for calculating the factorial of a number.

```
def fact(num):
   if num == 0:
      return 1
   else:
      return _____________________
```

a) num*fact(num-1)
b) (num-1)*(num-2)
c) num*(num-1)
d) fact(num)*fact(num-1)

Answer: a
Explanation: This is a classic example of a recursive function to calculate the factorial of a number.
In recursion, the function calls itself with a reduced value until it reaches the base case (num == 0).
So, the correct recursive step is:

```
return num * fact(num - 1)
```

5. What will be the output of the following Python code?

```
def test(i,j):
   if(i==0):
      return j
   else:
      return test(i-1,i+j)
```

```
print(test(4,7))
```
a) 13
b) 7
c) Infinite loop
d) 17

Answer: d
Explanation: The function test(i, j) recursively calls itself by decreasing i and adding i + j each time. Starting from test(4, 7), the values evolve as test(3,11), test(2,14), test(1,16), and finally test(0,17). When i becomes 0, it returns j, which is 17.

6. What will be the output of the following Python code?

```
l=[]
def convert(b):
    if(b==0):
        return l
    dig=b%2
    l.append(dig)
    convert(b//2)
convert(6)
l.reverse()
for i in l:
    print(i,end="")
```
a) 011
b) 110
c) 3
d) Infinite loop

Answer: b
Explanation: The function recursively divides the number by 2, appending

the remainder to list l. For 6, the remainders collected are [0, 1, 1]. After reversing, it becomes [1, 1, 0], which is the binary form of 6. Printing the list outputs 110.

7. What is tail recursion?

a) A recursive function that has two base cases
b) A function where the recursive functions leads to an infinite loop
c) A recursive function where the function doesn't return anything and just prints the values
d) A function where the recursive call is the last thing executed by the function

Answer: d
Explanation: In tail recursion, the recursive call is the last operation in the function before it returns the result. This allows some compilers or interpreters to optimize the recursion by reusing the current function's stack frame, making it more memory-efficient and avoiding stack overflow issues.

8. Observe the following Python code?

```
def a(n):
    if n == 0:
        return 0
    else:
        return n*a(n - 1)
def b(n, tot):
    if n == 0:
        return tot
    else:
        return b(n-2, tot-2)
```

a) Both a() and b() aren't tail recursive
b) Both a() and b() are tail recursive

c) b() is tail recursive but a() isn't
d) a() is tail recursive but b() isn't

Answer: c
Explanation: In tail recursion, the recursive call is the last operation in the function. In a(n), the multiplication (n * a(n – 1)) is performed after the recursive call, which prevents it from being tail recursive. On the other hand, in b(n, tot), the recursive call b(n – 2, tot – 2) is the last operation, making b() tail recursive.

9. Which of the following statements is false about recursion?

a) Every recursive function must have a base case
b) Infinite recursion can occur if the base case isn't properly mentioned
c) A recursive function makes the code easier to understand
d) Every recursive function must have a return value

Answer: d
Explanation: Not every recursive function needs to return a value. Some recursive functions, like those used for printing or performing actions (e.g., traversing a data structure), may not return anything—they just perform operations. Thus, option "Every recursive function must have a return value" is false. The other statements are true: a base case is essential to avoid infinite recursion, and recursion can simplify certain algorithms.

10. What will be the output of the following Python code?

```
def fun(n):
   if (n > 100):
      return n - 5
   return fun(fun(n+11));
```

```
print(fun(45))
```

a) 50
b) 100
c) 74
d) Infinite loop

Answer: b
Explanation: The function fun(n) uses nested recursion. For any input n ≤ 100, it keeps calling itself with n + 11 until n > 100, at which point it returns n – 5. Despite the complexity, the recursion stabilizes and always returns 100 for inputs ≤ 100. This is a variation of the McCarthy 91 function.

11. Recursion and iteration are the same programming approach.

a) True
b) False

Answer: b
Explanation: Recursion and iteration are not the same. Recursion involves a function calling itself to solve smaller instances of a problem. Iteration uses looping constructs (like for or while) to repeatedly execute a block of code. They can solve similar problems, but their approach, memory usage, and structure differ significantly.

12. What happens if the base condition isn't defined in recursive programs?

a) Program gets into an infinite loop
b) Program runs once
c) Program runs n number of times where n is the argument given to the function
d) An exception is thrown

Answer: a
Explanation: If a base condition is not defined in a recursive function, the function will keep calling itself indefinitely. This leads to an infinite recursion, eventually causing a stack overflow error or RecursionError in Python. The base condition is essential to stop the recursion at the right point.

13. Which of these is not true about recursion?

a) Making the code look clean
b) A complex task can be broken into sub-problems
c) Recursive calls take up less memory
d) Sequence generation is easier than a nested iteration

Answer: c
Explanation: Recursion usually takes up more memory compared to iteration because each recursive call adds a new frame to the call stack, which can lead to stack overflow if not handled properly (especially in deep recursion).

14. Which of these is not true about recursion?

a) It’s easier to code some real-world problems using recursion than non-recursive equivalent
b) Recursive functions are easy to debug
c) Recursive calls take up a lot of memory
d) Programs using recursion take longer time than their non-recursive equivalent

Answer: b
Explanation: While recursion can simplify certain problems (like tree traversals or factorial calculations), it can make debugging more difficult because of the multiple recursive calls, which can obscure the flow of execution and make it harder to track state across calls.

15. What will be the output of the following Python code?

```
def a(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return a(n-1)+a(n-2)
for i in range(0,4):
    print(a(i),end=" ")
```

a) 0 1 2 3
b) An exception is thrown
c) 0 1 1 2 3
d) 0 1 1 2

Answer: d
Explanation: The function a(n) calculates the Fibonacci sequence. For each value of n, it recursively computes the sum of the two previous numbers. For i = 0, 1, 2, 3, the results are 0, 1, 1, and 2 respectively, producing the output 0 1 1 2.

1. Which type of copy is shown in the following python code?

```
l1=[[10, 20], [30, 40], [50, 60]]
ls=list(l1)
ls
[[10, 20], [30, 40], [50, 60]]
```

a) Shallow copy
b) Deep copy
c) memberwise

d) All of the mentioned

Answer: a
Explanation: The list() constructor creates a shallow copy of the list. It copies the outer list structure, but not the inner nested lists—so both l1 and ls share references to the same inner lists. Changes to those nested lists via either reference will be reflected in both.

2. What will be the output of the following Python code?

```
l=[2, 3, [4, 5]]
l2=l.copy()
l2[0]=88
print(l)
print(l2)
```

a)
[88, 2, 3, [4, 5]]
[88, 2, 3, [4, 5]]
b)
[2, 3, [4, 5]]
[88, 3, [4, 5]]
c)
[88, 2, 3, [4, 5]]
[2, 3, [4, 5]]
d)
[2, 3, [4, 5]]
[2, 3, [4, 5]]

Answer: b
Explanation: l.copy() creates a shallow copy of list l. This means l2 has its own outer list, but the nested list [4, 5] is still shared. Changing l2[0] to 88

does not affect l, so l remains [2, 3, [4, 5]], while l2 becomes [88, 3, [4, 5]].

3. In _____ copy, the base address of the objects are copied. In _____ copy, the base address of the objects are not copied.

a) deep. shallow
b) memberwise, shallow
c) shallow, deep
d) deep, memberwise

Answer: c
Explanation: In a shallow copy, only the outer object is copied and its base address is referenced, so changes in nested objects affect both copies. In a deep copy, all objects (including nested ones) are recursively copied, so their base addresses are not the same, and modifications do not affect each other.

4. The nested list undergoes shallow copy even when the list as a whole undergoes deep copy.

a) True
b) False

Answer: a
Explanation: In Python, a shallow copy creates a new outer list, but the nested (inner) lists remain as references to the original objects. This means that changes made to the nested elements in the copied list will also reflect in the original list. Even though the outer list is copied, the inner lists are not duplicated unless a deep copy is explicitly performed using the copy module's deepcopy() function. Therefore, the nested lists

undergo a shallow copy even when the outer list seems to be deeply copied.

5. What will be the output of the following Python code and state the type of copy that is depicted?

```
l1=[2, 4, 6, 8]
l2=[1, 2, 3]
l1=l2
print(l2)
```

a) [2, 4, 6, 8], shallow copy
b) [2, 4, 6, 8], deep copy
c) [1, 2, 3], shallow copy
d) [1, 2, 3], deep copy

Answer: c
Explanation: The line l1 = l2 doesn't create a new copy; it just makes l1 refer to the same object as l2. This is a shallow copy, where both variables point to the same list in memory. Printing l2 gives [1, 2, 3].

6. What will be the output of the following Python code?

```
l1=[10, 20, 30]
l2=l1
print(id(l1)==id(l2))

l2=l1.copy()
print(id(l1)==id(l2))
```

a) False, False
b) False, True
c) True, True
d) True, False

Answer: d
Explanation: In the given code, l2 = l1 assigns l2 to reference the same list object as l1, so id(l1) == id(l2) returns True. However, when l2 = l1.copy() is used, it creates a new list with the same elements, resulting in a different object in memory. Therefore, id(l1) == id(l2) now returns False. This demonstrates the difference between assigning a reference and creating a shallow copy of a list.

7. What will be the output of the following Python code?

```
l1=[1, 2, 3, [4]]
l2=list(l1)
print(id(l1)==id(l2))
```

a) True
b) False
c) Error
d) Address of l1

Answer: b
Explanation: In the code, l2 = list(l1) creates a shallow copy of the list l1. This means l2 is a new list object with the same elements as l1, so id(l1) == id(l2) evaluates to False. Although the outer lists are different objects, they still share the same inner list [4] due to shallow copying.

8. What will be the output of the following Python code?

```
import copy
l1=[10, 20, 30, [40]]
l2=copy.deepcopy(l1)
l1[3][0]=90
print(l1)
print(l2)
```

a)
[10, 20, 30, [40]]

[10, 20, 30, 90]
b) Error
c)
[10, 20, 30 [90]]
[10, 20, 30, [40]]
d)
[10, 20, 30, [40]]
[10, 20, 30, [90]]

Answer: c
Explanation: The code uses copy.deepcopy() to create a completely independent copy of l1, including the nested list [40].
When l1[3][0] = 90 is executed, it modifies only l1, not l2, because they no longer share the same inner list.
As a result, the output is:
[10, 20, 30 [90]]
[10, 20, 30, [40]]

9. In _____ copy, the modification done on one list affects the other list. In _____ copy, the modification done on one list does not affect the other list.
a) shallow, deep
b) memberwise, shallow
c) deep, shallow
d) deep, memberwise

Answer: a
Explanation: In a shallow copy, the outer list is copied, but nested objects (like inner lists) are still referenced. So, changes to nested elements in

one list affect the other. In a deep copy, all objects are recursively copied, making the two lists completely independent. Hence, modifications in one list do not affect the other.

10. What will be the output of the following Python code?

```
l1=[1, 2, 3, (4)]
l2=l1.copy()
print(l2)
print(l1)
```

a)
[1, 2, 3, (4)]
[1, 2, 3, 4]
b)
[1, 2, 3, 4]
[1, 2, 3, (4)]
c)
[1, 2, 3, 4]
[1, 2, 3, 4]
d)
[1, 2, 3, (4)]
[1, 2, 3, (4)]

Answer: c
Explanation: Here,

- (4) is just an integer 4 enclosed in parentheses, not a tuple because a single-element tuple needs a trailing comma (4,).
- So, l1 is [1, 2, 3, 4] (the last element is integer 4, not a tuple).
- l2 = l1.copy() creates a shallow copy of the list.
- Printing l1 and l2 will both display [1, 2, 3, 4].

11. What will be the output of the following Python code?

```
def check(n):
    if n < 2:
        return n % 2 == 0
    return check(n - 2)
print(check(11))
```

a) False
b) True
c) 1
d) An exception is thrown

Answer: a
Explanation: The function check(n) recursively subtracts 2 until n < 2. Then it returns n % 2 == 0. For n = 11, the recursion reaches n = 1, and 1 % 2 == 0 is False. So, the output is False.

12. What is the base case in the Merge Sort algorithm when it is solved recursively?

a) n=0
b) n=1
c) A list of length one
d) An empty list

Answer: c
Explanation: In the recursive Merge Sort algorithm, the base case occurs when the list to be sorted has only one element. At this point, the list is already sorted by definition, so no further splitting or sorting is needed.

13. What will be the output of the following Python code?

```
a = [1, 2, 3, 4, 5]
b = lambda x: (b (x[1:]) + x[:1] if x else [])
print(b (a))
```

a) 1 2 3 4 5
b) [5, 4, 3, 2, 1]
c) []
d) Error, lambda functions can’t be called recursively

Answer: b
Explanation: The lambda function b is recursively called on the sliced list x[1:] until the list is empty (x == []). It builds the reversed list by appending the first element (x[:1]) at the end during each return. This effectively reverses the list. So, for a = [1, 2, 3, 4, 5], it returns [5, 4, 3, 2, 1].

1. What will be the output of the following Python code?

```
odd=lambda x: bool(x%2)
numbers=[n for n in range(10)]
print(numbers)
n=list()
for i in numbers:
    if odd(i):
        continue
    else:
        break
```

a) [0, 2, 4, 6, 8, 10]
b) [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
c) [1, 3, 5, 7, 9]
d) Error

Answer: b
Explanation: The list numbers is created using range(10), which gives [0, 1, 2, 3, 4, 5, 6, 7, 8, 9], and it is printed directly. The for loop that follows checks each number, and since the first number 0 is even, the else block

triggers break, exiting the loop immediately without modifying or printing anything. Hence, the printed output is just the original list.

2. What will be the output of the following Python code?

```
f=lambda x:bool(x%2)
print(f(20), f(21))
```

a) False True
b) False False
c) True True
d) True False

Answer: a
Explanation: The lambda function f = lambda x: bool(x % 2) returns True if x is odd (x % 2 == 1), and False if x is even (x % 2 == 0).

- f(20) → 20 % 2 = 0 → bool(0) → False
- f(21) → 21 % 2 = 1 → bool(1) → True

So the output is:

```
False True
```

3. What will be the output of the following Python code?

```
import functools
l=[1,2,3,4]
print(functools.reduce(lambda x,y:x*y,l))
```

a) Error
b) 10
c) 24
d) No output

Answer: c
Explanation: The code uses functools.reduce() to multiply all elements in

the list l = [1, 2, 3, 4].
reduce(lambda x, y: x * y, l) computes:

- 1 * 2 = 2
- 2 * 3 = 6
- 6 * 4 = 24

So the final result printed is 24.

4. What will be the output of the following Python code?

```
l=[1, -2, -3, 4, 5]
def f1(x):
   return x<2
m1=filter(f1, l)
print(list(m1))
```

a) [1, 4, 5]
b) Error
c) [-2, -3]
d) [1, -2, -3]

Answer: d
Explanation: The function f1(x) filters values less than 2 from the list l. Since 1, -2, and -3 meet this condition, they are included. The output is [1, -2, -3].

5. What will be the output of the following Python code?

```
l=[-2, 4]
m=map(lambda x:x*2, l)
print(m)
```

a) [-4, 16]
b) Address of m
c) Error
d)
-4

16

Answer: b
Explanation: The map() function returns a map object, which is an iterator. When printed directly without converting it to a list or iterating over it, it displays its memory address (e.g., <map object at 0x…>). To see the values, you'd need to do print(list(m)).

6. What will be the output of the following Python code?

```
l=[1, -2, -3, 4, 5]
def f1(x):
    return x<-1
m1=map(f1, l)
print(list(m1))
```

a) [False, False, False, False, False]
b) [False, True, True, False, False]
c) [True, False, False, True, True]
d) [True, True, True, True, True]

Answer: b
Explanation: The function checks if each element is less than -1. Only -2 and -3 satisfy this, so their results are True, while others are False. Hence, the output list is [False, True, True, False, False].

7. What will be the output of the following Python code?

```
l=[1, 2, 3, 4, 5]
m=map(lambda x:2**x, l)
```

```
print(list(m))
```
a) [1, 4, 9, 16, 25]
b) [2, 4, 8, 16, 32]
c) [1, 0, 1, 0, 1]
d) Error

Answer: b
Explanation: The map function applies lambda x: 2**x to each element in list l. So, it calculates powers of 2 for each element: 2^1, 2^2, 2^3, 2^4, 2^5, resulting in [2, 4, 8, 16, 32].

8. What will be the output of the following Python code?

```
import functools
l=[1, 2, 3, 4, 5]
m=functools.reduce(lambda x, y:x if x>y else y, l)
print(m)
```
a) Error
b) Address of m
c) 1
d) 5

Answer: d
Explanation: functools.reduce applies the lambda function cumulatively to the items in the list l. The lambda compares two values and returns the greater one. So it effectively finds the maximum value in the list, which is 5.

9. What will be the output of the following Python code?

```
l=[n for n in range(5)]
f=lambda x:bool(x%2)
print(f(3), f(1))
for i in range(len(l)):
```

```
    if f(l[i]):
        del l[i]
        print(i)
```

a)

True True

1

2

Error

b)

False False

1

2

c)

True False

1

2

Error

d)

False True

1

2

Answer: a

Explanation: The code prints True True because both 3 and 1 are odd numbers. It then tries to delete elements from the list while iterating over it using indices, which causes the list size to change and the loop to access an invalid index. This results in an IndexError after printing 1 and 2.

10. What will be the output of the following Python code?

```
m=reduce(lambda x: x-3 in range(4, 10))
print(list(m))
```

a) [1, 2, 3, 4, 5, 6, 7]
b) No output
c) [1, 2, 3, 4, 5, 6]
d) Error

Answer: d
Explanation: The code results in an error because reduce requires two arguments: a function with two parameters and an iterable. Here, only a lambda with one parameter is given, and no iterable is provided. Also, reduce is not imported, causing a NameError. Hence, the code fails with an error.

11. Which of the following numbers will not be a part of the output list of the following Python code?

```
def sf(a):
   return a%3!=0 and a%5!=0
m=filter(sf, range(1, 31))
print(list(m))
```

a) 1
b) 29
c) 16
d) 10

Answer: d
Explanation: The function sf(a) filters out numbers divisible by 3 or 5. Since 10 is divisible by 5, it gets excluded. The other options (1, 16, 29) are not divisible by 3 or 5, so they are included.

12. The single line equivalent of the following Python code?

```
l=[1, 2, 3, 4, 5]
def f1(x):
   return x<0
m1=filter(f1, l)
print(list(m1))
```

a) filter(lambda x:x<0, l)
b) filter(lambda x, y: x<0, l)
c) filter(reduce x<0, l)
d) reduce(x: x<0, l)

Answer: a
Explanation: The code shown above returns a new list containing only those elements from list l, which are less than 0. Since there are no such elements in the list l, the output of this code is: []. The single line equivalent of this code is filter(lambda x:x<0, l).

13. What will be the output of the following Python code?

```
print(list(map((lambda x:x^2), range(10))))
```

a) [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
b) Error
c) [2, 3, 0, 1, 6, 7, 4, 5, 10, 11]
d) No output

Answer: c
Explanation: The code uses the ^ operator, which performs a bitwise XOR, not exponentiation. For each element x in the range, it calculates x^2, which results in a different sequence of numbers than expected from squaring. This leads to the output [2, 3, 0, 1, 6, 7, 4, 5, 10, 11] based on the bitwise XOR operation between each number and 2.

14. What will be the output of the following Python code?

```
print(list(map((lambda x:x**2), filter((lambda x:x%2==0), range(10)))))
```

a) [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
b) [0, 4, 16, 36, 64]
c) Error
d) No output

Answer: b
Explanation: The code uses filter() to keep only even numbers from range(10), i.e., [0, 2, 4, 6, 8].
Then, map() applies the lambda function x**2 to each of these even numbers, producing [0, 4, 16, 36, 64].

15. The output of the following codes are the same.

```
print([x**2 for x in range(10)])
print(list(map((lambda x:x**2), range(10))))
```

a) True
b) False

Answer: a
Explanation: Both of the codes shown above print each whole number up to 10, raised to the power 2. Hence the output of both of these codes is: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]. Therefore, the statement is true.

1. What will be the output of the following Python code?

```
elements = [0, 1, 2]
def incr(x):
   return x+1
print(list(map(elements, incr)))
```

a) [1, 2, 3]
b) [0, 1, 2]
c) error
d) none of the mentioned

Answer: c
Explanation: The code causes an error because the map() function is used incorrectly. The correct syntax is map(function, iterable), but here elements (a list) is passed as the function, and incr (a function) as the iterable. Since a list isn't callable, this raises a TypeError.

2. What will be the output of the following Python code?

```
elements = [0, 1, 2]
def incr(x):
   return x+1
print(list(map(incr, elements)))
```

a) [1, 2, 3]
b) [0, 1, 2]
c) error
d) none of the mentioned

Answer: a
Explanation: The function incr(x) returns x + 1. The map() function applies incr to each element of the elements list [0, 1, 2]. This results in the list [1, 2, 3]. Hence, the output is [1, 2, 3].

3. What will be the output of the following Python code?

```
x = ['ab', 'cd']
print(list(map(upper, x)))
```

a) ['AB', 'CD']
b) ['ab', 'cd']
c) error
d) none of the mentioned

Answer: c
Explanation: The function upper is not defined or imported in the code. In

Python, strings have a method .upper() (e.g., ‘ab’.upper() returns ‘AB’), but upper by itself is not a built-in function unless explicitly imported or defined. Therefore, trying to use map(upper, x) will result in a NameError at runtime.

4. What will be the output of the following Python code?

```
def to_upper(k):
    return k.upper()
x = ['ab', 'cd']
print(list(map(upper, x)))
```

a) [‘AB’, ‘CD’]
b) [‘ab’, ‘cd’]
c) none of the mentioned
d) error

Answer: d
Explanation: The code will result in an error because the function upper is not defined anywhere. Even though to_upper(k) is defined correctly, it is not used in the map() call. Instead, the undefined name upper is passed to map(). This will result in a NameError. The correct way would be: print(list(map(to_upper, x)))

5. What will be the output of the following Python code?

```
def to_upper(k):
    return k.upper()
x = ['ab', 'cd']
print(list(map(to_upper, x)))
```

a) [‘AB’, ‘CD’]
b) [‘ab’, ‘cd’]
c) none of the mentioned
d) error

Answer: a
Explanation: The function to_upper(k) converts each string in the list to uppercase using k.upper(). The map() function applies to_upper to each element in [‘ab’, ‘cd’], resulting in [‘AB’, ‘CD’]. Hence, the output is uppercase versions of the original strings.

6. What will be the output of the following Python code?

```
def to_upper(k):
    k.upper()
x = ['ab', 'cd']
print(list(map(to_upper, x)))
```

a) [‘AB’, ‘CD’]
b) [‘ab’, ‘cd’]
c) [None, None]
d) error

Answer: c
Explanation: In the function to_upper(k), the line k.upper() performs the conversion to uppercase but does not return the result. Since there is no return statement, Python returns None by default. So, when map(to_upper, x) is executed, it applies to_upper to each element, but since to_upper returns None, the result is [None, None].

7. What will be the output of the following Python code?

```
x = ['ab', 'cd']
print(map(len, x))
```

a) [‘ab’, ‘cd’]
b) [2, 2]
c) [‘2’, ‘2’]
d) none of the mentioned

Answer: d
Explanation: The code map(len, x) returns a map object, not a list. To see the output, it must be converted using list(map(len, x)). So, as written, it just prints something like <map object at …> — not the actual list.

8. What will be the output of the following Python code?

```
x = ['ab', 'cd']
print(list(map(len, x)))
```

a) [‘ab’, ‘cd’]
b) [2, 2]
c) [‘2’, ‘2’]
d) none of the mentioned

Answer: b
Explanation: The map(len, x) function applies the len function to each element of the list x = [‘ab’, ‘cd’]. The length of both ‘ab’ and ‘cd’ is 2, so the result is [2, 2]. Wrapping it with list() converts the map object into a list.

9. What will be the output of the following Python code?

```
x = ['ab', 'cd']
print(len(map(list, x)))
```

a) [2, 2]
b) 2
c) 4
d) none of the mentioned

Answer: d
Explanation: In Python, map(list, x) returns a map object, not a list. You cannot directly apply len() to a map object unless it’s first converted to a list. So len(map(list, x)) raises a TypeError.

10. What will be the output of the following Python code?

```
x = ['ab', 'cd']
print(len(list(map(list, x))))
```

a) 2
b) 4
c) error
d) none of the mentioned

Answer: a
Explanation: map(list, x) applies list() to each element of x. Each string like ‘ab’ becomes [‘a’, ‘b’]. Converting the map object to a list gives [[‘a’, ‘b’], [‘c’, ‘d’]] which has length 2. So, len(list(map(list, x))) outputs 2.

1. What will be the output of the following Python code?

```
x = ['ab', 'cd']
print(len(list(map(list, x))))))
```

a) 2
b) 4
c) error
d) none of the mentioned

Answer: c
Explanation: The code has a syntax error due to an extra closing parenthesis)))) at the end. Python will raise a SyntaxError when trying to parse the code.

2. What will be the output of the following Python code?

```
x = ['ab', 'cd']
print(list(map(list, x)))
```

a) [‘a’, ‘b’, ‘c’, ‘d’]
b) [[‘ab’], [‘cd’]]
c) [[‘a’, ‘b’], [‘c’, ‘d’]]

d) none of the mentioned

Answer: c
Explanation: In the given Python code, the map() function is used to apply the list() function to each element of the list x, which contains the strings ‘ab’ and ‘cd’. The list() function, when applied to a string, converts it into a list of its individual characters. As a result, ‘ab’ becomes [‘a’, ‘b’] and ‘cd’ becomes [‘c’, ‘d’]. Therefore, the output of the code is a list of these lists: [[‘a’, ‘b’], [‘c’, ‘d’]].

3. What will be the output of the following Python code?

```
x = [12, 34]
print(len(list(map(len, x))))
```

a) 2
b) 1
c) error
d) none of the mentioned

Answer: c
Explanation: The given code will raise a TypeError. The map() function is attempting to apply len() to each element in the list x, which contains integers (12 and 34). Since len() is only valid for iterable types like strings, lists, etc., using it on integers results in an error.

4. What will be the output of the following Python code?

```
x = [12, 34]
print(len(list(map(int, x))))
```

a) 2
b) 1
c) error
d) none of the mentioned

Answer: a
Explanation: The list x = [12, 34] contains integers. The map(int, x) applies the int() function to each element, which has no effect since they are already integers. The list() converts the result to [12, 34]. Finally, len(…) returns the length of the list, which is 2.

5. What will be the output of the following Python code?

```
x = [12, 34]
print(len(''.join(list(map(int, x)))))
```

a) 4
b) 2
c) error
d) none of the mentioned

Answer: c
Explanation: The code tries to join integers using ”.join(), which expects a sequence of strings, not integers. Since map(int, x) gives integers, trying to join them with ” results in a TypeError.

6. What will be the output of the following Python code?

```
x = [12, 34]
print(len(''.join(list(map(str, x)))))
```

a) 4
b) 5
c) 6
d) error

Answer: a
Explanation: The code converts the integers [12, 34] to strings [’12’, ’34’] using map(str, x), then joins them into a single string ‘1234’. The len() function returns the length of this string, which is 4.

7. What will be the output of the following Python code?

```
x = [12, 34]
print(len(' '.join(list(map(int, x)))))
```

a) 4
b) 5
c) 6
d) error

Answer: d
Explanation: The code attempts to use ‘ ‘.join() on a list of integers: list(map(int, x)) gives [12, 34], but join() requires all elements to be strings, not integers. Since you can’t join integers directly, this raises a TypeError.

8. What will be the output of the following Python code?

```
x = [12.1, 34.0]
print(len(' '.join(list(map(str, x)))))
```

a) 6
b) 8
c) 9
d) error

Answer: c
Explanation: In the given code, the floating-point numbers in the list [12.1, 34.0] are first converted to strings using map(str, x), resulting in [‘12.1’, ‘34.0’]. These are then joined with a space using ‘ ‘.join(…), forming the string ‘12.1 34.0’. The length of this string is 9 characters, including the space.

9. What will be the output of the following Python code?

```
x = [12.1, 34.0]
```

```
print(' '.join(list(map(str, x))))
```
a) 12 1 34 0
b) 12.1 34
c) 121 340
d) 12.1 34.0

Answer: d
Explanation: In this code, each float in the list x = [12.1, 34.0] is converted to a string using map(str, x), resulting in [‘12.1’, ‘34.0’]. The ‘ ‘.join(…) function then joins these strings with a space, producing the final output ‘12.1 34.0’.

10. What will be the output of the following Python code?

```
x = [[0], [1]]
print(len(' '.join(list(map(str, x)))))
```
a) 2
b) 3
c) 7
d) 8

Answer: c
Explanation: The code converts each list inside x to a string, resulting in the strings “[0]” and “[1]”. Joining them with a space produces the string “[0] [1]”, which has 7 characters (including the space). Therefore, the output is 7.

1. What will be the output of the following Python code?

```
x = [[0], [1]]
print((' '.join(list(map(str, x)))))
```
a) (‘[0] [1]’,)
b) (’01’,)

c) [0] [1]
d) 01

Answer: c
Explanation: The code uses map(str, x) to convert each sublist [0] and [1] into their string representations “[0]” and “[1]”. Then ‘ ‘.join() joins these strings with a space, resulting in the output [0] [1].

2. What will be the output of the following Python code?

```
x = [[0], [1]]
print((' '.join(list(map(str, x))),))
```

a) (‘[0] [1]’,)
b) (’01’)
c) [0] [1]
d) 01

Answer: a
Explanation: In this code, map(str, x) converts each inner list [0] and [1] into strings: “[0]” and “[1]”. Then ‘ ‘.join(…) creates the string “[0] [1]”. Finally, it is wrapped in a tuple using the comma syntax, resulting in (‘[0] [1]’,).

3. What will be the output of the following Python code?

```
x = [34, 56]
print((''.join(list(map(str, x))),))
```

a) 3456
b) (3456)
c) (‘3456’)
d) (‘3456’,)

Answer: d
Explanation: The code converts each integer in the list x = [34, 56] to a string using map(str, x), resulting in [“34”, “56”]. These are then joined into a single string: “3456”. Finally, wrapping this in parentheses with a comma makes it a tuple containing the string: (‘3456’,).

4. What will be the output of the following Python code?

```
x = [34, 56]
print((''.join(list(map(str, x)))),)
```

a) 3456
b) (3456)
c) (‘3456’)
d) (‘3456’,)

Answer: a
Explanation: The expression ”.join(list(map(str, x))) converts each integer in the list x = [34, 56] into strings ([“34”, “56”]) and joins them into “3456”. The comma after the print function does not affect the output, as it is outside the parentheses of the print() function. So the final output is 3456.

5. What will be the output of the following Python code?

```
x = [34, 56]
print(len(map(str, x)))
```

a) [34, 56]
b) [’34’, ’56’]
c) 34 56
d) error

Answer: d
Explanation: In Python, map() returns an iterator, not a list. Calling len()

on a map object directly raises a TypeError because iterators do not have a predefined length. You must first convert it to a list using list(map(…)) before applying len().

6. What will be the output of the following Python code?

```
x = 'abcd'
print(list(map(list, x)))
```

a) [‘a’, ‘b’, ‘c’, ‘d’]
b) [‘abcd’]
c) [[‘a’], [‘b’], [‘c’], [‘d’]]
d) none of the mentioned

Answer: c
Explanation: In this Python code, the string ‘abcd’ is passed to the map() function along with list as the mapping function. The map() applies list() to each character of the string. Since strings are iterable, list(‘a’) becomes [‘a’], and similarly for the other characters. Therefore, the output is a list of lists: [[‘a’], [‘b’], [‘c’], [‘d’]].

7. What will be the output of the following Python code?

```
x = abcd
print(list(map(list, x)))
```

a) [‘a’, ‘b’, ‘c’, ‘d’]
b) [‘abcd’]
c) [[‘a’], [‘b’], [‘c’], [‘d’]]
d) none of the mentioned

Answer: d
Explanation: The given code will raise an error because the value abcd is not defined or enclosed in quotes. In Python, string literals must be enclosed in either single (‘abcd’) or double (“abcd”) quotes. Without

quotes, Python treats abcd as a variable name, and since it hasn't been defined, a NameError will be thrown.

8. What will be the output of the following Python code?

```
x = 1234
print(list(map(list, x)))
```

a) [1, 2, 3, 4]
b) [1234]
c) [[1], [2], [3], [4]]
d) none of the mentioned

Answer: d
Explanation: The given code will result in an error because map(list, x) expects x to be an iterable (like a string, list, etc.), but here x is an integer, which is not iterable. Since integers can't be directly used with map() in this way, Python will raise a TypeError.

9. What will be the output of the following Python code?

```
x = 1234
print(list(map(list, [x])))
```

a) [1, 2, 3, 4]
b) [1234]
c) [[1], [2], [3], [4]]
d) none of the mentioned

Answer: d
Explanation: Here, x is an integer (1234), and [x] is a list containing that integer: [1234]. When map(list, [x]) is called, it tries to apply list() to the integer 1234. But integers are not iterable, so this causes a TypeError. Hence, the code will throw an error and not produce any of the listed outputs.

10. What will be the output of the following Python code?

```
x = 'abcd'
print(list(map([], x)))
```

a) [‘a’, ‘b’, ‘c’, ‘d’]
b) [‘abcd’]
c) [[‘a’], [‘b’], [‘c’], [‘d’]]
d) none of the mentioned

Answer: d
Explanation: In map([], x), the first argument to map should be a function, but here it is an empty list [], which is not callable. This will raise a TypeError because Python expects a function to apply to each element of x. Hence, the output is an error and none of the given options match.

11. Is Python code compiled or interpreted?

a) Python code is only compiled
b) Python code is both compiled and interpreted
c) Python code is only interpreted
d) Python code is neither compiled nor interpreted

Answer: b
Explanation: Python code is both compiled and interpreted. When a Python script is run, it is first compiled into bytecode (.pyc files), which is a lower-level, platform-independent representation of the source code. This bytecode is then interpreted by the Python Virtual Machine (PVM), which executes the instructions line by line. Hence, Python involves a combination of compilation and interpretation.

12. Which of these is the definition for packages in Python?

a) A folder of python modules
b) A set of programs making use of Python modules
c) A set of main modules

d) A number of files containing Python definitions and statements

Answer: a
Explanation: In Python, a package is defined as a folder containing multiple Python modules, along with a special __init__.py file that indicates the directory is a package. Packages help in organizing related modules into a single directory hierarchy, making the codebase more modular and manageable.

13. Which of these is false about a package?

a) A package can have subfolders and modules
b) Each import package need not introduce a namespace
c) import folder.subfolder.mod1 imports packages
d) from folder.subfolder.mod1 import objects imports packages

Answer: b
Explanation: In Python, when you import a package, it introduces a namespace, which is crucial for keeping the package's components organized and avoiding name clashes. Each import ensures that the package or module resides within its namespace, which allows for clear organization and easy access to its components. Therefore, option "Each import package need not introduce a namespace" is incorrect.

1. Which of these definitions correctly describes a module?

a) Denoted by triple quotes for providing the specification of certain program elements
b) Design and implementation of specific functionality to be incorporated into a program
c) Defines the specification of how it is to be used
d) Any program that reuses code

Answer: b
Explanation: In Python, a module is essentially a file containing Python definitions and statements, which includes functions, classes, and variables. It is designed to provide specific functionality, which can then be reused across different parts of a program or even in different programs by importing the module.

2. Which of the following is not an advantage of using modules?

a) Provides a means of reuse of program code
b) Provides a means of dividing up tasks
c) Provides a means of reducing the size of the program
d) Provides a means of testing individual parts of the program

Answer: c
Explanation: Modules do not reduce the actual size of a program but help organize code better by dividing it into logical components. The main advantages include code reuse, task separation, and easier testing. The size of the overall program may stay the same or even increase slightly due to imports.

3. Program code making use of a given module is called a _____ of the module.

a) Client
b) Docstring
c) Interface
d) Modularity

Answer: a
Explanation: In Python, a program that uses functions or classes from a module is called a client of that module. The client interacts with the module through its public interface, using its functionalities without needing to know the internal implementation.

4. _____ is a string literal denoted by triple quotes for providing the specifications of certain program elements.
a) Interface
b) Modularity
c) Client
d) Docstring

Answer: d
Explanation: A docstring is a string literal enclosed in triple quotes (”’ or “””) used to document modules, functions, classes, or methods in Python. It provides specifications and helps users understand what the program element does.
5. Which of the following is true about top-down design process?
a) The details of a program design are addressed before the overall design
b) Only the details of the program are addressed
c) The overall design of the program is addressed before the details
d) Only the design of the program is addressed

Answer: c
Explanation: In the top-down design process, the system is broken down from the overall high-level design first, and then progressively into smaller, detailed components. This approach focuses on understanding the big picture before refining the details.
6. In top-down design every module is broken into same number of submodules.
a) True
b) False

Answer: b
Explanation: In top-down design, modules are broken down into submodules based on the functionality and complexity needed, and not necessarily into the same number of submodules. The division depends on logical separation, not equal partitioning.

7. All modular designs are because of a top-down design process.

a) True
b) False

Answer: b
Explanation: Not all modular designs result from a top-down approach. Modular design can also come from a bottom-up process or other design strategies, where modules are developed independently and then integrated. So, modularity isn't exclusive to top-down design.

8. What will be the output of the following Python code?

```
#mod1
def change(a):
    b=[x*2 for x in a]
    print(b)
#mod2
def change(a):
    b=[x*x for x in a]
    print(b)
from mod1 import change
from mod2 import change
#main
s=[1,2,3]
change(s)
```

a) [2,4,6]
b) [1,4,9]
c)
[2,4,6]
[1,4,9]
d) There is a name clash

Answer: d
Explanation: In the given code, both mod1 and mod2 define a function named change(). When from mod1 import change is followed by from mod2 import change, the second import overrides the first due to the same function name. This causes a name clash, as only one version of change() will be available in the current namespace, leading to confusion or unintended behavior.

9. Which of the following isn't true about main modules?

a) When a python file is directly executed, it is considered main module of a program
b) Main modules may import any number of modules
c) Special name given to main modules is: __main__
d) Other main modules can import main modules

Answer: d
Explanation: In Python, the main module is typically the module that is executed when the program starts. It is identified by the special name __main__. While a main module can import other modules, other main modules should not import the main module. This is because it could create circular dependencies and unintended behavior. Each module that is executed as the main module should be self-contained and not imported by other main modules.

10. Which of the following is not a valid namespace?
a) Global namespace
b) Public namespace
c) Built-in namespace
d) Local namespace

Answer: b
Explanation: In Python, namespaces are categorized into global, local, and built-in namespaces. The global namespace holds global variables, the built-in namespace contains Python's built-in functions and exceptions, and the local namespace stores local variables within functions. Public namespace is not a recognized type of namespace in Python, making it the incorrect choice.
11. Which of the following is false about "import modulename" form of import?
a) The namespace of imported module becomes part of importing module
b) This form of import prevents name clash
c) The namespace of imported module becomes available to importing module
d) The identifiers in module are accessed as: modulename.identifier

Answer: a
Explanation: In the "import modulename" form of import, the namespace of the imported module becomes available to the importing module, but it doesn't directly become a part of it. This form of import keeps the namespaces separate, allowing the importing module to access the identifiers of the imported module using the syntax modulename.identifier. By doing this, it prevents name clashes between the identifiers in the two modules.

12. Which of the following is false about “from-import” form of import?
a) The syntax is: from modulename import identifier
b) This form of import prevents name clash
c) The namespace of imported module becomes part of importing module
d) The identifiers in module are accessed directly as: identifier

Answer: b
Explanation: In the from-import form, specific identifiers (like functions, classes, or variables) are imported directly into the current namespace. While this allows direct access (e.g., identifier instead of module.identifier), it does not prevent name clashes. If two modules have identifiers with the same name and both are imported using from-import, the later one will overwrite the previous one
13. Which of the statements about modules is false?
a) In the “from-import” form of import, identifiers beginning with two underscores are private and aren’t imported
b) dir() built-in function monitors the items in the namespace of the main module
c) In the “from-import” form of import, all identifiers regardless of whether they are private or public are imported
d) When a module is loaded, a compiled version of the module with file extension .pyc is automatically produced

Answer: c
Explanation: In Python’s “from-import” form, identifiers that start with two underscores are considered private and are not imported by default. Only public identifiers are brought into the current namespace unless specifically stated otherwise. Therefore, it’s incorrect to say that all identifiers, including private ones, are imported automatically.

14. What will be the output of the following Python code?

```
from math import factorial
print(math.factorial(5))
```

a) 120
b) Nothing is printed
c) Error, method factorial doesn't exist in math module
d) Error, the statement should be: print(factorial(5))

Answer: d
Explanation: The code imports factorial directly from the math module, so you should call it as factorial(5), not math.factorial(5). Using math.factorial(5) causes an error because math itself wasn't imported—only the factorial function was.

15. What is the order of namespaces in which Python looks for an identifier?

a) Python first searches the global namespace, then the local namespace and finally the built-in namespace
b) Python first searches the local namespace, then the global namespace and finally the built-in namespace
c) Python first searches the built-in namespace, then the global namespace and finally the local namespace
d) Python first searches the built-in namespace, then the local namespace and finally the global namespace

Answer: b
Explanation: When Python encounters an identifier (like a variable or function name), it follows the LEGB rule to resolve it. It first looks in the Local namespace (inside the current function), then in the Enclosing namespace (if it's a nested function), followed by the Global namespace (top-level of the module), and finally the Built-in namespace (predefined

functions like len(), sum(), etc.). So, the correct search order for namespaces is: local → global → built-in.

1. What is returned by math.ceil(3.4)?

a) 3
b) 4
c) 4.0
d) 3.0

Answer: b
Explanation: math.ceil(3.4) returns the smallest integer greater than or equal to 3.4, which is 4. The ceil() function always rounds a number up to the nearest integer, and it returns the result as an integer, not a float.

2. What is the value returned by math.floor(3.4)?

a) 3
b) 4
c) 4.0
d) 3.0

Answer: a
Explanation: math.floor(3.4) returns the largest integer less than or equal to 3.4, which is 3. The floor() function always rounds a number down to the nearest integer and returns the result as an integer.

3. What will be the output of print(math.copysign(3, -1))?

a) 1
b) 1.0
c) -3
d) -3.0

Answer: d
Explanation: math.copysign(3, -1) returns a float value with the

magnitude of the first argument (3) and the sign of the second argument (-1). So, the result is -3.0 — the sign of -1 is applied to 3.

4. What is displayed on executing print(math.fabs(-3.4))?

a) -3.4
b) 3.4
c) 3
d) -3

Answer: b
Explanation: math.fabs(-3.4) returns the absolute value of the number as a float, even if the input is already a float. So, the output is 3.4.

5. Is the output of the function abs() the same as that of the function math.fabs()?

a) sometimes
b) always
c) never
d) none of the mentioned

Answer: a
Explanation: The built-in abs() function works with integers, floats, and complex numbers, while math.fabs() works only with floats and integers, always returning a float. So, for float or int inputs, both give the same numeric result, but abs() may return an int or float depending on input, while math.fabs() always returns a float. Hence, the outputs are the same only sometimes.

6. What is the value returned by math.fact(6)?

a) 720
b) 6
c) [1, 2, 3, 6]

d) error

Answer: d
Explanation: There is no function named math.fact() in the Python math module. The correct function for computing factorials is math.factorial(). Therefore, calling math.fact(6) will raise an AttributeError, resulting in an error.

7. What is the value of x if x = math.factorial(0)?
a) 0
b) 1
c) error
d) none of the mentioned

Answer: b
Explanation: The factorial of 0 is defined as 1 in mathematics. So, math.factorial(0) in Python returns 1, not an error or 0.
8. What is math.factorial(4.0)?
a) 24
b) 1
c) error
d) none of the mentioned

Answer: c
Explanation: The math.factorial() function in Python only accepts integers as its argument. Passing a float, like 4.0, will raise a ValueError. Even though 4.0 is mathematically equivalent to an integer, it is still treated as a float in Python.

9. What will be the output of print(math.factorial(4.5))?
a) 24
b) 120
c) error
d) 24.0

Answer: c
Explanation: The math.factorial() function only works with non-negative integers. Since 4.5 is a float, calling math.factorial(4.5) raises a ValueError. Factorials are undefined for non-integer values in this context.
10. What is math.floor(0o10)?
a) 8
b) 10
c) 0
d) 9

Answer: a
Explanation: 0o10 is the octal (base-8) representation of the number 8 in decimal. The math.floor() function returns the greatest integer less than or equal to the number, and since 8 is already an integer, math.floor(0o10) returns 8.
1. What does the function math.frexp(x) return?
a) a tuple containing the mantissa and the exponent of x
b) a list containing the mantissa and the exponent of x
c) a tuple containing the mantissa of x
d) a list containing the exponent of x

Answer: a
Explanation: The function math.frexp(x) breaks down the floating-point

number x into its mantissa and exponent, returning them as a tuple (mantissa, exponent) such that x = mantissa * 2**exponent and the mantissa is a float in the range [0.5, 1).

2. What is the result of math.fsum([.1 for i in range(20)])?

a) 2.0
b) 20
c) 2
d) 2.0000000000000004

Answer: a
Explanation: The function math.fsum() accurately sums floating-point numbers by minimizing precision errors. The list [0.1 for i in range(20)] contains twenty 0.1 values, and using math.fsum() returns the exact result 2.0. This is more precise than using the built-in sum() function.

3. What is the result of sum([.1 for i in range(20)]) in Python?

a) 2.0
b) 20
c) 2
d) 2.0000000000000004

Answer: d
Explanation: The expression sum([.1 for i in range(20)]) adds the float value 0.1 twenty times. While mathematically this should equal 2.0, Python (like most programming languages) uses binary floating-point representation, which cannot exactly represent 0.1. This leads to a small rounding error, resulting in 2.0000000000000004. To avoid such inaccuracies, Python provides math.fsum() for better precision with floats.

4. What is returned by math.isfinite(float('inf'))?

a) True

b) False
c) None
d) error

Answer: b
Explanation: The math.isfinite() function checks whether a number is finite (i.e., not infinity or NaN). float('inf') represents positive infinity, which is not finite, so math.isfinite(float('inf')) returns False.

5. What is returned by math.isfinite(float('nan'))?
a) True
b) False
c) None
d) error

Answer: b
Explanation: The function math.isfinite(x) returns True if x is a finite number, and False if x is infinity or NaN (Not a Number). When you use float('nan'), it creates a NaN value, which is not considered finite. Therefore, math.isfinite(float('nan')) returns False.
6. What is x if x = math.isfinite(float('0.0'))?
a) True
b) False
c) None
d) error

Answer: a
Explanation: math.isfinite(float('0.0')) checks if the number 0.0 is finite. Since 0.0 is a valid finite floating-point number, the function returns True.
7. What will be the output of the following Python code?

```
print(-float('inf') + float('inf'))
```
a) inf
b) nan
c) 0
d) 0.0

Answer: b
Explanation: In Python, adding -float(‘inf’) and float(‘inf’) results in an undefined operation (negative infinity plus positive infinity). This produces NaN (Not a Number), indicating an undefined or unrepresentable result in floating-point arithmetic.

8. What will be the output of the following Python code?

```
print(math.isinf(float('-inf')))
```
a) error, the minus sign shouldn’t have been inside the brackets
b) error, there is no function called isinf
c) True
d) False

Answer: c
Explanation: The function math.isinf() checks whether the given number is positive or negative infinity. Since float(‘-inf’) represents negative infinity, math.isinf(float(‘-inf’)) returns True.

9. What is the value of x if x = math.ldexp(0.5, 1)?

a) 1
b) 2.0
c) 0.5
d) none of the mentioned

Answer: d
Explanation: The value returned by ldexp(x, y) is x * (2 ** y). In the current case x is 1.0.

10. What is returned by math.modf(1.0)?

a) (0.0, 1.0)
b) (1.0, 0.0)
c) (0.5, 1)
d) (0.5, 1.0)

Answer: a
Explanation: The math.modf(x) function splits a floating-point number into its fractional and integer parts, both returned as floats. For x = 1.0, it returns (0.0, 1.0) — where 0.0 is the fractional part and 1.0 is the integer part.

1. What is the result of math.trunc(3.1)?

a) 3.0
b) 3
c) 0.1
d) 1

Answer: b
Explanation: The math.trunc(x) function returns the integer part of a number by truncating (removing) the fractional part, without rounding. So, math.trunc(3.1) returns 3, not 3.0.

2. What is the output of print(math.trunc('3.1'))?

a) 3
b) 3.0
c) error
d) none of the mentioned

Answer: c
Explanation: The output is an error because math.trunc() expects a numeric type (like int or float), but '3.1' is a string. Python raises a TypeError when trying to apply math.trunc() to a string, even if it looks like a number.

3. Which of the following is the same as math.exp(p)?

a) e ** p
b) math.e ** p
c) p ** e
d) p ** math.e

Answer: b
Explanation: The function math.exp(p) returns the value of e raised to the power p (i.e., e^p), where e is Euler's number (~2.718). This is equivalent to math.e ** p in Python.

4. What is returned by math.expm1(p)?

a) (math.e ** p) – 1
b) math.e ** (p – 1)
c) error
d) none of the mentioned

Answer: a
Explanation: The function math.expm1(p) returns the value of (e^p) – 1, where e is the base of natural logarithms. This function is more accurate for very small values of p than computing math.exp(p) – 1 directly.

5. What is the default base used when math.log(x) is found?

a) e
b) 10
c) 2

d) none of the mentioned

Answer: a
Explanation: The math.log(x) function in Python computes the natural logarithm of x, which means it uses base e (Euler's number, approximately 2.718). If you want to specify a different base, you can use math.log(x, base).

6. Which of the following aren't defined in the math module?
a) log2()
b) log10()
c) logx()
d) none of the mentioned

Answer: c
Explanation: The math module in Python provides logarithmic functions like math.log2() (log base 2), math.log10() (log base 10), and math.log() (natural log or log with custom base). However, there is no function named logx() in the module. Hence, logx() is not defined in the math module.

7. What is returned by int(math.pow(3, 2))?
a) 6
b) 9
c) error, third argument required
d) error, too many arguments

Answer: b
Explanation: The math.pow(3, 2) function returns the result of 3 raised to the power of 2, which is 9.0 (a float). Wrapping it with int() converts it to an integer, so the final result is 9.

8. What is output of print(math.pow(3, 2))?
a) 9
b) 9.0
c) None
d) None of the mentioned

Answer: b
Explanation: The function math.pow(3, 2) calculates 3 raised to the power of 2 and always returns a floating-point number. So the output is 9.0, not the integer 9.
9. What is the value of x if x = math.sqrt(4)?
a) 2
b) 2.0
c) (2, -2)
d) (2.0, -2.0)

Answer: b
Explanation: The math.sqrt() function returns the square root of a number as a float. So, math.sqrt(4) returns 2.0, not the integer 2. It also does not return negative roots or tuples.
10. What does math.sqrt(X, Y) do?
a) calculate the Xth root of Y
b) calculate the Yth root of X
c) error
d) return a tuple with the square root of X and Y

Answer: c
Explanation: The math.sqrt() function in Python takes only one argument, which must be a non-negative number. Passing two arguments like math.sqrt(X, Y) results in a TypeError. Therefore, it raises an error.

1. What will be the output of the following Python code?

```
import datetime
d=datetime.date(2016,7,24)
print(d)
```

a) Error
b) 2016-07-24
c) 2017-7-24
d) 24-7-2017

Answer: b
Explanation: The datetime.date(year, month, day) constructor creates a date object with the specified values. When printed, Python displays the date in the YYYY-MM-DD format by default. So, datetime.date(2016, 7, 24) outputs 2016-07-24.

2. What will be the output of the following Python code?

```
import datetime
d=datetime.date(2017,06,18)
print(d)
```

a) Error
b) 2017-06-18
c) 18-06-2017
d) 06-18-2017

Answer: a
Explanation: In Python, leading zeros in integer literals are not allowed (unless specifying an octal number using 0o prefix). Writing 06 is treated as a syntax error in Python 3. So, datetime.date(2017, 06, 18) raises a SyntaxError.

3. What will be the output of the following Python code if the system date is 18th August, 2016?

```
import datetime
tday=datetime.date.today()
print(tday.month)
```

a) August
b) Aug
c) 08
d) 18

Answer: c
Explanation: The tday object is a date object representing the current date. When tday.month is printed, it returns the month as an integer, not a string. If the system date is 18th August 2016, tday.month will return 8 (which is 08 if formatted), hence the correct answer should be 08.

4. What will be the output of the following Python code if the system date is 18th June, 2017 (Sunday)?

```
import datetime
tday=datetime.date.today()
print(tday)
```

a) 18-06-2017
b) 06-18-2017
c) 2017-06-18
d) Error

Answer: c
Explanation: The datetime.date.today() function returns the current local date in the format YYYY-MM-DD. So if the system date is 18th June 2017, the output will be 2017-06-18.

5. What will be the output of the following Python code if the system date is 18th June, 2017 (Sunday)?

```
import datetime
tday=datetime.date.today()
print(tday.weekday())
```

a) 6
b) 1
c) 0
d) 7

Answer: a
Explanation: The method tday.weekday() returns the day of the week as an integer, where Monday is 0 and Sunday is 6. So, if the date is 18th June 2017 (which was a Sunday), tday.weekday() will return 6.

6. What will be the output of the following Python code if the system date is 21st June, 2017 (Wednesday)?

```
import datetime
tday=datetime.date.today()
print(tday.isoweekday())
```

a) Wed
b) Wednesday
c) 2
d) 3

Answer: d
Explanation: The isoweekday() method returns the day of the week as an integer where Monday is 1 and Sunday is 7. Since 21st June 2017 was a Wednesday, the output will be 3.

7. Point out the error (if any) in the code shown below if the system date is 18th June, 2017?

```
import datetime
tday=datetime.date.today()
bday=datetime.date(2017,9,18)
till_bday=bday-tday
print(till_bday)
```

a) 3 months, 0:00:00
b) 90 days, 0:00:00
c) 3 months 2 days, 0:00:00
d) 92 days, 0:00:00

Answer: d
Explanation: The code subtracts two datetime.date objects — bday (18 Sept 2017) and tday (18 June 2017). This returns a timedelta object representing the number of days between the two dates.
From June 18 to September 18 is exactly 92 days, so the result is 92 days, 0:00:00.

8. The value returned when we use the function isoweekday() is _____ and that for the function weekday() is _____ if the system date is 19th June, 2017 (Monday).

a) 0,0
b) 0,1
c) 1,0
d) 1,1

Answer: c
Explanation: The function isoweekday() returns 1 for Monday, representing Monday as the first day of the week. The weekday() function returns 0 for Monday, where Monday is counted as day zero. Hence, for 19th June 2017 (Monday), the returned values are 1 and 0 respectively.

9. Which of the following will throw an error if used after the following Python code?

```
import datetime
tday=datetime.date.today()
bday=datetime.date(2017,9,18)
t_day=bday-tday
```

a) print(t_day.seconds)
b) print(t_day.months)
c) print(t_day.max)
d) print(t_day.resolution)

Answer: b
Explanation: The expression print(t_day.months) will throw an error because datetime.timedelta objects (like t_day) do not have a months attribute. They represent the difference between two dates in terms of days and seconds, not calendar months. Other attributes like seconds, max, and resolution are valid.

10. What will be the output of the following Python code if the system date is: 6/19/2017

```
import datetime
tday=datetime.date.today()
tdelta=datetime.timedelta(days=10)
print(tday+tdelta)
```

a) 2017-16-19
b) 2017-06-9
c) 2017-06-29
d) Error

Answer: c
Explanation: The code adds a timedelta of 10 days to the current date (19th June 2017). So, adding 10 days to 19th June results in 29th June 2017, which is displayed in the YYYY-MM-DD format as 2017-06-29.

1. The output of both of the print statements is the same.

```
import datetime
dt_1 = datetime.datetime.today()
dt_2 = datetime.datetime.now()
print(dt_1)
print(dt_2)
```

a) True
b) False

Answer: b
Explanation: Although both datetime.datetime.today() and datetime.datetime.now() return the current local date and time, they are not guaranteed to return exactly the same value down to the microsecond, since there's a very tiny time difference between the two function calls. Hence, their outputs will typically differ slightly, making the answer False.

2. Which of the following functions can be used to find the coordinated universal time, assuming that the datetime module has already been imported?

a) datetime.utc()
b) datetime.datetime.utc()
c) datetime.utcnow()
d) datetime.datetime.utcnow()

Answer: d
Explanation: To get the current Coordinated Universal Time (UTC), you use the utcnow() function from the datetime class within the datetime module. So the full call is datetime.datetime.utcnow().

3. What will be the output of the following Python code?

```
import time
print(time.asctime())
```

a) The number of hours passed since 1st January, 1970
b) The number of days passed since 1st January, 1970
c) The number of seconds passed since 1st January, 1970
d) The number of minutes passed since 1st January, 1970

Answer: c
Explanation: The time.time() function returns the current time as the number of seconds (including fractions of a second) since the Unix epoch, which is January 1, 1970, 00:00:00 UTC.

4. What will be the output of the following Python code, if the time module has already been imported?

```
def num(m):
        t1 = time.time()
        for i in range(0,m):
                print(i)
        t2 = time.time()
        print(str(t2-t1))

  num(3)
```

a)
0
1
2

The time taken for the execution of the code

b)

2

The time taken for the execution of the code

c)

0

1

2

UTC time

d)

2

UTC time

Answer: a

Explanation: The function num(m) prints numbers from 0 to m-1. It also calculates the time taken to execute the loop using time.time(), which returns the current time in seconds since the Unix epoch. The difference t2 – t1 gives the duration of the loop execution in seconds. Thus, the code prints 0, 1, 2, followed by the execution time.

5. What will be the output of the following Python code?

```
import time
print(time.asctime())
```

a) Current date only

b) UTC time

c) Current date and time

d) Current time only

Answer: c
Explanation: The function time.asctime() returns a string representing the current local time in the format: ‘Day Mon DD HH:MM:SS YYYY’. This includes both the date and time components, hence the output is the current date and time.

6. What will be the output of the following Python code?

```
import time
t=(2010, 9, 20, 8, 15, 12, 6)
print(time.asctime(t))
```

a) ‘20 Sep 2010 8:15:12 Sun’
b) ‘2010 20 Sept 08:15:12 Sun’
c) ‘Sun Sept 20 8:15:12 2010’
d) Error

Answer: d
Explanation: The time.asctime() function expects a time tuple with exactly 9 elements, but the given tuple has only 7. This causes Python to raise a TypeError due to the missing fields like yearday and dst. Therefore, the function call results in an error.

7. What will be the output of the following Python code?

```
import time
t=(2010, 9, 20, 8, 45, 12, 6, 0, 0)
print(time.asctime(t))
```

a) ‘Sep 20 2010 08:45:12 Sun’
b) ‘Sun Sep 20 08:45:12 2010’
c) ’20 Sep 08:45:12 Sun 2010’
d) ‘2010 20 Sep 08:45:12 Sun’

Answer: b
Explanation: The time.asctime(t) function converts a 9-element time

tuple into a readable string format: ‘Weekday Month Day HH:MM:SS Year’.
In the given tuple:
t = (2010, 9, 20, 8, 45, 12, 6, 0, 0)

- Year: 2010
- Month: 9 (September)
- Day: 20
- Hour: 8
- Minute: 45
- Second: 12
- Weekday: 6 (Sunday)

So, time.asctime(t) returns: ‘Sun Sep 20 08:45:12 2010’.

8. The sleep function (under the time module) is used to _____

a) Pause the code for the specified number of seconds
b) Return the specified number of seconds, in terms of milliseconds
c) Stop the execution of the code
d) Return the output of the code had it been executed earlier by the specified number of seconds

Answer: a
Explanation: The sleep function (under the time module) is used to pause the code for the specified number of seconds. The number of seconds is taken as an argument by this function.

9. What will be the output of the following Python code?

```
import time
for i in range(0,5):
        print(i)
        time.sleep(2)
```

a) After an interval of 2 seconds, the numbers 1, 2, 3, 4, 5 are printed all together

b) After an interval of 2 seconds, the numbers 0, 1, 2, 3, 4 are printed all together
c) Prints the numbers 1, 2, 3, 4, 5 at an interval of 2 seconds between each number
d) Prints the numbers 0, 1, 2, 3, 4 at an interval of 2 seconds between each number

Answer: d
Explanation: The time.sleep(2) function pauses execution for 2 seconds during each iteration of the loop. Since the range(0,5) generates values 0 to 4, each number is printed with a 2-second delay between them.

10. What will be the output if we try to extract only the year from the following Python code? (time.struct_time(tm_year=2017, tm_mon=6, tm_mday=25, tm_hour=18, tm_min=26, tm_sec=6, tm_wday=6, tm_yday=176, tm_isdst=0))

```
import time
t=time.localtime()
print(t)
```

a) t[1]
b) tm_year
c) t[0]
d) t_year

Answer: c
Explanation: The time.localtime() function returns a time.struct_time object, which is a tuple-like object. To extract the year, you can use index 0 (i.e., t[0]), since the structure follows the order: (tm_year, tm_mon, tm_mday, …). Therefore, t[0] correctly gives the year, which is 2017 in this case.

11. State whether true or false.

```
import time
s = time.time()
t = time.time()
print(s == t)
```
a) True
b) False

Answer: b
Explanation: The time.time() function returns the current time in seconds as a floating-point number. Since there's a very small delay between the two calls (s = time.time() and t = time.time()), the values of s and t will not be exactly equal, even if the difference is minimal. Thus, s == t evaluates to False.

1. To include the use of functions which are present in the random library, we must use the option:
a) import random
b) random.h
c) import.random
d) random.random

Answer: a
Explanation: To use functions from the random library in Python, you need to import the module first with the statement import random. This allows you to access its functions like random.random() or random.randint().
2. The output of the following Python code is either 1 or 2.
```
import random
print(random.randint(1,2))
```

a) True
b) False

Answer: a
Explanation: The function random.randint(a, b) returns a random integer N such that a ≤ N ≤ b. So, random.randint(1, 2) will randomly return either 1 or 2 — both inclusive. Hence, the output is either 1 or 2, making the statement True.

3. What will be the output of the following Python code?

```
import random
print(random.choice(2,3,4))
```

a) An integer other than 2, 3 and 4
b) Either 2, 3 or 4
c) Error
d) 3 only

Answer: c
Explanation: The random.choice() function expects a single sequence (like a list or tuple) as its argument, not multiple separate values. Passing random.choice(2,3,4) raises a TypeError because it receives multiple arguments instead of one iterable. The correct usage would be random.choice([2, 3, 4]).

4. What will be the output of the following Python code?

```
import random
print(random.choice([10.4, 56.99, 76]))
```

a) Error
b) Either 10.4, 56.99 or 76
c) Any number other than 10.4, 56.99 and 76

d) 56.99 only

Answer: b
Explanation: The function random.choice(sequence) returns a random element from the given sequence. In this case, the list [10.4, 56.99, 76] contains three elements. So, random.choice([10.4, 56.99, 76]) will randomly return either 10.4, 56.99, or 76.

5. What will be the output of the following Python function (random module has already been imported)?

```
import random
print(random.choice('sun'))
```

a) sun
b) u
c) either s, u or n
d) error

Answer: c
Explanation: The random.choice() function selects a random element from a sequence (like a list, tuple, or string). In this case, ‘sun’ is a string, and strings are sequences of characters. So random.choice(‘sun’) will return one of the characters: ‘s’, ‘u’, or ‘n’.

6. What will be the output of the following Python function, assuming that the random module has already been imported?

```
import random
print(random.uniform(3,4))
```

a) Error
b) Either 3 or 4
c) Any integer other than 3 and 4
d) Any decimal value between 3 and 4

Answer: d
Explanation: The function random.uniform(a, b) returns a random floating-point number N such that a ≤ N ≤ b. So, random.uniform(3, 4) will generate a random decimal value between 3 and 4, such as 3.274 or 3.987.

7. What will be the output of the following Python function if the random module has already been imported?

```
import random
print(random.randint(3.5,7))
```

a) Error
b) Any integer between 3.5 and 7, including 7
c) Any integer between 3.5 and 7, excluding 7
d) The integer closest to the mean of 3.5 and 7

Answer: a
Explanation: The function random.randint(a, b) requires both a and b to be integers. If you pass a float like 3.5 as an argument, Python will raise a TypeError. So, random.randint(3.5, 7) is invalid, and the code will result in an error.

8. Which of the following functions helps us to randomize the items of a list?

a) seed
b) randomise
c) shuffle
d) uniform

Answer: c
Explanation: The random.shuffle() function randomly rearranges the elements of a list in place. It's the standard way to mix up items in a list,

such as shuffling cards. Functions like seed, randomise, and uniform don't perform this task.

9. What will be the output of the following Python code?

```
import random

random.seed(3)
print(random.randint(1, 5))

random.seed(3)
print(random.randint(1, 5))
```

a) 3
b) 2
c) Any integer between 1 and 5, including 1 and 5
d) Any integer between 1 and 5, excluding 1 and 5

Answer: b
Explanation: The random.seed() function initializes the random number generator with a fixed seed value, making the random output deterministic (i.e., repeatable).
In the code:
random.seed(3)
random.randint(1,5) # Returns 2
random.seed(3)
random.randint(1,5) # Also returns 2
Since the seed is the same (3), the output of random.randint(1, 5) is reproducible and will always return 2. Therefore, the correct answer is 2.

10. What is the interval of the value generated by the function random.random(), assuming that the random module has already been

imported?
a) (0,1)
b) (0,1]
c) [0,1]
d) [0,1)

Answer: d
Explanation: The function random.random() generates a floating-point number greater than or equal to 0 and less than 1. This means it can return 0 but will never return 1, so the interval is [0, 1).

11. What will be the output of the following Python code?

```
import random
print(random.randrange(0,91,5))
```

a) 10
b) 18
c) 79
d) 95

Answer: a
Explanation: The random.randrange(0, 91, 5) function picks a random number from 0 up to 90 (91 excluded) in steps of 5. So, possible values are 0, 5, 10, …, 90. Among the options given, only 10 fits this pattern, making it the correct output.

12. Both the functions randint and uniform accept _____ parameters.
a) 0
b) 1
c) 3
d) 2

Answer: d
Explanation: Both functions belong to the random module and require two arguments to specify the range from which the random value is selected. For example, random.randint(a, b) returns an integer between a and b (inclusive), and random.uniform(a, b) returns a floating-point number between a and b.

13. The randrange function returns only an integer value.

a) True
b) False

Answer: a
Explanation: The random.randrange() function returns a randomly selected integer from the specified range. It does not return floating-point numbers.

14. What will be the output of the following Python code?

```
import random
print(random.randrange(1,100,10))
```

a) 32
b) 67
c) 91
d) 80

Answer: c
Explanation: The output of this function can be any value which is a multiple of 10, plus 1. Hence a value like 11, 21, 31, 41…91 can be the output. Also, the value should necessarily be between 1 and 100. The only option which satisfies this criteria is 91.

15. What will be the output of the following Python function, assuming that the random library has already been included?

```
import random
```

```
print(random.shuffle[1,2,24])
```
a) Randomized list containing the same numbers in any order
b) The same list, that is [1,2,24]
c) A list containing any random numbers between 1 and 24
d) Error

Answer: d
Explanation: The function shown above will result in an error because this is the incorrect syntax for the usage of the function shuffle(). The list should be previously declared and then passed to this function to get an output.
An example of the correct syntax:
```
l=['a','b','c','d']
random.shuffle(l)
print(l)
```
1. What the does random.seed(3) return?
a) True
b) None
c) 3
d) 1

Answer: b
Explanation: The function random.seed(3) is used to initialize the random number generator with a fixed seed value (in this case, 3) to produce reproducible results. However, it does not return any value — it only affects the internal state of the generator. Therefore, it returns None.
2. Which of the following cannot be returned by random.randrange(4)?
a) 0
b) 3
c) 2.3

d) 1

Answer: c
Explanation: The function random.randrange(4) returns a random integer from the range 0 to 3 (i.e., from 0 up to but not including 4). It cannot return floating-point numbers like 2.3. So, 2.3 is not a valid output.

3. Which of the following is equivalent to random.randrange(3)?

a) range(3)
b) random.choice(range(0, 3))
c) random.shuffle(range(3))
d) random.select(range(3))

Answer: b
Explanation: random.randrange(3) returns a random integer from 0 up to (but not including) 3 — i.e., 0, 1, or 2. This is equivalent to first creating the range range(0, 3) and then picking a random value from it using random.choice(). Hence, random.choice(range(0, 3)) gives the same result.

4. The function random.randint(4) can return only one of the following values. Which?

a) 4
b) 3.4
c) error
d) 5

Answer: c
Explanation: The function random.randint() requires two integer arguments: a lower and an upper bound. Calling random.randint(4) with

only one argument results in a TypeError because Python doesn't know the range to choose from.

5. Which of the following is equivalent to random.randint(3, 6)?

a) random.choice([3, 6])
b) random.randrange(3, 6)
c) 3 + random.randrange(3)
d) 3 + random.randrange(4)

Answer: d
Explanation: random.randint(3, 6) returns a random integer between 3 and 6, inclusive (i.e., 3, 4, 5, or 6). To achieve the same result using random.randrange, you can generate a number from 0 to 3 (using random.randrange(4)) and add 3 to it. This gives you values in the range 3 + 0 to 3 + 3, i.e., 3 to 6 — which is equivalent to random.randint(3, 6).

6. Which of the following will not be returned by random.choice("1 ,")?

a) 1
b) (space)
c) ,
d) none of the mentioned

Answer: d
Explanation: The function random.choice("1 ,") selects a random character from the string "1 ,", which contains three characterss: '1', ' ', and ','. All of the options — 1, space, and comma — are valid characters in the string, so none of them are excluded from being returned.

7. Which of the following will never be displayed on executing print(random.choice({0: 1, 2: 3}))?

a) 0
b) 1
c) error

d) none of the mentioned

Answer: c
Explanation: The function random.choice() requires a sequence like a list or string, but a dictionary is not a sequence. Passing a dictionary directly, as in random.choice({0: 1, 2: 3}), raises a TypeError. Therefore, no key or value like 0 will be displayed, and the function won't work as written.

8. What does random.shuffle(x) do when x = [1, 2, 3]?

a) error
b) do nothing, it is a placeholder for a function that is yet to be implemented
c) shuffle the elements of the list in-place
d) none of the mentioned

Answer: c
Explanation: random.shuffle(x) randomly rearranges the elements of the list x in-place, meaning it modifies the original list without returning a new one. For example, [1, 2, 3] may become [3, 1, 2].

9. Which type of elements are accepted by random.shuffle()?

a) strings
b) lists
c) tuples
d) integers

Answer: b
Explanation: random.shuffle() accepts only lists (or other mutable sequences) because it rearranges elements in-place. Immutable types like strings and tuples cannot be shuffled since their contents cannot be changed. Also, integers are not sequences, so they cannot be shuffled.

10. What is the range of values that random.random() can return?
a) [0.0, 1.0]
b) (0.0, 1.0]
c) (0.0, 1.0)
d) [0.0, 1.0)

Answer: d
Explanation: The function random.random() returns a floating-point number in the range [0.0, 1.0) — including 0.0 but excluding 1.0. It is used to generate random numbers for simulations and probabilistic models.
1. Which of the following functions can help us to find the version of python that we are currently working on?
a) sys.version
b) sys.version()
c) sys.version(0)
d) sys.version(1)

Answer: a
Explanation: The sys.version attribute (not a function) returns a string containing information about the Python version currently in use. For example, it might return something like “3.11.4 (main, …) …”. Since it’s an attribute, no parentheses are used.
2. Which of the following functions is not defined under the sys module?
a) sys.platform
b) sys.path
c) sys.readline
d) sys.argv

Answer: c
Explanation: The sys module provides functions and attributes like sys.platform, sys.path, and sys.argv, which are commonly used for platform identification, module search paths, and command-line arguments. However, sys.readline is not part of the sys module — it does not exist. Reading input is handled by functions like input() or sys.stdin.readline(), not sys.readline.

3. The output of the functions len(“abc”) and sys.getsizeof(“abc”) will be the same.

a) True
b) False

Answer: b
Explanation: The output of len(“abc”) and sys.getsizeof(“abc”) will not be the same. The len() function returns the number of characters in the string, which in this case is 3. On the other hand, sys.getsizeof() returns the total memory size in bytes that the string object occupies in memory, including additional overhead for managing the object. Because of this, the value returned by sys.getsizeof() is usually much larger than the character count, so the two outputs differ significantly.

4. What will be the output of the following Python code, if the code is run on Windows operating system?

```
import sys
if sys.platform[:2]== 'wi':
        print("Hello")
```

a) Error
b) Hello
c) No output

d) Junk value

Answer: b
Explanation: On Windows, sys.platform typically returns a string starting with ‘win’ (like ‘win32’). Since the code checks if the first two characters sys.platform[:2] equal ‘wi’, the condition is true on Windows, so it prints “Hello”.

5. What will be the output of the following Python code, if the sys module has already been imported?

```
sys.stdout.write("hello world")
```

a) helloworld
b) hello world10
c) hello world11
d) error

Answer: c
Explanation: The sys.stdout.write(“hello world”) function writes the string “hello world” directly to the standard output without adding a newline character. Unlike print(), it doesn’t automatically add anything extra. However, the output shows hello world11 because when sys.stdout.write() is used in an interactive environment like a Python shell, it returns the number of characters written — which is 11 for “hello world”. This return value (11) is then displayed by the interpreter, resulting in the output: hello world11.

6. What will be the output of the following Python code?

```
import sys
sys.stdin.readline()
Sanfoundry
```

a) ‘Sanfoundry\n’
b) ‘Sanfoundry’
c) ‘Sanfoundry10’
d) Error

Answer: a
Explanation: The method sys.stdin.readline() reads a full line from standard input including the newline character at the end. So, if the input is Sanfoundry followed by pressing Enter, the returned string will be ‘Sanfoundry\n’.

7. What will be the output of the following Python code?

```
import sys
eval(sys.stdin.readline())
"India"
```

a) India5
b) India
c) ‘India\n’
d) ‘India’

Answer: d
Explanation: sys.stdin.readline() reads the input line including the newline character, so it reads “India\n”. The eval() function evaluates this as a Python expression — since “India” (with quotes) is a valid string literal, eval() returns the string ‘India’ without the newline.

8. What will be the output of the following Python code?

```
import sys
eval(sys.stdin.readline())
Computer
```

a) Error
b) ‘Computer\n’

c) Computer8
d) Computer

Answer: a
Explanation: When eval() receives the input Computer without quotes, it tries to evaluate it as a variable or expression. Since Computer is not defined anywhere in the program, Python raises a NameError. Only quoted strings like “Computer” can be evaluated successfully as string literals. Hence, the code results in an error.

9. What will be the output of the following Python code?

```
import sys
sys.argv[0]
```

a) Junk value
b) ‘ ‘
c) No output
d) Error

Answer: b
Explanation: In Python, sys.argv[0] typically contains the name of the script that was executed. If you’re running code in an interactive environment or certain IDEs, it might return an empty string ‘ ‘. This can vary depending on the environment, but generally, it does not throw an error and returns either the script name or an empty string.

10. What will be the output of the following Python code?

```
import sys
sys.stderr.write(“hello”)
```

a) ‘hello’
b) ‘hello\n’
c) hello

d) hello5

Answer: d
Explanation: The statement sys.stderr.write(“hello”) writes the string “hello” to the standard error (stderr) without adding a newline. The write() function also returns the number of characters written, which is 5 in this case. If you print the return value of sys.stderr.write(“hello”), you see hello5 because “hello” is printed to stderr, and 5 (the return value) is printed to stdout.

11. What will be the output of the following Python code?

```
import sys
sys.argv
```

a) ‘ ‘
b) [ ]
c) [‘ ‘]
d) Error

Answer: c
Explanation: sys.argv is a list containing the command-line arguments passed to the Python script. By default, it always contains at least one element — the name of the script being executed. When run interactively or without arguments, sys.argv will be a list with a single empty string [”].

12. To obtain a list of all the functions defined under sys module, which of the following functions can be used?

a) print(sys)
b) print(dir.sys)
c) print(dir[sys])
d) print(dir(sys))

Answer: d
Explanation: The built-in dir() function returns a list of all the attributes and functions defined in a module. To list everything in the sys module, we use dir(sys) inside the print() function, i.e., print(dir(sys)).

13. The output of the function len(sys.argv) is _____

a) Error
b) 1
c) 0
d) Junk value

Answer: b
Explanation: sys.argv is a list that contains the command-line arguments passed to a Python script. It always includes at least one element—the script name—so len(sys.argv) is always at least 1, even if no additional arguments are provided.

1. What does os.name contain?

a) the name of the operating system dependent module imported
b) the address of the module os
c) error, it should've been os.name()
d) none of the mentioned

Answer: a
Explanation: os.name is a string that indicates the name of the operating system dependent module imported by Python (like 'posix', 'nt', or 'java'). It helps identify the underlying OS environment.

2. What does print(os.geteuid()) print?

a) the group id of the current process
b) the user id of the current process
c) both the group id and the user of the current process

d) none of the mentioned

Answer: b
Explanation: The function os.geteuid() returns the effective user ID of the current process, which is a unique number identifying the user running the process. It does not provide the group ID or both.

3. What does os.getlogin() return?

a) name of the current user logged in
b) name of the superuser
c) gets a form to login as a different user
d) all of the mentioned

Answer: a
Explanation: os.getlogin() returns the name of the user currently logged into the controlling terminal of the process. It does not provide superuser information or login forms.

4. What does os.close(f) do?

a) terminate the process f
b) terminate the process f if f is not responding
c) close the file descriptor f
d) return an integer telling how close the file pointer is to the end of file

Answer: c
Explanation: os.close(f) closes the low-level file descriptor f (an integer), not a file object. It is used when working with files at the OS level (e.g., after using os.open()). It does not terminate a process.

5. What does os.fchmod(fd, mode) do?

a) change permission bits of the file
b) change permission bits of the directory
c) change permission bits of either the file or the directory

d) none of the mentioned

Answer: a
Explanation: os.fchmod(fd, mode) changes the permission bits (like read, write, execute) of the file referred to by the file descriptor fd. It works only for files, not directories directly.

6. Which of the following functions can be used to read data from a file using a file descriptor?

a) os.reader()
b) os.read()
c) os.quick_read()
d) os.scan()
Answer: b
Explanation: os.read(fd, n) reads up to n bytes from the file associated with the file descriptor fd. It's a low-level function used for reading data directly from a file or device.

7. Which of the following returns a string that represents the present working directory?

a) os.getcwd()
b) os.cwd()
c) os.getpwd() d) os.pwd()

Answer: a
Explanation: os.getcwd() returns the current working directory as a string. It's a standard function in the os module used to find out where your Python script is running.

8. What does os.link() do?

a) create a symbolic link
b) create a hard link
c) create a soft link

d) none of the mentioned

Answer: b
Explanation: os.link(src, dst) creates a hard link pointing from dst to the file at src. A hard link makes dst another name for the same file content on disk, sharing the same inode.

9. Which of the following can be used to create a directory?

a) os.mkdir()
b) os.creat_dir()
c) os.create_dir()
d) os.make_dir()

Answer: a
Explanation: os.mkdir(path) is used to create a new directory at the specified path. It's a standard function in the os module for directory creation in Python.

10. Which of the following can be used to create a symbolic link?

a) os.symlink()
b) os.symb_link()
c) os.symblin()
d) os.ln()

Answer: a
Explanation: os.symlink(source, link_name) creates a symbolic (soft) link pointing to source with the name link_name. It's used to link to files or directories without duplicating their content.

1. What will be the output shape of the following Python code?

```
import turtle
t=turtle.Pen()
for i in range(0,4):
```

```
        t.forward(100)
        t.left(120)
```

a) square
b) rectangle
c) triangle
d) kite

Answer: c
Explanation: The code uses the Turtle graphics library to draw a shape. The t.left(120) command turns the turtle left by 120 degrees after moving forward by 100 units. Repeating this 4 times causes the turtle to trace an equilateral triangle, as turning by 120 degrees three times completes a 360-degree rotation, forming a triangle.

2. The number of lines drawn in each case, assuming that the turtle module has been imported:

```
Case 1:
for i in range(0,10):
        turtle.forward(100)
        turtle.left(90)
Case 2:
for i in range(1,10):
        turtle.forward(100)
        turtle.left(90)
```

a) 10, 9
b) 9, 10
c) 9, 9
d) 10, 10

Answer: a
Explanation: In Case 1, the loop runs from i = 0 to i = 9, which means it executes 10 times, drawing 10 lines. In Case 2, the loop runs from i = 1 to i = 9, executing 9 times, so it draws 9 lines. Hence, the number of lines drawn are 10 and 9 respectively.

3. The command which helps us to reset the pen (turtle):

a) turtle.reset
b) turtle.penreset
c) turtle.penreset()
d) turtle.reset()

Answer: d
Explanation: turtle.reset() clears the drawing, resets the turtle's state (position, heading, etc.) to the default, effectively resetting the pen. It must be called with parentheses since it's a function.

4. Fill in the blank such that the following Python code results in the formation of an inverted, equilateral triangle.

```
import turtle
t=turtle.Pen()
for i in range(0,3):
        t.forward(150)
        t.right(_____)
```

a) -60
b) 120
c) -120
d) 60

Answer: b
Explanation: To form an inverted equilateral triangle using turtle graphics, the turtle needs to turn right by 120 degrees after each side.

This makes each internal angle 60°, completing a 3-sided figure pointing downward.

5. What will be the output shape of the following Python code?

```
import turtle
t=turtle.Pen()
for i in range(1,4):
        t.forward(60)
        t.left(90)
```

a) Rectangle
b) Trapezium
c) Triangle
d) Square

Answer: d
Explanation: The turtle moves forward 60 units and turns left 90 degrees three times. Although only three sides are explicitly drawn, the fourth side is implied as the turtle completes the square's shape by aligning back to the starting point. The turning angles of 90° suggest a square structure.

6. What will be the output of the following Python code?

```
import turtle
t=turtle.Pen()
for i in range(0,4):
        t.forward(100)
        t.left(90)

t.penup()
t.left(90)
t.forward(200)
for i in range(0,4):
```

```
        t.forward(100)
        t.left(90)
```
a) Error
b) 1 square
c) 2 squares, at a separation of100 units, joined by a straight line
d) 2 squares, at a separation of 100 units, without a line joining them
Answer: b
Explanation: The code first draws a square by moving forward and turning left 90 degrees four times. Then, t.penup() lifts the pen, so when the turtle moves forward 200 units and turns, it does so without drawing any lines. Since the pen is never put down again using t.pendown(), the second loop moves the turtle but does not draw the second square. Therefore, only one square is visible.

7. Which of the following functions does not accept any arguments?

a) position
b) fillcolor
c) goto
d) setheading()

Answer: a
Explanation: The position() function (from the turtle module) does not require any arguments. It simply returns the current position of the turtle as a tuple (x, y). In contrast, fillcolor, goto, and setheading() all require arguments to specify color, coordinates, or direction respectively.

8. What will be the output of the following Python code?

```
import turtle
t=turtle.Pen()
t.goto(300,9)
t.position()
```

a) 300.00, 9.00
b) 9, 300
c) 300, 9
d) 9.00, 300.00

Answer: a
Explanation: The t.goto(300, 9) command moves the turtle to the position (300, 9). When t.position() is called, it returns the current position of the turtle as a tuple with float values. Hence, the output will be (300.00, 9.00) in float format.

9. What will be the output of the following Python code?

```
import turtle
t=turtle.Pen()
for i in range(0,5):
        t.left(144)
        t.forward(100)
```

a) Trapezium
b) Parallelepiped
c) Tetrahedron
d) Star

Answer: d
Explanation: The code uses a loop to draw 5 lines, each turning left by 144 degrees and moving forward 100 units. This specific turning angle and repetition count is a classic method to draw a five-pointed star using the turtle graphics module. The shape formed is not a regular polygon, but a star due to the crossing lines created by turning 144 degrees each time.

10. What will be the output of the following Python functions?

```
import turtle
```

```
t=turtle.Pen()
for i in range(0,3):
        t.forward(100)
        t.left(120)

t.back(100)
for i in range(0,3):
        t.forward(100)
        t.left(120)
```

a) Error
b) Two triangles, joined by a straight line
c) Two triangles, joined at one vertex
d) Two separate triangles, not connected by a line

Answer: c
Explanation: The first for loop draws an equilateral triangle with sides of 100 units. The command t.back(100) moves the turtle backward along its current heading without changing its direction, positioning it to start the next triangle from the same vertex. The second triangle begins from this new point and follows the same turning pattern, resulting in another triangle connected at the shared vertex.

1. What will be the output of the following Python code?

```
import turtle
t=turtle.Pen()
t.color(0,0,1)
t.begin_fill()
t.circle(15)
t.end_fill()
```

a) Error
b) A circle filled in with the colour red

c) A circle filled in with the colour blue
d) A circle filled in with the colour green

Answer: c
Explanation: The t.color(0, 0, 1) function sets the turtle's pen and fill color using RGB values, where (0, 0, 1) represents pure blue (with red and green set to zero). The begin_fill() and end_fill() functions are used to fill the shape drawn between them. In this case, a circle with a radius of 15 units is filled with the specified color. As a result, the output is a blue-filled circle.

2. Which of the following functions can be used to make the arrow black?

a) turtle.color(0,1,0)
b) turtle.color(1,0,0)
c) turtle.color(0,0,1)
d) turtle.color(0,0,0)

Answer: d
Explanation: The turtle.color(r, g, b) function sets the pen and fill color using RGB values where each value ranges from 0 to 1. The function turtle.color(0,1,0) will make the arrow green. The function turtle.color(1,0,0) will make the arrow red. The function turtle.color(0,0,1) will make the arrow blue. The function turtle.color(0,0,0) will make the arrow black.

3. What will be the output of the following Python code?

```
t=turtle.Pen()
t.color(1,1,1)
t.begin_fill()
for i in range(0,3):
        t.forward(100)
```

```
        t.right(120)
          t.end_fill()
```

a) Blank page
b) A triangle filled in with the colour yellow
c) A triangle which is not filled in with any colour
d) Error

Answer: a
Explanation: The t.color(1,1,1) sets the pen and fill color to white using RGB values, and the turtle graphics window also has a white background by default. So, even though a triangle is drawn and filled, it is the same color as the background, making it invisible, resulting in what appears to be a blank page.

4. What will be the output of the following Python code?

```
import turtle
t=turtle.Pen()
t.color(0,1,0)
t.begin_fill()
for i in range(0,4):
        t.forward(100)
        t.right(90)
```

a) A square filled in with the colour green
b) A square outlined with the colour green
c) Blank canvas
d) Error

Answer: b
Explanation: The code sets the pen color to green using t.color(0,1,0) and

begins a fill with t.begin_fill(), but does not call t.end_fill(), which is necessary to actually fill the shape. As a result, only the outline of the square is drawn in green, and the shape is not filled.

5. In which direction is the turtle pointed by default?

a) North
b) South
c) East
d) West

Answer: c
Explanation: By default, in Python's turtle module, the turtle starts at the center of the screen and faces East (i.e., towards the right side of the screen). This is the 0-degree direction in turtle graphics.

6. The command used to set only the x coordinate of the turtle at 45 units is:

a) reset(45)
b) setx(45)
c) xset(45)
d) xreset(45)

Answer: b
Explanation: The setx() function in the turtle module is used to set the turtle's x-coordinate to a specified value while keeping the y-coordinate unchanged. So, setx(45) moves the turtle horizontally to x = 45.

7. Which of the following functions returns a value in degrees, counterclockwise from the horizontal right?

a) heading()
b) degrees()
c) position()

d) window_height()

Answer: a
Explanation: The heading() function in the turtle module returns the current orientation of the turtle in degrees. The angle is measured counterclockwise from the horizontal right (which is 0 degrees).

8. What will be the output of the following Python code?

```
import turtle
t=turtle.Pen()
t.right(90)
t.forward(100)
t.heading()
```

a) 0.0
b) 90.0
c) 270.0
d) 360.0

Answer: c
Explanation: The turtle starts facing east (0 degrees). After turning right by 90 degrees, it faces south (270 degrees, since angles increase counterclockwise). Moving forward doesn't change the heading, so t.heading() returns 270.0.

9. What will be the output of the following Python code?

```
import turtle
t=turtle.Pen()
t.clear()
t.isvisible()
```

a) Yes
b) True
c) No

d) False

Answer: b
Explanation: The clear() method clears all drawings made by the turtle but does not hide the turtle itself. Therefore, t.isvisible() returns True because the turtle is still visible on the screen.

10. What will be the output of the following Python code?

```
import turtle
t=turtle.Pen()
t.forward(100)
t.left(90)
t.clear()
t.position()
```

a) 0.00, 90.00
b) 0.00, 0.00
c) 100.00, 90.00
d) 100.00, 100.00

Answer: d
Explanation: The clear() method only erases the drawing but does not move the turtle or change its position. After moving forward 100 units and turning left 90 degrees, the turtle's position remains at (100, 0). So, calling position() will return the current coordinates without any change from clear().

1. Which of the following functions results in an error?

a) turtle.shape("turtle")
b) turtle.shape("square")
c) turtle.shape("triangle")
d) turtle.shape("rectangle")

Answer: d
Explanation: The turtle.shape() function accepts only specific shape names like “turtle”, “square”, and “triangle”. Since “rectangle” is not a valid predefined shape in the turtle module, calling turtle.shape(“rectangle”) results in an error.

2. What will be the output of the following Python code?

```
import turtle
t=turtle.Pen
t.tilt(75)
t.forward(100)
```

a) A straight line of 100 units tiled at 75 degrees from the horizontal
b) A straight line of 100 units tilted at 15 degrees from the horizontal
c) A straight line of 100 units lying along the horizontal
d) Error

Answer: d
Explanation: In the code, t = turtle.Pen assigns the class Pen itself to t instead of creating an instance (which should be t = turtle.Pen()). Also, the tilt() method is not a standard method of the turtle pen object. Due to these issues, the code will cause an error when executed.

3. What will be the output of the following Python code?

```
import turtle
t=turtle.Pen()
t.backward(100)
t.penup()
t.right(45)
t.isdown()
```

a) True
b) False

c) Yes
d) No

Answer: b
Explanation: The method penup() lifts the pen, so the turtle stops drawing while moving. Once penup() is called, the pen is no longer down. We did not use the pendown() function to lower the pen again. Therefore, when we call turtle.isdown(), it returns False because the pen is up.

4. The function used to alter the thickness of the pen to 'x' units:

a) turtle.width(x)
b) turtle.span(x)
c) turtle.girth(x)
d) turtle.thickness(x)

Answer: a
Explanation: The function turtle.width(x) is used to set the thickness of the turtle's pen to x units, controlling how thick the lines drawn by the turtle will be.

5. What will be the output of the following Python code?

```
import turtle
t=turtle.Pen()
t.goto(100,0)
t.towards(0,0)
```

a) 0.0
b) 180.0
c) 270.0
d) 360.0

Answer: b
Explanation: The turtle starts at the default position (0,0). After moving to (100,0) with t.goto(100,0), the turtle's position is at x=100, y=0. The function t.towards(0,0) returns the angle (in degrees) from the current position (100,0) toward the point (0,0). Since the target point is directly to the left, the angle is 180 degrees.

6. What will be the output of the following Python code?

```
import turtle
t=turtle.Pen()
t.position()
(100.00,0.00)
t.goto(100,100)
t.distance(100,0)
```

a) 0.0
b) Error
c) 100.0, 100.0
d) 100.0

Answer: d
Explanation: Initially, t.position() returns the turtle's current position, which is (100.00, 0.00), indicating that the turtle is already at (100, 0). Then, t.goto(100, 100) moves the turtle to the position (100, 100). Finally, t.distance(100, 0) calculates the vertical distance between the current position (100, 100) and the point (100, 0), which is 100 units. Therefore, the output is 100.0.

7. The output of the following Python code will result in a shape similar to the alphabet _____

```
import turtle
t=turtle.Turtle()
```

```
t1=turtle.Turtle()
t.left(45)
t1.left(135)
t.forward(100)
t1.forward(100)
```

a) V
b) Inverted V
c) X
d) T

Answer: a
Explanation: The two turtles start facing different directions — one is turned left by 45°, and the other by 135°. When both move forward the same distance, their paths form two straight lines that meet at a point, creating a shape similar to the letter “V”.

8. The output of the following Python code is similar to the alphabet _____

```
import turtle
t=turtle.Pen()
t1=turtle.Pen()
t2=turtle.Pen()
t.forward(100)
t1.forward(100)
t2.forward(100)
t1.left(90)
t1.forward(75)
t2.right(90)
t2.forward(75)
```

a) X
b) N

c) T
d) M

Answer: c
Explanation: The first turtle draws a vertical line (forward 100), the second turtle moves forward then turns left 90° and moves 75 (forming the top horizontal line), and the third turtle moves forward then turns right 90° and moves 75. Together, these lines form a shape resembling the letter T.

9. The following Python code will result in an error.

```
import turtle
t=turtle.Pen()
t.speed(-45)
t.circle(30)
```

a) True
b) False

Answer: a
Explanation: The speed() function in the turtle module accepts values from 0 to 10 (or special values like 0 for fastest). Passing a negative value like -45 is invalid and will cause an error.

10. What will be the output of the following Python code?

```
import turtle
t=turtle.Pen()
t.goto(50,60)
t1=t.clone()
t1.ycor()
```

a) 0.0
b) 50.0
c) 60.0

d) Error

Answer: c
Explanation: The turtle t moves to the position (50, 60). When t1 = t.clone() is called, t1 is created at the same position as t, which is (50, 60). So, t1.ycor() returns the y-coordinate of t1, which is 60.0.

11. What will be the output shape of the following Python code?

```
import turtle
t=turtle.Pen()
for i in range(0,6):
        t.forward(100)
        t.left(60)
```

a) Hexagon
b) Octagon
c) Pentagon
d) Heptagon

Answer: a
Explanation: The loop runs 6 times, and after each forward movement of 100 units, the turtle turns left by 60 degrees. Since the sum of external angles for any polygon is 360°, dividing 360° by 60° gives 6 sides, which forms a hexagon.

12. What will be the output of the following Python code?

```
import turtle
t=turtle.Pen()
t.resizemode("user")
t.resizemode()
```

a) user
b) auto
c) nonresize

d) error

Answer: c
Explanation: The resizemode() function in the turtle module can both set and get the current resizing behavior of the turtle. When t.resizemode(“user”) is called, it sets the resizing mode to “user”, allowing manual control over the turtle’s size. A subsequent call to t.resizemode() returns the current mode, which is “user”. This is one of three valid modes: “auto”, “user”, and “noresize” (default).

1. The process of pickling in Python includes:

a) conversion of a list into a datatable
b) conversion of a byte stream into Python object hierarchy
c) conversion of a Python object hierarchy into byte stream
d) conversion of a datatable into a list

Answer: c
Explanation: Pickling in Python refers to the process of serializing a Python object structure (like lists, dictionaries, or custom objects) into a byte stream so that it can be saved to a file or transmitted over a network. This byte stream can later be “unpickled” to reconstruct the original object using the pickle module.

2. To sterilize an object hierarchy, the _____ function must be called. To desterilize a data stream, the _____ function must be called.

a) dumps(), undumps()
b) loads(), unloads()
c) loads(), dumps()
d) dumps(), loads()

Answer: d
Explanation: To serialize (convert to byte stream) a Python object hierarchy, you use pickle.dumps() or pickle.dump() (for writing to a file). To deserialize (reconstruct the object from byte stream), you use pickle.loads() or pickle.load(). These are the core functions of the pickle module for object persistence in Python.

3. Pick the correct statement regarding pickle and marshal modules.

a) The pickle module supports primarily .pyc files whereas marshal module is used to sterilize Python objects
b) The pickle module keeps track of the objects that have already been sterilized whereas the marshal module does not do this
c) The pickle module cannot be used to sterilize user defined classes and their instances whereas marshal module can be used to perform this task
d) The format of sterilization of the pickle module is not guaranteed to be supported across all versions of Python. The marshal module sterilization is compatible across all the versions of Python

Answer: b
Explanation: The pickle module is more powerful and flexible than marshal. It can serialize a wider range of Python objects and also keeps track of objects already serialized to avoid infinite recursion with self-referencing objects. On the other hand, the marshal module is mainly used for writing and reading Python's compiled bytecode (.pyc files) and does not support reference tracking or custom classes.

4. Which of the following attributes in the pickle module returns the highest supported protocol number for pickling in the Python version?

pickle.HIGHEST_PROTOCOL

a) pickle.PROTOCOL_VERSION
b) pickle.MAX_PROTOCOL

c) pickle.HIGHEST_PROTOCOL
d) pickle.TOP_PROTOCOL

Answer: c
Explanation: pickle.HIGHEST_PROTOCOL gives the highest protocol number available for the current version of Python, ensuring the most efficient pickling.

5. Which of the following Python codes will result in an error?

```
object = 'a'
```

a) pickle.dumps(object)
b) pickle.dumps(object, 3)
c) pickle.dumps(object, 3, True)
d) pickle.dumps('a', 2)

Answer: c
Explanation: The pickle.dumps() function's third parameter (fix_imports) must be passed as a keyword argument, not positionally. So, pickle.dumps(object, 3, True) causes a TypeError because True is given as a positional argument. The correct way is pickle.dumps(object, 3, fix_imports=True).

6. Which of the following functions can be used to find the protocol version of the pickle module currently being used?

a) pickle.DEFAULT
b) pickle.CURRENT
c) pickle.CURRENT_PROTOCOL
d) pickle.DEFAULT_PROTOCOL

Answer: d
Explanation: The function pickle.DEFAULT_PROTOCOL can be used to find

the protocol version of the pickle module currently being used by the system.

7. The output of the following two Python codes is exactly the same.

```
import pickle

object = 'a'

code1 = pickle.dumps('a', 3)
code2 = pickle.dumps(object, 3)

print(code1 == code2)
```

a) True
b) False

Answer: a
Explanation: Since object is assigned the string ‘a’, both pickle.dumps(‘a’, 3) and pickle.dumps(object, 3) serialize the same string. Therefore, the resulting byte outputs are identical, making the comparison True.

8. Which of the following functions can accept more than one positional argument?

a) pickle.dumps
b) pickle.loads
c) pickle.loadfile
d) pickle.load

Answer: a
Explanation: pickle.dumps() can accept more than one positional argument, such as the object to pickle and the protocol version (e.g., pickle.dumps(obj, 3)). Other functions like pickle.loads() and pickle.load()

accept only one positional argument; additional parameters must be passed as keyword arguments.

9. Which of the following functions raises an error when an unpicklable object is encountered by Pickler?

a) pickle.PickleError
b) pickle.PicklingError
c) pickle.UnpickleError
d) pickle.UnpicklingError

Answer: b
Explanation: The pickle.PicklingError is raised when the Pickler encounters an object that cannot be pickled (e.g., open file handles, sockets, or lambda functions).

- pickle.UnpicklingError is raised when loading (not saving) fails.
- pickle.PickleError is the base class for both pickling and unpickling errors, but it's not raised directly.

10. The pickle module defines _____ exceptions and exports _____ classes.

a) 2, 3
b) 3, 4
c) 3, 2
d) 4, 3

Answer: c
Explanation: The pickle module defines 3 exceptions (PickleError, PicklingError, and UnpicklingError) to handle serialization errors. It also exports 2 classes (Pickler and Unpickler) used for customizing the pickling and unpickling process.

11. Which of the following cannot be pickled?

a) Functions which are defined at the top level of a module with lambda

b) Functions which are defined at the top level of a module with def
c) Built-in functions which are defined at the top level of a module
d) Classes which are defined at the top level of a module

Answer: a
Explanation: Lambda functions cannot be pickled because they are anonymous and don't have a qualified name that can be referenced by the pickle module. Only functions, classes, and objects defined at the top level of a module using def or class can be pickled reliably.

12. If __getstate__() returns _____ the __setstate__() module will not be called on pickling.

a) True value
b) False value
c) ValueError
d) OverflowError

Answer: b
Explanation: If __getstate__() returns a false value (like None, False, 0, or an empty container), the __setstate__() method will not be called during unpickling. This is because there's no state to restore, so Python skips the call.

13. Lambda functions cannot be pickled because:

a) Lambda functions only deal with binary values, that is, 0 and 1
b) Lambda functions cannot be called directly
c) Lambda functions cannot be identified by the functions of the pickle module
d) All lambda functions have the same name, that is, <lambda>

Answer: d
Explanation: Lambda functions cannot be pickled because they are

anonymous and all have the same name (<lambda>). This makes it impossible for the pickle module to reference and recreate them properly during unpickling.

14. The module _____ is a comparatively faster implementation of the pickle module.

a) cPickle
b) nPickle
c) gPickle
d) tPickle

Answer: a
Explanation: cPickle is a faster, C-implemented version of the pickle module that improves serialization speed while maintaining the same interface. It's commonly used in Python 2; in Python 3, the standard pickle module uses the C implementation by default.

15. The copy module uses the _____ protocol for shallow and deep copy.

a) pickle
b) marshal
c) shelve
d) copyreg

Answer: a
Explanation: The copy module in Python uses the pickle protocol to perform deep copying of objects. This allows it to serialize and reconstruct complex objects, ensuring that the copied object is a completely independent duplicate of the original.

1. Which module in Python supports regular expressions?

a) re
b) regex

c) pyregex
d) none of the mentioned

Answer: a
Explanation: The re module in Python provides support for regular expressions, enabling pattern matching and text manipulation. It is the standard library module used for regex operations in Python.

2. Which of the following creates a pattern object?

a) re.create(str)
b) re.regex(str)
c) re.compile(str)
d) re.assemble(str)

Answer: c
Explanation: The function re.compile(str) compiles a regular expression pattern into a pattern object, which can then be used for matching using its methods like .match(), .search(), etc. This improves performance if the same pattern is used multiple times.

3. What does the function re.match do?

a) matches a pattern at the start of the string
b) matches a pattern at any position in the string
c) such a function does not exist
d) none of the mentioned

Answer: a
Explanation: re.match() checks for a match only at the beginning of the string. If the pattern is found right at the start, it returns a match object; otherwise, it returns None. For matching patterns anywhere in the string, re.search() is used instead.

4. What does the function re.search do?

a) matches a pattern at the start of the string
b) matches a pattern at any position in the string
c) such a function does not exist
d) none of the mentioned

Answer: b
Explanation: re.search() scans through the entire string and returns a match object for the first occurrence of the pattern anywhere in the string. Unlike re.match(), it does not require the pattern to be at the start.

5. What will be the output of the following Python code?

```
import re
sentence = 'we are humans'
matched = re.match(r'(.*) (.*?) (.*)', sentence)
print(matched.groups())
```

a) (‘we’, ‘are’, ‘humans’)
b) (we, are, humans)
c) (‘we’, ‘humans’)
d) ‘we are humans’

Answer: a
Explanation: The regex pattern (.*) (.*?) (.*) breaks the sentence into three groups separated by spaces:

- (.*) — matches any characters greedily until the last space (captures ‘we’ in this case because of the lazy second group),
- (.*?) — matches any characters lazily until the next space (captures ‘are’),
- (.*) — matches the rest of the string (captures ‘humans’).

So, matched.groups() returns a tuple of three strings: (‘we’, ‘are’, ‘humans’).

6. What will be the output of the following Python code?

```
import re
sentence = 'we are humans'
matched = re.match(r'(.*) (.*?) (.*)', sentence)
print(matched.group())
```

a) (‘we’, ‘are’, ‘humans’)
b) (we, are, humans)
c) (‘we’, ‘humans’)
d) we are humans

Answer: d
Explanation: matched.group() returns the entire matched portion of the string, which is “we are humans”. To get the individual parts captured by the groups, you would use matched.groups() or specify group numbers like matched.group(1).

7. What will be the output of the following Python code?

```
import re
sentence = 'we are humans'
matched = re.match(r'(.*) (.*?) (.*)', sentence)
print(matched.group(2))
```

a) ‘are’
b) ‘we’
c) ‘humans’
d) ‘we are humans’

Answer: a
Explanation: The regex splits the sentence into three groups separated by

spaces. The second group (.*?) captures the word ‘are’, so matched.group(2) returns ‘are’.

8. What will be the output of the following Python code?

```
import re
sentence = 'horses are fast'
regex = re.compile('(?P<animal>\w+) (?P<verb>\w+) (?P<adjective>\w+)')
matched = re.search(regex, sentence)
print(matched.groupdict())
```

a) {‘animal’: ‘horses’, ‘verb’: ‘are’, ‘adjective’: ‘fast’}
b) (‘horses’, ‘are’, ‘fast’)
c) ‘horses are fast’
d) ‘are’

Answer: a
Explanation: The output is {‘animal’: ‘horses’, ‘verb’: ‘are’, ‘adjective’: ‘fast’} because the regex uses named groups (?P<name>) to capture words in the sentence. The groupdict() method returns a dictionary mapping the group names to their matched substrings.

9. What will be the output of the following Python code?

```
import re
sentence = 'horses are fast'
regex = re.compile('(?P<animal>\w+) (?P<verb>\w+) (?P<adjective>\w+)')
matched = re.search(regex, sentence)
print(matched.groups())
```

a) {‘animal’: ‘horses’, ‘verb’: ‘are’, ‘adjective’: ‘fast’}
b) (‘horses’, ‘are’, ‘fast’)
c) ‘horses are fast’

d) ‘are’

Answer: b
Explanation: matched.groups() returns a tuple of all matched groups without their names. So it outputs (‘horses’, ‘are’, ‘fast’) as a simple tuple of strings. Named groups only affect groupdict(), not groups().

10. What will be the output of the following Python code?

```
import re
sentence = 'horses are fast'
regex = re.compile('(?P<animal>\w+) (?P<verb>\w+)
(?P<adjective>\w+)')
matched = re.search(regex, sentence)
print(matched.group(2))
```

a) {‘animal’: ‘horses’, ‘verb’: ‘are’, ‘adjective’: ‘fast’}
b) (‘horses’, ‘are’, ‘fast’)
c) ‘horses are fast’
d) ‘are’

Answer: d
Explanation: The regex captures three words as named groups: ‘animal’, ‘verb’, and ‘adjective’. Using matched.group(2) returns the second captured group, which corresponds to ‘are’.

1. The character Dot (that is, ‘.’) in the default mode, matches any character other than _____

a) caret
b) ampersand
c) percentage symbol
d) newline

Answer: d
Explanation: In regular expressions, the dot (.) matches any single character except the newline character (\n) by default. To include newlines, you must use the re.DOTALL or re.S flag.

2. The expression a{5} will match _____ characters with the previous regular expression.

a) 5 or less
b) exactly 5
c) 5 or more
d) exactly 4

Answer: b
Explanation: The expression a{5} in regular expressions matches exactly 5 occurrences of the character a. It does not match fewer or more than 5. For example, it matches ‘aaaaa’ but not ‘aaaa’ or ‘aaaaaa’.

3. _____ matches the start of the string.
_____ matches the end of the string.

a) ‘^’, ‘$’
b) ‘$’, ‘^’
c) ‘$’, ‘?’
d) ‘?’, ‘^’

Answer: a
Explanation: ‘^’ (carat) matches the start of the string. ‘$’ (dollar sign) matches the end of the string. For example, ^Hello matches any string that starts with “Hello”, and world$ matches any string that ends with “world”.

4. Which of the following will result in an error?

a)p = re.compile("d")

p.search("door")
b) p = re.escape(‘hello’)
c) p = re.subn()
d) p = re.purge()

Answer: c
Explanation: re.subn() requires at least three arguments: the pattern, replacement, and string. Calling it with no arguments (as in re.subn()) will raise a TypeError.

5. What will be the output of the following Python code?

```
import re
print(re.split('\W+', 'Hello, hello, hello.'))
```

a) [‘Hello’, ‘hello’, ‘hello.’]
b) [‘Hello, ‘hello’, ‘hello’]
c) [‘Hello’, ‘hello’, ‘hello’, ‘.’]
d) [‘Hello’, ‘hello’, ‘hello’, ”]

Answer: d
Explanation: The re.split(‘\W+’, ‘Hello, hello, hello.’) function splits the string at one or more non-word characters (\W+), which includes punctuation like commas and periods.

- ‘Hello, hello, hello.’ gets split at the commas and period.
- This results in: [‘Hello’, ‘hello’, ‘hello’, ”] — the empty string is due to splitting at the end after the period.

6. What will be the output of the following Python function?

```
import re
print(re.findall("hello world", "hello", 1))
```

a) [“hello”]
b) [ ]
c) hello

d) hello world

Answer: b
Explanation: The function re.findall(“hello world”, “hello”, 1) is trying to find the exact string “hello world” inside the string “hello”. Since “hello world” does not exist in “hello”, the result is an empty list [].

7. Choose the function whose output can be: <_sre.SRE_Match object; span=(4, 8), match=’aaaa’>.

a) re.search(‘aaaa’, “alohaaaa”, 0)
b) re.match(‘aaaa’, “alohaaaa”, 0)
c) re.match(‘aaa’, “alohaaa”, 0)
d) re.search(‘aaa’, “alohaaa”, 0)

Answer: a
Explanation: The output <_sre.SRE_Match object; span=(4, 8), match=’aaaa’> means that the pattern ‘aaaa’ was found in the string starting at position 4 and ending at 8 (i.e., the substring ‘aaaa’ exists in the string “alohaaaa” starting at index 4).

- re.search(‘aaaa’, “alohaaaa”, 0) will find the pattern ‘aaaa’ in “alohaaaa” starting from position 0.
- re.match() would only check at the start of the string, so it wouldn’t find ‘aaaa’ unless it starts at index 0.

8. Which of the following functions clears the regular expression cache?

a) re.sub()
b) re.pos()
c) re.purge()
d) re.subn()

Answer: c
Explanation: The function re.purge() is used to clear the regular

expression cache in Python. This cache stores compiled regular expressions for faster reuse, and re.purge() clears it to free up memory or reset the cache.

9. What will be the output of the following Python code?

```
import re
re.ASCII
```

a) 8
b) 32
c) 64
d) 256

Answer: d
Explanation: The constant re.ASCII in Python has a value of 256. It is used as a flag to make regular expressions perform ASCII-only matching instead of Unicode when using \w, \d, etc.

10. Which of the following functions results in case insensitive matching?

a) re.A
b) re.U
c) re.I
d) re.X

Answer: c
Explanation: The re.I flag (also written as re.IGNORECASE) enables case-insensitive matching in regular expressions. It allows characters like a and A to be treated as equal when matching patterns.

1. What will be the output of the following Python code?

```
import re
print(re.compile('hello', re.X))
```

a) [‘h’, ‘e’, ‘l’, ‘l’, ‘o’]
b) re.compile(‘hello’, re.VERBOSE)
c) Error
d) Junk value

Answer: b
Explanation: The flag re.X is a shorthand for re.VERBOSE, which allows you to write regular expressions more clearly with whitespace and comments. So, re.compile(‘hello’, re.X) is equivalent to re.compile(‘hello’, re.VERBOSE).

2. What will be the output of the following Python code?

```
import re
print(re.split('[a-c]', '0a3B6', re.I))
```

a) Error
b) [‘a’, ‘B’]
c) [‘0’, ‘3B6’]
d) [‘a’]

Answer: c
Explanation: The re.split(‘[a-c]’, ‘0a3B6’, re.I) splits the string at any letter from ‘a’ to ‘c’, case-insensitively (due to re.I). So, it splits at ‘a’, resulting in parts: ‘0’ and ‘3B6’.

3. What will be the output of the following Python code?

```
import re
print(re.sub('morning', 'evening', 'good morning'))
```

a) ‘good evening’
b) ‘good’
c) ‘morning’

d) ‘evening’

Answer: a
Explanation: The re.sub() function replaces all occurrences of the pattern ‘morning’ with ‘evening’ in the given string ‘good morning’. Hence, the output is ‘good evening’.

4. The function re.error raises an exception if a particular string contains no match for the given pattern.

a) True
b) False

Answer: b
Explanation: The re.error exception is raised for invalid regex patterns, not for when there is no match. If no match is found, functions like re.search() simply return None without raising an error.

5. What will be the output of the following Python code?

```
import re
print(re.escape('new**world'))
```

a) ‘new world’
b) ‘new**world’
c) ‘**’
d) ‘new’, ‘*’, ‘*’, ‘world’

Answer: b
Explanation: The re.escape() function escapes all non-alphanumeric characters in the string by adding backslashes before them. So the asterisks ** are escaped as **, resulting in ‘new**world’.

6. What will be the output of the following Python code?

```
import re
print(re.fullmatch('hello', 'hello world'))
```

a) No output
b) []
c) <_sre.SRE_Match object; span=(0, 5), match='hello'>
d) Error

Answer: a
Explanation: The re.fullmatch() function attempts to match the entire string against the given pattern. In the code:

```
re.fullmatch('hello', 'hello world')
```

The string 'hello world' does not exactly match the pattern 'hello' because it has extra characters (' world'). Hence, re.fullmatch() returns None, which produces no output when printed.

7. Choose the option wherein the two choices do not refer to the same option.

a)
  re.I
  re.IGNORECASE
b)
  re.M
  re.MULTILINE
c)
  re.X
  re.VERBOSE
d)
  re.L
  re.LOWERCASE

Answer: d
Explanation: Options like re.I and re.IGNORECASE, re.M and re.MULTILINE, and re.X and re.VERBOSE are equivalent pairs in Python's

re module—they are just aliases for the same flags. However, re.LOWERCASE does not exist in the re module, while re.L is a valid flag used for locale-dependent matching.

8. The difference between the functions re.sub and re.subn is that re.sub returns a _____ whereas re.subn returns a _____

a) string, list
b) list, tuple
c) string, tuple
d) tuple, list

Answer: c
Explanation: The re.sub() function returns just the modified string after all replacements. In contrast, re.subn() returns a tuple containing the modified string and the number of substitutions made. This helps track how many replacements occurred.

9. What will be the output of the following Python code?

```
import re
print(re.split('mum', 'mumbai*', 1))
```

a) Error
b) [”, ‘bai*’]
c) [”, ‘bai’]
d) [‘bai*’]

Answer: b
Explanation: The re.split(‘mum’, ‘mumbai*’, 1) function splits the string ‘mumbai*’ at the first occurrence of the pattern ‘mum’. Since ‘mum’

occurs at the start, the string is split into two parts: an empty string before ‘mum’ and ‘bai*’ after it. Hence, the output is [”, ‘bai*’].

10. What will be the output of the following Python code?

```
import re
print(re.findall('good', 'good is good'))
print(re.findall('good', 'bad is good'))
```

a)
[‘good’, ‘good’]
[‘good’]
b)
(‘good’, ‘good’)
(good)
c)
(‘good’)
(‘good’)
d)
[‘good’]
[‘good’]

Answer: a
Explanation: The re.findall() function returns a list of all matches of the given pattern in the string. In the first call, ‘good’ appears twice in ‘good is good’, so it returns [‘good’, ‘good’]. In the second call, ‘good’ appears once in ‘bad is good’, so it returns [‘good’].

1. What will be the output of the following Python code?

```
import re
print(re.split(r'(n\d)=', 'n1=3.1, n2=5, n3=4.565'))
```

a) Error
b) [”, ‘n1’, ‘3.1, ‘, ‘n2’, ‘5, ‘, ‘n3’, ‘4.565’]
c) [‘n1’, ‘3.1, ‘, ‘n2’, ‘5, ‘, ‘n3’, ‘4.565’]

d) [‘3.1, ‘, ‘5, ‘, ‘4.565’]

Answer: b
Explanation: The regex r'(n\d)=’ captures patterns like ‘n1=’, splitting the string at these points and including the captured group (e.g., ‘n1’, ‘n2’) in the result. re.split() inserts each match and its surrounding text into the result list. Hence, it returns [”, ‘n1’, ‘3.1, ‘, ‘n2’, ‘5, ‘, ‘n3’, ‘4.565’].

2. The function of re.search is _____
a) Matches a pattern at the start of the string
b) Matches a pattern at the end of the string
c) Matches a pattern from any part of a string
d) Such a function does not exist

Answer: c
Explanation: re.search() function scans through the entire string and returns the first match of the pattern from any part of the string, not just the beginning or end. Unlike re.match() (which checks only at the start), re.search() finds matches anywhere in the string.
3. Which of the following functions creates a Python object?
a) re.compile(str)
b) re.assemble(str)
c) re.regex(str)
d) re.create(str)

Answer: a
Explanation: The re.compile(str) function compiles a regular expression pattern into a regular expression object, which can then be used for matching using methods like .match(), .search(), etc. Other options like re.assemble, re.regex, and re.create do not exist in the re module.

4. Which of the following pattern matching modifiers permits whitespace and comments inside the regular expression?

a) re.L
b) re.S
c) re.U
d) re.X

Answer: d
Explanation: The re.X modifier (also known as re.VERBOSE) allows you to write more readable regular expressions by permitting whitespace and comments within the pattern. This is helpful when dealing with complex regex patterns.

5. What will be the output of the following Python code?

```
import re
s = 'welcome home'
m = re.match(r'(.*)(.*?)', s)
print(m.group())
```

a) (‘welcome’, ‘home’)
b) [‘welcome’, ‘home’]
c) welcome home
d) [‘welcome’ // ‘home’]

Answer: c
Explanation: The pattern r'(.*)(.*?)’ used with re.match on the string ‘welcome home’ captures the entire string in two groups:

- (.*) captures as much as possible (greedy).
- (.*?) then captures the remainder (which would be an empty string in this case).

But m.group() without any index returns the entire matched string, which is ‘welcome home’.

6. The function of re.match is _____

a) Error
b) Matches a pattern anywhere in the string
c) Matches a pattern at the end of the string
d) Matches a pattern at the start of the string

Answer: d
Explanation: The re.match() function in Python checks for a match only at the beginning of the string. It does not search the entire string for the pattern. If the pattern is not found at the start, it returns None.

7. The special character \B matches the empty string, but only when it is _____

a) at the beginning or end of a word
b) not at the beginning or end of a word
c) at the beginning of the word
d) at the end of the word

Answer: b
Explanation: The special character \B in regular expressions matches the empty string, but only when it is not at the beginning or end of a word. In contrast, \b matches the position at the start or end of a word (i.e., a word boundary), while \B does the opposite.

8. What will be the output of the following Python code?

```
import re
s = "A new day"
m = re.match(r'(.*)(.*?)', s)
print(m.group(2))
```

```
print(m.group(0))
```

a)
No output
A new day
b)
No output
No output
c)
[‘A’, ‘new’, ‘day’]
(‘A’, ‘new’, ‘day’)
d)
Error
[‘A’, ‘new’, ‘day’]

Answer: a
Explanation: The regular expression r'(.*)(.*?)’ is applied using re.match(), which matches from the start of the string.

- (.*) matches as much as possible (greedy).
- (.*?) matches as little as possible (non-greedy), but since .* already consumes everything, .*? matches an empty string.

So, group(2) refers to the second group (.*?), which is an empty string. group(0) returns the entire matched string, which is “A new day”.

9. Which of the following special characters matches a pattern only at the end of the string?

a) \B
b) \X
c) \Z

d) \A

Answer: c
Explanation: \Z matches only at the end of the string, ensuring the pattern appears right before the string ends. Unlike $, which can match before a newline, \Z strictly anchors the match to the absolute string end. This makes it useful for precise end-of-string matching.

10. The output of the following two Python codes are the same.

```
import re
p = re.compile('hello')
r = p.match('hello everyone')
print(r.group(0))

r = re.match('hello', 'hello everyone')
print(r.group(0))
```

a) True
b) False

Answer: a
Explanation: Both codes use re.match() to match the pattern ‘hello’ at the start of the string ‘hello everyone’.

- In the first, the pattern is compiled with re.compile() and then matched.
- In the second, the pattern is passed directly to re.match().

Both return a match object where .group(0) is ‘hello’, so the outputs are the same.

11. What will be the output of the following Python code?

```
import re
print(re.match('sp(.*)am', 'spam'))
```

a) <_sre.SRE_Match object; span=(1, 4), match='spam'>
b) <_sre.SRE_Match object; span=(0, 4), match='spam'>
c) No output
d) Error

Answer: b
Explanation: The pattern sp(.*)am matches the entire string 'spam' from start to end because (.*) matches zero characters between 'sp' and 'am'. Hence, re.match returns a match spanning (0, 4) with 'spam'.

12. Which of the following special characters represents a comment (that is, the contents of the parenthesis are simply ignores)?

a) (?:…)
b) (?=…)
c) (?!…)
d) (?#…)

Answer: d
Explanation: The special sequence (?#…) is used to insert comments inside a regular expression pattern, and anything inside these parentheses is ignored by the regex engine.

13. Which of the codes shown below results in a match?

a) re.match('George(?=Washington)', 'George Washington')
b) re.match('George(?=Washington)', 'George')
c) re.match('George(?=Washington)', 'GeorgeWashington')
d) re.match('George(?=Washington)', 'Georgewashington')

Answer: c
Explanation: The pattern George(?=Washington) uses a positive lookahead (?=Washington), which means it matches 'George' only if it is

immediately followed by ‘Washington’ without including ‘Washington’ in the match.

- It matches ‘George’ only if ‘Washington’ follows right after, with exact case and no space.
- ‘GeorgeWashington’ fits this condition, so it matches.
- ‘George Washington’ (with a space) or ‘Georgewashington’ (different case) do not match.

1. What will be the output of the following Python code?

```
import re
print(re.split(r'(a)(t)', 'Maths is a difficult subject'))
```

a) [‘M a t h s i s a d i f f i c u l t s u b j e c t’]
b) [‘Maths’, ‘is’, ‘a’, ‘difficult’, ‘subject’]
c) ‘Maths is a difficult subject’
d) [‘M’, ‘a’, ‘t’, ‘hs is a difficult subject’]

Answer: d
Explanation: The regex re.split(r'(a)(t)’, ‘Maths is a difficult subject’) splits the string at the first match of ‘a’ followed by ‘t’, which occurs in ‘Maths’. Since capturing groups are used, the matched groups ‘a’ and ‘t’ are also included in the output. Hence, the result is [‘M’, ‘a’, ‘t’, ‘hs is a difficult subject’].

2. The output of the following two Python codes are the same.

```
import re

# CODE 1
print(re.split(r'(a)(t)', 'The night sky'))

# CODE 2
```

```
print(re.split(r'\s+', 'The night sky'))
```
a) True
b) False

Answer: b
Explanation: Code 1 splits using the pattern (a)(t), which looks for ‘at’ with capturing groups—this pattern doesn’t exist in ‘The night sky’, so it returns the whole string. Code 2 splits on whitespace using \s+, so it returns [‘The’, ‘night’, ‘sky’].

3. What will be the output of the following Python code?

```
import re
s = 'abc123 xyz666 lmn-11 def77'
print(re.sub(r'\b([a-z]+)(\d+)', r'\2\1:', s))
```
a) ‘123abc: 666xyz: lmn-11 77def:’
b) ‘77def: lmn-11: 666xyz: 123abc’
c) ‘abc123:’, ‘xyz666:’, ‘lmn-11:’, ‘def77:’
d) ‘abc123: xyz666: lmn-11: def77’

Answer: a
Explanation: Here,

- \b([a-z]+)(\d+) matches word boundaries (\b) where a sequence of lowercase letters ([a-z]+) is followed by digits (\d+).
- The replacement \2\1: swaps the digits and letters, then adds a colon.
- ‘abc123’ becomes ‘123abc:’, ‘xyz666’ becomes ‘666xyz:’, ‘def77’ becomes ’77def:’.
- ‘lmn-11’ is not matched because there’s a hyphen between letters and digits.
- So, the final result is: ‘123abc: 666xyz: lmn-11 77def:’

4. What will be the output of the following Python code?

```
import re
print(re.subn('A', 'X', 'AAAAAA', count=4))
```

a) ‘XXXXAA, 4’
b) (‘AAAAAA’, 4)
c) (‘XXXXAA’, 4)
d) ‘AAAAAA, 4’

Answer: c
Explanation: The re.subn() function performs substitutions and returns a tuple containing the new string and the number of substitutions made. In the code re.subn(‘A’, ‘X’, ‘AAAAAA’, count=4), the first 4 occurrences of ‘A’ are replaced with ‘X’, resulting in (‘XXXXAA’, 4).

5. What will be the output of the following Python code?

```
import re
n = re.sub(r'\w+', 'Hello', 'Cats and dogs')
print(n)
```

a)
Hello
Hello
Hello
b) ‘Hello Hello Hello’
c) [‘Hello’, ‘Hello’, ‘Hello’]
d) (‘Hello’, ‘Hello’, ‘Hello’)

Answer: b
Explanation: The regular expression \w+ matches each word consisting of alphanumeric characters. The re.sub() function replaces each of those words with ‘Hello’. In the string ‘Cats and dogs’, there are three words, so the output becomes ‘Hello Hello Hello’.

6. What will be the output of the following Python code?

```
import re
w = re.compile('[A-Za-z]+')
print(w.findall('It will rain today'))
```

a) ‘It will rain today’
b) (‘It will rain today’)
c) [‘It will rain today’]
d) [‘It’, ‘will’, ‘rain’, ‘today’]

Answer: d
Explanation: The pattern [A-Za-z]+ matches continuous sequences of letters (words without digits or punctuation). findall() finds all such matches in ‘It will rain today’, returning them as a list.
So the output is [‘It’, ‘will’, ‘rain’, ‘today’].

7. In the functions re.search.start(group) and re.search.end(group), if the argument groups not specified, it defaults to _____

a) Zero
b) None
c) One
d) Error

Answer: a
Explanation: In the functions re.search().start(group) and re.search().end(group), if the group argument is not specified, it defaults to 0, which refers to the entire match.

8. What will be the output of the following Python code?

```
import re
print(re.split(r'\s+', 'Chrome is better than explorer', maxsplit=3))
```

a) [‘Chrome’, ‘is’, ‘better’, ‘than’, ‘explorer’]
b) [‘Chrome’, ‘is’, ‘better’, ‘than explorer’]

c) (‘Chrome’, ‘is’, ‘better’, ‘than explorer’)
d) ‘Chrome is better’ ‘than explorer’

Answer: b
Explanation: The pattern \s+ splits the string at spaces, but with maxsplit=3, it stops after three splits. So, the first three words are separated, and the rest (‘than explorer’) stays together as the last element. Hence, the output is [‘Chrome’, ‘is’, ‘better’, ‘than explorer’].

9. What will be the output of the following Python code?

```
import re
a=re.compile('[0-9]+')
print(a.findall('7 apples and 3 mangoes'))
```

a) [‘apples’ ‘and’ ‘mangoes’]
b) (7, 3)
c) [‘7’, ‘3’]
d) Error

Answer: c
Explanation: The regular expression [0-9]+ is designed to match one or more digits in a row. When applied to the string ‘7 apples and 3 mangoes’ using the findall() function, it scans through the text and extracts all occurrences of digit sequences. As a result, it finds the numbers ‘7’ and ‘3’ and returns them as a list of strings: [‘7’, ‘3’].

1. Which of the following functions returns a dictionary mapping group names to group numbers?

a) re.compile.group
b) re.compile.groupindex
c) re.compile.index
d) re.compile.indexgroup

Answer: b
Explanation: re.compile.groupindex is an attribute of the compiled regular expression object that returns a dictionary mapping named group names to their corresponding group numbers.

2. Which of the following statements regarding the output of the function re.match is incorrect?

a) ‘pq*’ will match ‘pq’
b) ‘pq?’ matches ‘p’
c) ‘p{4}, q’ does not match ‘pppq’
d) ‘pq+’ matches ‘p’

Answer: d
Explanation: ‘pq+’ matches ‘p’ is incorrect because the pattern ‘pq+’ requires at least one ‘q’ after ‘p’. The + quantifier means “one or more” of the preceding character. So ‘p’ alone doesn’t match ‘pq+’; it would need to be at least ‘pq’.

3. The following Python code snippet results in an error.

```
import re
c=re.compile(r'(\d+)(\[A-Z]+)([a-z]+)')
print(c.groupindex)
```

a) True
b) False

Answer: b
Explanation: The code does not result in an error. While the regex pattern itself is valid, groupindex will simply return an empty dictionary ({}) in this case because there are no named groups in the regular expression. Named groups use the syntax (?P<name>…). Since (\d+)(\[A-

Z]+)([a-z]+) contains only unnamed groups, groupindex works, but it has nothing to map.

4. Which of the following functions does not accept any argument?

a) re.purge
b) re.compile
c) re.findall
d) re.match

Answer: a
Explanation: The function re.purge() clears the internal cache of the re module and does not take any arguments. All other functions listed, like re.compile, re.findall, and re.match, require at least a pattern argument.

5. What will be the output of the following Python code?

```
import re
a = re.compile('0-9')
print(a.findall('3 trees'))
```

a) []
b) [‘3’]
c) Error
d) [‘trees’]

Answer: a
Explanation: The regular expression ‘0-9’ is interpreted literally, looking for the exact substring “0-9”, not any digit between 0 and 9. In the string ‘3 trees’, there’s no literal “0-9” substring, so findall() returns an empty list.

6. Which of the following lines of code will not show a match?

a) re.match(‘ab*’, ‘a’)
b) re.match(‘ab*’, ‘ab’)
c) re.match(‘ab*’, ‘abb’)

d) re.match(‘ab*’, ‘ba’)

Answer: d
Explanation: The pattern ‘ab*’ matches the letter ‘a’ followed by zero or more ‘b’ characters. In option re.match(‘ab*’, ‘ba’), the string starts with ‘b’ instead of ‘a’, so it doesn’t match the pattern. Hence, no match is returned.

7. What will be the output of the following Python code?

```
import re
m = re.search('a', 'The blue umbrella')
print(m.re.pattern)
```

a) {}
b) ‘The blue umbrella’
c) ‘a’
d) No output

Answer: c
Explanation: The re.search(‘a’, ‘The blue umbrella’) finds the first match of ‘a’ in the string. The expression m.re.pattern accesses the pattern string used to compile the regular expression. In this case, the pattern is ‘a’.

8. What will be the output of the following Python code?

```
import re
print(re.sub('Y', 'X', 'AAAAAA', count=2))
```

a) ‘YXAAAA’
b) (‘YXAAAA’)
c) (‘AAAAAA’)
d) ‘AAAAAA’

Answer: d
Explanation: The function re.sub('Y', 'X', 'AAAAAA', count=2) attempts to replace the first two occurrences of 'Y' with 'X' in the string 'AAAAAA'. However, since 'Y' does not appear in the string, no replacements are made, and the original string is returned unchanged.

1. To open a file c:\scores.txt for reading, we use _____

a) infile = open("c:\scores.txt", "r")
b) infile = open("c:\\scores.txt", "r")
c) infile = open(file = "c:\scores.txt", "r")
d) infile = open(file = "c:\\scores.txt", "r")

Answer: b
Explanation: In Python strings, backslashes (\) are escape characters, so to specify a Windows path like c:\scores.txt, you need to escape the backslash by using double backslashes (\\). Thus, open("c:\\scores.txt", "r") correctly opens the file for reading.

2. To open a file c:\scores.txt for writing, we use _____

a) outfile = open("c:\scores.txt", "w")
b) outfile = open("c:\\scores.txt", "w")
c) outfile = open(file = "c:\scores.txt", "w")
d) outfile = open(file = "c:\\scores.txt", "w")

Answer: b
Explanation: To open a file for writing in Python on Windows, you need to specify the correct file path. Since backslashes are escape characters, each backslash in the path should be doubled (\\).
So, "c:\\scores.txt" correctly represents the file path, and "w" mode opens the file for writing.

3. To open a file c:\scores.txt for appending data, we use _____

a) outfile = open("c:\\scores.txt", "a")

b) outfile = open(“c:\\scores.txt”, “rw”)
c) outfile = open(file = “c:\scores.txt”, “w”)
d) outfile = open(file = “c:\\scores.txt”, “w”)

Answer: a
Explanation: To open a file for appending, use mode “a” which adds data to the file without overwriting existing content. The file path on Windows must use double backslashes (\\) to avoid escape errors. Thus, open(“c:\\scores.txt”, “a”) correctly opens the file for appending.

4. Which of the following statements are true?
a) When you open a file for reading, if the file does not exist, an error occurs
b) When you open a file for writing, if the file does not exist, a new file is created
c) When you open a file for writing, if the file exists, the existing file is overwritten with the new file
d) All of the mentioned

Answer: d
Explanation: When opening a file in read mode, an error occurs if the file doesn’t exist. In write mode, Python creates a new file if it’s missing and overwrites it if it already exists. So, all these statements about file modes are correct.
5. To read two characters from a file object infile, we use _____
a) infile.read(2)
b) infile.read()
c) infile.readline()
d) infile.readlines()

Answer: a
Explanation: infile.read(2) reads exactly two characters from the file. The read(n) method reads n number of characters from the file object. Other options are used to read the entire file or lines, not specific character counts.

6. To read the entire remaining contents of the file as a string from a file object infile, we use _____

a) infile.read(2)
b) infile.read()
c) infile.readline()
d) infile.readlines()

Answer: b
Explanation: infile.read() without any arguments reads the entire remaining contents of the file and returns it as a string. It reads from the current file pointer position to the end of the file.

7. What will be the output of the following Python code?

```
f = None
for i in range (5):
   with open("data.txt", "w") as f:
      if i > 2:
         break
print(f.closed)
```

a) True
b) False
c) None
d) Error

Answer: a
Explanation: The with statement ensures the file is automatically closed

when the block is exited, even if break is executed. When i becomes 3, the condition i > 2 is true, so the loop breaks, but before that, the with block closes the file. Therefore, f.closed is True.

8. To read the next line of the file from a file object infile, we use _____

a) infile.read(2)
b) infile.read()
c) infile.readline()
d) infile.readlines()

Answer: c
Explanation: infile.readline() reads the next line from the file, stopping at the newline character. It returns that line as a string, including the newline at the end (if present). This is useful for processing a file line by line.

9. To read the remaining lines of the file from a file object infile, we use _____

a) infile.read(2)
b) infile.read()
c) infile.readline()
d) infile.readlines()

Answer: d
Explanation: infile.readlines() reads all the remaining lines from the file and returns them as a list of strings, where each string is one line from the file. This is useful when you want to process or store all the lines at once.

10. The readlines() method returns _____

a) str
b) a list of lines

c) a list of single characters
d) a list of integers

Answer: b
Explanation: The readlines() method reads all the remaining lines in a file and returns them as a list of strings, where each string represents one line (including the newline character \n if present).

1. Which Python function is used to read a single line of text from the user input?

a) print()
b) input()
c) open()
d) read()

Answer: b
Explanation: The input() function reads a single line of text entered by the user from standard input (keyboard) and returns it as a string.

2. What will be the output of the following Python code?

```
str = input("Enter your input: ");
print ("Received input is : ", str)
```

a)
Enter your input: Hello Python
Received input is : Hello Python

b)
Enter your input: Hello Python
Received input is : Hello
Enter your input: Hello Python
Received input is : Python

d) None of the mentioned

Answer: a
Explanation: The input() function reads the entire line of text entered by the user (including spaces) as a string. So typing Hello Python returns “Hello Python”. The print() statement then displays the full input after the message.

3. What will be the output of the following Python code?

```
str = input("Enter your input: ");
print ("Received input is : ", str)
```

a)
Enter your input: [x*5 for x in range(2,10,2)]
Received input is : [x*5 for x in range(2,10,2)]
b)
Enter your input: [x*5 for x in range(2,10,2)]
Received input is : [10, 30, 20, 40]
c)
Enter your input: [x*5 for x in range(2,10,2)]
Received input is : [10, 10, 30, 40]
d) None of the mentioned

Answer: a
Explanation: The input() function reads user input as a string, not as a Python expression. So when you enter [x*5 for x in range(2,10,2)], it treats it as a string rather than evaluating it as a list comprehension. Thus, it prints the exact input string.

4. Which one of the following is not attributes of file?

a) closed
b) softspace
c) rename

d) mode

Answer: c
Explanation: closed, softspace, and mode are built-in attributes of a file object that provide information about the file's state or how it was opened. rename is not an attribute of a file object; it's a separate function used to change a file's name.

5. What is the use of tell() method in python?

a) tells you the current position within the file
b) tells you the end position within the file
c) tells you the file is opened or not
d) none of the mentioned

Answer: a
Explanation: The tell() method returns the current position (offset) of the file pointer within the file, indicating where the next read or write will occur.

6. What is the current syntax of rename() a file?

a) rename(current_file_name, new_file_name)
b) rename(new_file_name, current_file_name,)
c) rename(()(current_file_name, new_file_name))
d) none of the mentioned

Answer: a
Explanation: The rename() function from the os module is used to rename files. The correct syntax is: os.rename(current_file_name, new_file_name), where the first argument is the existing file name and the second is the new name you want to assign.

7. What is the current syntax of remove() a file?

a) remove(file_name)

b) remove(new_file_name, current_file_name,)
c) remove(() , file_name))
d) none of the mentioned

Answer: a
Explanation: The remove() function from the os module is used to delete a file. The correct syntax is: os.remove(file_name), where file_name is the name (or path) of the file to be deleted.

8. What will be the output of the following Python code?

```
fo = open("foo.txt", "rw+")
print("Name of the file: ", fo.name)

# Assuming file has following 5 lines
# This is 1st line
# This is 2nd line
# This is 3rd line
# This is 4th line
# This is 5th line

for index in range(5):
   line = fo.next()
   print("Line No %d - %s" % (index, line))

# Close opened file
fo.close()
```

a) Compilation Error
b) Syntax Error
c) Displays Output
d) None of the mentioned

Answer: b
Explanation: The given Python code contains syntax issues that prevent it from running correctly:

- **Invalid File Mode (“rw+”)** → “rw+” is not a valid mode in Python. Use “r+” instead.
- **Incorrect print Statement** → The syntax used for print is incorrect. print should be written properly as a function.
- **Incorrect Use of next()** → The method next() is not directly available for file objects in this way. The correct way to read the next line is next(fo).

9. What is the use of seek() method in files?
a) sets the file’s current position at the offset
b) sets the file’s previous position at the offset
c) sets the file’s current position within the file
d) none of the mentioned

Answer: a
Explanation: The seek() method is used to move the file pointer to a specific position within the file. This position is defined by the offset (number of bytes from the beginning, or another reference point if specified). It’s useful for reading or writing at specific locations in a file.
10. What is the use of truncate() method in file?
a) truncates the file size
b) deletes the content of the file
c) deletes the file size
d) none of the mentioned

Answer: a
Explanation: The truncate() method is used to resize the file to a specified size. If no size is specified, it truncates the file at the current file pointer

position. This can effectively reduce the file's content by cutting off data beyond the set size.

1. Which is/are the basic I/O connections in file?

a) Standard Input
b) Standard Output
c) Standard Errors
d) All of the mentioned

Answer: d
Explanation: The basic I/O connections in files include Standard Input (stdin), Standard Output (stdout), and Standard Errors (stderr), which are fundamental streams used for input and output operations.

- Standard Input is used to receive input data (usually from the keyboard).
- Standard Output is used to display output data (usually to the screen).
- Standard Errors is used to output error messages separately from standard output, allowing error handling and logging.

2. What will be the output of the following Python code? (If entered name is sanfoundry)

```
import sys
print('Enter your name: ', end='')
name = ''
while True:
   c = sys.stdin.read(1)
   if c == '\n':
      break
   name += c
```

```
print('Your name is:', name)
```

a) sanfoundry

b)

Enter your name: sanfoundry

Your name is: sanfoundry

c) Your name is: san

d) Enter your name: sanfoundry Your name is: sanfoundry

Answer: b

Explanation: This code reads one character at a time from standard input using sys.stdin.read(1).

It continues reading until a newline character (\n) is encountered (when the user presses Enter).

If the input is sanfoundry, then name becomes sanfoundry. The final print statement displays:

3. What will be the output of the following Python code?

```
import sys
sys.stdout.write(' Hello\n')
sys.stdout.write('Python\n')
```

a) Compilation Error

b) Runtime Error

c) Hello Python

d)

Hello

Python

Answer: d

Explanation: The sys.stdout.write() function writes text to the output without adding a newline automatically. Since both strings in the code

include \n, each line is printed on a new line as written. Therefore, the output is:
Hello
Python

4. Which of the following mode will refer to binary data?
a) r
b) w
c) +
d) b

Answer:d
Explanation: The ‘b’ mode in file operations refers to binary mode. It’s used when reading or writing binary files (like images, videos, or executable files). For example:

- ‘rb’ opens a file for reading in binary mode.
- ‘wb’ opens a file for writing in binary mode.

Other modes like ‘r’, ‘w’, and ‘+’ relate to text file operations or read/write permissions, not specifically binary data.

5. What is the pickling?
a) It is used for object serialization
b) It is used for object deserialization
c) None of the mentioned
d) All of the mentioned

Answer: a
Explanation: Pickling is the process of converting a Python object into a byte stream, also known as serialization. This enables the object to be saved to a file or transmitted over a network. The reverse process,

converting the byte stream back into a Python object, is called unpickling or deserialization. The pickle.dump() function is used to serialize (pickle) an object, while pickle.load() is used to deserialize (unpickle) it.

6. What is unpickling?

a) It is used for object serialization
b) It is used for object deserialization
c) None of the mentioned
d) All of the mentioned

Answer: b
Explanation: Unpickling is the process of deserializing a pickled (serialized) object — that is, converting the byte stream back into a Python object. It is the reverse of pickling and is done using functions like pickle.load().

7. What is the correct syntax of open() function?

a) file = open(file_name [, access_mode][, buffering])
b) file object = open(file_name [, access_mode][, buffering])
c) file object = open(file_name)
d) none of the mentioned

Answer: a
Explanation: The correct and general syntax of the open() function in Python is:

```
file = open(file_name [, access_mode][, buffering])
```

- file_name: Name of the file to be opened.
- access_mode: Optional; specifies the mode in which the file is opened (‘r’, ‘w’, ‘a’, ‘rb’, etc.).
- buffering: Optional; sets buffering policy.

8. What will be the output of the following Python code?

```
fo = open("foo.txt", "wb")
```

```
print("Name of the file: ", fo.name)
fo.flush()
fo.close()
```

a) Compilation Error
b) Runtime Error
c) No Output
d) Flushes the file when closing them

Answer: d
Explanation: The code opens a file in binary write mode and prints its name. The flush() method ensures any buffered data is written to the file. When close() is called, it automatically flushes the buffer as well, so the file is safely saved.

9. Correct syntax of file.writelines() is?

a) file.writelines(sequence)
b) fileObject.writelines()
c) fileObject.writelines(sequence)
d) none of the mentioned

Answer: c
Explanation: The writelines() method is called on a file object and takes a sequence (like a list or tuple) of strings as its argument. It writes each string in the sequence to the file without adding newlines automatically. So, the proper syntax is:

```
fileObject.writelines(sequence)
```

10. Correct syntax of file.readlines() is?

a) fileObject.readlines(sizehint);
b) fileObject.readlines();
c) fileObject.readlines(sequence)

d) none of the mentioned

Answer: a
Explanation: The readlines() method reads all the lines from a file and returns them as a list of strings. It optionally accepts a sizehint argument, which is an approximate number of bytes to read. The correct syntax is:
fileObject.readlines(sizehint)
where sizehint is optional.

1. In file handling, what does this terms means “r, a”?

a) read, append
b) append, read
c) write, append
d) none of the mentioned

Answer: a
Explanation: In file handling, “r” mode opens a file for reading, while “a” mode opens a file for appending, allowing new data to be added at the end of the file without overwriting existing content.

2. What is the use of “w” in file handling?

a) read
b) write
c) append
d) none of the mentioned

Answer: b
Explanation: The “w” mode in file handling is used to write to a file. If the file already exists, it overwrites the content. If the file doesn’t exist, it creates a new file.

3. What is the use of “a” in file handling?

a) read

b) write
c) append
d) none of the mentioned

Answer: c
Explanation: In file handling, the mode “a” stands for append. It opens the file for writing but adds new data at the end of the file without deleting the existing content. If the file doesn’t exist, it creates a new one.

4. Which function is used to read all the characters?
a) read()
b) readcharacters()
c) readall()
d) readchar()

Answer: a
Explanation: The function to read all characters from a file is read(). It reads the entire content of the file as a single string. The other options are not valid Python file methods.
5. Which function is used to read single line from file?
a) readline()
b) readlines()
c) readstatement()
d) readfullline()

Answer: a
Explanation: The readline() function reads one line at a time from a file. It returns the next line in the file each time it is called.

6. Which function is used to write all the characters?
a) write()
b) writecharacters()
c) writeall()
d) writechar()

Answer: a
Explanation: The write() function is used to write a string of characters to a file. It can be used to write a single character, a word, a sentence, or an entire block of text, depending on the input given. The other options are not valid Python file functions.
7. Which function is used to write a list of string in a file?
a) writeline()
b) writelines()
c) writestatement()
d) writefullline()

Answer: b
Explanation: The writelines() function writes a list (or any iterable) of strings to a file without adding newline characters automatically. To write each string on a new line, you must include \n at the end of each string. It efficiently writes multiple lines in one call.
8. Which function is used to close a file in python?
a) close()
b) stop()
c) end()
d) closefile()

Answer: a
Explanation: The close() function is used to close an open file in Python. It

is important to close a file after completing operations on it to free up system resources.

9. Is it possible to create a text file in python?

a) Yes
b) No
c) Machine dependent
d) All of the mentioned

Answer: a
Explanation: Yes, it is possible to create a text file in Python using the open() function with modes like “w” (write) or “a” (append). If the specified file does not exist, Python will create it automatically.

10. Which of the following are the modes of both writing and reading in binary format in file?

a) wb+
b) w
c) wb
d) w+

Answer: a
Explanation: Here is the description below

- “w”: Opens a file for writing only in text mode. Overwrites the file if it exists or creates a new one.
- “wb”: Opens a file for writing only in binary mode. Overwrites the file if it exists or creates a new one.
- “w+”: Opens a file for both writing and reading in text mode. Overwrites if the file exists or creates a new one.
- “wb+”: Opens a file for both writing and reading in binary mode. Overwrites if the file exists or creates a new one.

1. Which of the following is not a valid mode to open a file?
a) ab
b) rw
c) r+
d) w+

Answer: b
Explanation: In Python, file modes like ab, r+, and w+ are valid. ab opens a file for appending in binary mode, r+ opens a file for both reading and writing, and w+ opens a file for both writing and reading (overwriting the existing file or creating a new one). However, rw is not a valid file mode in Python and will raise a ValueError if used.
2. What is the difference between r+ and w+ modes?
a) no difference
b) both r+ and w+ place the pointer at the beginning of the file, but w+ truncates (clears) the file, while r+ keeps its content
c) in w+ the pointer is initially placed at the beginning of the file and the pointer is at the end for r+
d) depends on the operating system

Answer: b
Explanation: Both r+ and w+ open the file for reading and writing. However, r+ requires the file to exist and does not truncate its content — it keeps the original data. On the other hand, w+ creates a new file if it doesn't exist or clears the contents of the file if it does. In both cases, the file pointer is placed at the beginning of the file.
3. How do you get the name of a file from a file object (fp)?
a) fp.name
b) fp.file(name)
c) self.__name__(fp)

d) fp.__name__()

Answer: a
Explanation: In Python, every file object has an attribute .name that stores the name of the file associated with that file object. So, fp.name returns the name of the file opened using that file object.

4. Which of the following is not a valid attribute of a file object (fp)?
a) fp.name
b) fp.closed
c) fp.mode
d) fp.size

Answer: d
Explanation: In Python, file objects have attributes like:

- fp.name: the name of the file,
- fp.closed: a boolean indicating whether the file is closed,
- fp.mode: the mode in which the file was opened.

However, fp.size is not a valid attribute of a file object. If you want to get the size of a file, you would typically use os.path.getsize(filename) instead.

5. How do you close a file object (fp)?
a) close(fp)
b) fclose(fp)
c) fp.close()
d) fp.__close__()

Answer: c
Explanation: The correct way to close a file object fp in Python is fp.close(). This method ensures that any buffered output is flushed to the

file and releases the file resource. It's the standard and recommended way to close a file in Python.

6. How do you get the current position within the file?

a) fp.seek()
b) fp.tell()
c) fp.loc
d) fp.pos

Answer: b
Explanation: The correct method to get the current position within a file is fp.tell(). This function returns the current position of the file pointer in bytes from the beginning of the file. It is commonly used to monitor the progress of reading or writing operations.

7. How do you rename a file?

a) fp.name = 'new_name.txt'
b) os.rename(existing_name, new_name)
c) os.rename(fp, new_name)
d) os.set_name(existing_name, new_name)

Answer: b
Explanation: In Python, to rename a file, the correct method is os.rename(old_name, new_name). This updates the file's name on the filesystem. Modifying the fp.name attribute does not affect the actual file, and functions like os.set_name or using the file object directly with os.rename() are not valid.

8. How do you delete a file?

a) del(fp)
b) fp.delete()
c) os.remove('file')

d) os.delete(‘file’)

Answer: c
Explanation: The correct way to delete a file in Python is os.remove(‘file’). This function from the os module permanently deletes the file specified by the given path. It does not move it to the recycle bin. Make sure to import the os module before using it:
import os
os.remove('file.txt')

9. How do you change the file position to an offset value from the start?

a) fp.seek(offset, 0)
b) fp.seek(offset, 1)
c) fp.seek(offset, 2)
d) none of the mentioned

Answer: a
Explanation: The seek() method is used to change the file’s current position. The second argument specifies the reference point: 0 means from the beginning of the file, 1 means from the current position, and 2 means from the end of the file. Therefore, fp.seek(offset, 0) moves the pointer to offset bytes from the start.

10. What happens if no arguments are passed to the seek function?

a) file position is set to the start of file
b) file position is set to the end of file
c) file position remains unchanged
d) error

Answer: d
Explanation: The seek() function requires at least one argument (the

offset). If called without arguments, it raises a TypeError because the mandatory offset parameter is missing.

1. Which function is called when the following Python code is executed?

```
f = foo()
format(f)
```

a) format()
b) __format__()
c) str()
d) __str__()

Answer: b
Explanation: When format(f) is called, Python internally invokes the special method f.__format__(). This is different from str(f) or f.__str__() which are used for string conversion. The __format__() method defines how the object should be formatted using the format() function.

2. Which of the following Python code will print True?

```
a = foo(2)
b = foo(3)
print(a < b)
```

a)

```
class foo:
    def __init__(self, x):
        self.x = x
    def __lt__(self, other):
        if self.x < other.x:
            return False
        else:
            return True
```

b)

```
class foo:
   def __init__(self, x):
      self.x = x
   def __less__(self, other):
      if self.x > other.x:
         return False
      else:
         return True
```

c)

```
class foo:
   def __init__(self, x):
      self.x = x
   def __lt__(self, other):
      if self.x < other.x:
         return True
      else:
         return False
```

d)

```
class foo:
   def __init__(self, x):
      self.x = x
   def __less__(self, other):
      if self.x < other.x:
         return False
      else:
         return True
```

Answer: c
Explanation: Python uses the special method __lt__(self, other) to implement the < operator. The code that defines __lt__() and returns

self.x < other.x will correctly compare the two foo objects. Given a = foo(2) and b = foo(3), this comparison results in 2 < 3, which is True.

3. Which function overloads the + operator?

a) __add__()
b) __plus__()
c) __sum__()
d) none of the mentioned

Answer: a
Explanation: The function that overloads the + operator in Python is __add__(). When you use the + operator between two objects, Python internally calls the __add__() method of the first object to perform the addition. Other options like __plus__() and __sum__() do not exist as special methods in Python.

4. Which operator is overloaded by __invert__()?

a) !
b) ~
c) ^
d) –

Answer: b
Explanation: The __invert__() method is used to overload the bitwise NOT operator (~) in Python. When you use ~x, Python internally calls x.__invert__() to compute the result.

5. Which function overloads the == operator?

a) __eq__()
b) __equ__()
c) __isequal__()

d) none of the mentioned

Answer: a
Explanation: The __eq__() method is used to overload the == operator in Python. When you compare two objects using ==, Python internally calls obj1.__eq__(obj2) to determine equality.

6. Which operator is overloaded by __lg__()?

a) <
b) >
c) !=
d) none of the mentioned

Answer: d
Explanation: There is no special method __lg__() in Python for operator overloading. Comparison operators like < and > are overloaded using __lt__() and __gt__() respectively. Hence, __lg__() does not overload any operator.

7. Which function overloads the >> operator?

a) __more__()
b) __gt__()
c) __ge__()
d) none of the mentioned

Answer: d
Explanation: The >> (right shift) operator is overloaded in Python using the __rshift__() method. None of the given options (like __gt__() or __ge__()) are used for shifting operations. Hence, the correct answer is “none of the mentioned”.

8. Let A and B be objects of class Foo. Which functions are called when print(A + B) is executed?

a) __add__(), __str__()
b) __str__(), __add__()
c) __sum__(), __str__()
d) __str__(), __sum__()

Answer: a
Explanation: When print(A + B) is executed, Python first evaluates the expression A + B, which calls the __add__() method of class Foo. The result of this addition (which is usually another object) is then passed to print(), which calls the __str__() method on it to convert it to a string for display.

9. Which operator is overloaded by the __or__() function?

a) ||
b) |
c) //
d) /

Answer: b
Explanation: The __or__() function is used to overload the bitwise OR (|) operator in Python. This allows custom behavior when using | with objects of user-defined classes.

10. Which function overloads the // operator?

a) __div__()
b) __ceildiv__()
c) __floordiv__()
d) __truediv__()

Answer: c
Explanation: The __floordiv__() method is used to overload the floor

division operator (//) in Python. It defines the behavior of // when used between objects of a custom class.

1. _____ represents an entity in the real world with its identity and behaviour.

a) A method
b) An object
c) A class
d) An operator

Answer: b
Explanation: An object represents a real-world entity with state (identity) and behavior in object-oriented programming. It is an instance of a class and encapsulates data and functions that operate on that data.

2. _____ is used to create an object.

a) class
b) constructor
c) User-defined functions
d) In-built functions

Answer: b
Explanation: A constructor is a special method (usually __init__ in Python) used to create and initialize an object of a class. It is automatically invoked when a new object is created using the class.

3. What will be the output of the following Python code?

```
class test:
    def __init__(self,a="Hello World"):
        self.a=a
```

```
    def display(self):
        print(self.a)
obj=test()
obj.display()
```

a) The program has an error because constructor can't have default arguments
b) Nothing is displayed
c) "Hello World" is displayed
d) The program has an error display function doesn't have parameters

Answer: c
Explanation: The constructor __init__ has a default argument "Hello World", which is perfectly valid in Python. When obj = test() is called without arguments, the default value is used. Then obj.display() prints the value of self.a, which is "Hello World".

4. What is setattr() used for?

a) To access the attribute of the object
b) To set an attribute
c) To check if an attribute exists or not
d) To delete an attribute

Answer: b
Explanation: The setattr() function in Python is used to assign a value to an object's attribute. It takes three arguments: the object, the name of the attribute as a string, and the value to set. For example, setattr(obj, 'x', 10) sets obj.x = 10.

5. What is getattr() used for?

a) To access the attribute of the object
b) To delete an attribute
c) To check if an attribute exists or not

d) To set an attribute

Answer: a
Explanation: The getattr() function is used to retrieve the value of an attribute from an object. It takes the object and attribute name (as a string) as arguments. For example, getattr(obj, ‘name’) returns the value of obj.name.

6. What will be the output of the following Python code?

```
class change:
    def __init__(self, x, y, z):
        self.a = x + y + z

x = change(1,2,3)
y = getattr(x, 'a')
setattr(x, 'a', y+1)
print(x.a)
```

a) 6
b) 7
c) Error
d) 0

Answer: b
Explanation: The change class constructor initializes self.a to the sum of x, y, and z, which is 1 + 2 + 3 = 6.

- getattr(x, ‘a’) retrieves the value 6.
- setattr(x, ‘a’, y + 1) updates x.a to 7.
- So, print(x.a) outputs 7.

7. What will be the output of the following Python code?

```
class test:
    def __init__(self,a):
```

```
        self.a=a

    def display(self):
        print(self.a)
obj=test()
obj.display()
```

a) Runs normally, doesn't display anything
b) Displays 0, which is the automatic default value
c) Error as one argument is required while creating the object
d) Error as display function requires additional argument

Answer: c
Explanation: The __init__ method in the test class requires one argument a besides self. When creating the object with obj = test(), no argument is passed, which leads to a TypeError because Python expects one argument. Hence, the program results in an error due to the missing required argument during object creation.

8. Is the following Python code correct?

```
class A:
    def __init__(self,b):
        self.b=b
    def display(self):
        print(self.b)
obj=A("Hello")
del obj
```

a) True
b) False

Answer: a
Explanation: Yes, the code is correct. The class A is defined properly with

an __init__ method and a display method. An object obj of class A is created with the argument “Hello”. Using del obj deletes the reference to the object without causing any error. Hence, the answer is True.

9. What will be the output of the following Python code?

```
class test:
    def __init__(self):
        self.variable = 'Old'
        self.Change(self.variable)
    def Change(self, var):
        var = 'New'
obj=test()
print(obj.variable)
```

a) Error because function change can’t be called in the __init__ function
b) ‘New’ is printed
c) ‘Old’ is printed
d) Nothing is printed

Answer: c
Explanation: In the __init__ method, self.variable is set to ‘Old’. Then, self.Change(self.variable) is called, passing the string ‘Old’ as an argument. Inside the Change method, the parameter var is assigned ‘New’, but this only changes the local variable var and does not affect self.variable. Therefore, when print(obj.variable) is called, it prints the original value ‘Old’.

10. What is Instantiation in terms of OOP terminology?

a) Deleting an instance of class
b) Modifying an instance of class
c) Copying an instance of class
d) Creating an instance of class

Answer: d
Explanation: Instantiation in Object-Oriented Programming (OOP) refers to the process of creating a specific object (instance) from a class. When a class is instantiated, memory is allocated for the new object, and its constructor (like __init__ in Python) is called to initialize it. Thus, instantiation means creating an instance of a class.

11. What will be the output of the following Python code?

```
class fruits:
   def __init__(self, price):
      self.price = price
obj=fruits(50)

obj.quantity=10
obj.bags=2

print(obj.quantity+len(obj.__dict__))
```

a) 12
b) 52
c) 13
d) 60

Answer: c
Explanation: Here,

- obj.quantity is set to 10.
- obj.__dict__ stores all instance attributes as a dictionary. Here it contains ‘price’, ‘quantity’, and ‘bags’, so its length is 3.
- The expression obj.quantity + len(obj.__dict__) evaluates to 10 + 3 = 13.

12. What will be the output of the following Python code?

```
class Demo:
  def __init__(self):
    pass

  def test(self):
    print(__name__)

obj = Demo()
obj.test()
```

a) Exception is thrown
b) __main__
c) Demo
d) test

Answer: b
Explanation: In Python, the special variable __name__ represents the name of the current module. When a script is run directly (not imported), __name__ is set to '__main__'. In this code, the function test() prints the value of __name__. Since the script is being executed as the main program, the output will be '__main__'.

1. The assignment of more than one function to a particular operator is _____

a) Operator over-assignment
b) Operator overriding
c) Operator overloading
d) Operator instance

Answer: c
Explanation: Operator overloading allows a class to define or customize the behavior of standard operators (such as +, -, *) for its own objects by

implementing special methods like __add__(), __sub__(), and others. This enables the same operator to perform different operations depending on the context or the types of objects involved.

2. Which of the following is not a class method?

a) Non-static
b) Static
c) Bounded
d) Unbounded

Answer: a
Explanation: The three different class methods in Python are static, bounded and unbounded methods. A non-static method typically refers to an instance method, which is not considered a class method. Class methods are those that take the class (cls) as the first argument and are marked with the @classmethod decorator. Static methods (@staticmethod) and unbounded methods can still be associated with classes.

3. What will be the output of the following Python code?

```
def add(c,k):
   c.test=c.test+1
   k=k+1
class A:
   def __init__(self):
      self.test = 0
def main():
   Count=A()
   k=0

   for i in range(0,25):
```

```
    add(Count,k)
  print("Count.test=", Count.test)
  print("k =", k)
main()
```

a) Exception is thrown

b)

Count.test=25

k = 25

c)

Count.test=25

k = 0

d)

Count.test=0

k = 0

Answer: c

Explanation: The add function increases the object's attribute test by 1 each time, so after 25 calls, Count.test becomes 25. However, k inside add is a local copy and changing it does not affect the original k in main(), so k stays 0.

4. Which of the following Python code creates an empty class?

a)

```
class A:
  return
```

b)

```
class A:
  pass
```

c)

```
class A:
```
d) It is not possible to create an empty class

Answer: b
Explanation: In Python, an empty class cannot have no statements inside the body—this causes a syntax error. The pass statement acts as a placeholder and allows defining an empty class without error. Using return inside the class body is invalid syntax, and leaving the class body completely empty also raises a syntax error.

5. Is the following Python code valid?

```
class B(object):
  def first(self):
    print("First method called")
  def second():
    print("Second method called")
ob = B()
B.first(ob)
```

a) It isn't as the object declaration isn't right
b) It isn't as there isn't any __init__ method for initializing class members
c) Yes, this method of calling is called unbounded method call
d) Yes, this method of calling is called bounded method call

Answer: c
Explanation: Yes, the code is valid. Calling B.first(ob) is an example of an unbounded method call, where the method is called on the class and you explicitly pass the instance (ob) as the first argument (self). Note that second() is invalid here because it lacks the self parameter.

6. What are the methods which begin and end with two underscore characters called?

a) Special methods

b) In-built methods
c) User-defined methods
d) Additional methods

Answer: a
Explanation: Methods that begin and end with double underscores (like __init__, __str__, __lt__) are called special methods (or sometimes magic methods). They define special behavior in Python classes.

7. Special methods need to be explicitly called during object creation.

a) True
b) False

Answer: b
Explanation: Special methods (like __init__) are automatically called by Python during object creation or certain operations; you don't need to call them explicitly. For example, __init__ runs automatically when you create an instance.

8. What will be the output of the following Python code?

```
class demo():
   def __repr__(self):
      return '__repr__ built-in function called'
   def __str__(self):
      return '__str__ built-in function called'
s=demo()
print(s)
```

a) Error
b) Nothing is printed
c) __str__ built-in function called
d) __repr__ built-in function called

Answer: c
Explanation: When you print an object, Python first tries to call its __str__() method to get a user-friendly string representation. If __str__() is not defined, then it falls back to __repr__(). Since both methods are defined here, print(s) calls s.__str__(), so the output is __str__ built-in function called.

9. What will be the output of the following Python code?

```
class demo():
   def __repr__(self):
      return '__repr__ built-in function called'
   def __str__(self):
      return '__str__  built-in function called'
s=demo()
print(s)
```

a) __str__ built-in function called
b) __repr__ built-in function called
c) Error
d) Nothing is printed

Answer: b
Explanation: In Python, __repr__() is called when repr() is used or when an object is displayed in the interpreter. In the code, repr(s) explicitly calls __repr__(), so the output is __repr__ built-in function called.

10. What is hasattr(obj,name) used for?

a) To access the attribute of the object
b) To delete an attribute
c) To check if an attribute exists or not
d) To set an attribute

Answer: c
Explanation: The hasattr(obj, name) function checks whether the object obj has an attribute with the name name. It returns True if the attribute exists, otherwise False.

11. What will be the output of the following Python code?

```
class stud:
  def __init__(self, roll_no, grade):
    self.roll_no = roll_no
    self.grade = grade
  def display (self):
    print("Roll no : ", self.roll_no,  ", Grade: ", self.grade)
stud1 = stud(34, 'S')
stud1.age=7
print(hasattr(stud1, 'age'))
```

a) Error as age isn’t defined
b) True
c) False
d) 7

Answer: b
Explanation: In Python, attributes can be added to objects dynamically. In the given code, stud1.age = 7 adds an age attribute to the stud1 object. The function hasattr(stud1, ‘age’) then checks for the existence of this attribute, returning True since it was successfully added.

12. What is delattr(obj,name) used for?

a) To print deleted attribute
b) To delete an attribute
c) To check if an attribute is deleted or not
d) To set an attribute

Answer: b
Explanation: The delattr(obj, name) function in Python is used to delete an attribute from an object. If the attribute specified by name exists, it will be removed from the object obj.

13. __del__ method is used to destroy instances of a class.

a) True
b) False

Answer: a
Explanation: Yes, the __del__ method in Python is known as a destructor. It is automatically called when an object is about to be destroyed, typically during garbage collection. This method can be used to clean up resources like closing files or network connections.

14. What will be the output of the following Python code?

```
class student:
  'Base class for all students'
  def __init__(self, roll_no, grade):
    self.roll_no = roll_no
    self.grade = grade
  def display (self):
    print("Roll no : ", self.roll_no,  ", Grade: ", self.grade)
print(student.__doc__)
```

a) Exception is thrown
b) __main__
c) Nothing is displayed
d) Base class for all students

Answer: d
Explanation: In Python, a docstring is a special string placed right after the definition of a class, function, or module. It describes the purpose of

that block and can be accessed using the __doc__ attribute. Here, student.__doc__ returns “Base class for all students”.

15. What does print(Test.__name__) display (assuming Test is the name of the class)?

a) ()
b) Exception is thrown
c) Test
d) __main__

Answer: c
Explanation: Test.__name__ returns the name of the class as a string. So, if the class is defined as class Test:, then print(Test.__name__) will output “Test”.

1. Which of the following best describes inheritance?

a) Ability of a class to derive members of another class as a part of its own definition
b) Means of bundling instance variables and methods in order to restrict access to certain class members
c) Focuses on variables and passing of variables to functions
d) Allows for implementation of elegant software that is well designed and easily modified

Answer: a
Explanation: Inheritance is a fundamental concept in object-oriented programming that allows a class (called a derived or child class) to inherit properties and behaviors (methods and attributes) from another class (called a base or parent class). This promotes code reuse and establishes a natural hierarchy between classes.

2. Which of the following statements is wrong about inheritance?

a) Protected members of a class can be inherited

b) The inheriting class is called a subclass
c) Private members of a class can be inherited and accessed
d) Inheritance is one of the features of OOP

Answer: c
Explanation: Private members (with double underscores) are inherited but cannot be directly accessed by subclasses due to name mangling. Only public and protected members are accessible in subclasses. Hence, private members are hidden to maintain encapsulation.

3. What will be the output of the following Python code?

```
class Demo:
   def __new__(self):
      self.__init__(self)
      print("Demo's __new__() invoked")
   def __init__(self):
      print("Demo's __init__() invoked")
class Derived_Demo(Demo):
   def __new__(self):
      print("Derived_Demo's __new__() invoked")
   def __init__(self):
      print("Derived_Demo's __init__() invoked")
def main():
   obj1 = Derived_Demo()
   obj2 = Demo()
main()
```

a)
Derived_Demo’s __init__() invoked
Derived_Demo's __new__() invoked
Demo's __init__() invoked

Demo's __new__() invoked
b)
Derived_Demo's __new__() invoked
Demo's __init__() invoked
Demo's __new__() invoked
c)
Derived_Demo's __new__() invoked
Demo's __new__() invoked
d)
Derived_Demo's __init__() invoked
Demo's __init__() invoked

Answer: b
Explanation: In Python, the __new__() method is responsible for creating a new instance before __init__() initializes it. In the code, Derived_Demo.__new__() is called and does not return an instance, so __init__() is not executed for Derived_Demo. For Demo, the __new__() method explicitly calls __init__() inside it, so both get executed in order.

4. What will be the output of the following Python code?

```
class Test:
    def __init__(self):
        self.x = 0
class Derived_Test(Test):
    def __init__(self):
        self.y = 1
def main():
    b = Derived_Test()
    print(b.x,b.y)
```

```
main()
```
a) 0 1
b) 0 0
c) Error because class B inherits A but variable x isn't inherited
d) Error because when object is created, argument must be passed like Derived_Test(1)

Answer: c
Explanation: In Python, if a child class (Derived_Test) defines its own __init__() method, it overrides the parent's __init__() (Test). Since super().__init__() is not called, the attribute x is never initialized, leading to an AttributeError when accessing b.x.

5. What will be the output of the following Python code?

```
class A():
    def disp(self):
        print("A disp()")
class B(A):
    pass
obj = B()
obj.disp()
```

a) Invalid syntax for inheritance
b) Error because when object is created, argument must be passed
c) Nothing is printed
d) A disp()

Answer: d
Explanation: Class B inherits from class A and does not override the disp() method, so it uses the one defined in A. When obj.disp() is called on an instance of B, it prints "A disp()".

6. All subclasses are a subtype in object-oriented programming.
a) True
b) False

Answer: b
Explanation: Not all subclasses are subtypes. A subclass becomes a subtype only if it adheres to the Liskov Substitution Principle — meaning it can be used in place of its superclass without altering the program's correctness. If it violates expected behavior, it's not a valid subtype even though it's a subclass.
7. When defining a subclass in Python that is meant to serve as a subtype, the subtype Python keyword is used.
a) True
b) False

Answer: b
Explanation: Python does not have a keyword called subtype. To define a subclass that acts as a subtype, you simply use inheritance syntax like class SubClass(SuperClass):. Whether it acts as a subtype depends on if it properly extends the behavior of the superclass, not on a special keyword.
8. Suppose B is a subclass of A, to invoke the __init__ method in A from B, what is the line of code you should write?
a) A.__init__(self)
b) B.__init__(self)
c) A.__init__(B)
d) B.__init__(A)

Answer: a
Explanation: To explicitly call the __init__ method of a superclass (A)

from a subclass (B), you use A.__init__(self). This ensures the base class is properly initialized when the subclass is instantiated. Alternatively, you can also use super().__init__() in modern Python.

9. What will be the output of the following Python code?

```
class Test:
    def __init__(self):
        self.x = 0
class Derived_Test(Test):
    def __init__(self):
        Test.__init__(self)
        self.y = 1
def main():
    b = Derived_Test()
    print(b.x,b.y)
main()
```

a) Error because class B inherits A but variable x isn't inherited
b) 0 0
c) 0 1
d) Error, the syntax of the invoking method is wrong

Answer: c
Explanation: In the subclass Derived_Test, the constructor explicitly calls the base class constructor using Test.__init__(self), so self.x is initialized to 0. Then, self.y is set to 1. Therefore, print(b.x, b.y) outputs: 0 1.

10. What will be the output of the following Python code?

```
class A:
    def __init__(self, x= 1):
        self.x = x
class der(A):
    def __init__(self,y = 2):
```

```
        super().__init__()
        self.y = y
def main():
    obj = der()
    print(obj.x, obj.y)
main()
```

a) Error, the syntax of the invoking method is wrong
b) The program runs fine but nothing is printed
c) 1 0
d) 1 2

Answer: d
Explanation: The subclass der calls the superclass A's constructor using super().__init__(), which sets self.x = 1 (default value). Then self.y = 2 is set in der's constructor. So print(obj.x, obj.y) outputs: 1 2.

11. What does built-in function type do in context of classes?

a) Determines the object name of any value
b) Determines the class name of any value
c) Determines class description of any value
d) Determines the file name of any value

Answer: b
Explanation: The built-in type() function in Python returns the class/type of the given object or value. For example, type(5) returns <class 'int'>, indicating the object's class.

12. Which of the following is not a type of inheritance?

a) Double-level
b) Multi-level
c) Single-level

d) Multiple

Answer: a
Explanation: There is no inheritance type called “Double-level” in object-oriented programming. Common inheritance types include Single-level, Multi-level, Multiple, Hierarchical, and Hybrid inheritance.

13. What does built-in function help do in context of classes?

a) Determines the object name of any value
b) Determines the class identifiers of any value
c) Determines class description of any built-in type
d) Determines class description of any user-defined built-in type

Answer: c
Explanation: The built-in help() function in Python provides a detailed description of modules, classes, functions, and methods. When used on a built-in type or class, it displays the documentation, including available methods and usage, helping developers understand its functionality.

14. What will be the output of the following Python code?

```
class A:
   def one(self):
      return self.two()

   def two(self):
      return 'A'

class B(A):
   def two(self):
      return 'B'
obj1=A()
obj2=B()
```

```
print(obj1.two(),obj2.two())
```

a) A A
b) A B
c) B B
d) An exception is thrown

Answer: b
Explanation: obj1 is an instance of class A, so two() returns ‘A’. obj2 is an instance of subclass B, which overrides two(), so it returns ‘B’.

15. What type of inheritance is illustrated in the following Python code?

```
class A():
    pass
class B():
    pass
class C(A,B):
    pass
```

a) Multi-level inheritance
b) Multiple inheritance
c) Hierarchical inheritance
d) Single-level inheritance

Answer: b
Explanation: Class C inherits directly from two classes A and B, which means it has multiple parent classes. This is called multiple inheritance, where a subclass inherits features from more than one superclass.

1. What type of inheritance is illustrated in the following Python code?

```
class A():
    pass
class B(A):
    pass
```

```
class C(B):
    pass
```

a) Multi-level inheritance
b) Multiple inheritance
c) Hierarchical inheritance
d) Single-level inheritance

Answer: a
Explanation: Class B inherits from A, and class C inherits from B, forming a chain of inheritance. This is called multi-level inheritance, where a class is derived from a subclass, creating multiple levels in the hierarchy.

2. What does single-level inheritance mean?

a) A subclass derives from a class which in turn derives from another class
b) A single superclass inherits from multiple subclasses
c) A single subclass derives from a single superclass
d) Multiple base classes inherit a single derived class

Answer: c
Explanation: Single-level inheritance means a subclass inherits directly from one superclass only. It forms a simple parent-child relationship without multiple levels or multiple parents.

3. What will be the output of the following Python code?

```
class A:
    def __init__(self):
        self.__i = 1
        self.j = 5

    def display(self):
        print(self.__i, self.j)
```

```
class B(A):
    def __init__(self):
        super().__init__()
        self.__i = 2
        self.j = 7
c = B()
c.display()
```

a) 2 7
b) 1 5
c) 1 7
d) 2 5

Answer: c
Explanation: Here,

- self.__i in both classes A and B is name-mangled to be class-specific (_A__i and _B__i), so they are treated as separate variables.
- In B.__init__(), super().__init__() sets self._A__i = 1 and self.j = 5.
- Then B.__init__() sets a new self._B__i = 2 and updates self.j = 7.
- In display(), self.__i refers to self._A__i, which is still 1, and self.j is 7.

4. Which of the following statements isn't true?

a) A non-private method in a superclass can be overridden
b) A derived class is a subset of superclass
c) The value of a private variable in the superclass can be changed in the subclass
d) When invoking the constructor from a subclass, the constructor of

superclass is automatically invoked

Answer: c
Explanation: Private variables in Python (defined with __varname) are name-mangled, meaning they can't be directly accessed or modified from a subclass. You can't change them without using a public/protected setter method or by knowing and accessing their mangled name, which defeats encapsulation. So, statement "The value of a private variable in the superclass can be changed in the subclass" is false.

5. What will be the output of the following Python code?

```
class A:
    def __init__(self,x):
        self.x = x
    def count(self,x):
        self.x = self.x+1
class B(A):
    def __init__(self, y=0):
        A.__init__(self, 3)
        self.y = y
    def count(self):
        self.y += 1
def main():
    obj = B()
    obj.count()
    print(obj.x, obj.y)
main()
```

a) 3 0
b) 3 1
c) 0 1

d) An exception in thrown

Answer: b
Explanation: Class B inherits from class A. When obj = B() is created, it calls A.__init__ with x=3, so self.x = 3. Then, B's count() method is called, which increments self.y from 0 to 1. The overridden count() in B is used, so self.x remains unchanged. Hence, the output is 3 1.

6. What will be the output of the following Python code?

```
class A:
        pass
class B(A):
        pass
obj=B()
print(isinstance(obj,A))
```

a) True
b) False
c) Wrong syntax for isinstance() method
d) Invalid method for classes

Answer: a
Explanation: The object obj is an instance of class B, which inherits from class A. Since B is a subclass of A, obj is also considered an instance of A. Therefore, isinstance(obj, A) returns True.

7. Which of the following statements is true?

a) The __new__() method automatically invokes the __init__ method
b) The __init__ method is defined in the object class
c) The __eq() method is defined in the object class
d) The __repr__() method is defined in the object class

Answer: d
Explanation: __repr__() is a method in Python's base object class that returns a string representation of an object. Other options are incorrect because __init__() isn't defined in object, __eq() is misspelled, and __new__() doesn't automatically call __init__().

8. Method issubclass() checks if a class is a subclass of another class.

a) True
b) False

Answer: a
Explanation: The issubclass(ClassA, ClassB) function returns True if ClassA is a subclass of ClassB, either directly or indirectly. It is a built-in Python method used to check class relationships.

9. What will be the output of the following Python code?

```
class A:
    def __init__(self):
        self.__x = 1
class B(A):
    def display(self):
        print(self.__x)
def main():
    obj = B()
    obj.display()
main()
```

a) 1
b) 0
c) Error, invalid syntax for object declaration
d) Error, private class member can't be accessed in a subclass

Answer: d
Explanation: The variable __x in class A is private due to name mangling (it becomes _A__x). It cannot be accessed directly as self.__x in subclass B. Trying to do so causes an AttributeError.

10. What will be the output of the following Python code?

```
class A:
    def __init__(self):
        self._x = 5
class B(A):
    def display(self):
        print(self._x)
def main():
    obj = B()
    obj.display()
main()
```

a) Error, invalid syntax for object declaration
b) Nothing is printed
c) 5
d) Error, private class member can't be accessed in a subclass

Answer: c
Explanation: The variable _x is a protected member (single underscore), which can be accessed in subclasses. So, B can access and print _x without any error, outputting 5.

11. What will be the output of the following Python code?

```
class A:
    def __init__(self,x=3):
        self._x = x
class B(A):
    def __init__(self):
```

```
        super().__init__(5)
    def display(self):
        print(self._x)
def main():
    obj = B()
    obj.display()

main()
```

a) 5
b) Error, class member x has two values
c) 3
d) Error, protected class member can’t be accessed in a subclass

Answer: a
Explanation: Class B inherits from class A and explicitly calls super().__init__(5) in its constructor, setting _x to 5. Since _x is a protected member (not private), it is accessible in the subclass, and display() prints 5.

12. What will be the output of the following Python code?

```
class A:
    def test1(self):
        print(" test of A called ")
class B(A):
    def test(self):
        print(" test of B called ")
class C(A):
    def test(self):
        print(" test of C called ")
class D(B,C):
    def test2(self):
```

```
        print(" test of D called ")
obj=D()
obj.test()
```

a)
test of B called
test of C called
b)
test of C called
test of B called
c) test of B called
d) Error, both the classes from which D derives has same method test()

Answer: c
Explanation: Class D inherits from B and C. When obj.test() is called, Python uses the Method Resolution Order (MRO), which checks parents from left to right. Since B is listed before C, D uses B's test() method, printing "test of B called".

13. What will be the output of the following Python code?

```
class A:
    def test(self):
        print("test of A called")
class B(A):
    def test(self):
        print("test of B called")
        super().test()
class C(A):
    def test(self):
        print("test of C called")
        super().test()
class D(B,C):
```

```
    def test2(self):
        print("test of D called")
obj=D()
obj.test()
```

a)
test of B called
test of C called
test of A called
b)
test of C called
test of B called
c)
test of B called
test of C called
d) Error, all the three classes from which D derives has same method test()

Answer: a
Explanation: Class D inherits from B and C. When obj.test() is called, the method resolution order (MRO) is: D -> B -> C -> A.

- B.test() is called first, prints “test of B called”, then calls super().test().
- super() in B refers to C (next in MRO), so C.test() is called, which prints “test of C called”, then calls super().test().
- super() in C refers to A, so A.test() is called, which prints “test of A called”.

1. Which of the following best describes polymorphism?
a) Ability of a class to derive members of another class as a part of its own definition
b) Means of bundling instance variables and methods in order to

restrict access to certain class members
c) Focuses on variables and passing of variables to functions
d) Allows for objects of different types and behaviour to be treated as the same general type

Answer: d
Explanation: Polymorphism in object-oriented programming allows objects of different classes to be treated through the same interface. For example, different classes may have their own implementation of a method like draw(), but they can all be called through a common base class reference, making the code more flexible and extensible.

2. What is the biggest reason for the use of polymorphism?

a) It allows the programmer to think at a more abstract level
b) There is less program code to write
c) The program will have a more elegant design and will be easier to maintain and update
d) Program code takes up less space

- Answer: c
  Explanation: Polymorphism helps hide details (abstraction) and can reduce repeated code, but its main benefit is making programs more flexible and easier to manage. It lets different types of objects be used in the same way, through a common interface. This makes it easier to add new features or make changes without touching existing code, which helps the program grow and stay organized.

3. What is the use of duck typing?

a) More restriction on the type values that can be passed to a given method
b) No restriction on the type values that can be passed to a given method

c) Less restriction on the type values that can be passed to a given method
d) Makes the program code smaller

- Answer: c
  Explanation: Duck typing focuses on whether an object behaves like a certain type (i.e., has the required methods/attributes), rather than its actual class. This allows more flexible and less restrictive coding.

.

4. What will be the output of the following Python code?

```
class A:
    def __str__(self):
        return '1'
class B(A):
    def __init__(self):
        super().__init__()
class C(B):
    def __init__(self):
        super().__init__()
def main():
    obj1 = B()
    obj2 = A()
    obj3 = C()
    print(obj1, obj2,obj3)
main()
```

a) 1 1 1
b) 1 2 3
c) ‘1’ ‘1’ ‘1’

d) An exception is thrown

Answer: a
Explanation: All three objects obj1, obj2, and obj3 are instances of classes that ultimately inherit from class A, which defines the __str__() method returning '1'. Therefore, printing any of these instances (whether of class A, B, or C) invokes the __str__() method of class A, resulting in the output 1 for each.

5. What will be the output of the following Python code?

```
class Demo:
    def __init__(self):
        self.x = 1
    def change(self):
        self.x = 10
class Demo_derived(Demo):
    def change(self):
        self.x=self.x+1
        return self.x
def main():
    obj = Demo_derived()
    print(obj.change())

main()
```

a) 11
b) 2
c) 1
d) An exception is thrown

Answer: b
Explanation: The object obj is an instance of Demo_derived, which

inherits the __init__ method from Demo, initializing self.x to 1. When obj.change() is called, it uses the change() method defined in Demo_derived, which increments self.x by 1 (resulting in 2) and returns the updated value, 2.

6. A class in which one or more methods are only implemented to raise an exception is called an abstract class.

a) True
b) False

Answer: a
Explanation: An abstract class often contains one or more methods that are not fully implemented and typically raise exceptions like NotImplementedError to enforce that subclasses must override them. This helps define a common interface while leaving the implementation details to derived classes.

7. Overriding means changing behaviour of methods of derived class methods in the base class.

a) True
b) False

Answer: b
Explanation: Overriding means changing the behavior of base class methods in the derived class, not the other way around. The derived class provides its own implementation of a method already defined in the base class.

8. What will be the output of the following Python code?

```
class A:
   def __repr__(self):
      return "1"
class B(A):
```

```
    def __repr__(self):
        return "2"
class C(B):
    def __repr__(self):
        return "3"
o1 = A()
o2 = B()
o3 = C()
print(obj1, obj2, obj3)
```

a) 1 1 1
b) 1 2 3
c) '1' '1' '1'
d) An exception is thrown

Answer: d
Explanation: In the print() statement, the variables obj1, obj2, obj3 are used instead of o1, o2, o3. Since obj1, obj2, and obj3 were never defined anywhere in the program, Python will raise a NameError, stating that these variables are not found.

9. What will be the output of the following Python code?

```
class A:
    def __init__(self):
        self.multiply(15)
        print(self.i)

    def multiply(self, i):
        self.i = 4 * i;
class B(A):
    def __init__(self):
        super().__init__()
```

```
    def multiply(self, i):
        self.i = 2 * i;
obj = B()
```

a) 15
b) 60
c) An exception is thrown
d) 30

Answer: d
Explanation: When obj = B() is created, the constructor B.__init__() is invoked, which calls super().__init__(), leading to the execution of A.__init__(). Within A.__init__(), the method self.multiply(15) is called. Since obj is an instance of class B, and class B overrides the multiply() method, B.multiply(15) is executed. This method sets self.i = 2 * 15 = 30.

10. What will be the output of the following Python code?

```
class Demo:
    def check(self):
        return " Demo's check "
    def display(self):
        print(self.check())
class Demo_Derived(Demo):
    def check(self):
        return " Derived's check "
Demo().display()
Demo_Derived().display()
```

a)
Demo's check
Derived's check

b)
Demo's check
Demo's check
c) Derived’s check Demo’s check
d) Syntax error

Answer: a
Explanation: Demo().display() calls Demo’s display() method, which in turn calls self.check(). Since self refers to an instance of Demo, it invokes Demo.check(), printing “Demo’s check”. On the other hand, Demo_Derived().display() also uses the inherited display() method from Demo, but this time self refers to an instance of Demo_Derived. Therefore, self.check() calls the overridden method in Demo_Derived, printing “Derived’s check”.
Output:
Demo's check
Derived's check

11. What will be the output of the following Python code?

```
class A:
    def __init__(self):
        self.multiply(15)
    def multiply(self, i):
        self.i = 4 * i;
class B(A):
    def __init__(self):
        super().__init__()
        print(self.i)

    def multiply(self, i):
        self.i = 2 * i;
```

```
obj = B()
```
a) 15
b) 30
c) An exception is thrown
d) 60

Answer: b
Explanation: When B() is instantiated, it calls A's __init__() via super(). Inside A.__init__(), the method self.multiply(15) is called — but since self is an instance of B, B's multiply() is invoked.
So, self.i = 2 * 15 = 30, and print(self.i) outputs 30.

12. What will be the output of the following Python code?

```
class Demo:
    def __check(self):
        return " Demo's check "
    def display(self):
        print(self.check())
class Demo_Derived(Demo):
    def __check(self):
        return " Derived's check "
Demo().display()
Demo_Derived().display()
```

a) Demo's check Derived's check
b) Demo's check Demo's check
c) Derived's check Demo's check
d) Error

Answer: d
Explanation: The method __check() is private due to the double underscore (__), making it inaccessible outside the class. The display()

method incorrectly tries to call self.check(), which does not exist, leading to an AttributeError at runtime.

13. What will be the output of the following Python code?

```
class A:
    def __init__(self, x, y):
        self.x = x
        self.y = y
    def __str__(self):
        return 1
    def __eq__(self, other):
        return self.x * self.y == other.x * other.y
obj1 = A(5, 2)
obj2 = A(2, 5)
print(obj1 == obj2)
```

a) False
b) 1
c) True
d) An exception is thrown

Answer: c
Explanation: The __eq__ method is overridden to compare the products of x and y for two A objects. Since 5 * 2 equals 2 * 5, obj1 == obj2 evaluates to True.

14. What will be the output of the following Python code?

```
class A:
    def one(self):
        return self.two()
    def two(self):
        return 'A'
class B(A):
```

```
    def two(self):
        return 'B'
obj2=B()
print(obj2.two())
```

a) A
b) An exception is thrown
c) A B
d) B

Answer: d
Explanation: obj2 is an instance of class B, which overrides the two() method to return ‘B’. So, calling obj2.two() returns ‘B’.

15. Which of the following statements is true?

a) A non-private method in a superclass can be overridden
b) A subclass method can be overridden by the superclass
c) A private method in a superclass can be overridden
d) Overriding isn’t possible in Python

Answer: a
Explanation: In Python, non-private (public or protected) methods in a superclass can be overridden by subclasses to provide new or extended behavior. Private methods (with double underscores) are name-mangled and cannot be directly overridden.

Which of these is not a fundamental features of OOP?

a) Encapsulation
b) Inheritance
c) Instantiation
d) Polymorphism

Answer: c
Explanation: Encapsulation, Inheritance, and Polymorphism are fundamental features of Object-Oriented Programming (OOP). Instantiation is the process of creating an object from a class, but it is not considered a fundamental OOP concept.

Which of the following is the most suitable definition for encapsulation?

a) Ability of a class to derive members of another class as a part of its own definition

b) Means of bundling instance variables and methods in order to restrict access to certain class members

c) Focuses on variables and passing of variables to functions

d) Allows for implementation of elegant software that is well designed and easily modified

Answer: b
Explanation: Encapsulation is the concept of wrapping data (variables) and code (methods) together while restricting direct access to some of the object's components to protect the integrity of the data.

What will be the output of the following Python code?

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

def display(self):
return self.__b
obj = Demo()
print(obj.a)
```

The program has an error because there isn’t any function to return self.a
b) The program has an error because b is private and display(self) is returning a private member
c) The program runs fine and 1 is printed
d) The program has an error as you can’t name a class member using __b

Answer: c
Explanation: The attribute a is public and can be accessed directly with obj.a, so 1 is printed. The private variable __b is accessed only inside the class and doesn’t affect accessing a.

What will be the output of the following Python code?

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

def display(self):
return self.__b

obj = Demo()
print(obj.__b)
```

The program has an error because there isn’t any function to return self.a
b) The program has an error because b is private and display(self) is returning a private member
c) The program has an error because b is private and hence can’t be printed

d) The program runs fine and 1 is printed

Answer: c
Explanation: The code throws an error because __b is a private variable and cannot be accessed directly outside the class. Attempting to print obj.__b raises an AttributeError due to name mangling. Accessing it through a method like display() would work, but direct access is not allowed.

Methods of a class that provide access to private members of the class are called as _____ and _____

a) getters/setters
b) __repr__/__str__
c) user-defined functions/in-built functions
d) __init__/__del__

Answer: a
Explanation: Getters and setters are special methods used to access (get) and update (set) private variables of a class. They provide a controlled way to read or modify private data, ensuring data encapsulation and validation when needed.

Which of these is a private data field?

```
def Demo:
def __init__(self):
__a = 1
self.__b = 1
self.__c__ = 1
__d__= 1
```

__a
b) __b
c) __c__

d) __d__

Answer: b
Explanation: self.__b is a private instance variable due to name mangling in Python (it becomes _Demo__b).
__a and __d__ are local variables inside the constructor, not class fields.
__c__ uses double underscores before and after, which by convention is reserved for special methods or attributes, not private fields.

What will be the output of the following Python code?

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

def get(self):
        return self.__b

obj = Demo()
print(obj.get())
```

The program has an error because there isn't any function to return self.a
b) The program has an error because b is private and display(self) is returning a private member
c) The program has an error because b is private and hence can't be printed
d) The program runs fine and 1 is printed

Answer: d
Explanation: The private variable __b is accessed within the class using

the get() method. Since get() returns self.__b, calling obj.get() correctly returns 1. Accessing private variables from within class methods is allowed, so the program runs without any error and prints 1.

What will be the output of the following Python code?

```
class Demo:
def __init__(self):
    self.a = 1
    self.__b = 1
def get(self):
    return self.__b
obj = Demo()
obj.a=45
print(obj.a)
```

The program runs properly and prints 45

b) The program has an error because the value of members of a class can't be changed from outside the class

c) The program runs properly and prints 1

d) The program has an error because the value of members outside a class can only be changed as self.a=45

Answer: a

Explanation: The public variable a can be changed directly from outside the class. Setting obj.a = 45 updates its value, so print(obj.a) outputs 45 without any error.

Private members of a class cannot be accessed.

a) True

b) False

Answer: b

Explanation: Private members (with double underscores) cannot be

accessed directly outside the class, but they can still be accessed indirectly through name mangling or via public methods like getters and setters.

The purpose of name mangling is to avoid unintentional access of private class members.

a) True

b) False

Answer: a

Explanation: Name mangling in Python changes the name of private variables (those with double underscores) internally to prevent accidental access or modification from outside the class, helping to protect private members.

What will be the output of the following Python code?

```
class fruits:
def __init__(self):
self.price = 100
self.__bags = 5
def display(self):
print(self.__bags)
obj=fruits()
obj.display()
```

The program has an error because display() is trying to print a private class member

b) The program runs fine but nothing is printed

c) The program runs fine and 5 is printed

d) The program has an error because display() can't be accessed

Answer: c
Explanation: The private variable __bags is accessible inside the class methods, so display() can print its value 5 without any error.

What will be the output of the following Python code?

```
class student:
def __init__(self):
self.marks = 97
self.__cgpa = 8.7
def display(self):
print(self.marks)
obj=student()
print(obj._student__cgpa)
```

The program runs fine and 8.7 is printed
b) Error because private class members can't be accessed
c) Error because the proper syntax for name mangling hasn't been implemented
d) The program runs fine but nothing is printed

Answer: a
Explanation: The private variable __cgpa is name-mangled by Python to _student__cgpa. Accessing it directly using obj._student__cgpa works and prints 8.7.

Which of the following is false about protected class members?

a) They begin with one underscore
b) They can be accessed by subclasses
c) They can be accessed by name mangling method
d) They can be accessed within a class

Answer: c
Explanation: Protected members (single underscore prefix) are a

convention indicating they should be treated as non-public but are still accessible directly and by subclasses. Name mangling is only applied to private members (double underscore prefix), not protected members.

What will be the output of the following Python code?

```
class objects:
def __init__(self):
self.colour = None
self._shape = "Circle"

def display(self, s):
self._shape = s
obj=objects()
print(obj._objects_shape)
```

The program runs fine because name mangling has been properly implemented

b) Error because the member shape is a protected member

c) Error because the proper syntax for name mangling hasn't been implemented

d) Error because the member shape is a private member

Answer: b

Explanation: The attribute _shape is a protected member (single underscore), not private, so name mangling does not apply. Accessing obj._objects_shape causes an AttributeError because the name is incorrect — the correct attribute is obj._shape.

How many except statements can a try-except block have?

a) zero

b) one

c) more than one

d) more than zero

Answer: d
Explanation: A try block can have one or more except clauses to handle different exceptions. It must have at least one except block to catch errors.

When will the else part of try-except-else be executed?

a) always
b) when an exception occurs
c) when no exception occurs
d) when an exception occurs in to except block

Answer: c
Explanation: In a try-except-else block, the else part is executed only if no exception is raised in the try block. If an exception occurs, the except block runs instead, and the else block is skipped.

Is the following Python code valid?

```
try:
# Do something
except:
# Do something
finally:
# Do something
```

no, there is no such thing as finally
b) no, finally cannot be used with except
c) no, finally must come before except
d) yes

Answer: d
Explanation: In Python, it is perfectly valid to use a try-except-finally block. The finally clause is always executed, regardless of whether an exception occurred or was handled.

Is the following Python code valid?

```
try:
# Do something
except:
# Do something
else:
# Do something
```

no, there is no such thing as else
b) no, else cannot be used with except
c) no, else must come before except
d) yes

Answer: d
Explanation: In Python, the try-except-else construct is valid. The else block is executed only if no exception occurs in the try block.

Can a single except block handle multiple exceptions in Python?

a) Yes, like except TypeError, SyntaxError
b) Yes, like except [TypeError, SyntaxError]
c) Yes, like except (TypeError, SyntaxError)
d) No

Answer: c
Explanation: In Python, a single except block can handle multiple exceptions by grouping them inside parentheses as a tuple. This syntax allows one block to catch and handle different exception types together.

Example:

```
try:
# code that may raise TypeError or SyntaxError
except (TypeError, SyntaxError):
# handle both exceptions here
```

When is the finally block executed?

a) when there is no exception
b) when there is an exception
c) only if some condition that has been specified is satisfied
d) always

Answer: d
Explanation: The finally block is always executed, regardless of whether an exception occurs or not. It is typically used for cleanup actions like closing files or releasing resources, ensuring that such code runs no matter what happens in the try or except blocks.

What will be the output of the following Python code?

```
def foo():
try:
return 1
finally:
return 2
k = foo()
print(k)
```

1
b) 2
c) 3
d) error, there is more than one return statement in a single try-finally block

Answer: b
Explanation: In Python, if both the try block and the finally block contain return statements, the return in the finally block overrides the one in the try block. So, even though return 1 is in the try, the function ends up returning 2 because of the finally block.

What will be the output of the following Python code?

```
def foo():
try:
print(1)
finally:
print(2)
foo()
```

a)
1
2

1
c) 2
d) none of the mentioned

Answer: a
Explanation: In this code, the try block executes and prints 1. The finally block always executes, regardless of whether an exception occurs, so it prints 2. Therefore, the output is both 1 and 2, printed in order.

What will be the output of the following Python code?

```
try:
if '1' != 1:
raise "someError"
else:
print("someError has not occurred")
```

```
except "someError":
print ("someError has occurred")
```

someError has occurred
b) someError has not occurred
c) invalid code
d) none of the mentioned

Answer: c
Explanation: In Python, exceptions must be derived from the built-in BaseException class (usually subclasses of Exception). You cannot raise a string like “someError”. Also, the except clause expects an exception type, not a string. This code will raise a TypeError at runtime, making it invalid.

What happens when ‘1’ == 1 is executed?

a) we get a True
b) we get a False
c) an TypeError occurs
d) a ValueError occurs

Answer: b
Explanation: In Python, ‘1’ is a string and 1 is an integer. Comparing them using == returns False, because they are of different data types and do not hold the same value in Python’s type system.

The following Python code will result in an error if the input value is entered as -5.

```
x = int(input("Enter a number: "))
print("You entered:", x)

if x < 0:
assert False, 'Spanish'
```

True
b) False

Answer: a
Explanation: The statement assert False, ‘Spanish’ will always raise an AssertionError with the message ‘Spanish’ regardless of any input because the condition is explicitly False. Therefore, if this line runs, it will result in an error (AssertionError).

What will be the output of the following Python code?

```
x=10
y=8
assert x>y, 'X too small'
```

Assertion Error
b) 10 8
c) No output
d) 108

Answer: c
Explanation: The code shown above results in an error if and only if x<y. However, in the above case, since x>y, there is no error. Since there is no print statement, hence there is no output.

What will be the output of the following Python code?

```
#generator
def f(x):
yield x+1
g=f(8)
print(next(g))
```

8
b) 9

c) 7
d) Error

Answer: b
Explanation: The function f(x) is a generator that yields x + 1. When f(8) is called, it returns a generator object g. Calling next(g) executes the function up to the yield statement, which yields 8 + 1 = 9. Hence, the output is 9.

What will be the output of the following Python code?

```
def f(x):
yield x+1
print("test")
yield x+2
g=f(9)
```

Error
b) test
c)
test
10
12
No output

Answer: d
Explanation: The generator f(9) is defined, and g is assigned the generator object, but since next() is never called on g, the generator's code never runs, so nothing is printed or yielded. Therefore, the output is none (no output).

What will be the output of the following Python code?

```
def f(x):
yield x+1
```

```
print("test")
yield x+2
g=f(10)
print(next(g))
print(next(g))
```

No output

b)

11

test

12

c)

11

test

11

Answer: b

Explanation: The first call to next(g) runs the generator until it reaches the first yield, which returns x + 1 → 11. The second call to next(g) resumes execution right after the first yield, prints “test”, and then reaches the second yield, which returns x + 2 → 12.

Therefore, the output is:

11

test

12

What will be the output of the following Python code?

```
def a():
try:
f(x, 4)
finally:
print('after f')
```

```
print('after f?')
a()
```

No output
b) after f?
c)

```
after f
error
```

after f

Answer: c
Explanation: The function f is called inside the try block with x undefined, which raises a NameError. The finally block executes and prints “after f”. However, since the exception is not caught (no except block), the error propagates, and the line print(‘after f?’) is never executed. Therefore, the program results in an error after printing “after f”.
The output will be:
after f
Traceback (most recent call last):
...
NameError: name 'x' is not defined

What will be the output of the following Python code?

```
def f(x):
for i in range(5):
yield i
g=f(8)
print(list(g))
```

[0, 1, 2, 3, 4]
b) [1, 2, 3, 4, 5, 6, 7, 8]
c) [1, 2, 3, 4, 5]

d) [0, 1, 2, 3, 4, 5, 6, 7]

Answer: a
Explanation: The parameter x is not used inside the function f(). The function simply yields numbers from 0 to 4 using range(5). So, converting the generator g to a list prints [0, 1, 2, 3, 4].

The error displayed in the following Python code is?

```
import itertools
l1=(1, 2, 3)
l2=[4, 5, 6]
l=itertools.chain(l1, l2)
print(next(l1))
```

'list' object is not iterator
b) 'tuple' object is not iterator
c) 'list' object is iterator
d) 'tuple' object is iterator

Answer: b
Explanation: The error occurs because l1 is a tuple, which is iterable but not an iterator. The next() function requires an iterator, so calling next(l1) directly raises a TypeError. To fix it, use next(iter(l1)).

Which of the following is not an exception handling keyword in Python?

a) try
b) except
c) accept
d) finally

Answer: c
Explanation: In Python, the keywords used for exception handling are

try, except, else, and finally. The word accept is not a valid Python keyword for exception handling and will result in a syntax error if used.
What will be the output of the following Python code?

```
g = (i for i in range(5))
print(type(g))
```

class <'loop'>
b) class <'iteration'>
c) class <'range'>
d) class <'generator'>

Answer: d
Explanation: The expression (i for i in range(5)) creates a generator object in Python. When you check its type using type(g), it returns <class 'generator'>, which indicates that g is a generator. Generators are used for lazy iteration and are created using generator expressions or functions with yield.

What happens if the file is not found in the following Python code?

```
a=False
while not a:
try:
f_n = input("Enter file name")
i_f = open(f_n, 'r')
except:
print("Input file not found")
```

No error
b) Assertion error
c) Input output error
d) Name error

Answer: a
Explanation: If the file is not found, the open(f_n, ‘r’) line will raise a FileNotFoundError, which is caught by the except block. The message “Input file not found” is printed, and the program continues due to the while not a: loop. Since the exception is handled gracefully, no unhandled error is raised, making the answer “No error.”

What will be the output of the following Python code?

```
lst = [1, 2, 3]
print(lst[3])
```

NameError
b) ValueError
c) IndexError
d) TypeError

Answer: c
Explanation: The list lst has elements at indices 0, 1, and 2. Trying to access lst[3] is out of range, as index 3 does not exist. This raises an IndexError, which occurs when a sequence subscript is out of range.

What will be the output of the following Python code?

```
print(t[5])
```

IndexError
b) NameError
c) TypeError
d) ValeError

Answer: b
Explanation: In the expression t[5], the variable t is used without being defined anywhere in the code. This results in a NameError, which

occurs when a local or global name is not found. Python raises this error when it encounters a name that it doesn't recognize.

What will be the output of the following Python code?

```
print(4 + '3')
```

NameError
b) IndexError
c) ValueError
d) TypeError

Answer: d

Explanation: The expression 4 + '3' tries to add an integer (4) and a string ('3'), which are incompatible types for the + operator in Python. This results in a TypeError.

What will be the output of the following Python code?

```
print(int('65.43'))
```

ImportError
b) ValueError
c) TypeError
d) NameError

Answer: b

Explanation: The string '65.43' represents a floating-point number, not an integer. The int() function cannot directly convert a string containing a decimal point to an integer, so it raises a ValueError. To convert such a string, you must first convert it to a float, then to an int.

Compare the following two Python codes shown below and state the output if the input entered in each case is -6?

CODE 1

```
import math
num=int(input("Enter a number of whose factorial you want to find"))
```

```
print(math.factorial(num))
```

CODE 2

```
num=int(input("Enter a number of whose factorial you want to find"))
print(math.factorial(num))
```

ValueError, NameError
b) AttributeError, ValueError
c) NameError, TypeError
d) TypeError, ValueError

Answer: a
Explanation: In Code 1, the math module is imported, but calling math.factorial(-6) raises a ValueError because factorials are only defined for non-negative integers. In Code 2, since math is not imported, calling math.factorial(-6) raises a NameError as Python doesn't recognize the name math.

What will be the output of the following Python code?

```
def getMonth(m):
if m<1 or m>12:
raise ValueError("Invalid")
print(m)
getMonth(6)
```

ValueError
b) Invalid
c) 6
d) ValueError("Invalid")

Answer: c
Explanation: The function getMonth checks whether the value of m is outside the range 1 to 12. Since 6 falls within this valid range, the

condition m < 1 or m > 12 evaluates to False, so the raise statement is not executed. The function then proceeds to print m, which outputs 6.

What will be the output of the following Python code if the input entered is 6?

```
valid = False
while not valid:
try:
n=int(input("Enter a number"))
while n%2==0:
    print("Bye")
valid = True
except ValueError:
print("Invalid")
```

Bye (printed once)
b) No output
c) Invalid (printed once)
d) Bye (printed infinite number of times)

Answer: d

Explanation: When n is 6 (an even number), the inner while n%2 == 0: loop will run infinitely because n never changes inside the loop. Therefore, “Bye” will be printed endlessly, causing an infinite loop.

Identify the type of error in the following Python codes?

```
Print(“Good Morning”)
print(“Good night)
```

Syntax, Syntax
b) Semantic, Syntax
c) Semantic, Semantic
d) Syntax, Semantic

Answer: a
Explanation: Print(“Good Morning”) causes a SyntaxError because Python is case-sensitive, and the correct function name is print (all lowercase).
print(“Good night) also results in a SyntaxError due to a missing closing quotation mark, making the string improperly formatted.
Both are syntax errors, not semantic errors, because they violate the rules of Python syntax.

Which of the following statements is true?

a) The standard exceptions are automatically imported into Python programs
b) All raised standard exceptions must be handled in Python
c) When there is a deviation from the rules of a programming language, a semantic error is thrown
d) If any exception is thrown in try block, else block is executed

Answer: a
Explanation: Python’s standard exceptions (like ValueError, TypeError, etc.) are built-in and available by default, so you don’t need to import them explicitly. The other options are incorrect because:
Not all raised exceptions must be handled.
Semantic errors are different from exceptions.
The else block runs only if no exception occurs in the try block.

Which of the following is not a standard exception in Python?

a) NameError
b) IOError
c) AssignmentError
d) ValueError

Answer: c
Explanation: AssignmentError is not a standard exception in Python. Common standard exceptions include NameError (when a variable is not defined), IOError (input/output operation failure), and ValueError (when a function receives an argument of the right type but inappropriate value).
Syntax errors are also known as parsing errors.
a) True
b) False

Answer: a
Explanation: Syntax errors occur when the code violates the rules of the programming language grammar. They are also called parsing errors because the parser fails to interpret the code structure correctly.
An exception is _____
a) an object
b) a special function
c) a standard module
d) a module

Answer: a
Explanation: In Python, exceptions are objects that are instances of classes derived from the BaseException class. When an error occurs, an exception object is created and can be handled using try-except blocks.
_____ exceptions are raised as a result of an error in opening a particular file.
a) ValueError
b) TypeError
c) ImportError

d) IOError

Answer: d
Explanation: IOError exceptions occur when there is a failure in input/output operations, such as errors in opening, reading, or writing a file. If a file cannot be opened (e.g., because it does not exist or permission is denied), an IOError is raised.

Which of the following blocks will be executed whether an exception is thrown or not?

a) except
b) else
c) finally
d) assert

Answer: c
Explanation: The finally block in Python is always executed, regardless of whether an exception was raised or handled. It is typically used for cleanup actions that must happen no matter what.