SC5

1. Definitions and Short Answers

1. What is the data type of (1, 2, 3)?

> 💡 tuple

2. If s = 'ABCDE', what is the value of
   - s[0]
   - s[1]
   - s[-1]
   - s[1:4]
   - s[-5:-2]
   - s[:2]
   - s[-3:]
   - s[:]
   - s[0:0]
   - s[1:4:2]
   - s[-1:0:-1]

> 💡 ‘A’ ; ‘B’ ; ‘E’ ; ‘BCD’ ; ‘ABC’ ; ‘AB’ ; ‘CDE’ ; ‘ABCDE’ ; ‘’ ; ‘BD’ ; ‘EDCB’

3. If S = ['h', 'e', 'l', 'l', 'o'], what is the value of S after executing the statement S[1:2] = ['a']?

> 💡 S = [’h’, ‘a’, 'l', 'l', 'o']

4. If T = ('h', 'e', 'l', 'l', 'o'), which of the following is allowed or not allowed and why?

    - T[3] = 'z'
    - T = ('w', 'o', 'r', 'l', 'd')
    - T = T[2:-1]

💡 No, Yes, Yes

5. What is the value of

    - list('apple')
    - tuple('apple')
    - set('apple')

💡 ['a', 'p', 'p', 'l', 'e']
('a', 'p', 'p', 'l', 'e')
{'l', 'e', 'a', 'p'}

6. What is the value of

    - str(['a', 'p', 'p', 'l', 'e']) # ['a', 'p', 'p', 'l', 'e']
    - str(('a', 'p', 'p', 'l', 'e')) # ('a', 'p', 'p', 'l', 'e')
    - str({'a', 'p', 'p', 'l', 'e'}) # {'e', 'p', 'l', 'a'}

💡 `str(['a', 'p', 'p', 'l', 'e'])` will return the string representation of the list object `['a', 'p', 'p', 'l', 'e']`, which is `"['a', 'p', 'p', 'l', 'e']"`.

`str(('a', 'p', 'p', 'l', 'e'))` will return the string representation of the tuple object `('a', 'p', 'p', 'l', 'e')`, which is `"('a', 'p', 'p', 'l', 'e')"`.

`str({'a', 'p', 'p', 'l', 'e'})` will return the string representation of the set object `{'a', 'p', 'p', 'l', 'e'}`, which can have different orderings on different executions, for example `{'p', 'l', 'e', 'a'}`.

7. What is the value of
    - list(('a', 'p', 'p', 'l', 'e'))
    - tuple(['a', 'p', 'p', 'l', 'e'])
    - set(['a', 'p', 'p', 'l', 'e'])

💡 `list(('a', 'p', 'p', 'l', 'e'))` will return a list object `['a', 'p', 'p', 'l', 'e']`.

`tuple(['a', 'p', 'p', 'l', 'e'])` will return a tuple object `('a', 'p', 'p', 'l', 'e')`.

`set(['a', 'p', 'p', 'l', 'e'])` will return a set object `{'a', 'p', 'l', 'e'}`.

8. What is the result of
    - 'Apple' < 'apple'
    - 'Apple' <= 'apple'
    - 'Apple' == 'apple'
    - 'Apple' >= 'apple'
    - 'Apple' > 'apple'
    - 'Apple' != 'apple'

💡 T, T, F, F, F, T

9. What is the result of
    - 'Apple' < 'adventure'
    - 'apple' < 'adventure'
    - 'apple' < 'Adventure'
    - 'apple' < 'bee'
    - 'apple' < 'Bee'

- 'Apple' < 'bee'
- 'Apple' < 'Bee'

💡 T, F, F, T, F, T, T

10. What is the result of
    - ('apple', 0) < ('apple', 2)
    - ('apple', 0, 3) < ('apple', 1)
    - ['apple', 2, 2] < ['apple', 2, 1, 5]
    - ['apple', 3] < ['oranges', 0]

💡 T, T, F, T

11. What is the result of
    - 's' in 'school'
    - 'hoo' in 'school'
    - 'S' in 'school'
    - 'ol' in 'school'
    - 'k' not in 'school'
    - 's' not in 'School'

💡 T, T, F, T, T, T

12. What is the result of
    - 's' in ['s', 'c', 'h', 'o', 'o', 'l']
    - ['s'] in ['s', 'c', 'h', 'o', 'o', 'l']

- ['s'] in [['s'], ['c'], ['h'], ['o'], ['o'], ['l']]
- 'hoo' in ['s', 'c', 'h', 'o', 'o', 'l']
- ['h', 'o', 'o'] in ['s', 'c', 'h', 'o', 'o', 'l']
- ('h', 'o', 'o') in ['s', 'c', ('h', 'o', 'o'), 'l']
- ('h', 'o', 'o') not in ('s', 'c', ('h', 'o', 'o'), 'l')
- 'ol' in ['s', 'c', 'h', 'o', 'ol']
- 's' in ['S', 'c', 'h', 'o', 'o', 'l']

> 💡 T, F, T, F, F, T, F, T, F

13. What is the result of

- 'sch' + 'ool'
- [1, 2, 3] + [4, 5, 6]
- (1, 2, 3) + (4, 5, 6)

> 💡 'school'
> [1, 2, 3, 4, 5, 6]
> (1, 2, 3, 4, 5, 6)

14. What is the result of

- 'sch' + 'o' * 10 + 'l'
- 'do' * 5
- ['s'] + ['o'] * 5 + ['l']

> 💡 'schoooooooooool'
> 'dododododo'
> ['s', 'o', 'o', 'o', 'o', 'o', 'l']

15. How do you express a tuple literal of a single element? For example, how do you write a tuple literal that has the same value as tuple([1])?

> 💡 t = (1,)

16. Suppose you have x = 1, 2, 3What is the value of type(x)?

> 💡 <class 'tuple'>

17. Suppose you have L = ['f', 'r', 'o', 'g']What is the new value of L after executing each of the following statements in order?
    - L.append('s')
    - L.extend(['p', 'o', 'n', 'd'])
    - L.insert(4, ' ')
    - L.reverse()
    - L.sort()
    - L.remove('o')
    - L.pop()
    - L.pop(0)
    - L.clear()
    - L.append('z')

💡 ['f', 'r', 'o', 'g', 's']
['f', 'r', 'o', 'g', 's', 'p', 'o', 'n', 'd']
['f', 'r', 'o', 'g', ' ', 's', 'p', 'o', 'n', 'd']
['d', 'n', 'o', 'p', 's', ' ', 'g', 'o', 'r', 'f']
[' ', 'd', 'f', 'g', 'n', 'o', 'o', 'p', 'r', 's']
[' ', 'd', 'f', 'g', 'n', 'o', 'p', 'r', 's']
's'
' '
[]
['z']

18. If T = (1, 3, 5, 7, 9, 11), Can you call del(T[1])? why or why not? Can you call del(T)? What is the effect?

💡 TypeError: 'tuple' object doesn't support item deletion
We can call del(T)

19. Suppose L = list('hello') and separately M = list('hello'). After executingL.reverse() M = M[::-1]
    - is L == M evaluate to True or False?
    - What is the difference between these two ways of reversing elements in a list?

💡 True
The difference between `L.reverse()` and `M = M[::-1]` is that `L.reverse()` reverses the elements of the list in place, meaning that it modifies the original list object `L`. On the other hand, `M[::-1]` creates a new list object `M` that contains the same elements as the original list `M`, but in reverse order. The original list `M` remains unchanged.

20. if T = tuple('hello'), are the following statements allowed in Python? Why or why not?

- T.reverse()
- T = T[::-1]

💡 NO. Tuples are immutable, meaning that their elements cannot be modified. Therefore, the reverse() method, which modifies the order of the elements in place, cannot be used with tuples.
YES. This creates a new tuple that contains the elements of T in reverse order. Because tuples are immutable, the original tuple T cannot be modified. However, this statement is allowed and creates a new tuple that can be assigned to a new variable or to T itself.

21. What is a **stack** as a data structure? What is another name (4-letter initialism) for a stack? How can a stack be implemented using a list? Show how **push** and **pop** can be accomplished by calling list methods.

💡 FIFO
append + pop

22. What is a **queue** as a data structure? What is another name (4-letter initialism) for a queue? How can a queue be implemented using a list? Show how enqueue and dequeue can be accomplished by calling list methods.

💡 FILO
append + pop(0)

23. Show how a **tuple** can be used to implement
    - a stack's push and pop functionality
    - a queue's enqueue and dequeue functionality
    - Is a tuple more or less efficient than a list for implementing the stack and queue data structures? Why?

💡

```
stack = ()
stack += (1,)  # push 1
stack += (2,)  # push 2
stack += (3,)  # push 3
top = stack[-1]  # peek top
stack = stack[:-1]  # pop top

queue = ()
queue += (1,)  # enqueue 1
queue += (2,)  # enqueue 2
queue += (3,)  # enqueue 3
front = queue[0]  # peek front
queue = queue[1:]  # dequeue front
```

24. What do these built-in functions do?

- max(['h', 'e', 'l', 'l', 'o'])
- min('hello')
- sum([2, 3, 4, 5, 6])
- sum(range(10))
- any(['', 'apples', 'oranges', 'banana'])
- any([0, '', 0.0, [], ()])
- any(['0', '', 0.0, [], ()])
- any([0, ' ', 0.0, [], ()])
- all(['', 'apples', 'oranges', 'banana'])
- all([' ', 'apples', 'oranges', 'bananas'])
- all([0, '', 0.0, [], ()])

💡 o
e
20
45
True
False
True
True
False
True
False

25. ★★★What is the **non-mutation** version of the following statements? Assume L is a list

- L.sort()
- L.reverse()
- L.extend([1, 2, 3])
- del(L[1])
- L.pop()

💡 sorted(L)
list(reversed(L))
L = L + [1,2,3]
L = L[:1] + L[2:]
L = L[:-1]

26. How do you use **list comprehension** to create a list with values

- ['*', '**', '***', '****', '*****']
- [1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096]

💡 ['*'*i for i in range(1, 6)]
[2 ** i for i in range(13)]

27. How do you use **two-level list comprehension** to create a multiplication table in the following format: [(1, 1, 1), (1, 2, 2), (1, 3, 3), … (1, 9, 9), (2, 1, 2), (2, 2, 4), (2, 3, 6), (2, 4, 8), … (2, 9, 18), (3, 1, 3), (3, 2, 6), (3, 3, 9), … (3, 9, 27), (4, 1, 4) … (4, 9, 36), (5, 1, 5), … (9, 9, 81)]

💡 [(i, j, i*j) for i in range(1, 10) for j in range(1, 10)]

28. How do you use list comprehension with filter to generate the list of upper-case letters except 'A', 'E', 'I', 'O', 'U'?

```
[chr(i) for i in range(65, 65+26) \
  if chr(i) not in ['A', 'E', 'I', 'O', 'U']]
```

29. After executing the following sequence of statements:
x = 3
y = x
x = 4
what is the value of y?

💡 3

30. After executing the following sequence of statements
x = [1, 2, 3]
y = x
x = [4, 5, 6]
what is the value of y?

💡 [1, 2, 3]

31. After executing the following sequence of statements
    x = [1, 2, 3]
    y = x
    x[1] = 4
    what is the value of y?

💡 The value of `y` will be `[1, 4, 3]`.

This is because `x` and `y` both refer to the same list object in memory. When we execute the statement `y = x`, we are not creating a new list. Rather, we are simply creating a new reference to the same list object that `x` refers to. Therefore, when we modify the list object by changing the value of its second element to `4`, this change is reflected in both `x` and `y`.

32. After executing the following sequence of statements
    x = [1, 2, 3]
    y = **x[:]**
    x[1] = 4
    what is the value of y?

💡 [1, 2, 3]

33. After executing the following sequence of statements
    x = [1, 2, 3]
    y = x
    y[:] = [4, 5, 6]
    what is the value of x?

💡 [4, 5, 6]

34. After executing the following sequence of statements
    z = ['a', 'b']
    x = [1, z, 3]
    z.append('c')
    what is the value of x?

> 💡 [1, ['a', 'b', 'c'], 3]

35. After executing the following sequence of statements
    z = ['a', 'b']
    x = [1, z, 3]
    y = x
    z.append('c')
    what is the value of y?

> 💡 [1, ['a', 'b', 'c'], 3]

36. After executing the following sequence of statements
    z = ['a', 'b']
    x = [1, z, 3]
    y = x[:]
    z.append('c')
    what is the value of y?

> 💡 [1, ['a', 'b'], 3]

37. After executing the following sequence of statements
    z = ['a', 'b']
    x = [1, z, 3]
    y = **x[:]**
    x[0] = 4

z.append('c')
what is the value of y?

💡 [1, ['a', 'b', 'c'], 3]

38. After executing the following sequence of statementsimport copy
    z = ['a', 'b']
    x = [1, z, 3]
    y = **copy.copy**(x)
    x[0] = 4
    z.append('c')
    what is the value of y?

💡 [1, ['a', 'b', 'c'], 3]

39. After executing the following sequence of statementsimport copy
    z = ['a', 'b']
    x = [1, z, 3]
    y = **copy.deepcopy**(x)
    x[0] = 4
    z.append('c')
    what is the value of y?

💡 [1, ['a', 'b'], 3]

40. What is the **type** of {}?

💡 dictionary

41. What is the expression for an **empty set**?

> 💡 set()

42. Which of the following can or cannot be a **member of a set**? Why?
    - 'hello'
    - 23
    - 44.27
    - 5e-3
    - 2+4j
    - ['Mary', 'had', 'a', 'little', 'lamb']
    - ('Mary', 'had', 'a', 'little', 'lamb')
    - {'Mary', 'had', 'a', 'little', 'lamb'}
    - {'Sun': 0, 'Mon': 1, 'Tue': 2, 'Wed': 3}
    - True
    - False
    - ()
    - []
    - {}
43. What is the value of len(set('hello'))?

> 💡 4

44. What is the value of each of the following expressions?
    - {1, 2} - {2, 3}
    - {1, 2} | {2, 3}
    - {1, 2} & {2, 3}

- {1, 2} ^ {2, 3}

💡 {1}
{1, 2, 3}
{2}
{1, 3}

45. What is the result of the following comparisons?

    - {1, 2, 3} > {2, 3}
    - {1, 2, 3} < {1, 2, 4}
    - {1, 2, 2, 3} == {1, 2, 3}
    - {1, 2, 4} != {4, 2, 1}

💡 True
False
True
False

46. Assume S = {1, 2, 3}, what is the difference between
    S = S | {3, 4} and
    S |= {3, 4}?

47. Assume D = {'Sun': 0, 'Mon': 1, 'Tue': 2, 'Wed': 3}

    - What is the value of D['Mon']?
    - What is the value of D after D['Thu'] = 4?
    - Continuing with the previous statement, what is the value of D after D['Sun'] = 7?
    - What happens if you attempt print(D['Fri'])?

💡 1
{'Sun': 0, 'Mon': 1, 'Tue': 2, 'Wed': 3, ‘Thu’: 4}
{'Sun': 0, 'Mon': 1, 'Tue': 2, 'Wed': 3, ‘Thu’: 4, ‘Sun’: 7}
KeyError: 'Fri’

48. Assume D = {'Sun': 0, 'Mon': 1, 'Tue': 2, 'Wed': 3}
    - What is the value of D.keys()
    - What is the value of D.values()
    - What is the value of D.items()

💡 dict_keys(['Sun', 'Mon', 'Tue', 'Wed'])
dict_values([0, 1, 2, 3])
dict_items([('Sun', 0), ('Mon', 1), ('Tue', 2), ('Wed', 3)])

49. Assuming D = {}, which of the following is legal or not legal in Python? If not legal, why not?
    - D[()] = 10 legal
    - D[''] = {} legal
    - D[0] = '' legal
    - D[{}] = ()

    💡 Not legal. This statement will raise a `TypeError`
     because the key `{}`
     is a dictionary, and dictionaries are not hashable in Python. Only hashable objects can be used as keys in a dictionary.

    - D[[]] = set()

💡 Not legal. This statement will raise a `TypeError` because the key `[]` is a list, and lists are not hashable in Python. Only hashable objects can be used as keys in a dictionary.

- D[:] = range(10) Not legal
- D[-1] = [-1] legal
- D[(())] = [{}] legal

50. How do you use dictionary comprehension to create a reverse mapping? For example, suppose D = {'Sun': 0, 'Mon': 1, 'Tue': 2, 'Wed': 3, 'Thu': 4, 'Fri': 5, 'Sat': 6}, create its reverse mapping whose value should be {0: 'Sun', 1: 'Mon', 2: 'Tue', 3: 'Wed', 4: 'Thu', 5: 'Fri', 6: 'Sat'}?

💡 D = {'Sun': 0, 'Mon': 1, 'Tue': 2, 'Wed': 3, 'Thu': 4, 'Fri': 5, 'Sat': 6}

reverse_D = {v: k for k, v in D.items()}

print(reverse_D)