1. Which line swaps the contents of variables a and b?

(a) \(a = b = b = a\)

(b) \(a, b = b, a\)

(c) \(a, b = a, b\)

(d) \(a = b = (a, b)\)

2. What is the output of the following program?

\[
s = "hi!"
print(s * len(s))
\]

(a) 9

(b) !!!

(c) hi!hi!hi!

(d) There is no output; the code produces an error

3. The following program produces an error when run:

```python
class Foo:
    def bar(s, x, y):
        s.w = x + y
        return s.w

f = Foo()
f.bar(f, 5, 10)
```

Which of the following would fix the error?

(a) Adding an `__init__` method to `Foo` that initializes the `w` attribute

(b) Renaming the first parameter of `bar` (and other references to `s`) to `self`

(c) Removing the parameter `s` from the definition of `bar`

(d) Removing the argument `f` in the call to method `bar`
def binary_search(lst, x): # lst is sorted in ascending order
    lo = 0
    hi = len(lst)-1
    while lo <= hi:
        mid = (lo + hi) // 2
        if lst[mid] < x:
            ___________________________ #1
        elif lst[mid] > x:
            ___________________________ #2
        else:
            return True # x is found in lst
            return False # x is not found in lst

Which of the following correctly fill in blanks #1 and #2 (in that order)?

(a) mid = lo + 1   /   mid = hi - 1
(b) lo = hi - mid   /   hi = lo + mid
(c) hi = mid + 1   /   lo = mid - 1
(d) lo = mid + 1   /   hi = mid - 1

5. What is the output of the following program?

```python
def gen(n):
    for x in range(n):
        print('Yielding', x)
        yield x
        print('Yielded', x)

g = gen(10)
print(next(g))
```

(a) **Yielding 0**
    0
(b) **Yielding 0**
    0
    **Yielded 0**
(c) **Yielding 0**
    **Yielded 0**
(d) **Yielding 0**
    **Yielded 0**
    0
6. What does the following list comprehension evaluate to?

\[
[x + y \text{ for } x \text{ in } \text{range}(1, 4) \text{ for } y \text{ in } \text{range}(2, 6) \text{ if } x < y]
\]

(a) \([3, 4, 5, 5, 6, 7, 6, 7, 8]\)
(b) \([3, 4, 5, 6, 5, 6, 7, 7, 8]\)
(c) \([3, 4, 5, 6, 7, 5, 6, 7, 8, 7, 8, 9, 9, 10]\)
(d) \([3, 4, 5, 5, 6, 7, 6, 7, 8, 9, 7, 8, 9, 10]\)

7. What are the contents of \(lst\) at the end of the following program?

```python
d = {'the': ['a', 'is'], 'a': ['is', 'this'], 'is': ['the', 'a']}
lst = ['the']
while lst[-1] in d:
    for w in d[lst[-1]]:
        lst.append(w)
```

(a) ['the', 'a', 'is', 'the', 'a', 'is', 'this']
(b) ['the', 'a', 'is', 'this']
(c) ['the', 'is', 'a', 'is', 'this']
(d) ['this', 'is', 'a', 'the', 'is', 'a', 'this']

8. The following method should return true iff the provided list \(\text{lst}\) contains any duplicate elements:

```python
def has_repeats(lst):
    d = {}
    for x in lst:
        if ________________:  #1
            return True
        else:
            ________________  #2
    return False
```

Which of the following correctly fill in blanks \#1 and \#2 (in that order)?

(a) \(x \text{ in } d\)  /  \(d[x] = x\)
(b) \(x \text{ not in } d\)  /  \(\text{del } d[x]\)
(c) \(x \text{ in } d\text{.values()}\)  /  \(d[x] = \text{lst}\)
(d) \(x \text{ in } d\text{.items()}\)  /  \(d[\text{lst}] = x\)
Consider the following class definition and subsequent code:

```python
class Bar:
    def __init__(self):
        self.data = {}

    def __getitem__(self, x):
        return self.data[x]

    def __delitem__(self, x):
        self.data[x] = x

    def __setitem__(self, x, y):
        self.data[x] = y

bar = Bar()
bar['a'] = 'b'
bar['c'] = bar['a']
del bar['a']
```

What are the contents of `bar.data` at the end of the program?

(a) {'c': 'a'}
(b) {'a': 'a', 'c': 'c'}
(c) {'b': 'b', 'a': 'c'}
(d) {'a': 'a', 'c': 'b'}
10. Given that iterable is an iterable object, which of the following emulates the behavior of a for loop to iterate over its contents?

(a) 

```python
it = iter(iterable)
while True:
    try:
        x = next(it)
        # do something with x
    except StopIteration:
        break
```

(b) 

```python
it = iterable
while True:
    i = iter(it)
    x = next(i)
    # do something with x
    if not i:
        break
```

(c) 

```python
it = next(iterable)
while True:
    try:
        x = iter(it)
        # do something with x
    except StopIteration:
        break
```

(d) 

```python
it = iter(iterable)
while True:
    x = next(it)
    # do something with x
else:
    raise StopIteration
```
11. What is the worst-case run-time complexity of a method that uses binary search to determine if a given element is not in a sorted, array-backed list of N elements?
   (a) $O(1)$
   (b) $O(\log N)$
   (c) $O(N)$
   (d) $O(N^2)$

12. What is the worst-case run-time complexity of creating a new array-backed list that contains the elements of one array-back list followed by that of another array-backed list, given that there are a total of N elements?
   (a) $O(1)$
   (b) $O(\log N)$
   (c) $O(N)$
   (d) $O(N^2)$

13. What is the worst-case run-time complexity of deleting the last element (i.e., in the largest index) of an array-backed list of N elements?
   (a) $O(1)$
   (b) $O(\log N)$
   (c) $O(N)$
   (d) $O(N^2)$

14. What is the worst-case run-time complexity of finding and removing the element with the minimum value from an unsorted array-backed list of N elements?
   (a) $O(1)$
   (b) $O(\log N)$
   (c) $O(N)$
   (d) $O(N^2)$

15. What is the worst case time complexity of inserting an element into a sorted array-backed list of N elements, such that the list remains sorted after insertion?
   (a) $O(1)$
   (b) $O(\log N)$
   (c) $O(N)$
   (d) $O(N^2)$
16. What is the worst-case runtime complexity of the following function?

```python
def fA(N):
    lst = []
    for i in range(N):
        for _ in range(N):
            lst.append(i)
    return lst
```

(a) $O(1)$
(b) $O(\log N)$
(c) $O(N)$
(d) $O(N^2)$

17. What is the worst-case runtime complexity of the following function?

```python
def fB(lst):  # lst is a Python list of length N
    n = 1
    while lst[0] == lst[n]:
        n += 1
    return n
```

(a) $O(1)$
(b) $O(\log N)$
(c) $O(N)$
(d) $O(N^2)$

18. What is the worst-case runtime complexity of the following function?

```python
def fC(lst):  # lst is a Python list of length N
    n = 0
    uniques = []
    for x in lst:
        if x in uniques:
            n += 1
        else:
            uniques.append(x)
    return n
```

(a) $O(1)$
(b) $O(\log N)$
(c) $O(N)$
(d) $O(N^2)$
19. What is the worst-case runtime complexity of the following function?

```python
def fD(N):
    res = 0
    for val in range(N // 1024):
        res = res + val
    return res
```

(a) $O(1)$
(b) $O(\log N)$
(c) $O(N)$
(d) $O(N^2)$

20. What is the worst-case runtime complexity of the following function?

```python
def fE(N):
    res = 1
    while True:
        if N == 0:
            return res
        else:
            res = res * N
            N = N // 2
```

(a) $O(1)$
(b) $O(\log N)$
(c) $O(N)$
(d) $O(N^2)$

21. Which choice correctly completes the following method that reverses the contents of an array-backed list?

```python
def reverse(self):
    for i in range(len(self) // 2):
        ________________________________
```

(a) self[i], self[i+1] = self[i+1], self[i]
(b) self[i], self[len(self)-i-1] = self[len(self)-i-1], self[i]
(c) self[len(self)-1], self[i] = self[len(self)-1], self[i-1]
(d) self[i+1], self[i] = self[i], self[i+1]
22. Which choice correctly completes the following method to delete the first \( n \) elements from an array-backed list?

```python
def drop(self, n):
    for i in range(n):
        del self.data[len(self)-1]
```

(a) for i in range(0, n):
    self[i] = self[i+1]

(b) for i in range(0, len(self)):
    self[i-n] = self[i]

(c) for i in range(n, len(self)-1):
    self[i+1-n] = self[n]

(d) for i in range(n, len(self)):
    self[i-n] = self[i]

23. Which choice correctly completes the following method that returns an iterator over successive, non-overlapping pairs of elements (as tuples) from an array-backed list? (If there are an odd number of elements, the last element will be omitted.)

```python
def pairs(self):
    for i in range(0, len(self), 2):
        if i+1 < len(self):
            yield self[i], self[i+1]
```

(a) for i in range(0, len(self)):
    if i+1 < len(self):
        yield self[i], self[i+1]

(b) for i in range(0, len(self), 2):
    if i+1 < len(self):
        yield self[i], self[i+1]

(c) for i in range(1, len(self), 2):
    yield self[i+1], self[i-1]

(d) for i in range(0, len(self)-2, 2):
    yield self[i], self[i+2]
def forever(self):
    while True:
        for i in range(len(self)):
            yield self[i]

    for i in range(len(self)):
        yield self[i]
        for j in range(len(self)):
            yield self[j]

(c) while True:
    j = 0
    for i in range(j, len(self)):
        yield self[i]
    j += 1

(d) for i in range(len(self), -1, -1):
    for j in range(len(self)):
        yield self[j]
        yield self[i]

24. Which choice correctly completes the following method that returns an "infinite" iterator that repeatedly cycles through the elements of an array-backed list, starting with the first?