Answersheet:
Python

1. The correct answer is **a) [1]**
   [1, 2]
   [3]
   [1, 2, 4].

   Explanation:

   1. The first call `func(1)` appends `1` to the default list `y`, resulting in `[1]`.
   2. The second call `func(2)` appends `2` to the same list `y` (which already contains `[1]`), resulting in `[1, 2]`.
   3. The third call `func(3, [])` uses a new list `y`, resulting in `[3]`.
   4. The fourth call `func(4)` appends `4` to the default list `y` (which contains `[1, 2]`), resulting in `[1, 2, 4]`.

2. The correct answer is **b) [1, 10, 3, 4]**.

   Explanation:

   - In Python, assigning `y = x` does not create a copy of the list `x`. Instead, `y` and `x` both refer to the same list object in memory.
   - When `x[1] = 10` is executed, it modifies the list that both `x` and `y` refer to.
   - Therefore, `print(y)` will output `[1, 10, 3, 4]`.

3. The correct answer is **b) 10**.

   Explanation:

   - The class `B` inherits from class `A`.
   - When an instance of `B` is created, the `__init__` method of `B` is called.
   - Inside the `__init__` method of `B`, `super().__init__()` is called, which initializes the `value` attribute to `5` by calling the `__init__` method of `A`.
   - After that, the `value` attribute is set to `10` in the `__init__` method of `B`.
   - Therefore, `print(obj.value)` outputs `10`.

4. The correct answer is **b) 10**.

   Explanation:

   - The `outer` function defines a variable `x` with the value `5`.
   - The `inner` function is defined within `outer` and uses the `nonlocal` keyword to indicate that `x` refers to the variable `x` in the nearest enclosing scope (which is `outer`'s `x`).
   - The `inner` function sets `x` to `10`.
   - When `inner` is called within `outer`, it modifies `outer`'s `x` to `10`.
   - Therefore, `outer` returns `10`, and `print(outer())` outputs `10`.

5. The correct answer is **a) [1, 2, 3]**.

   Explanation:

   - The statement `` `y = x[:]` `` creates a shallow copy of the list `` `x` ``.
   - Modifying `` `x[0]` `` to `` `10` `` does not affect the list `` `y` `` because `` `y` `` is a separate copy of the original list.
   - Therefore, `` `print(y)` `` outputs `` `[1, 2, 3]` ``.