Итераторы Python: ленивое чтение и исчерпание

1. В Python it=iter([7,8]); next(it). Что вернёт следующий next(it)?

1. [7,8]
2. 8
3. 7
4. None

Правильный ответ: 8

2. Что возникает при next(it) после исчерпания обычного итератора без default?

1. KeyError
2. SyntaxError
3. Возвращается 0 всегда
4. StopIteration

Правильный ответ: StopIteration

3. Каков результат next(iter([]),'конец')?

1. StopIteration
2. 'конец'
3. 0
4. []

Правильный ответ: 'конец'

4. Что получится из list(zip([1,2,3],['a','b'])) без strict?

1. [(1,'a'),(2,'b'),(3,None)]
2. [(1,'a'),(2,'b')]
3. [(1,2,3),('a','b')]
4. Возникнет ошибка длины обязательно

Правильный ответ: [(1,'a'),(2,'b')]

5. Для enumerate(['x','y'],start=1) первая пара какая?

1. (1,'y')
2. (0,'x')
3. (1,'x')
4. ('x',1)

Правильный ответ: (1,'x')

6. Генераторное выражение (x*x for x in range(3)) немедленно создаёт список всех квадратов?

1. Да, всегда
2. Да, но только отрицательные
3. Нет, оно не может выдавать числа
4. Нет, значения выдаются по запросу

Правильный ответ: Нет, значения выдаются по запросу

7. Что даст list(itertools.chain([1,2],[3]))?

1. [1,2,3]
2. [1,3,2]
3. [3,2,1]
4. [[1,2],[3]]

Правильный ответ: [1,2,3]

8. Что даст list(itertools.islice(range(10),2,5))?

1. [2,3,4]
2. [2,3,4,5]
3. [0,1,2,3,4]
4. [5,6,7]

Правильный ответ: [2,3,4]

9. Один генератор уже полностью прочитан list(g). Что даст повторный list(g)?

1. []
2. Ту же последовательность заново
3. Обратную последовательность
4. Всегда TypeError

Правильный ответ: []

10. Почему ленивый проход полезен для большого файла, если строки обрабатываются независимо?

1. Не нужно держать все строки одновременно в памяти
2. Он гарантирует отсутствие любых ошибок
3. Он всегда шифрует файл
4. Он автоматически исправляет кодировку

Правильный ответ: Не нужно держать все строки одновременно в памяти