# 17차 연습문항

## **파이썬 기본문항: 고급함수**

|  |
| --- |
| 수강생 공지 사항   * 제출은 프로그래머스를 통해 해주시기 바랍니다([링크](https://campus.programmers.co.kr/app/courses/24550/curriculum)). * 파일명은 아래와 같은 형식으로 제출해주세요   + 교육생번호\_이름\_교과목\_문항\_N차시.pdf   ex) DR-11111\_홍길동\_파이썬\_연습문항\_1차시.pdf  ex) DR-11111\_홍길동\_파이썬\_추가문항\_1차시.pdf   * 답은 "write your answer"에 적어주세요. 다만 코딩 문제의 경우 output까지 답에 포함시켜 주세요.   ex)     * 답은 캡쳐를 하셔도 되고 텍스트로 넣으셔도 됩니다. * 마감 기한은 문제가 나간 주 **일요일 23:59까지**입니다. |

### 이터레이터(Iterator)에 대해 맞게 설명한 것을 고르시오

a. 이터레이터는 순회 가능한(iterable) 객체에서 값을 하나씩 가져오는 객체입니다. \_\_iter\_\_() 메서드와 \_\_next\_\_() 메서드를 구현하여 만들 수 있습니다.

b. 이터레이터는 for 루프에서 사용할 수 없으며, 오직 while 루프에서만 사용할 수 있습니다.

c. 이터레이터는 한 번 생성된 후에 요소를 변경할 수 있습니다.

d. 이터레이터는 요소를 순회할 때마다 다시 초기화되어야 하며, 초기화되지 않으면 동일한 요소를 반복해서 가져옵니다.

**답**

|  |
| --- |
| A |

### 리스트 numbers를 이터레이터로 만들고 이터레이터 함수 next를 통해 리스트 요소의 값을 출력하는 프로그램을 작성하시오.

numbers = [1, 2, 3, 4, 5]

**답**

|  |
| --- |
| # 리스트 정의  numbers = [1, 2, 3, 4, 5]  # 리스트의 이터레이터 생성  iterator = iter(numbers)    # 이터레이터를 사용하여 항목을 하나씩 출력  print(next(iterator)) # 출력: 1  print(next(iterator)) # 출력: 2  print(next(iterator)) # 출력: 3  print(next(iterator)) # 출력: 4  print(next(iterator)) # 출력: 5  print(next(iterator))  # 더 이상 항목이 없으면 StopIteration 예외 발생# print(next(iterator))  # 주석 해제 시 StopIteration 예외 발생 |

### 문제2의 해답에서 여러개의 next() 호출을 try/except문으로 감싸고 반복이 끝났을 때 StopIteration 예외를 감지하여 프로그램을 종료되도록 수정하시오.

**답**

|  |
| --- |
| # 리스트 정의  numbers = [1, 2, 3, 4, 5]  # 이터레이터 생성  iter\_numbers = iter(numbers)    # 이터레이터를 통해 값을 출력하는 반복문  try:      while True:          value = next(iter\_numbers)          print(value)  except StopIteration:      pass  numbers = [1, 2, 3, 4, 5] |

### 다음 중 이터레이터의 강점에 대한 설명 중 틀린 답을 고르시오.

a. 메모리 효율성: 이터레이터는 한 번에 모든 요소를 메모리에 저장하지 않고 필요할 때마다 값을 생성하여 반환하므로 대용량 데이터를 처리할 때 메모리 사용량을 줄일 수 있다.

b. 지연 평가 (Lazy Evaluation): 이터레이터는 다음 값을 필요로 할 때까지 계산을 지연시킬 수 있어서 데이터 스트림을 효율적으로 처리할 수 있다.

c. 다양한 데이터 타입 지원: 파이썬의 이터레이터는 다양한 데이터 타입과 사용자 정의 객체에 대해 동일한 방식으로 작동하여 코드의 일반성과 재사용성을 높일 수 있다.

d. 인덱스 접근: 이터레이터는 현재 위치를 기억하지만 인덱스로 직접 접근할 수 없으므로 인덱스 기반 접근을 지원하지 않는다.

**답**

|  |
| --- |
| d |

### 사용자 정의 클래스를 이용하여 이터레이터를 구현하고, 주어진 리스트의 값을 반복적으로 출력하는 프로그램을 작성하라.

numbers = [1, 2, 3, 4, 5]

- MyIterator 클래스는 이터레이터 프로토콜에 따라 \_\_iter\_\_()와 \_\_next\_\_() 메서드를 구현합니다.

- \_\_init\_\_() 메서드에서는 초기화를 수행하고, 입력으로 받은 데이터(data)와 현재 인덱스(index)를 초기화합니다.

- \_\_iter\_\_() 메서드는 이터레이터 객체 자체를 반환합니다. 이는 이터레이터 프로토콜을 따르기 위해 필요한 메서드입니다.

- \_\_next\_\_() 메서드는 다음 값을 반환합니다. 현재 인덱스(index)를 사용하여 데이터에서 값을 가져온 후, 인덱스를 증가시킵니다. 데이터를 모두 순회한 경우 StopIteration 예외를 발생시켜 반복을 멈춥니다.

- MyIterator 객체를 생성하고, for 반복문을 사용하여 이터레이터를 순회하면서 각 값을 출력합니다.

**답**

|  |
| --- |
| class MyIterator:      def \_\_init\_\_(self, data):          self.data = data          self.index = 0        def \_\_iter\_\_(self):          return self        def \_\_next\_\_(self):          if self.index >= len(self.data):              raise StopIteration          value = self.data[self.index]          self.index += 1          return value    # 사용 예제  numbers = [1, 2, 3, 4, 5]  iterator = MyIterator(numbers)    for num in iterator:        print(num) |

### 문제5번에서 만들어진 이터레이터는 리스트의 요소들에 대해 순차적인 접근만 허용한다. 인덱스로 접근할 수 있는 방법을 찾아 프로그램을 수정하라.

- 클래스에서 \_\_getitem\_\_ 메소드를 구현하면 인덱스로 접근 가능.

**답**

|  |
| --- |
| class MyIterator:        def \_\_init\_\_(self, data):              self.data = data              self.index = 0        def \_\_iter\_\_(self):              return self        def \_\_next\_\_(self):              if self.index >= len(self.data):                   raise StopIteration             value = self.data[self.index]             self.index += 1             return value        def \_\_getitem\_\_(self, index):                   if index < 0 or index >= len(self.data):                         raise IndexError("Index out of range")                   return self.data[index]    # 사용 예제  numbers = [1, 2, 3, 4, 5]  iterator = MyIterator(numbers)    for num in iterator:        print(num)  # 인덱스로 접근하여 값 출력  print("Index access:")  print(iterator[0])    # 출력: 1  print(iterator[3])    # 출력: 4 |

### 텍스트 파일의 각 줄을 반환하는 이터레이터 함수 file\_line\_iterator를 구현하라.

- file\_line\_iterator 함수는 파일 경로를 입력으로 받고, 이터레이터를 반환합니다.

- 반환된 이터레이터는 next() 함수를 통해 파일의 각 줄을 순서대로 반환해야 합니다. 파일의 모든 줄을 반환한 후에는 StopIteration 예외를 발생시켜야 합니다.

- 파일을 열고 닫는 관리는 적절히 처리되어야 합니다.

**답**

|  |
| --- |
| def file\_line\_iterator(file\_path):      try:          with open(file\_path, 'r') as file:              for line in file:                  yield line.strip()      except FileNotFoundError:          print(f"Error: The file '{file\_path}' was not found.")          raise StopIteration      except Exception as e:          print(f"Error while reading file '{file\_path}': {str(e)}")          raise StopIteration  # 예제 파일을 생성하고 테스트해 보겠습니다.  with open('example.txt', 'w') as f:      f.write("First line\n")      f.write("Second line\n")      f.write("Third line\n")  # 파일 경로를 입력하여 이터레이터를 생성하고 사용합니다.  iterator = file\_line\_iterator('example.txt')  for line in iterator:      print(line) |

### 아래 조건에 맞추어 피보나치 수열의 숫자를 생성하는 이터레이터를 구현하라.

* 피보나치 수열은 다음과 같은 규칙에 따라 정의되는 수열입니다:
  + 첫 번째 숫자는 0입니다.
  + 두 번째 숫자는 1입니다.
  + 세 번째 숫자부터는 바로 직전의 두 숫자의 합으로 정의됩니다.
* FibonacciIterator 클래스는 초기화 메서드 \_\_init\_\_에서 피보나치 수열의 처음 두 숫자를 self.a와 self.b에 초기화합니다.
* \_\_next\_\_ 메서드는 현재 피보나치 수(self.a)를 반환하고, 다음 피보나치 수를 계산하여 self.a와 self.b를 업데이트합니다.
* **수열이 100을 넘어가면 멈춥니다.**

**답**

|  |
| --- |
| class FibonacciIterator:      def \_\_init\_\_(self):          self.a, self.b = 0, 1      def \_\_iter\_\_(self):          return self      def \_\_next\_\_(self):          a, self.b = self.a, self.b          if a > 100: # 종료 조건                  raise StopIteration          self.a, self.b = self.b, a + self.b          return a  # 사용 예시  fib\_iter = FibonacciIterator()  for number in fib\_iter:      print(number) |

### 제네레이터의 개념과 동작 방식에 대한 설명 중 틀린 것은?

a. 제네레이터는 함수 내에서 yield 키워드를 사용하여 값을 반환할 수 있습니다.

b. 제네레이터는 호출될 때마다 이전 상태를 기억하고, 다음 값을 생성할 수 있습니다.

c. 제네레이터는 이터레이터를 생성해주는 함수입니다.

d. 제네레이터는 오직 한 번만 실행될 수 있습니다.

**답**

|  |
| --- |
| D  제네레이터는 호출될 때마다 코드 실행을 시작하고, yield를 만나면 일시 중지됩니다. 이후 다시 호출되면 이전 상태부터 진행되며, 이 과정을 반복합니다. |

### 문제8에서 이터레이터를 이용하여 피보나치 수열을 생성했었다. 여기서는 제네레이터를 이용해서 피보나치 수열을 구현하라.

**답**

|  |
| --- |
| def fibonacci():      a, b = 0, 1      while True:          yield a          a, b = b, a + b  # fibonacci 제네레이터를 이용하여 피보나치 수열의 처음 10개 값을 출력합니다.  fib\_gen = fibonacci()  for \_ in range(10):      print(next(fib\_gen), end = ', ') # 0, 1, 1, 2, 3, 5, 8, 13, 21, 34 출력 |

### 제네레이터를 사용하여 데이터를 필터링하여 출력하는 프로그램을 작성하라.

- ‘filter\_even’ 제네레이터는 리스트 nums에서 짝수만 필터링하여 yield를 통해 반환합니다.

- nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

**답**

|  |
| --- |
| def filter\_even(numbers):      for num in numbers:          if num % 2 == 0:              yield num  # filter\_even 제네레이터를 이용하여 짝수만 출력합니다.  nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  even\_gen = filter\_even(nums)  for even\_num in even\_gen:      print(even\_num) # 2, 4, 6, 8, 10 출력 |