# **17차 연습문**항

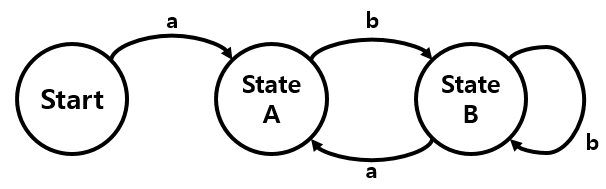
## **파이썬 추가문항: 고급함수**

| 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. |
| --- |

### **1.유한 상태 기계를 시뮬레이션하는 제네레이터를 작성하세요. 이 유한 상태 기계는 문자열을 입력으로 받아 특정 패턴을 인식합니다. 예를 들어, 상태 기계는 입력 문자열에서 "ab" 패턴을 인식해야 한다. Example Code의 “???”를 알맞게 채워서 코드를 완성하라.**

<FSM 정의>

* 상태: START, STATE\_A, STATE\_B, ACCEPT
* 전환:
  + START에서 'a'를 읽으면 STATE\_A로 전환
  + STATE\_A에서 'b'를 읽으면 STATE\_B로 전환
  + STATE\_B에서 'a'를 읽으면 STATE\_A로 전환
  + STATE\_B에서 'b'를 읽으면 STATE\_B로 유지
  + 그 외의 경우에는 START로 전환



<설명>

* 제네레이터 초기화: fsm\_generator 함수는 제네레이터로, 처음 호출 시 초기 상태를 'START'로 설정하고 대기합니다.
* 상태 전환:
  + yield state를 통해 현재 상태를 호출자에게 반환합니다.
  + 호출자는 fsm.send(char)를 사용하여 다음 입력 문자를 제네레이터로 보냅니다.
  + 제네레이터는 입력 문자를 받아 현재 상태와 입력 문자에 따라 상태를 전환합니다.
* 패턴 인식: 입력 문자열을 순차적으로 제네레이터에 보내고 각 상태 전환을 출력하여 "ab" 패턴이 인식되는 과정을 추적합니다.

| Example Code |
| --- |
| #no.1  def fsm\_generator():  state = 'START'  while True:  input\_char = yield state  if state == 'START':  if input\_char == 'a':  state = 'STATE\_A'  elif state == 'STATE\_A':  if input\_char == 'b':  state = 'STATE\_B'  else:  state = 'START'  elif state == 'STATE\_B':  if input\_char == 'a':  state = 'STATE\_A'  elif input\_char == 'b':  state = 'STATE\_B'  else:  state = 'START'  # 사용 예시: 문자열을 입력하여 상태 변화를 추적  fsm = fsm\_generator()  next(fsm) # 제네레이터 초기화  input\_string = "ababab"  for char in input\_string:  state = fsm.send(char)  print(f"Input: {char}, State: {state}") |

|  |
| --- |

### **2.제네레이터를 활용해서 텍스트 파일 검색 및 필터링 시스템을 구현하라.**

<함수 정의>

- 외부 함수: process\_line 함수는 각 줄을 받아 추가 처리를 수행합니다.

- 제네레이터:

. read\_lines 제네레이터는 파일을 줄 단위로 읽어옵니다.

. filter\_lines 제네레이터는 특정 키워드를 포함하는 줄을 필터링합니다.

- 제네레이터 사용:

. read\_lines 제네레이터를 초기화하여 파일에서 줄을 읽어옵니다.

. filter\_lines 제네레이터를 사용하여 키워드를 포함하는 줄만 필터링합니다.

. 필터링된 각 줄을 process\_line 함수로 전달하여 처리합니다.

| Example Code |
| --- |
| #no.2  def process\_line(line):  print(line.strip())  def read\_lines(filename):  with open(filename,'r') as file:  while line := file.readline():  yield line  def filter\_line(lines,keyword):  for line in lines:  if keyword in line:  yield line  filename = 'show.txt'  lines = read\_lines(filename)  keyword='Second'  filtered\_lines=filter\_line(lines,keyword)  for line in filtered\_lines:  process\_line(line) |

|  |
| --- |

### **3.다음 중 코루틴에 대한 설명으로 맞는 것을 고르시오.**

A. 코루틴은 함수 내부에서 yield 키워드를 사용하여 정의되며, async def 키워드는 사용할 수 없다.

B. 코루틴은 await 키워드를 사용하여 실행을 중단하고 다른 비동기 작업의 완료를 기다릴 수 있다.

C. 코루틴은 호출될 때마다 새로운 인스턴스를 생성하지 않고 항상 동일한 인스턴스를 사용한다.

D. 코루틴은 실행 중에 중단될 수 없으며, 연속적으로 실행된다.

| B번입니다. |
| --- |

### **4.다음 중 코루틴을 사용하는 적절한 상황에 대한 설명으로 옳은 것을 고르시오.**

A. 코루틴은 데이터베이스 쿼리를 수행하는 동안 사용하여 쿼리 결과를 즉시 반환하도록 한다.

B. 코루틴은 CPU 집약적인 작업을 수행할 때 사용하여 작업 속도를 높인다.

C. 코루틴은 I/O-bound 작업을 처리할 때 사용하여 비동기적으로 다른 작업을 수행할 수 있게 한다.

D. 코루틴은 프로그램의 시작과 종료 시점을 제어하기 위해 사용된다.

| C번입니다. |
| --- |

### **5.일반함수에서 코루틴을 호출하는 간단한 프로그램을 작성해 보시오.**

- simple\_coroutine() 함수는 비동기 코루틴으로 정의됩니다. 임의의 시작메세지를 출력하고 1초간 대기 후(asyncio.sleep 함수 사용) 임의의 종료메세지를 출력합니다.

- call\_coroutine() 함수는 일반 함수로 정의되며, 내부에서 다음을 수행합니다:

. simple\_coroutine() 함수를 호출하여 코루틴 객체를 생성합니다.

. asyncio.run(coroutine)을 사용하여 coroutine 코루틴을 실행합니다. asyncio.run() 은 이벤트 루프를 생성하고 코루틴을 실행하는 역할을 합니다.

- call\_coroutine() 함수를 호출하여 결과를 확인합니다.

| Example Code |
| --- |
| #no.5  import asyncio  async def simple\_coroutine(): # aync def 는 코루틴 함수를 정의  # Write your code here  print('coroutine 시작')  await asyncio.sleep(1)  print('coroutine 종료')  def call\_coroutine():  # Write your code here  coroutine = simple\_coroutine()  asyncio.run(coroutine)  call\_coroutine() |

|  |
| --- |

### **6.여러 학생의 성적을 비동기적으로 처리하여 평균 성적을 계산하는 프로그램을 작성하세요. 각 학생은 이름과 점수를 가지고 있습니다. 성적 데이터는 리스트로 주어집니다.**

**<요구사항>**

- calculate\_average(scores) 함수를 정의합니다. 이 함수는 비동기 코루틴으로 구현 되어야 합니다.

- 각 학생의 점수를 비동기적으로 받아서 평균 점수를 계산합니다.

- 모든 학생의 평균 점수를 계산한 후에 출력합니다.

scores = [ {"name": "Alice", "score": 85},

{"name": "Bob", "score": 70},

{"name": "Charlie", "score": 90}]

| Example Code |
| --- |
| #no.6  import asyncio  # 각 학생의 점수 데이터  scores = [  {"name": "Alice", "score": 85},  {"name": "Bob", "score": 70},  {"name": "Charlie", "score": 90}  ]  # 비동기 코루틴으로 평균 성적을 계산하는 함수  async def calculate\_average(scores):  # Write your code here  total = 0  for score in scores:  total += score['score']  average = total/len(scores)  return average  async def main():  average = await calculate\_average(scores) # await는 구버전의 yield from  print(f"전체 학생들의 평균 점수는 {average:.2f}입니다.")  # asyncio.run을 사용하여 메인 함수 실행  asyncio.run(main()) |

|  |
| --- |