중간고사 대체 과제

-TOY LANGUAGE-

컴퓨터 소프트웨어과
1733022 정현수

1. 자료구조의 정의

사람들이 사물을 편리하고 효율적으로 사용하기 위해 정리하여 관리하는 것과 같이 컴퓨터도 자료들을 정리하고 조직화하여 복잡한 자료들을 효율적으로 처리하는 여러 가지 구조가 존재한다. 이러한 구조들을 묶어 분류한 것을 자료구조라고 한다. 자료구조는 숫자와 문자, 문자열을 취급하는 단순 자료구조와 여러 자료들을 한꺼번에 보관하는 복합 자료구조로 나누어진다. 해당 프로그램에서는 단순 자료구조인 문자열과 복합 자료구조의 스택과 리스트를 자료구조로 사용했다.

2. 문자열과 리스트, 그리고 스택

문자열은 기본적으로 제공되는 단순 자료구조이다. 기본적인 단순 자료형에는 정수형(int, long), 실수형float, double), 문자형 등이 있다. 여기서는 실수나 음수를 판별하기 위해 문자열 내장함수를 사용했다.

리스트는 C언어의 배열이 발전된 형태로 여러 개의 데이터를 하나로 묶어서 저장할 수 있는 자료구조이다. 파이썬의 리스트는 동적 배열로 구현되어 크기에 대한 부담이 없는 것이 특징이다. 또한 리스트만의 내장 함수가 제공되어 기존의 배열보다 편리하게 활용할 수 있다는 장점이 있다.

스택은 가장 늦게 입력된 자료가 가장 늦게 출력되는 후입선출의 형태로 일어나는 자료 구조를 의미한다. 따라서 항목 접근을 위한 통로는 리스트의 후단만 열어두는데 이를 스택의 상단이라고 부른다. 이 때, 상단에서만 삽입과 삭제 연산이 이루어지기 때문에 O(1)으로 매우 효율적인 시간복잡도를 가지게 된다.

이미 정의가 되어 있는 자료구조인 리스트와 문자열과는 달리 스택은 추상자료형으로 배열과 연결리스트를 통해 구현을 해주어야 한다. 일반적으로 배열로 스택을 구현하는 것은 배열의 사이즈에 따라 스택의 크기가 정해지기 때문에 삽입 연산이 비효율적일 수 있다. 그렇기 때문에 연결리스트로 구현하여 스택의 크기가 제한되는 것을 해결하기도 한다. 그러나 파이썬에서는 배열의 역할을 하는 리스트가 동적 배열이기 때문에 리스트만으로 충분히 스택을 구현할 수 있다. 따라서 해당 프로그램에서는 리스트를 이용해서 스택을 구현해주었다.

3. 리스트 및 문자열 내장 함수

- 문자열 내장 함수

■ isdigit()
  - 문자열이 숫자인지를 판별해주는 함수이다.
  - 음수과 실수는 판별해주지 못하므로 따로 구현해주어야 한다.

■ isalpha()
  - 문자열이 알파벳으로 구성되었는지 판별해주는 함수이다.

■ replace()
  - 문자열에 있는 문자를 다른 문자로 바꾸어줄 때 사용하는 함수이다.

■ lsrtip()
  - 좌측을 기준으로 매개변수로 들어간 문자를 제거해주는 함수이다.
  - 만약 매개변수가 없다면 공백을 제거하게 된다.

- 리스트 내장 함수

■ join():
  - 리스트를 하나의 문자열 형태로 이어주는 함수이다.
  - 스택에 쌓인 요소들을 하나의 문자열로 바꿔줄 때 사용했다.
  - 매개변수로는 리스트가 들어간다.

■ append():
  - 매개변수에 들어간 함목을 리스트의 후단에 삽입해주는 함수이다.
  - 스택의 push 메소드를 해당 함수로 구현했다.

■ pop()
  - 리스트의 후단에 있는 항목을 삭제하면서 값을 반환해주는 함수이다.
  - 스택의 pop 메소드를 구현할 때 사용되었다.

■ partition()
  - 매개 변수를 기준으로 문자열을 분리해주는 함수이다.

해당 프로그램을 구현하는데 사용된 리스트 및 문자열의 내장 함수의 이름과 그에 대한 설명이다. 해당 함수들은 getTokens 함수나 checkSyntax 함수에서 빈번하게 사용될 것이다.

3-1. 스택 클래스에 사용된 메소드

```
1  class Stack:
2      def __init__(self):
3          self.top = []
4
5      def isEmpty(self):
6          return len(self.top) == 0
7
8      def push(self, item):
9          self.top.append(item)
10
11     def pop(self):
12         if not self.isEmpty():
13             return self.top.pop(-1)
14
15     def peek(self):
16         return self.top[-1]
17
18     def clear(self):
19         self.top = []
```

스택은 추상 자료형으로 객체 지향 언어에서는 클래스를 만들어서 구현하게 된다. 클래스는 객체를 찍어내는 설계도이며 속성을 나타내는 멤버 변수와 기능(동작)을 나타내는 메소드로 구성된다. 그렇다면 스택이 수행하는 기능은 무엇일까? 스택의 주요 기능은 삽입(push)과 삭제(pop) 연산일 것이다. 여기에 부수적으로 스택의 저장된 내용을 모두 삭제해주는 기능과 가장 최근에 저장된 항목을 살펴보는 기능이 있다면 프로그램 구현에 있어서 매우 큰 도움이 될 것이다. 이에 따라 해당 프로그램에서는 이러한 기능들을 스택 클래스의 메소드로 구현했다. 그렇다면 구현한 함수들에 대해서 설명해보도록 하겠다.

■ __init__ 메소드(생성자)

```
1      def __init__(self):
2          self.top = []
```

스택을 생성할 때 호출되는 _init_ 메소드이다. _init_메소드는 일종의 생성자 역할을 수행한다. 또한 객체가 생성될 때 _init_에서 작성해놓은 멤버 변수를 선언하거나 메소드를 호출한다. 여기서 _init_ 메소드의 첫 번째 인자와 내용에는 반드시 객체 자신을 가리키는 self를 써주어야 한다. 스택 클래스에서는 _init_ 메소드의 내용으로 top이라는 이름의 리스트를 멤버 변수로 선언해주었다.

■ isEmplty 메소드

```
1      def isEmpty(self):
2          return len(self.top) == 0
```

스택이 비어있는지 확인해주는 메소드이다. 마찬가지로 클래스의 멤버 메소드로 작성되었기에 첫 번째 인자는 self를 작성해주었다. 해당 메소드에서는 len(self.top) == 0을 기준으로 리스트 top의 크기가 0이면 True, 0이 아니면 False를 반환하도록 작성했다.

■ push 메소드

```
1      def push(self, item):
2          self.top.append(item)
```

스택에 항목 item을 추가해주는 메소드이다. 해당 함수에서는 리스트의 내장함수 append를 이용해서 item을 리스트의 후단(스택의 상단)에 추가해 주었다.

■ pop 메소드

```
1       def pop(self):
2           if not self.isEmpty():
3               return self.top.pop(-1)
```

스택의 상단에 있는 항목을 삭제하면서 해당 항목을 반환해주는 메소드이다. 이 때, 스택이 비어있다면 에러가 발생되기 때문에 스택이 비어있는지 확인하고 비어있지 않으면 리스트의 내장함수인 pop을 이용해서 마지막 값을 삭제시키면서 해당 값을 반환하도록 작성해주었다.

■ peek 메소드

```
1       def peek(self):
2           return self.top[-1]
```

스택의 상단에 있는 항목의 내용을 반환해주는 메소드이다. 마지막 내용을 반환시켜주면 되기에 self.top[-1]를 작성하고 이를 반환시켜 주도록 했다.

여기까지 스택 클래스에서 구현한 모든 메소드를 살펴보았다. 해당 프로그램에서는 거의 모든 함수에서 스택을 사용했다. 따라서 해당 프로그램에서의 스택은 매우 중요한 역할을 하는 자료구조라고 볼 수 있을 것이다. 그렇다면 프로그램에서 사용되는 함수는 무엇이고 각 함수에서 스택을 어떻게 활용되었을까? 이제부터 프로그램 구현을 위해 사용된 함수들을 설명해보도록 하겠다.

3-2. 프로그램 함수 설명

(1) 문자열을 토큰 단위로 리스트에 저장해주는 getTokens 함수

```
1   def getTokens(input):
2       tokens = []
3       stack = Stack()
4
5       for ch in input:
6           if ch == '(':
7               tokens.append(ch)
8           elif ch == ')' or ch == ' ':
9               if not stack.isEmpty():
10                  tokens.append(''.join(stack.top))
11                  stack.clear()
12
13              if ch == ')':
14                  tokens.append(ch)
15          else: stack.push(ch)
16
17      if not stack.isEmpty():
18          tokens.append(''.join(stack.top))
19
20      return tokens
```

사용자가 하나의 문자열을 입력받으면 컴퓨터가 이해할 수 있는 최소한의 의미를 지닌 단위로 분할해야 한다. 이 때, 분할된 요소들을 토큰이라고 부른다. getTokens 함수에서는 앞에서 설명한 것처럼 문자열을 토큰 단위로 분리하고 리스트에 저장하여 반환하는 역할을 수행한다. 해당 함수에서는 이를 위해 스택을 활용했다.

우선, tokens라는 이름의 리스트와 스택을 생성했다. 그러고 나서 for문을 통해 input 문자열을 한글자씩 순회하면서 일정한 조건에 따라 토큰들이 만들어지도록 작성했다.

이 때, for문의 수행 과정은 다음과 같다.

■ ch가 왼쪽 괄호를 가리키고 있다면 tokens에 저장한다.
■ ch가 오른쪽 괄호나 공백이라면 아래의 과정을 거친다.
  - 스택에 저장된 문자들을 연결하여 tokens에 저장한다.
  - clear 메소드를 이용하여 스택의 모든 내용을 삭제한다.
  - 만약 ch가 오른쪽 괄호라면 tokens에 ch를 저장한다.
■ 이외의 문자라면 스택에 저장한다.

이런 과정을 거치고 for문이 끝나게 되면 스택이 비어있는지 확인하는 작업을 거친다. 이 때, 빈 스택이 아니라면 스택에 저장된 문자들을 연결하여 tokens에 저장한다. 이렇게 빈 스택인지 확인하는 작업까지 끝나게 되면 tokens를 반환하는 것으로 함수의 수행이 종료 된다.

(2) 음수 판별 및 실수 판별 함수

일반적으로 숫자를 판별해주는 isdigit 함수는 자연수의 경우, True를 반환해주지만 이외의 값은 모두 False로 반환해주기 때문에 실수와 음수를 제대로 판별해주지 못한다. 따라서 음수와 실수의 여부를 판별해주기 위해 작성하게 되었다. 해당 함수들은 숫자 위치의 문법을 체크하는데 사용된다.

■ 음수를 판별해주는 isNagative 함수

```
1  def isNagative(token):
2      if token[0] == '-' and token[1:].isdigit():
3          return True
4      return False
```

숫자 위치의 토큰이 음수인지를 판별하는 함수이다. 첫글자가 '-'이면서 이후의 인덱스부터 마지막 문자까지 숫자라면 음수이므로 True를 반환, 아니라면 False를 반환하도록 작성했다.

■ 실수를 판별해주는 isFloat 함수

```
1  def isFloat(token):
2      try:
3          num1, _, num2 = token.partition('.')
4          if (isNagative(num1) or num1.isdigit()) and num2.isdigit():
5              return True
6      except:
7          pass
8      return False
```

숫자 위치의 토큰이 실수인지를 판별하는 함수이다. 우선 변수 3개를 선언하고 문자열 내장 함수 partition을 사용해여 '.'을 기준으로 분리한 문자들을 저장해준다. 만약 token의 값이 실수라면 num1과 num2에는 숫자, _에는 점이 저장될 것이다. 그렇게 변수가 저장되면 num1과 num2가 숫자인지 판별해서 맞으면 True를 반환해주고, 그렇지 않다면 False를 반환해주도록 했다. 이 때, num1은 음수가 나올 수 있으므로 isNagative 함수도 사용해주었다. 또한 해당 연산을 수행할 수 없는 경우(ex. ...., a1a2, .1a.a.21a), 예외 처리로 에러를 발생시켜 해당 수행문을 건너뛰도록 처리하였다.

(3) 문법적으로 잘못된 부분이 없는지 체크하는 checkSyntax 함수

```
1  def checkSyntax(tokens):
2      stack = Stack()
3      numOpCnt = 0
4
```

```
5       for token in tokens:
6           if token == '(':
7               stack.push(token)
8           elif token == ')':
9               if stack.isEmpty():
10                  return BRACKET_ERROR1
11              else: stack.pop()
12          else:
13              if numOpCnt % 2 == 0:
14                  try:
15                      if token.isdigit() or isNagative(token):
16                          numOpCnt += 1
17                      elif token[0] == '-' and token[1] == '-' and token.lstrip('-').isdigit():
18                          return NUM_ERROR1
19                      elif token.count('-') >= 1 and not token.lstrip('-').isdigit() :
20                          return NUM_ERROR2
21                      elif isFloat(token):
22                          raise ThatIsFloatError
23                      else:
24                          raise UseForbiddenChracters
25                  except ThatIsFloatError:
26                      return NUM_ERROR3
27                  except UseForbiddenChracters:
28                      return NUM_ERROR4
29              else:
30                  if token == "MINUS":
31                      numOpCnt += 1
32                  else:
33                      return OPERATOR_ERROR
34
35      if numOpCnt % 2 == 0: return EXPRERESSION_ERROR
36      elif stack.isEmpty(): return None
37      else: return BRACKET_ERROR2
```

SyntaxCheck 함수는 getTokens 함수를 통해 나누어진 토큰들의 문법이 올바른지 확인하는 기능을 한다. 즉, 토큰들이 저장된 리스트를 기준으로 숫자 위치와 연산자 위치의 문법이 맞는지를 판단하며, 스택을 활용하여 괄호가 제대로 표기되었는지 확인하는 동작을 수행하게 된다.

그럼 괄호에 대한 에러 처리 알고리즘을 설명하도록 하겠다. 우선, 왼쪽 괄호가 나오면 스택에 저장하고 새로운 토큰이 오른쪽 괄호라면 이전에 저장된 왼쪽 괄호를 스택에서 삭제한다. 만약, 오른쪽 괄호가 나왔는데 스택에 왼쪽 괄호가 없으면 오른쪽 괄호가 먼저 나왔다는 에러(BRACEKT_ERROR1)를 반환하고 for문이 다 돌았음에도 불구하고 스택에 괄호(왼쪽 괄호)가 남아있다면 왼쪽과 오른쪽 괄호의 개수가 다르다는 에러(BRACKET_ERROR2)를 반환하도록 한다. 여기까지가 괄호에 대한 에러 처리 알고리즘이다. 다음으로 숫자와 연산자의 에러 처리 알고리즘을 알아보도록 하겠다.

for문이 도는 과정에서 왼쪽 괄호와 오른쪽 괄호가 아닌 다른 토큰이 나온다면 분기는 2가지로 나뉘어진다. 숫자 위치와 연산자 위치를 기준으로 에러를 판별하는 방식이 다르기 때문이다. 여기서는 이를 구현하기 위해 numOpCnt를 선언하고 초기값을 0으로 설정했다. 이 때, 숫자나 연산자 위치의 토큰에 대한 에러 검사를 마치고 무사히 for문을 돌게되면 numOpCnt의 값이 1씩 증가하게 된다.

numOpCnt를 선언했다면 숫자와 연산자의 위치를 판별하는 알고리즘을 생각해야 한다. 일반적으로 괄호를 제외한 수식의 순서는 숫자->연산자->숫자의 순서로 이어지게 된다. 그렇다면 현재 위치

를 2로 나누었을 때 1이라면 숫자위치, 0이라면 연산자 위치를 가리킨다는 것을 유추해 볼 수 있을 것이다. 해당 프로그램에서는 이러한 알고리즘을 사용하기 위해 numOpCnt를 활용했다. 다만, numOpCnt는 초기값이 0부터 시작되므로 numOpCnt를 2로 나누었을 때 0이면 숫자 위치, 1이면 연산자 위치를 가리키게 될 것이다.

그렇다면 numOpCnt를 2로 나눈 값이 0일 때(숫자 위치) 어떤 과정을 거치는지 살펴 보도록 하겠다. 우선, try문이 나오게 된다. 해당 프로그램에서는 2가지의 에러 처리를 구현해야한다. 구현 조건은 아래와 같다.

■ 숫자 위치에 실수가 나오면 에러 처리
■ 알파벳과 '-'를 제외한 각종 기호가 입력되면 에러 처리

```
1  class ThatIsFloatError(Exception):
2      pass
3
4  class UseForbiddenChracters(Exception):
5      pass
```

이를 구현하기 위해 예외 처리를 위한 클래스를 작성했다. 여기서 에러를 처리하는 클래스는 파이썬에서 제공하는 Exeception의 클래스의 상속을 받는다. 이렇게 작성된 클래스는 raise문을 통해 에러를 처리할 때 사용하게 된다. 만약 raise문을 통해 에러가 발생하게 되면 except문에서 해당 에러 문구를 반환해주면 되기 때문에 클래스는 기본적인 틀만 작성했다.

본론으로 돌아가서 try문 다음에는 다중 if문이 나오게 된다. 다중 if문의 수행과정은 다음과 같다. 먼저 토큰이 자연수이거나 음수라면 정상적인 토큰이므로 numOpCnt의 값을 1 증가시킨다. 반면에 숫자 앞에 '-' 문자를 중복 입력되었거나 숫자 사이에 '-' 문자가 들어갔을 경우, 각각 그에 대한 에러(소스 코드 상단에 선언된 상수)를 반환하도록 한다. 만약 여기까지 진행했는데도 해당되는 조건이 없다면 토큰이 실수인지의 여부를 판별하게 된다. 이 때, isFloat 함수의 반환 값이 True라면 실수를 입력했다는 에러(ThaisFloatError)를, 그렇지 않다면 알파벳과 '-' 이외의 기호를 입력한 것이므로 에러(UseForbiddenChracters)를 raise문을 통해 발생시키게 된다. 그리고 except 문을 통해 raise문으로 발생 시킨 에러들에 대한 처리로 각각 해당되는 에러를 반환하게 된다.

이렇게 모든 과정을 거치게 되치게 되면 numOpCnt를 2로 나눈 마지막 값은 1을 가리키게 된다. 완전한 수식은 숫자와 연산자를 합한 개수가 홀수이기 때문이다. 이에 따라 numOpCnt % 2의 값이 0을 가리킨다면 완벽한 수식이 아니므로 그에 따른 에러를, 그렇지 않다면 None을 반환하도록 했다.

■ checkSyntax 함수에 사용된 에러 상수

```
1  BRACKET_ERROR1 = "오른 괄호가 먼저 나왔습니다."
2  BRACKET_ERROR2 = "왼괄호의 개수가 더 많습니다."
3
4  NUM_ERROR1 = "마이너스가 중복 입력되었습니다."
5  NUM_ERROR2 = "숫자 사이에 '-'가 들어갔습니다."
6  NUM_ERROR3 = "실수를 입력하셨습니다."
7  NUM_ERROR4 = "알파벳 혹은 '-' 이외의 기호를 입력하셨습니다."
8
9  EXPRERESSION_ERROR = "완전한 수식을 입력하세요"
10 OPERATOR_ERROR = "UNDEFINED"
```

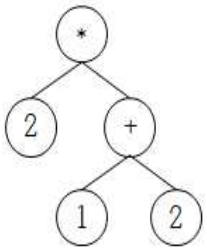
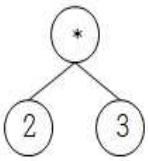
checkSyntax 함수에 사용된 에러 상수들이다. 파이썬에서는 상수를 선언할 수 있는 방법이 없다. 따라서 상수라는 것을 알려주기 위해 어쩔 수 없이 변수를 사용하는 대신 변수의 이름을 모두 대문자로 작성하도록 해야 한다. 만약 에러가 있을 경우에 위에 있는 상수 중에서 해당되는 에러에 관한 문자열이 저장된 상수를 반환하게 된다.

(4) 중위식을 후위식으로 변환해주는 getPostfix 함수와 eval 함수

과제의 내용 마지막 부분에는 구현 방법에 대한 힌트가 제시되어 있다. 힌트는 중위 표기식을 후위 표기식으로 변환하고 결과값을 구해주는 방법에 대한 내용이며 다음과 같은 기술되어 있었다.

■ 입력 받은 중위 표기식을 후위 표기식으로 변환한다.
■ 스택을 이용하여 결과 값을 계산한다.
■ Postorder로 스택에 저장하며, 연산자 MINUS가 나오면 연산결과 값을 스택에 저장한다.

이를 토대로 구현을 하기 위해서는 우선 POSTORDER에 대해서 알아봐야할 필요성이 있다. POSTORDER는 후휘 순회(POSTORDER TRAVERSAL)을 뜻한다. 후회 순위를 이해하기 위해서는 피연산자와 연산자의 관계를 알아봐야 한다. 일반적으로 수식은 연산자와 피연산자로 구분한다. 여기서 우리는 피연산자보다는 연산자를 보고 계산의 방향을 결정하고 결과 값을 도출하는 형식으로 계산을 수행한다. 즉, 2개의 피연산자는 연산자에 종속된 관계를 맺고 있다는 것이다. 위의 이론을 토대로 2 * (1 + 2)를 예시로 삼아 우리는 아래와 같은 이진 트리를 만들어 낼 수 있다

		
우리가 알고 있는 수식을 트리 형태로 만듦	오른쪽의 '+' 연산자에 종속된 피연산자의 결과값을 도출	마지막으로 '*' 연산자에 종속된 피연산자를 해당 연산자를 통해 계산 최종 결과값으로 6이 나옴

이렇게 수식을 이진트리로 만들어 보았다. 이제부터 후위 순회에 대해서 설명하도록 하겠다. 후위 순회는 가장 왼쪽의 서브 트리부터 오른쪽 서브 트리, 마지막으로 가장 최정점인 루트 순으로 순회하는 것을 말한다. 즉, 위의 표를 예를 들어보면 2 -> 1 -> 2 -> + -> * 순으로 순회하게 된다는 것이다. 이를 리스트에 순서대로 저장해보면 2 * (1 + 2)를 후위 표기식으로 만들어 리스트에 저장한 것과 똑같다는 것을 알 수 있다. 결과적으로 스택을 이용해서 후위 표기식으로 바꾸어주고 이를 토대로 연산 작업을 해주면 된다는 것이다. 해당 프로그램에서는 스택을 활용하여 중위표기식을 후위표기식으로 바꾸어서 이를 리스트의 형태로 반환하는 getPostfix 함수와 이를 토대로 연산 과정을 수행하여 결과값을 도출하고 반환해주는 eval 함수로 나누어서 작성했다.

■getPostfix 함수

```
1  def getPostfix(expr):
2      stack = Stack()
3      output = []
4
5      for term in expr:
6          if term == '(':
7              stack.push(term)
8          elif term == ')':
9              while not stack.isEmpty():
10                 op = stack.pop()
11                 if op == '(':
12                     break
13                 else:
14                     output.append(op)
15         elif term == "MINUS":
16             if not stack.isEmpty() and stack.peek() != '(':
17                 output.append(stack.pop())
```

```
18                  stack.push(term)
19              else:
20                  output.append(term)
21
22          while not stack.isEmpty():
23              output.append(stack.pop())
24
25          return output
```

중위 표기식으로 정렬된 리스트를 스택을 활용하여 후위 표기식 방식으로 바꾸고 리스트에 저장하여 반환해주는 함수이다. 가장 먼저 스택을 생성하고 후위 표기식으로 바꾸면서 저장해 나갈 리스트 output을 선언했다. 그러고 나서 토큰의 길이만큼 반복문(for문)을 수행하게 된다. 반복문을 수행하는 과정에서 다중 분기문을 만나게 되는데 그에 따른 수행은 다음과 같다.

먼저 term이 왼쪽 괄호라면 스택에 저장해준다. 반면에 term이 오른쪽 괄호라면 스택이 비어질 때까지 하나씩 pop하여 op에 저장하고 op의 값이 왼쪽 괄호인지 여부를 판별하고 그렇지 않다면 리스트 output에 저장한다. 이때 op의 값이 왼쪽 괄호라면 반복문(while not stack.isEmpty())을 빠져나오게된다. 마지막으로 왼쪽 괄호와 오른쪽 괄호, 모두 해당되지 않으면 리스트 output에 term을 저장한다. 여기까지가 for문에서 수행하는 구문이다.

이러한 과정을 거쳐서 for문이 끝나고 스택이 비어있는지 확인해서 비어있지 않으면 스택이 비어질 때까지 output에 순차적으로 저장해주면 된다. 이렇게 모든 과정이 끝나면 output을 반환하게 된다.

■eval 함수

```
1   def eval(expr):
2       stack = Stack()
3
4       for token in expr:
5           if token == "MINUS":
6               val2 = int(stack.pop())
7               val1 = int(stack.pop())
8               stack.push(val1 - val2)
9           else:
10              stack.push(token)
11
12      return stack.pop()
```

eval 함수는 비교적 쉬운 구성으로 이루어져있다. token에 저장된 데이터가 "MINUS"가 나올때까지(즉, 숫자가 나오는 동안) 스택에 푸시하고 "MINUS"가 나왔을 때, 가장 최근에 있는 값과 그 이전의 값을 각각 pop해서 변수 val2와 val1에 저장하고 val1과 val2의 차를 구한 다음에 push를 통해 스택에 저장해준다. 이러한 작업을 for문을 통해 반복 수행하게 되면 스택에는 하나의 결과 값만 남게 된다. 이러한 과정이 모두 끝나면 사용자가 입력한 수식의 결과로 스택에 남아있는 값을 pop하여 반환하도록 작성되었다.

(5) 메뉴 화면을 출력해주는 programInterface 함수

```
1   def programInterface():
2       print("=====================================================")
3       print("1. File Load")
4       print("2. Interaction Mode")
5       print("3. Exit")
6       print("=====================================================")
```

programInterface 함수는 메뉴화면를 출력해주는 함수다. 프로그램 시작되고 while문(무한 루프)에 들어서게 되면 제일 먼저 호출받게 된다. 이 때, 사용자는 programInterface 함수에서 출력되는 메시지들을 보고 어떤

메뉴를 선택할 것인지 결정하게 된다.

(6) 파일을 로드해주는 fileLoader 함수

```
1  def fileLoader(filename):
2      try:
3          infile = open(filename, "r")
4      except FileNotFoundError:
5          print("₩n파일을 찾을 수가 없습니다.")
6          return None
7
8      lines = infile.readlines()
9      infile.close()
10
11     return lines
```

파일을 로드해주는 기능이 구현된 fileLoader 함수다. 먼저 try문을 통해 매개 변수로 받아온 사용자가 입력한 문자열이 저장된 fileName을 open 함수를 통해 읽기 전용으로 불러들여 변수 infile에 저장을 시도한다. 이때, 파일이 존재하지 않으면 파이썬에서 기본으로 제공해주는 에러인 FileNotFoundError가 발생하게 되며, 파일을 찾을 수 없다는 문구가 출력되면서 None값을 반환한다. 만약 에러가 발생되지 않는다면 readlines() 함수를 통해서 lines라는 리스트에 줄 단위(개행문자 기준)로 차례대로 저장되게 된다. 그리고 나서 파일을 닫아주고 lines를 반환하게 되면 fileLoader 함수의 역할은 끝나게 된다.

(7) 파일의 내용과 결과를 출력해주는 showFileResult 함수

```
1  def showFileResult(lines):
2      lineNum = 0
3
4      print("₩n파일 내용은")
5      print("-----------------------------------------------------")
6
7      for line in lines:
8          lineNum += 1
9          print(lineNum, "행 :", line[:-1])
10
11     lineNum = 0
12
13     print("-----------------------------------------------------")
14     print("입니다.₩n")
15     print("출력 결과는")
16     print("-----------------------------------------------------")
17
18     for line in lines:
19         lineNum += 1
20         print(lineNum, "행 결과 :", operation(line[:-1]))
21
22     print("-----------------------------------------------------")
23     print("입니다.₩n")
```

줄 단위로 받아온 파일의 내용을 출력하고 한줄씩 결과를 출력하는 함수다. 가장 만저 , 라인의 번호를 나타내는 변수 lineNum에 0으로 초기화를 해준다. 그리고 나서 for문을 돌려서 루프가 끝날때마다 라인의 번호를 1을 증가시키면서 해당 라인의 내용을 리스트 lines에서 가져와 출력해나간다. 첫 번째 for 루프가 다 끝나게되면 파일의 내용들이 모두 출력되게 된다. 같은 방법으로 lineNum을 0으로 초기화하고 두 번재 for문을 돌리면서 operation함수를 활용하여 해당 요소의 결과를 출력해 나간다. 이때, 개행 문자를 제거하기 위해 operation의 인자를 line[:-1]로 주었다. 이렇게 두 번째 for문이 끝나게되면 첫 번째 for문과 마찬가지로 에러나 결과가 모두 출력되게 된다.

(8) 한 줄의 문자열을 입력받고 결과를 출력해주는 interactinMode 함수

```
1  def interactionMode():
2      inputStr = input("₩n문장을 입력하세요 >> ")
3      print("₩n결과 :", operation(inputStr))
```

사용자가 메뉴 화면에서 인터프리터(2번) 메뉴를 선택했을 때 호출되는 함수이다. 가장 먼저 사용자로부터 문자을 입력받고 이를 매개변수로한 operation 함수를 호출하게 된다. operation 함수는 에러 체크와 연산 작업을 수행하는 함수로 결과 값으로 에러의 내용이나 계산 결과 값을 반환하는 역할을 한다. 해당 함수에서는 operation 함수를 호출하고 그에 대한 결과 값을 출력하도록 했다.

(9) 연산 작업을 수행해주는 operation 함수

```
1  def operation(inputStr):
2      tokens = getTokens(inputStr)
3      syntaxError = checkSyntax(tokens)
4
5      if syntaxError == None:
6          expr = getPostfix(tokens)
7          result = eval(expr)
8          return result
9      return syntaxError
```

operation 함수는 사용자가 입력한 수식이 올바른지 판별해주고 에러가 있다면 에러 문구를 반환하고, 그렇지 않다면 후위표기식 변환을 거쳐 계산한 결과값을 반환해주는 역할을 한다. 이 때, 에러가 없다면 syntaxError의 값은 None를 가리키게 된다. 따라서 조건문을 통해 syntaxError가 None일 때, getPostfix와 eval 함수로 계산 작업을 수행하고 결과 값을 반환하도록 했다. 물론, syntaxError가 None이 아니라면 syntaxError를 반환하게 된다.

(10) Main

```
1  if __name__ == "__main__":
2      while True:
3          programInterface()
4
5          inputNum = int(input("메뉴를 선택 하세요 >> "))
6
7          if inputNum == 1:
8              filename = input("₩n파일명을 입력하세요 >> ")
9              lines = fileLoader(filename)
10             if lines != None:
11                 showFileResult(lines)
12         elif inputNum == 2:
13             interactionMode()
14         elif inputNum == 3:
15             break
16         else:
17             print("주어진 범위에 맞는 번호를 선택하세요!!")
```

main 함수이 역할을 하는 파트이다. 파이썬에서는 main 함수를 조건문을 통해 구현하게 된다.

우선, main 함수가 실행이 되면 무한 반복문이 시작되는 것을 알려주는 while문이 나온다. 그렇게 무한 루프가 시작되면 먼저 메뉴의 출력을 담당하는 programInterface 함수를 호출하게 된다. 그리고 나서 메뉴 번호를 입력하라는 문구가 출력되고 사용자로부터 번호를 입력받아 inputNum에 저장하게 된다. 이 때, 입력받은 값은 타입이 문자열이기 때문에 편의상 int형으로 바꾸어 주었다.

사용자로부터 번호를 입력받았으면 다중 if문을 거치게 된다. 가장 먼저 inputNum이 1을 가리킨다면 파일 모드를 선택한 것이므로 파일의 이름을 입력하라는 문구를 출력하고 값을 입력받아 fileName에 저장한다. 다음으로 이를 매개 변수로 한 fileLoader 함수를 호출하여 파일을 로드하는 과정을 거친다. 이 때, 파일 로드에 실패(파일이 없을 경우)하게 되면 에러가 발생되어 None값을 리턴한다. 반면에, 로드에 성공하게 되면 파일의 내용을 개행문자를 기준으로 분할되어 저장된 리스트가 반환되어 lines에 저장된다. 이러한 과정이 끝나게 되면 정상적으로 lines에 리스트가 저장되었는지 확인하고 showFileResult 함수를 호출하여 파일의 내용과 결과 값을 출력한다. 만약 lines가 리스트가 아닌 None이라면 showFileResult 함수는 실행되지 않고 초기의 메뉴 화면으로 돌아가게 된다.

다음으로 inpuNum의 값이 2를 가리킨다면 interactionMode 함수를 호출한다. 이 때, interactionMode 함수는 한 줄의 문자열로 된 수식을 입력받아 operation 함수를 통해 연산 작업을 수행하고 반환된 에러나 결과 값을 출력하게 된다.

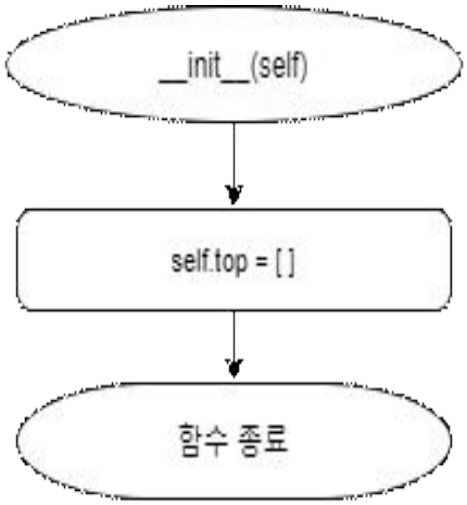
끝으로 inputNum의 값이 3이라면 종료를 선택한 것이므로 반복루프를 빠져나가게 되어 프로그램을 종료하게 된다. 여기서 inputNum의 값이 1~3이 아니라면 메뉴 선택의 범위를 벗어난 것이므로 사용자에게 올바른 번호를 입력하라는 문구를 출력한다.

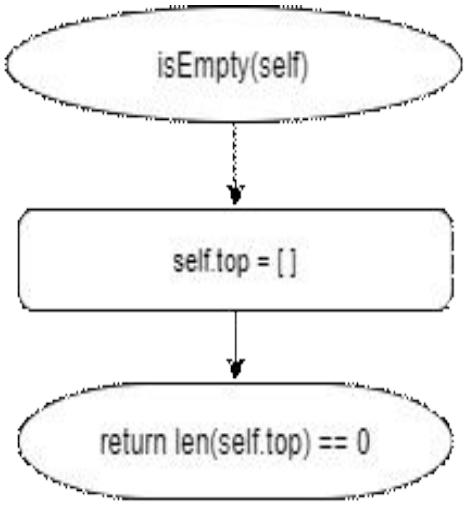
이렇게 조건문을 끝나게 되면 다시 처음으로 돌아가서 사용자로부터 메뉴 입력을 받은 시점으로 되돌아가게 된다. 이 때, 반복문이 무한 루프를 돌기 때문에 3번을 입력할 때 까지 프로그램은 같은 작업을 반복하게 될 것이다.

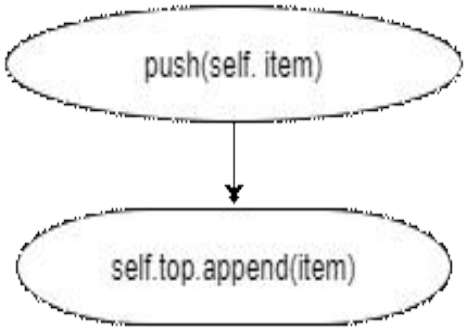
4. 프로그램 실행 결과

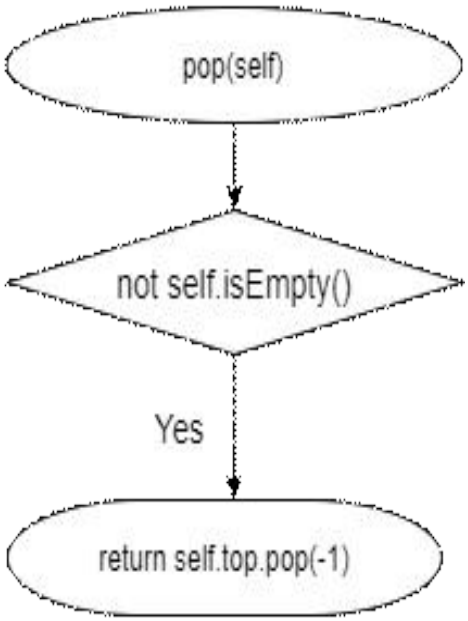
5. 흐름도

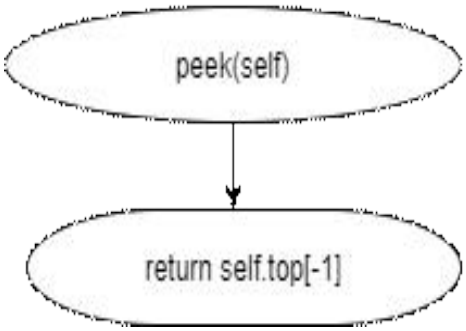
■ 스택 클래스의 메소드 흐름도

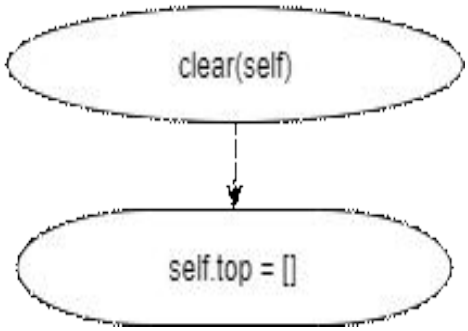




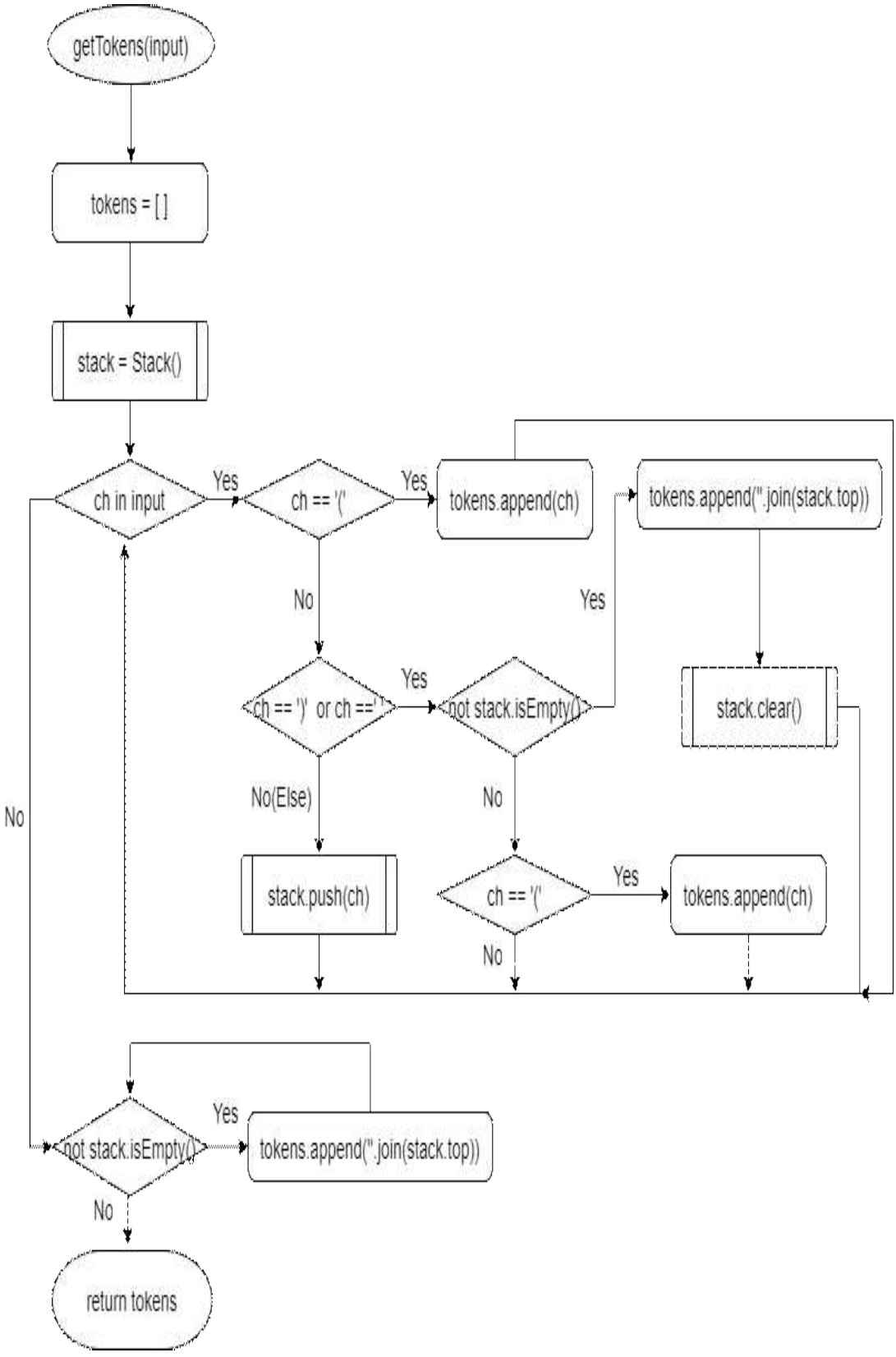




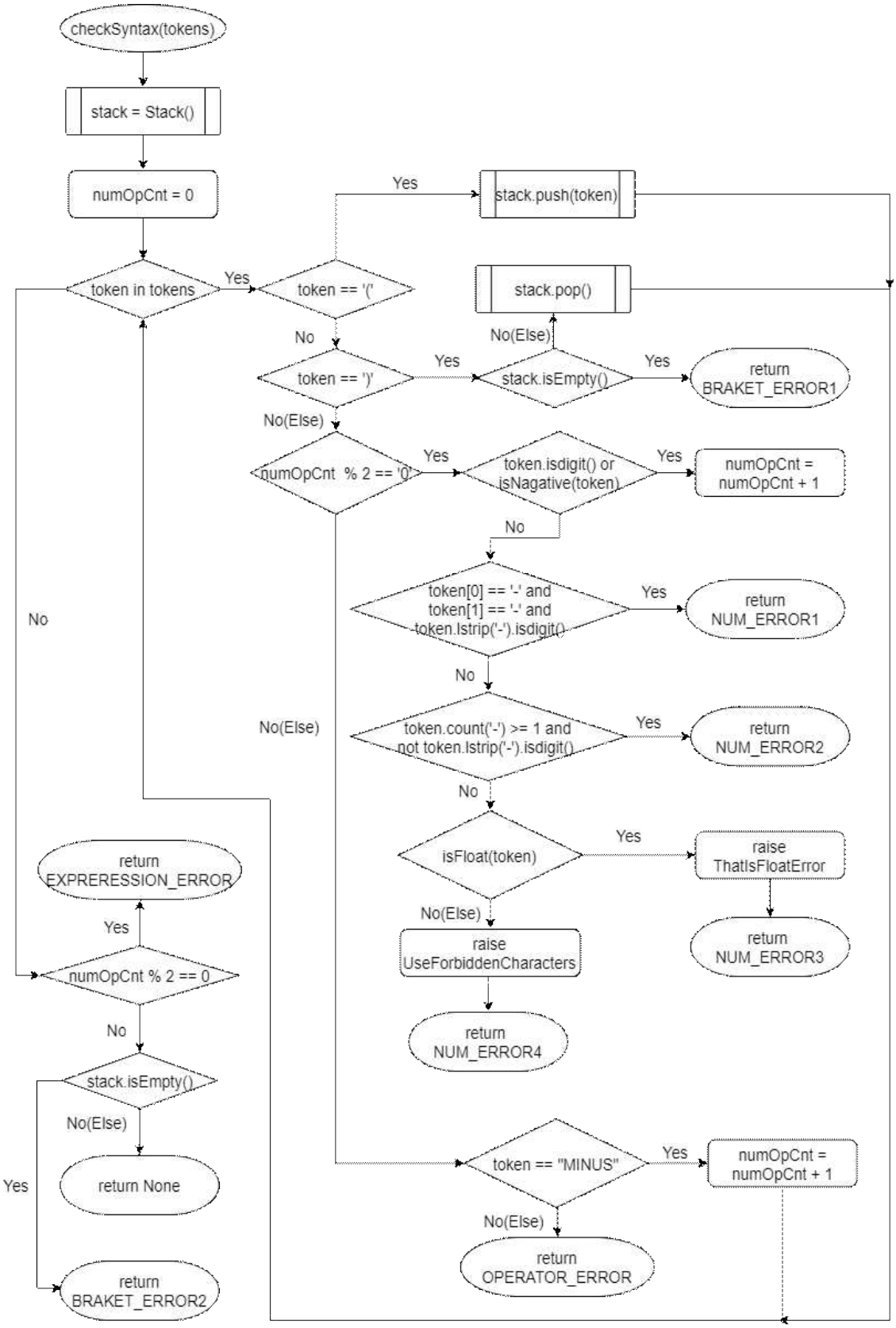




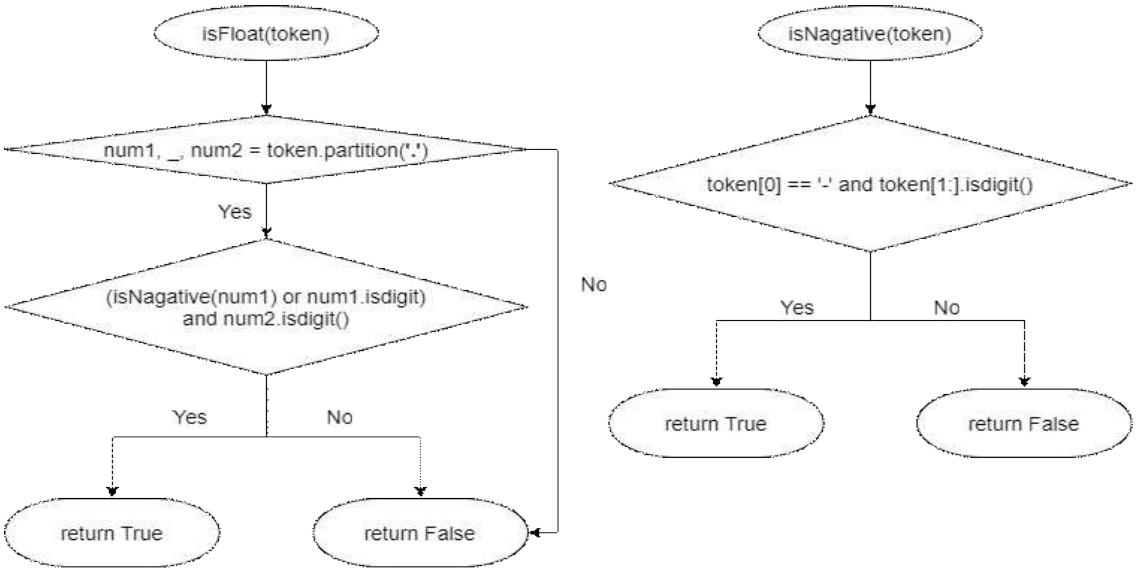
■ **getTokens 함수 흐름도**



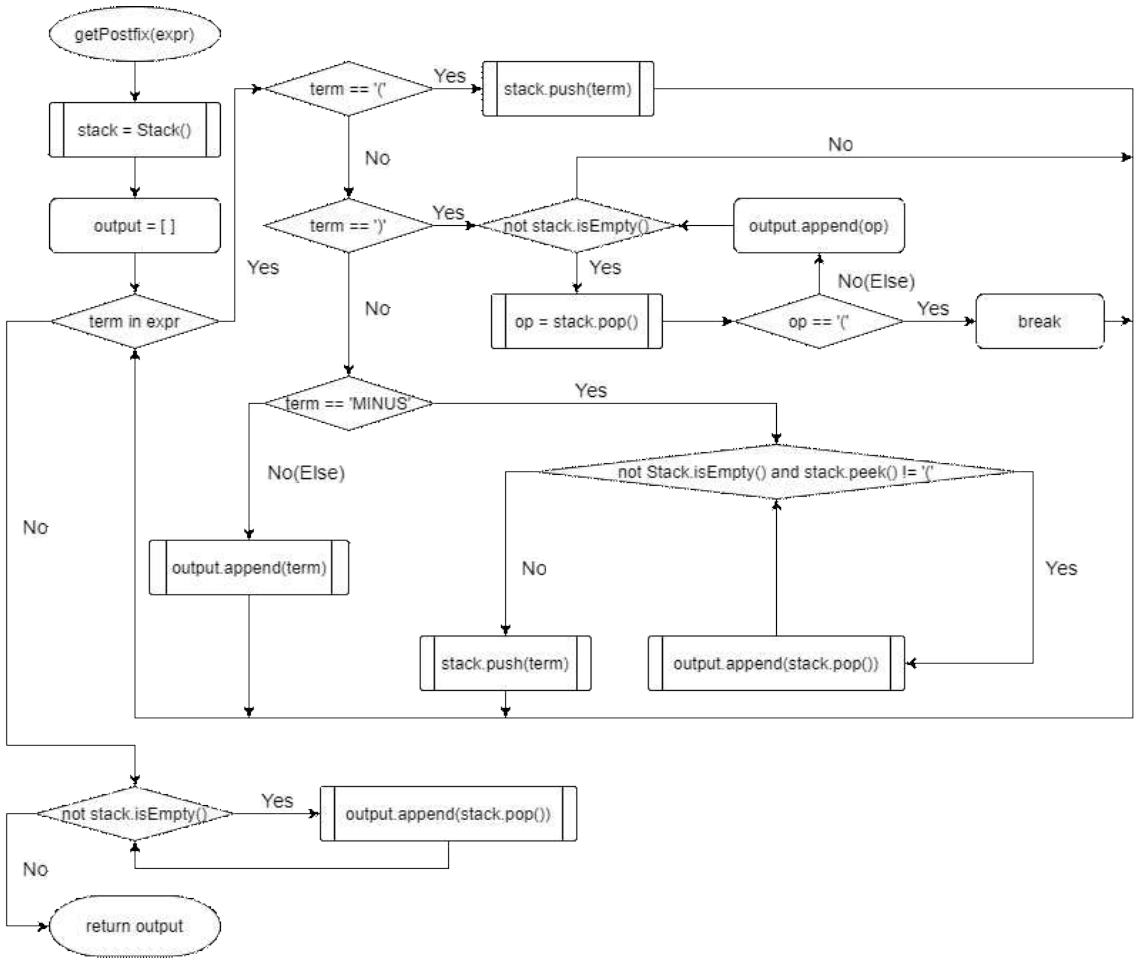
■ checkSyntax 함수 흐름도



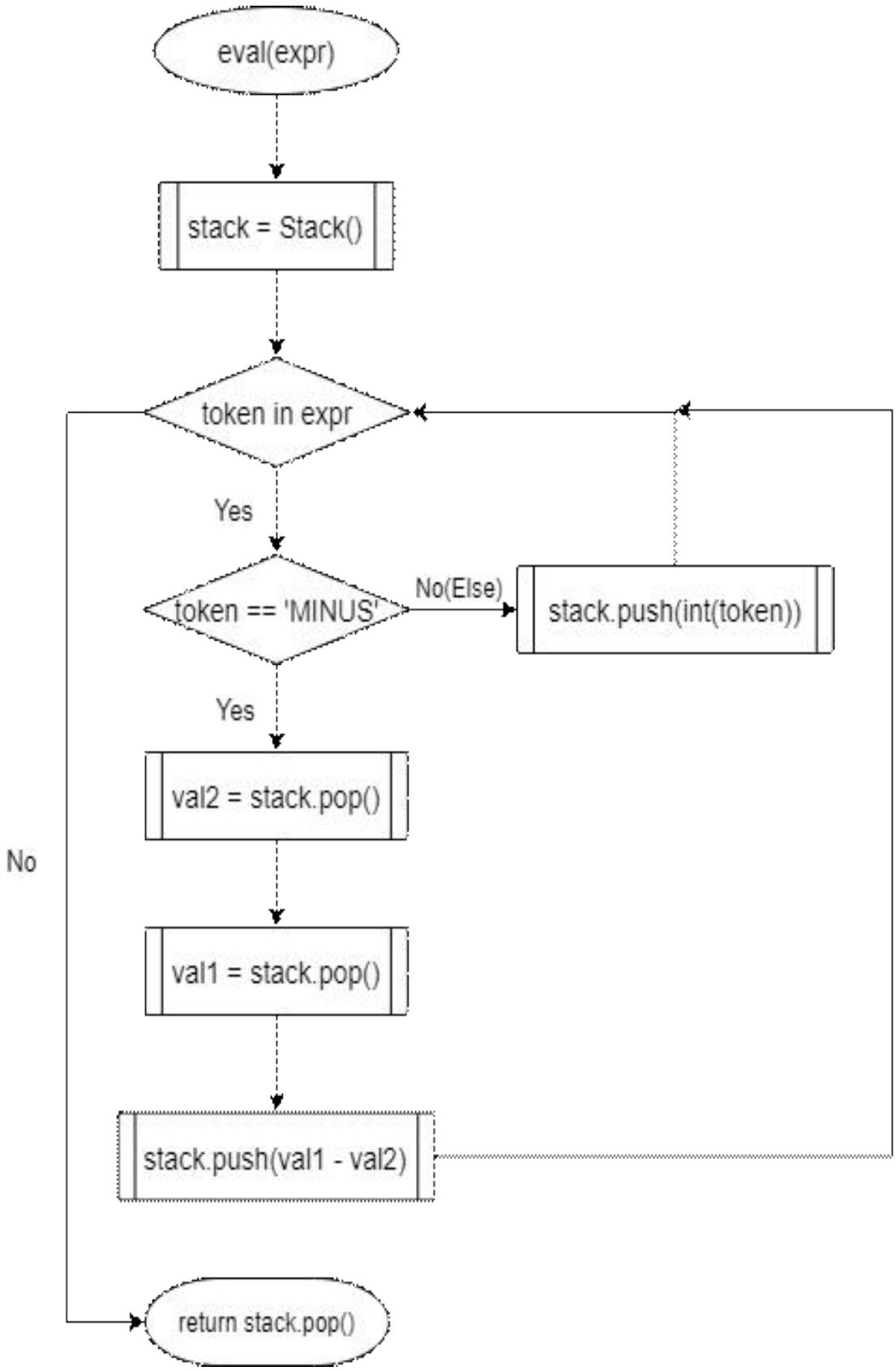
■ isFloat / isNagative 함수 흐름도



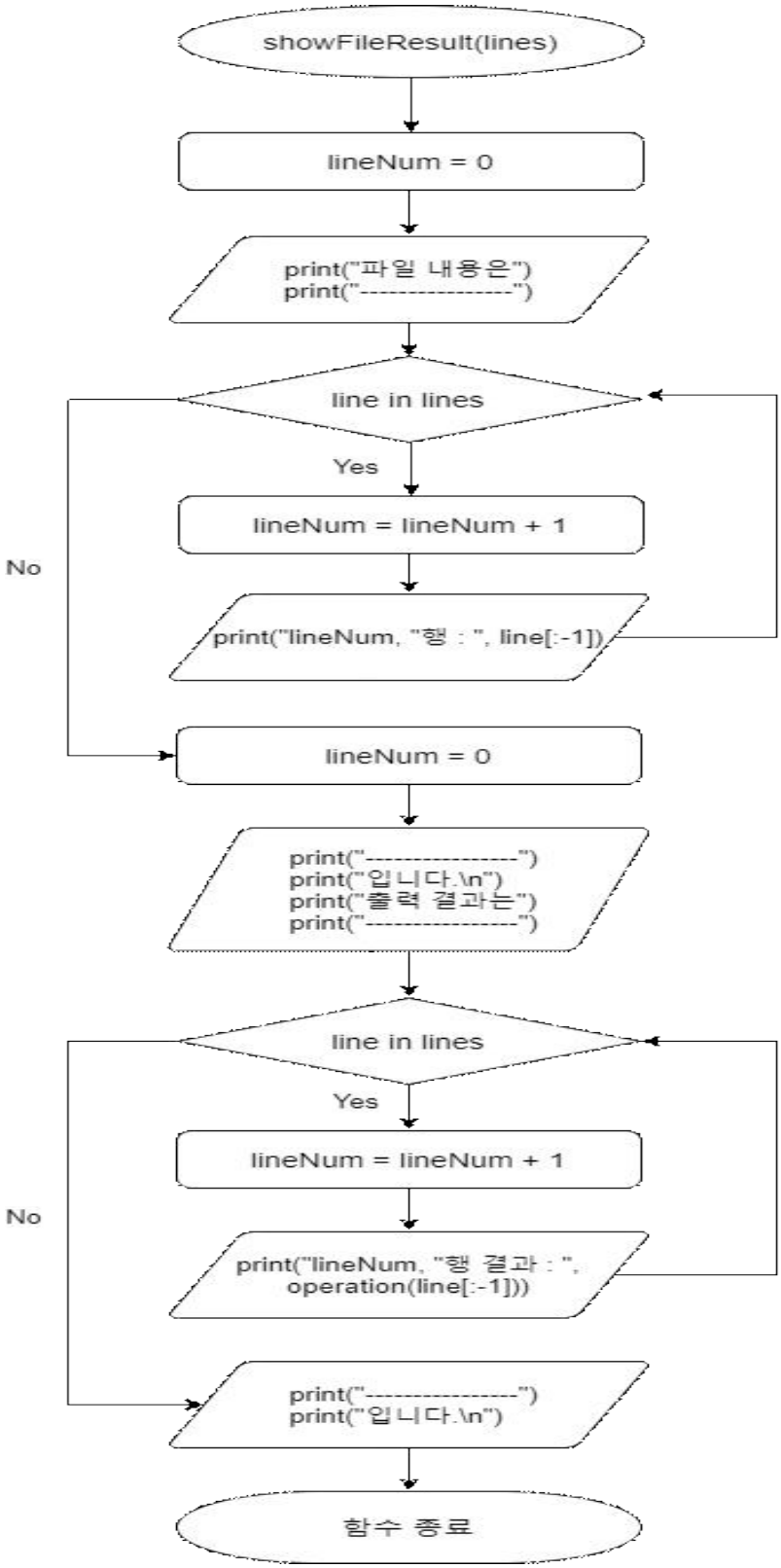
■ getPostfix 함수 흐름도



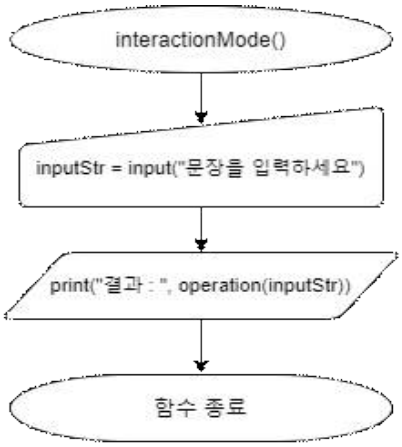
■ **eval 함수 흐름도**

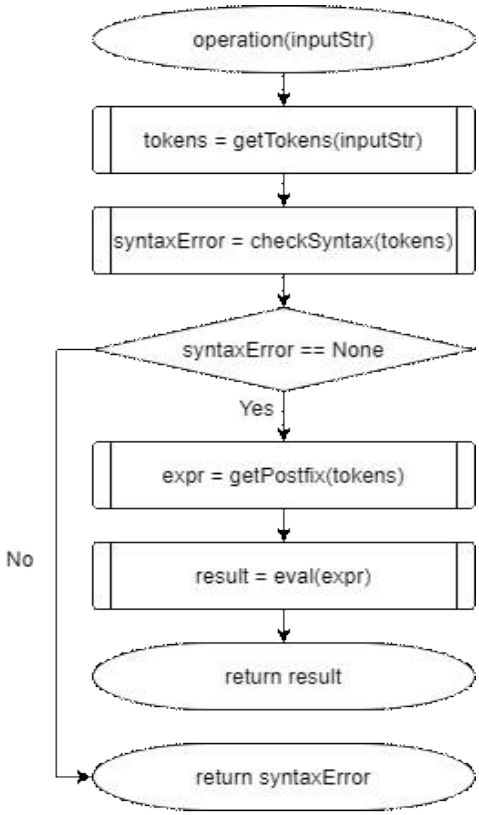

eval(expr)
stack = Stack()
token in expr
Yes
token == 'MINUS'
No(Else)
stack.push(int(token))
Yes
val2 = stack.pop()
val1 = stack.pop()
stack.push(val1 - val2)
No
return stack.pop()

■ showFileResult 함수 흐름도

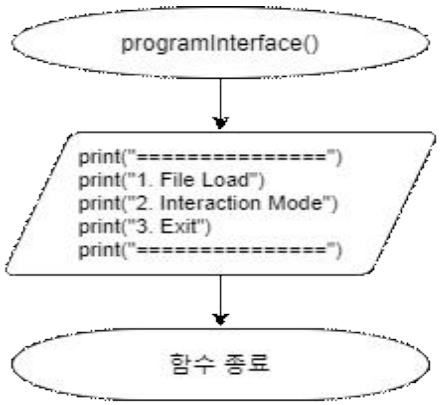


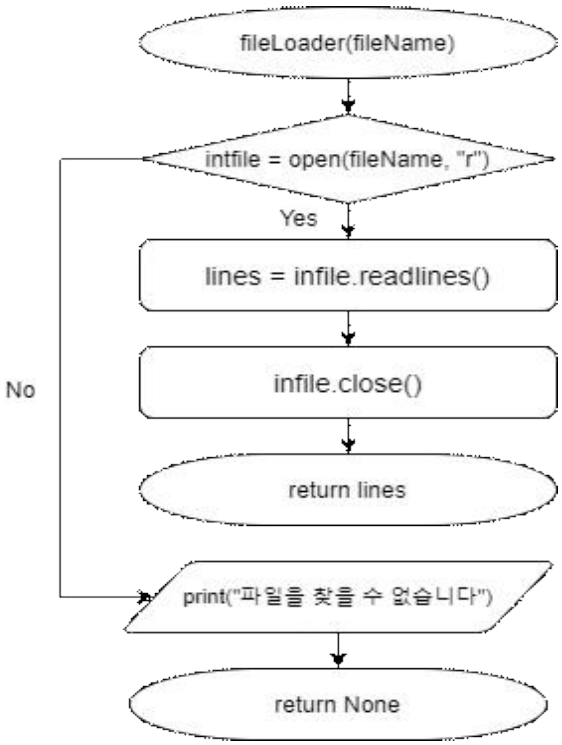
■ interactionMode / operation 함수 흐름도



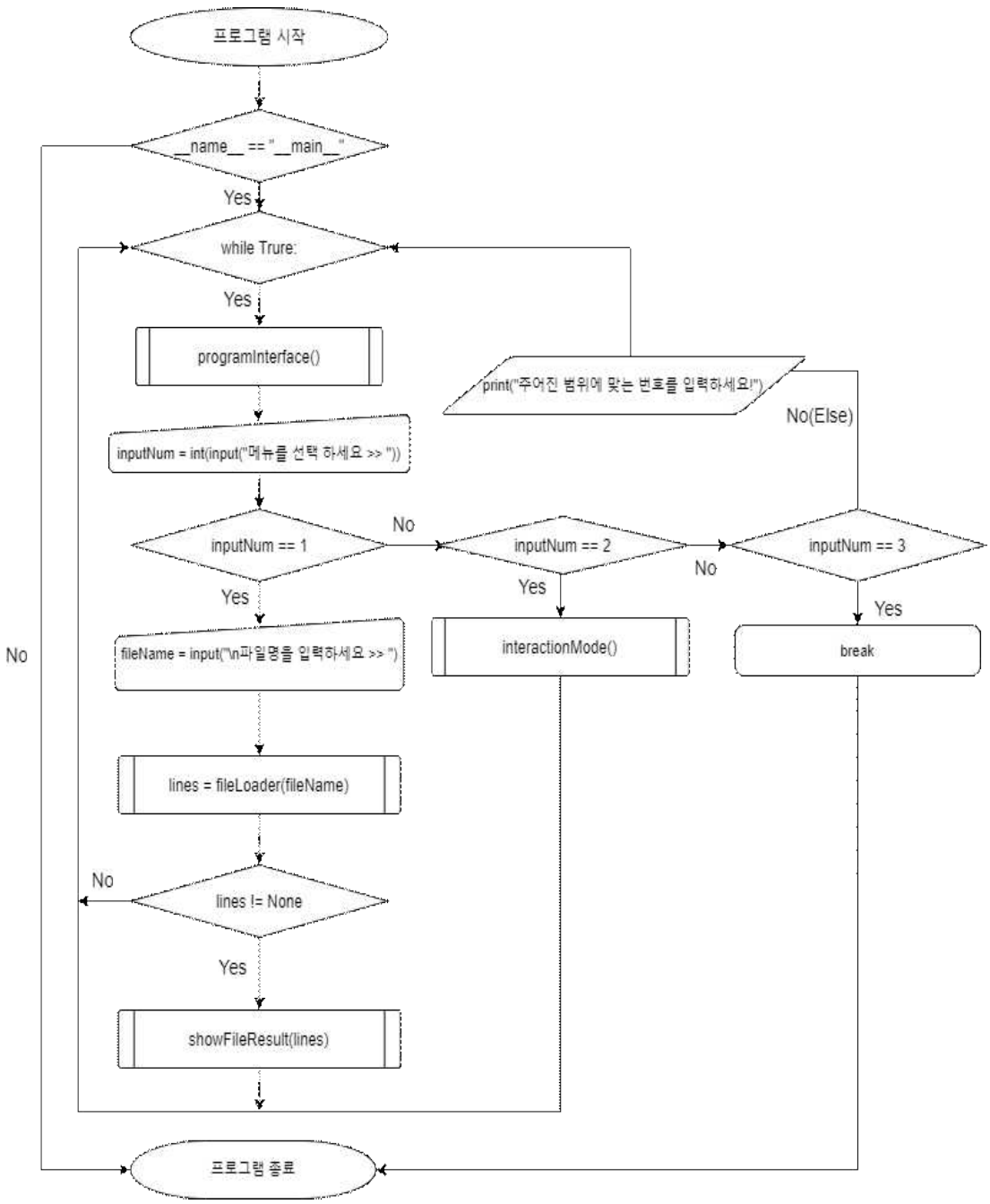


■ programInterface / fileLoader 함수 흐름도





■ main 흐름도


프로그램 시작
__name__ == "__main__"
Yes
while Trure:
Yes
programInterface()
inputNum = int(input("메뉴를 선택 하세요 >> "))
inputNum == 1
No
inputNum == 2
No
inputNum == 3
Yes
Yes
Yes
fileName = input("\n파일명을 입력하세요 >> ")
interactionMode()
break
No
lines = fileLoader(fileName)
No
lines != None
Yes
showFileResult(lines)
print("주어진 범위에 맞는 번호를 입력하세요!")
No(Else)
No
프로그램 종료