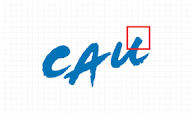


**PROGRAMMING LANGUAGE**

Team Project





**학과:소프트웨어학부**

**이름:정설희, 임건, 유효창**

**학번:20193574, 20192848, 20190551**

**2020.12.06**

**< LISP >**

**“안녕, 내 이름은 리습! 날 소개해볼게~”**

**LISP란 .. ?**

-Lisp는 프로시저를 정의하고 다른 프로시저들로부터 호출할 수 있는 재귀적 성격을 가져 간단한 구문을 가지며, 자료 형태의 요구가 적고, 인터프리터를 사용하여 대화적인 환경을 제공한다는 면이 특징이다.

Lisp에서 하나의 표현식은 (괄호) 로 감싸져있다. 어떤 한 표현식 안에 다른 표현식이 포함될 수 도 있다. 예) ( SETQ X (CAR ‘(1 2 3)))

Lisp에서는 데이터를 숫자, 문자열, 리스트 등으로 처리한다.

모든 연산이 함수를 통해 이루어진다. 이 때, 연산자가 앞에 오고 이후에 적합한 피연산자들이 나열된다.

영문 대소문자 구분 없이 처리한다.

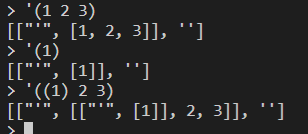
Comment 기호로는 ;를 사용한다.

Lisp에서는 “ ‘ “ 기호를 사용하여 심볼과 리스트를 다른 데이터들과 구분한다.

LIST INTERPRETER 설계:

우리 팀은 파이썬 언어를 사용하여 LISP 인터프리터를 구현하였다. 이는, 파이썬이 다른 언어들에 비해 자료형에 제약이 적기 때문에 LISP의 리스트를 구현하기에 적합하다 판단하였기 때문이다.

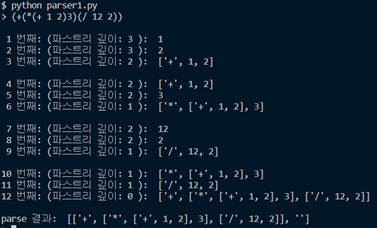
리스트 자료형은 [ “ ‘ “ , [ 리스트의 요소들] ] 로 파싱하기로 약속했다.    
( 예) (1 2 3) -> [ “’”, [1, 2, 3]] ) 이는, 파싱 된 토큰을 이후 인터프리터에서 관리하기 용이하게 하기 위해서이다.

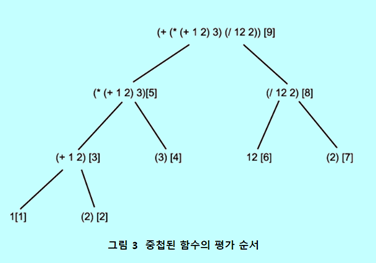


[표현식이 parse.py를 거쳐 파싱된 형식]

Lisp\_interpreter를 구현하기 위해 주로 parse->lex->interpreter의 과정을 거치지만, parse 과정을 먼저 거치게 되고 이 모듈에서 처리된 것들을 interpreter.py에서 lexer과정과 동시에 해석하여 실행하는 순서로 설계하였다.

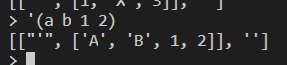
이 때, parser.py를 거친 표현식은 parse tree 의 형식에 따라 계층적으로 구성된다.





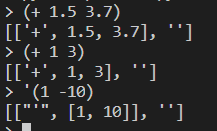
[ (+(\*(+ 1 2) 3) (/ 12 2)) 의 표현식을 입력 받았을 때, parse의 결과 ]

숫자(정수, 실수)와 문자의 입력을 구분하여, 숫자가 입력 된 경우 int 혹은 float형으로 변환하여 저장하고, 문자는 문자열로 저장된다. 이 때, LISP는 대소문자 구별을 하지 않으므로, 모두 대문자로 저장된다. (.Upper() 함수 사용)



[ 문자와 숫자가 파싱되는 형식 ]

또한, 정수(양수 / 음수) , 실수 를 구분해서 받아낸다.



[ 실수 양수 정수가 파싱되는 형식 ]

LISP INTERPRETER 구현 방법:

**parser.py**

 터미널에서 사용자에게 입력 받는 코드 한 줄 한 줄은 문자열로 생각하고 parsing을 진행한다.

parser는 크게 두 가지, value\_parser, key\_parser로 구분된다.

각각의 keyword들(‘define’,’+’,’if’ 등)을 기준으로 parsing이 진행된다.

LISP에선 항상 keyword들이 먼저 나오고 그 다음 피연산자들이 나오기 때문에 parse반환을 리스트의 형태로 [keyword, 나머지 부분]으로 잘라 반환하게 된다.

이 부분은 각각의 parser함수들에 구현이 되어 있다. keyword에 해당하는 index를 기준으로 문자열을 처리하게 된다.

이 모든 parser이 구현된 상태에서 최종적으로 expression을 나타내기 위해 expression\_parser함수에서 parsing 작업을 시작한다.

 피연산자 부분에서도 다른 식이 나올 수 있는데, 이 식도 LISP의 재귀적인 특징으로 똑같이 [keyword, 나머지 부분]를 파싱하여 앞서 존재하는 연산자 나머지 부분의 원소로 parsing 되는 것이다.

닫힌 괄호가 나올 때 까지 코드의 부분이 끝나지 않으므로, expression\_parser를 재귀적으로 실행시켜 모든 괄호에 쌓인 expression에 대해 parsing이 진행된다.

**그 다음으로 심볼과 리스트들을 처리해주기 위한 parser가 따로 존재한다.(다음 페이지)**

**quote\_parser(data)**

“ ‘ “ 가 가장 처음에 오면 위 함수를 호출한다.

“ ‘ “ 가 가장 처음에 오는 경우는, ‘(1 2 3) 과 같이 리스트인 경우와, ‘X 와 같이 심볼인 경우가 있는데, “ ‘ “ 다음 요소가  ‘ ( ‘ 이면 리스트로, 그가 아니면 심볼로 처리해준다.

|  |
| --- |
| def quote\_parser(data):      if data[0] == "'": # 첫 요소가 " ' ' 이면..         if data[1] == '(':  # 그 다음요소가 '(' 이면 -> LIST             ...         else: # 아니면 -> 심볼             ... |

**data[1] == ‘(‘ 인 경우엔,** 임시 list *tmp*에 “ ‘ “ 를 요소로 추가하여 리스트의 시작임을 알리고, 괄호 다음 문자부터 list\_parser로 넘겨 새로운 리스트를 생성한다. 이후에, tmp에 이 결과를 넣어주고, 잘린 곳을 기점으로 return 해준다.

**data[1] != ‘(‘인 경우엔,** **정규식**을 사용하여 공백이 올 때 까지(‘/w+’) 잘라줘 심볼로 처리해준다.

**list\_parser(data)**

위 함수는 quote\_parser(data)에서 리스트 생성을 위해 호출되는 함수이다. 우선, 리스트를 저장할 빈 리스트 L을 생성해준다.

이 때, while문을 통해 data가 한 글자씩 읽혀짐에 따라,

**1. 리스트 안에 또 리스트가 들어 오는 경우 (s ‘(‘의 입력 )**

**2. 리스트가 끝나는 경우 ( ‘)’ 의 입력)**

**3. 공백이 아닌 경우**

의 경우로 나누어준다.

**1번의 경우에는,** 괄호 다음부부터 끝까지 ⓐlist\_parser로 다시 넣어준다.**(재귀함수)** 임시사용 리스트인 T에 “‘“를 넣어주고, ⓐ의 리턴값을(리스트형) T의 요소로 추가해준다. 그 후, L에 T를 요소로 추가해주고, index를 추가해준다.

**2번의 경우에는,** index를 1 더해준 후, [L, index] 를 return 해 준다. 이는, 리스트 안에 리스트가 있을 때, inner리스트 자체와 그 다음으로 읽을 index를 동시에 return해 주기 위해서이다.

**3번의 경우에는,** not in 을 이용하여, 해당 문자가 공백인지 아닌지를 판별해준다. 이후, **정규식 표현**을 이용하여 공백 혹은 특수문자가 오기 전까지 읽어주고, index값을 이용해 그 때 까지 잘라준다.

**이 파싱된 결과를 반환하여 interpreter.py에서 반환된 리스트를 사용해 최종적으로 코드를 실행한다.**

**Interpreter.py**

eval(x) 함수:

* lexer 와 interpreter 역할 모두 다 하는 함수

이 함수는 interpreter.py의 끝판왕 함수이다. 사용자가 코드를 입력하면, 그 코드를 parser.py에서 파싱한 후 eval함수가 어떤 코드인 지 판단하고 실행한다.

숫자, 문자, 리스트에 대해 판별해주고, 파싱한 token에 정보에 따라 밑에 구현된 함수로 이동하여 작업을 수행한다.

+ , - , \*, / :

* ‘+’:

1.우선 인자로 들어온 x를 token과 나머지 부분으로 나눈다.

2.나머지 부분에 있는 모든 피연산자에 대해 eval를 먼저 거치고 검사한 부분이 숫자로 판단되면 저장소에 하나씩 더해준다.

3.만약 숫자가 아니라면 올바르지 않은 자료형이므로 ERROR를 출력한다.

* ‘-’:

1.우선 인자로 들어온 x를 token과 나머지 부분으로 나눈다.

2.나머지 부분에 있는 모든 피연산자에 대해 eval를 먼저 거치고 검사한 부분이 숫자로 판단되면 저장소에 하나씩 빼준다.

3.만약 숫자가 아니라면 올바르지 않은 자료형이므로 ERROR를 출력한다.

* ‘\*’:

1.우선 인자로 들어온 x를 token과 나머지 부분으로 나눈다.

2.나머지 부분에 있는 모든 피연산자에 대해 eval를 먼저 거치고 검사한 부분이 숫자로 판단되면 첫 번째 숫자를 저장소에 더해주고 그 이후 숫자에 대해 저장소에 하나씩 곱해준다.

3.만약 숫자가 아니라면 올바르지 않은 자료형이므로 ERROR를 출력한다.

* ‘/’:

1.우선 인자로 들어온 x를 token과 나머지 부분으로 나눈다.

2.나머지 부분에 있는 모든 피연산자에 대해 eval를 먼저 거치고 검사한 부분이 숫자로 판단되면 첫 번째 숫자에 대해선 저장소에 저장하고 그 이후 숫자에 대해선 하나씩 나눠준다.

3.만약 나눔수가 0이거나 숫자가 아닐 경우 ERROR를 출력한다.

IF:

* 첫번째 인자인 조건식이 참이면 두번째 인자를 실행, 거짓이면 세번째 인자를 실행

1. 인자로 들어온 x를 test(조건식), conseq(참:실행문), alt(거짓:실행문)으로 나뉘어 받는다.
2. alt의 argument가 2개 이상이면은 error처리를 해준다.
3. test(조건식)을 eval로 test의 값을 얻는다.
4. test의 값이 True 또는 False의 값이 아니라면은 조건 연산자가 아니므로 ERROR 처리
5. 조건연산식 test의 값이 True면은 conseq exp를 eval에 넘긴다.
6. 조건연산식 test의 값이 False면은 alt exp를 eval에 넘긴다.
7. if문은 해당 실행문의 값을 리턴한다.

COND:

* else의 경우가 없는 if문이 여러개 있는 함수

1. 위 if문과 구문이 비슷하다.
2. 먼저 여러 argument를 받아온다.
3. argument의 개수만큼 for문을 돌려 각각의 argument(IF문)을 처리해준다.
4. 각 IF문을 처리해주는 방법은 위의 IF 함수와 동일하다.

SETQ:

* 심볼에 값을 저장하는 함수

1. 우선 인자로 들어온 x를 token과 나머지 부분(여러개의 arguments)을 나눈다.
2. argument가 2개가 아니면 error 처리를 해준다.
3. 인자로 들어온 x를 var과 exp로 나누어준 후, mem 딕셔너리에 var을 key로, exp를 value로 추가해준다.
4. 이 때, var이 스트링이 아니면 에러처리 해준다. (숫자, 심볼, 리스트 등등)

LIST:

* 원소들을 모아 새로운 리스트 구조를 생성해 내는 함수
* ‘를 붙이면 심볼 그 자체로, ‘를 붙이지 않으면 그 값을 계산해 리스트에 넣는다.

1. 우선 인자로 들어온 x를 token과 나머지 부분(여러개의 arguments)을 나눈다.
2. argument들을 list\_procedure 함수로 보내준다.

CAR:

* 리스트의 첫번째 인소를 반환하는 함수

1. 두번재 인자를 carList라는 이름으로 받아온다.
2. 그리고 car\_procedure 함수를 실행한다.
3. carList가 리스트를 담고있는 변수일 수도, 아니면 진자 리스트일 수도 있다.
4. 만약 변수일 경우 mem에서 변수명으로 접근하여 리스트를 가져와야한다.
5. 따라서 isList()함수를 통해 mem을 통해 접근할지 아니면 그냥 리스트를 가져올지 결정한다.
6. 위의 과정을 걸쳐 리스트를 가져오게되면은 첫번째 원소를 return한다.

CDR:

* 리스트의 첫번째 인소를 제외한 모든 원소를 리스트로 반환하는 함수

1. 두번째 인자를 cdrList라는 이름으로 받아온다.
2. 그리고 cdr\_procedure 함수를 실행한다
3. carList가 리스트를 담고있는 변수일 수도, 아니면 진자 리스트일 수도 있다.
4. 만약 변수일 경우 mem에서 변수명으로 접근하여 리스트를 가져와야한다.
5. 따라서 isList()함수를 통해 mem을 통해 접근할지 아니면 그냥 리스트를 가져올지 결정한다.
6. 위의 과정을 걸쳐 리스트를 가져오게되면은 첫번째 원소를 제외한 리스트를 return한다

CADDR:

* CAR, CDR, CDR을 연속적으로 사용하는 함수

1. 어떤 리스트 x에 대해서 car\_procedure 함수의 인자로 넘겨주어 호출한다.
2. 그리고 cder\_procedure 함수를 2번 연속 호출하면 CADDR 값을 받을 수 있다.

NTH:

* 입력한 숫자에 해당하는 LIST의 원소를 반환하는 함수

1.우선 인자로 들어온 x를 token과 나머지 부분을 나눈다.

2.그 후 LIST에 해당하는 원소를 eval에 거치고 LIST인지 판단하는 함수를 거치고

프로그램 내 저장소에 LIST 이름에 해당하는 LIST가 존재한다면 그 LIST의 원소를 반환하고

터미널에 입력한 LIST라면 그 LIST에 해당하는 원소를 반환한다.

3.만약에 index나 LIST의 입력이 적절하지 않거나, LIST INDEX의 벗어난다면 ERROR를 출력한다.

CONS:

* 기존의 리스트에 새로운 원소를 추가하여 리스트를 만드는 함수

1.우선 인자로 들어온 x를 token과 나머지 부분을 나눈다.

2.그 후 새로운 원소와 LIST에 대해 eval을 거치고, 새로운 원소의 자료형을 확인해준 뒤 임시 리스트에 추가해준다. 만약 새로운 원소로 들어온 이름이 심볼이고 저장소에 저장되어 있다면 저장소에서 값을 가져온 뒤 임시 리스트에 추가한다.

3. 인자로 들어온 LIST에 대해서도 프로그램 내 저장소에 있는 지 확인하고, 만약 사용자가 입력한 LIST라면 그 대로 임시 리스트에 추가하고 반환한다.

4.만약 전달된 인자가 3개 이상이거나, 새로운 원소가 아무것도 없거나, LIST의 형태로 인자가 전달되지 않았다면 ERROR를 출력한다.

REVERSE:

* 입력받은 리스트를 거꾸로 뒤집어주는 함수

1.우선 인자로 들어온 x를 token과 나머지 부분을 나눈다.

2.인자로 들어온 LIST를 프로그램 내 저장소에 존재하는 지 확인한 후, 거꾸로 뒤집어주고 LIST를 표현하는 방식으로 저장하여 반환한다.

3.만약 인자로 들어온 것이 LIST형이 아니라면 ERROR를 출력한다.

APPEND:

* 리스트들을 합쳐주는 함수

1. 인자들을 한번에 모두 받아 append\_procedure 함수로 넘겨준다.
2. append\_procedure 함수에서 리스트를 하나씩 for문으로 돌린다.
3. 새로운 리스트를 만들고 그 리스트에 기존 for문으로 돌리는 리스트를 하나씩 append 해준다.
4. 모두 append를 했으면 그 리스트를 return해준다.

LENGTH:

* 입력받은 LIST의 원소 개수를 반환하는 함수

1.우선 인자로 들어온 x를 token과 나머지 부분을 나눈다.

2.인자로 들어온 LIST를 eval를 거치고 프로그램 내 저장소에 존재하는 지 확인한 후,

그 LIST의 개수를 반환한다.

3.만약에 전달된 인자가 LIST가 아니면 ERROR를 출력한다.

MEMBER:

* 세번째 인자인 리스트에서 두번째 인자의 위치를 찾고 그 인자부터 끝까지 출력 함수

1. 찾고자하는 word와 리스트로 인자를 파싱해서 member 명령들을 진행한다.
2. word도 변수거나 어떤 함수일 수 있으므로 eval로 한번 넘긴다.
3. 파이썬 list의 .index() 함수를 이용해 word를 원소로 갖는 리스트의 원소 인덱스를 바로 찾아준다.
4. 그리고 새로운 리스트를 만들어 그 원소 인덱스부터 끝 인덱스까지 반복문을 돌려 리스트에 APPEND해주고 이를 return해준다.

ASSOC:

* 두번째 인자인 튜플 리스트에서 첫번째 인자를 키로 갖는 튜플을 찾는 함수

1. key와 튜플 리스트를 다시 eval에 넣어준다. 최종 value를 갖게 하기 위해
2. 튜플을 한개씩 for문으로 가져온다.
3. 만약 그 튜플의 키값이 key값과 같다면은 그 튜플을 출려해주도록 한다.

REMOVE:

* 첫 번째 인자로 두 번째 인자로 받은 리스트에서 찾아 모두 제거하는 함수

1.우선 인자로 들어온 x를 token과 나머지 부분을 나눈다.

2.나머지 부분에 대해 먼저 eval을 거치고 LIST내에 전달받은 원소가 있는 지 확인한다.

3.해당 원소가 LIST에 존재하지 않을 때까지 제거해준다.

4.만약 LIST입력이 되지 않거나 제거할 대상이 LIST에 존재하지 않는다면 ERROR를 출력한다.

SUBST:

* sublist라는 리스트에서 원하는 단어 word\_sub을 word로 대체하는 함수

1. sublist에서 리스트의 함수 .index()를 활용해 word\_sub이 위치한 인덱스를 얻는다.
2. 인덱스를 얻었으면 해당 리스트의 인덱스를 word로 대입하여 대체한다.

ATOM:

* 전달된 인자가 ATOM(심볼) 일 때 참(true)를 반환하는 함수

1. 우선 인자로 들어온 x를 token과 나머지 부분으로 나눈다.
2. 나머지 부분이 1개가 아니면 입력값은 1개여야 한다는 ERROR를 출력한다.
3. argument를 eval을 거치고, 이가 리스트형이거나 숫자인 경우에는 NIL을 리턴하고, str인 경우엔 True를 리턴한다.

NULL:

* 전달된 인자가 NIL일 때 참을 반환하는 함수

1.우선 인자로 들어온 x를 token과 나머지 부분을 나눈다.

2.전달된 인자가 ‘’, [] , NIL임을 판별한다.

NUMBERP:

* 전달된 인자가 숫자일때 참을 반환하는 함수

1.우선 인자로 들어온 x를 token과 나머지 부분을 나눈다.

2.전달된 인자를 numberp\_procedure에서 숫자인지 아니면 프로그램 내 저장되어 있는 값인 지 판별한다.

3.인자가 숫자이면 참을 반환한다.

ZEROP:

* 전달된 인자가 0일때 참을 반환하는 함수

1.우선 인자로 들어온 x를 token과 나머지 부분을 나눈다.

2.전달된 인자를 zerop\_procedure에서 숫자와 프로그램 내 저장되어 있는 값인 지 판별하고, 그 값이 0인 지 판별한다.

3.인자가 숫자이면서 0이면 참을 반환하고, 숫자가 아니라면 ERROR를 출력한다.

MINUSP:

* 전달된 인자가 음수일 때 참을 반환하는 함수

1.우선 인자로 들어온 x를 token과 나머지 부분을 나눈다.

2.전달된 인자를 eval과 numberp\_procedure에 거쳐 숫자인 지 아닌 지 판별하고

반환된 숫자의 부호를 판별한다.

3.만약 전달된 인자가 숫자가 아니라면 ERROR를 출력한다.

EQUAL:

* 전달된 인자들이 같으면 참을 반환하는 함수

1.우선 인자로 들어온 x를 token과 나머지 부분을 나눈다.

2.두 인자를 eval에 거치게 하여 같은 형태로 만들어 준 다음, 그 둘의 동치를 판별한다.

3.만약 두 인자의 자료형이 다르거나 인자가 3개 이상 입력됐다면 ERROR를 출력한다.

<,>=:

* 1.우선 인자로 들어온 x를 token과 나머지 부분을 나눈다.

2.두 인자를 eval에 거치게 하여 같은 형태로 만들어 준 다음, 그 둘의 대소를 판별한다.

3.만약 두 인자의 자료형이 다르거나 인자가 3개 이상 입력됐다면 ERROR를 출력한다.

STRINGP:

* 1.우선 인자로 들어온 x를 token과 나머지 부분을 나눈다.

2.인자를 eval함수에 거치고 string형인 지 판별한다.

3.만약 인자가 2개 이상 입력되었다면 ERROR를 출력한다.

printlist 함수:

* 리스트를 (1 2 3)과 같은 방식으로 print 해주기 위한 함수

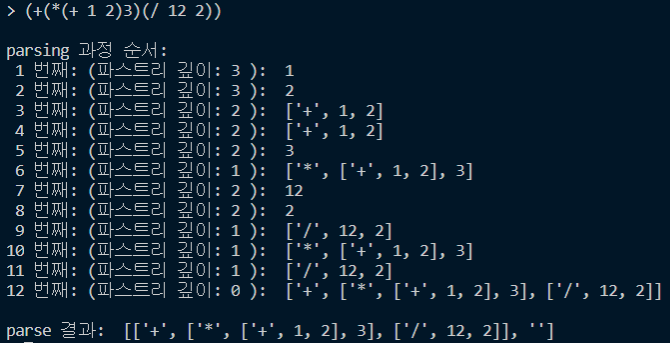
1. 전달받은 배열의 첫 요소가 “ ‘ “ 이고, 그 다음 요소가 리스트의 형태인지 확인해준다.
2. 맞다면, tmp 스트링을 만들어주고 “(“로 초기화한다.
3. for문을 진행하는데, 0으로 초기화된 index의 값을 1씩 증가시키며 for문을 진행한다.
4. 해당 요소가 리스트가 아닌 경우엔, tmp 에 해당 값을 이어준다. 이 때, 띄어쓰기를 위해 index가 1인지 판별하고, 스트링형인지 판별해 스트링형이 아니라면 스트링형으로 바꿔 tmp에 이어준다.
5. 리스트인 경우엔, 다시 printlist(l)함수를 불러온다.**(재귀함수)**
6. 해당 for문을 빠져 나오면 tmp에 “)”를 이어주고, tmp를 return한다.

isList 함수:

list\_procedure(\*args):

* LIST 함수 처리를 위한 함수

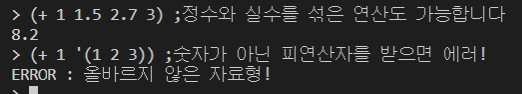
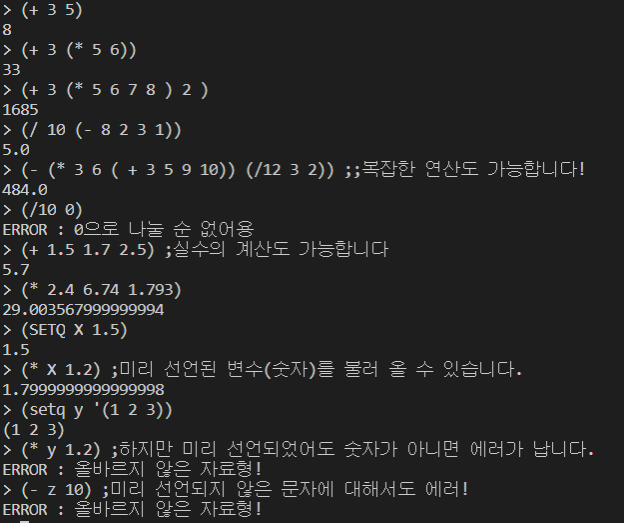
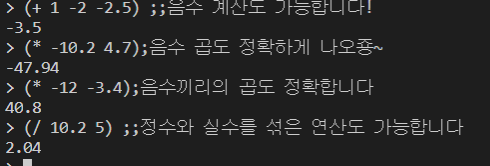
1. 인자의 개수가 0개이면 ERROR
2. 리스트의 첫 요소를 “ ‘ “ 로 하기 위해 임시 리스트 T = [“‘“] 선언한다.
3. 빈 리스트 L 를 선언해 준다.
4. for문을 통해 args들을 차례로 받아오고, 이를 eval함수에 돌렸을때 None이면 ERROR 처리를 해준다. 에러가 나지 않은 값에 대해서 L에 append 해준다.
5. T에 L을 요소로 추가하고, T를 return 해준다.

**LISP의 파싱  
*저희의 파서는 left-most derivation의 과정 순서도 보여줍니다.   
해당 기능은 interpreter.py가 아닌 parser1.py에서 실행해야합니다.***

**주석**  

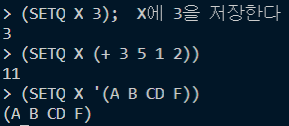
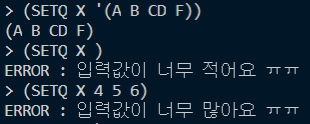


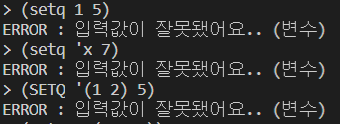

**LISP의 산술 연산**

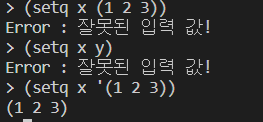
**사칙 연산(+, -, \*, / )**  


**2.LISP의 기본 함수**

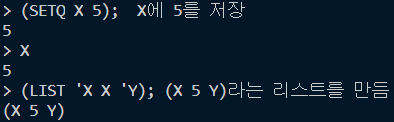
**SETQ :   
*심볼에 값을 저장(Binding)***

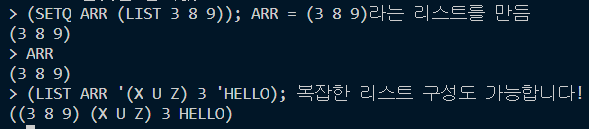
  






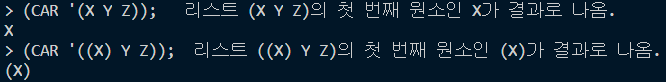
**LIST :   
*원소들을 모아서 새로운 리스트 구조를 생성해 내는 함수이다.  `를 안붙이면 그 값을 계산해서 리스트에 넣는다.***





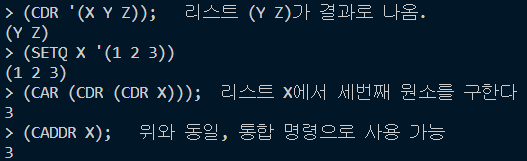


**CAR :   
*리스트의 첫번째 원소를 가져온다.***

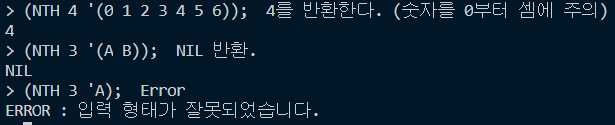


**CDR :   
*리스트의 첫번째 원소를 제외한 나머지를 결과로 생성한다.***

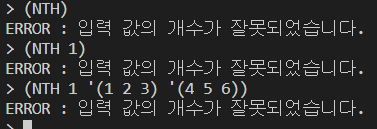
**CAR과 CDR을 혼합 사용할 수 있다**



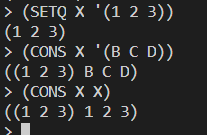
**NTH :  
*N번째 원소를 반환한다. 두번째 argument가 리스트가 아니면 에러***

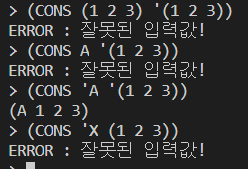


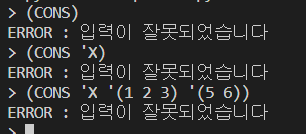


****

**CONS :  
*기존의 리스트에 새로운 원소를 추가하여 리스트를 만듬. 새로 추가되는 원소는 리스트의 첫번째 원소가 된다.***

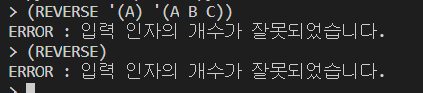




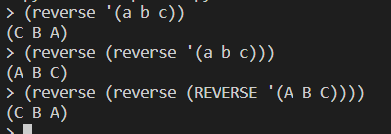


**REVERSE :  
*주어진 리스트 안의 원소의 순서를 거꾸로 바꾼다.***







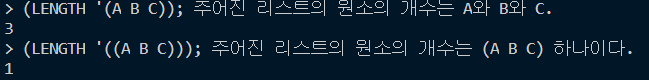


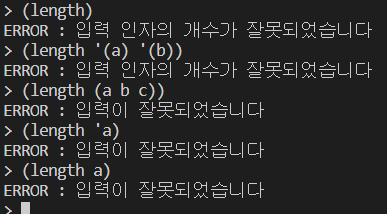
**APPEND :   
*주어진 여러개의 리스트들을 하나의 리스트로 만든다.***

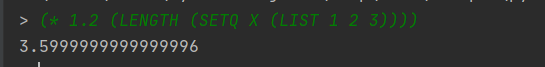


**LENGTH :   
*주어진 리스트 내의 원소 개수를 값으로 반환한다.***

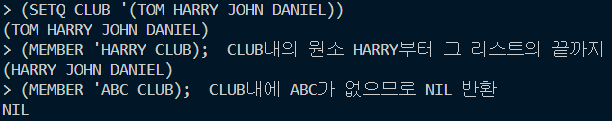


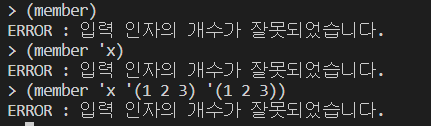




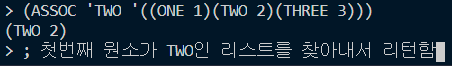
****

**MEMBER :   
*주어진 리스트 내에 어떤 원소가 있는지 확인하는 함수이다. 만약 찾고자 하는 원소가 주어진 리스트 내에 존재하면 그 원소부터 리스트 끝까지가 결과 값으로 반환된다. 리스트 내에 찾고자 하는 원소가 없을 경우 NIL이 반환된다.***

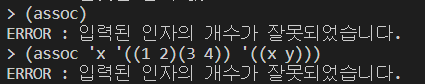


****

**ASSOC :   
*리스트를 원소로 갖는 리스트에서 원소 리스트의 첫번째 원소를 데이터베이스에서의 KEY처럼 사용하여 원하는 리스트를 찾을 수 있는 함수이다. 작은 데이터베이스 구축에 용이하게 쓸 수 있다.***

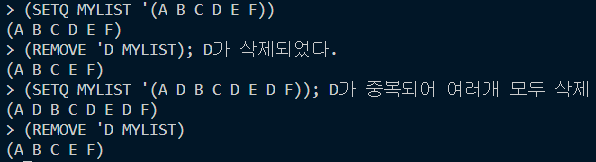


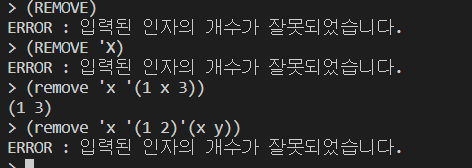


****

**REMOVE :   
*첫 번째 인자를 두 번째 인자로 받는 리스트에서 찾아 모두 제거하는 함수이다.***

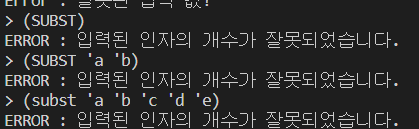
***제거한 결과를 보여주기만 할 뿐, 그 결과값으로 리스트를 새로 저장하지는 않는다.***





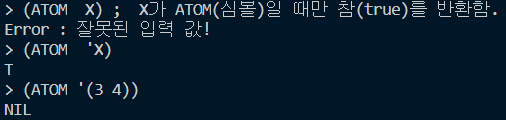
**SUBST :   
*세 번째 인자에서 두 번째 인자를 찾아 첫 번째 인자로 대치한다.***



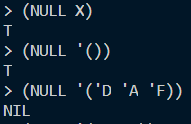


**LISP의 Predicate 함수**

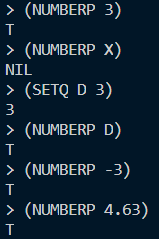
**> (ATOM  X) ;  X가 ATOM(심볼)일 때만 참(true)를 반환함.**

****

**> (NULL X) ;  X가 NIL일 때만 참(true)을 반환함.**

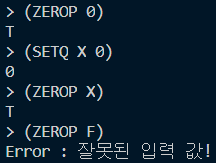
****

**> (NUMBERP X) ;  X가 숫자일 때만 참(true)을 반환함.**

****

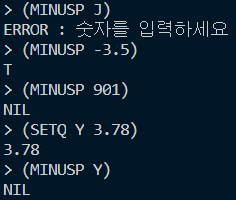
**> (ZEROP X) ;  X가 0일 때만 참(true)을 반환함.**

**X가 숫자가 아니면 ERROR 발생.**

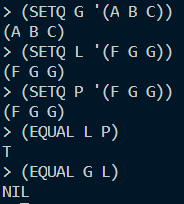
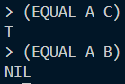
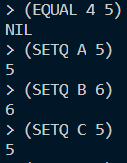
****

**> (MINUSP X) ; X가 음수일 때만 참(true)을 반환함.**

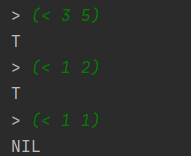
**X가 숫자가 아니면 ERROR 발생.**

****

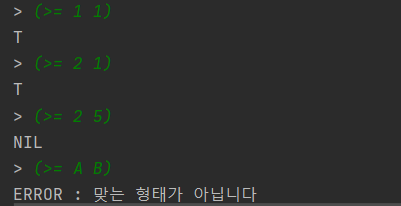
**> (EQUAL X Y) ;  X와 Y가 같으면 참(true)을 반환함.**

****

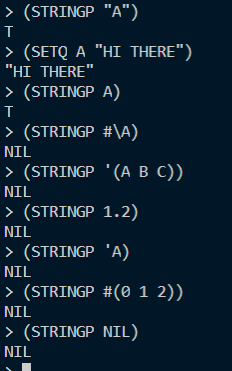
**> (< X Y) ;  X < Y 이면 참(true)을 반환함.**

****

**> (>= X Y) ;  X >= Y 이면 참(true)을 반환함.**

****

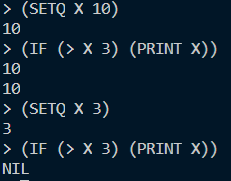
**> (STRINGP X) ;  X가 STRING일 때만 참(true)을 반환함.**

****

**LISP의 조건문**

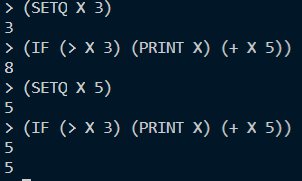
·         IF

> (IF (> X 3) (PRINT X)) ;  X값이 3보다 크면 X값을 프린트 아니면 NIL.



> (IF (> X 3) (PRINT X) (+ X 5)) ; X값이 3보다 크면 X값이 프린트되고

                                ; 그렇지 않으면 X에 5를 더하게 됨.



·         COND

 (COND (<제 1 조건문> <수행문1>)

         (<제 2 조건문> <수행문2>)

         (<제 3 조건문> <수행문3>)

