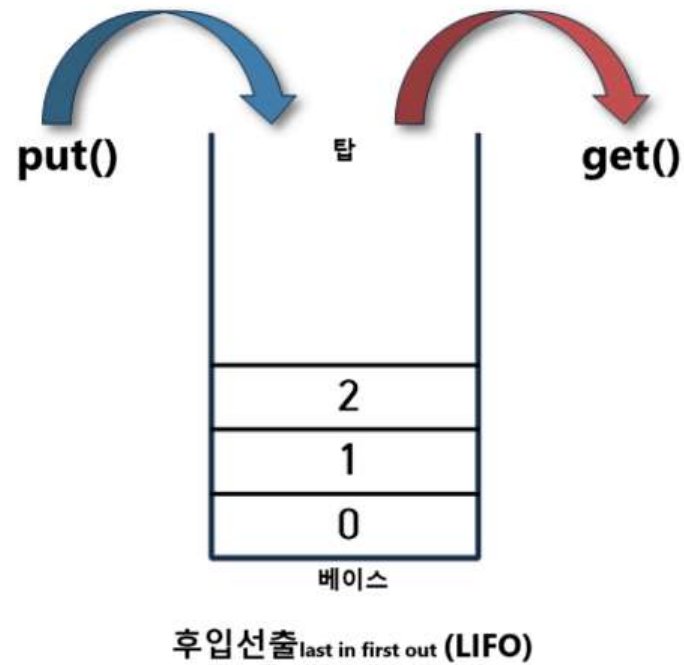


스택(Stack)

스택의 정의

- 항목의 추가 및 삭제: **탑**_{top}이라 불리는 한 쪽 끝에서만 허용
- 탑: 가장 나중에 추가된 항목의 위치
- 베이스: 남아 있는 항목 중에서 가장 먼저 추가된 항목의 위치
- **후입선출**_{last-in first-out}(LIFO) 원리 따름



스택 자료구조 구현

- 항목의 추가와 삭제를 탑이라고 부르는 곳에서만 처리
- 하지만 그 이외에는 큐와 동일한 기능 지원
- 파이썬에서 스택 자료구조로 제공되는 `queue.LifoQueue` 클래스가 큐 자료구조인 `queue.Queue`와 동일한 이름의 메서드를 제공
- `Queue` 추상 자료형을 구상 클래스로 상속하는 방식으로 스택 자료구조를 선언

```
In [1]: class Queue:
    def __init__(self, maxsize=0):
        raise NotImplementedError(
            """아래 두 변수 구현 필요
            self._maxsize=maxsize
            self._container=비어 있는 모음 객체. 항목 저장.
            """)

    def qsize(self):
        return len(self._container)

    def empty(self):
        return not self._container

    def full(self):
        if self._maxsize <= 0:
            return False
        elif self.qsize() < self._maxsize:
            return False
        else:
            return True

    def put(self, item):
        raise NotImplementedError

    def get(self):
        raise NotImplementedError
```

```
In [2]: class Stack(Queue):
        def __init__(self, maxsize=0):
            """
            - 새로운 스택 생성
            - _maxsize: 최대 항목 수. 0은 무한대 의미.
            - _container: 항목 저장 장치. list 활용
            """
            self._maxsize = maxsize
            self._container = list() # 비어있는 리스트

        def __repr__(self):
            """스택 표기법: stack([1, 2, 3]) 등등"""
            return f"stack({self._container})"

        def put(self, item):
            """
            _maxsize를 못 채웠을 경우에만 항목 추가
            """
            if not self.full():
                self._container.append(item)
            else:
                print("추가되지 않아요!")

        def get(self):
            """머리 항목 삭제 후 반환"""
            return self._container.pop()
```

In [3]: s = Stack(maxsize=4)

```
s.put(4)
s.put("dog")
s.put(True)
print(s)
s.put(8.4)
print(s.full())
print(s)
s.put("하나 더?")
print(s)
print(s.get())
print(s.get())
print(s.qsize())
print(s)
print(s.empty())
```

```
stack([4, 'dog', True])
True
stack([4, 'dog', True, 8.4])
추가되지 않아요!
stack([4, 'dog', True, 8.4])
8.4
True
2
stack([4, 'dog'])
False
```

queue 모듈의 LifoQueue 클래스

In [4]: `import queue`

put() 메서드 속도 비교

```
In [5]: %%time

n = 100_000
q1 = Stack(maxsize=0)

for k in range(n):
    q1.put(k)
```

CPU times: user 6.81 ms, sys: 0 ns, total: 6.81 ms
Wall time: 6.95 ms

```
In [6]: %%time

n = 100_000
q2 = queue.LifoQueue(maxsize=0)

for k in range(n):
    q2.put(k)
```

CPU times: user 32.7 ms, sys: 299 μ s, total: 33 ms
Wall time: 32.7 ms

get() 메서드 속도 비교

In [7]: %%time

```
for k in range(n):  
    q1.get()
```

CPU times: user 0 ns, sys: 4.53 ms, total: 4.53 ms
Wall time: 4.53 ms

In [8]: %%time

```
for k in range(n):  
    q2.get()
```

CPU times: user 34 ms, sys: 0 ns, total: 34 ms
Wall time: 33.8 ms

스택 활용: 괄호 매칭 여부 판정

괄호 종류

괄호 종류	기호	활용 예제
소괄호	(,)	튜플, 수식
중괄호	{, }	사전, 집합
대괄호	[,]	리스트

괄호 매칭 여부 판정

파이썬은 코드를 실행하기 전에 구문 검사를 진행

괄호의 짝이 맞지 않으면 바로 구문 오류(SyntaxError)를 발생시킴

```
In [9]: x = [1, 2, (5, 3), ('a':7, 'b':9)]
```

```
Cell In[9], line 1
      x = [1, 2, (5, 3), ('a':7, 'b':9)]
                        ^
SyntaxError: closing parenthesis '}' does not match opening parenthesis
'('
```

```
In [10]: x = [1, 2, (5, 3), {'a':7, 'b':9}]
```

괄호 문자열

표현식에 사용된 괄호만 고려

```
[(){}]
```

괄호가 중첩되어 사용되면 괄호 매칭 여부의 판단이 보다 어려워짐.

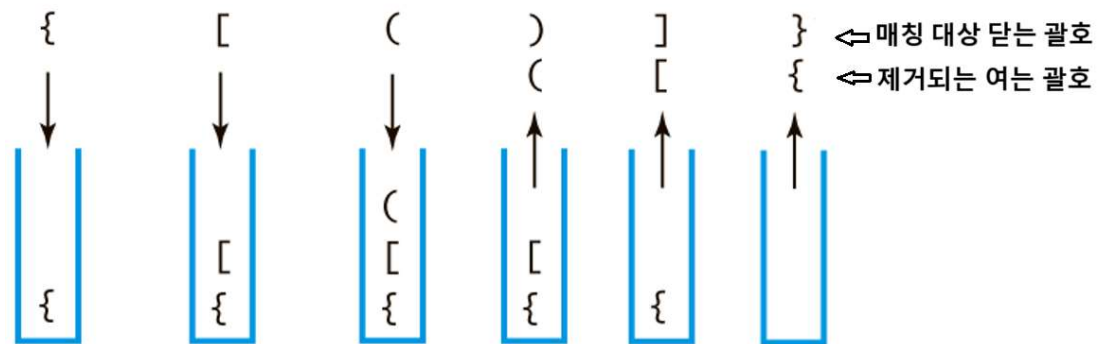
```
{{([[]])}()}  
[[{{(())}}]]  
[] [] [] () {}  
([])  
(([]))  
[{()]}
```

괄호 문자열 대상 괄호 매칭 여부 판정

스택 자료구조를 활용하여 괄호만으로 구성된 문자열에 대해 괄호 매칭 여부를 판정

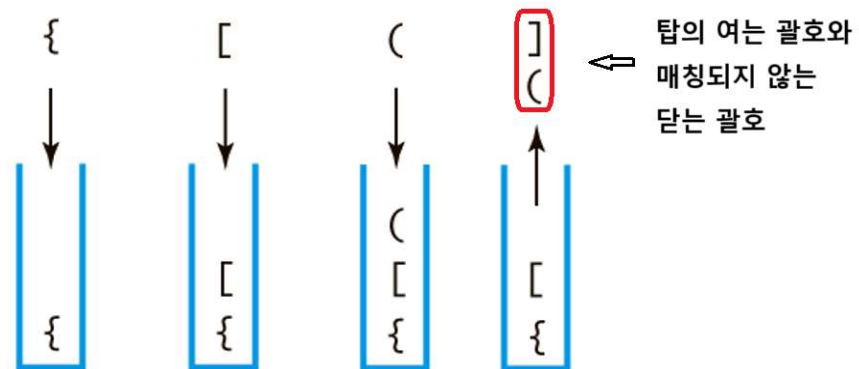
- 비어있는 스택을 하나 생성
- 괄호 문자열에 포함된 기호에 대해 아래 작업 반복
 - 여는 괄호: 스택에 추가
 - 닫는 괄호: 스택의 탑 항목 삭제

예제: `{[(())]}`에 대해 괄호 매칭 여부 판정



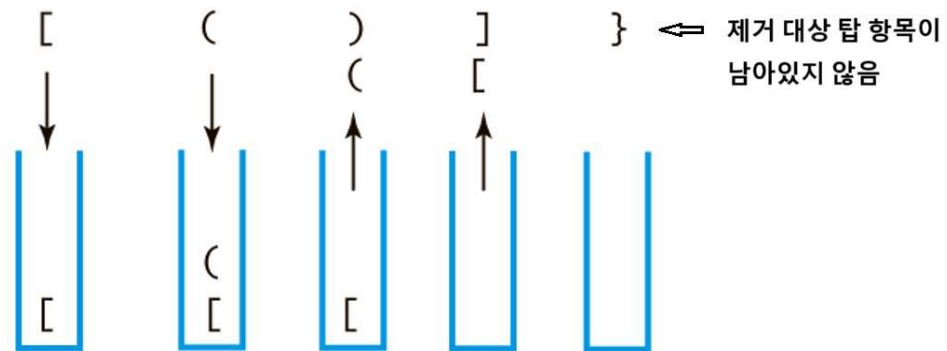
괄호 매칭이 거짓인 경우 1

`{[(())]}`의 경우: 탑에 위치한 여는 괄호와 표현식에 있는 닫는 괄호가 달라서 여는 괄호와 닫는 괄호가 매칭되지 않음



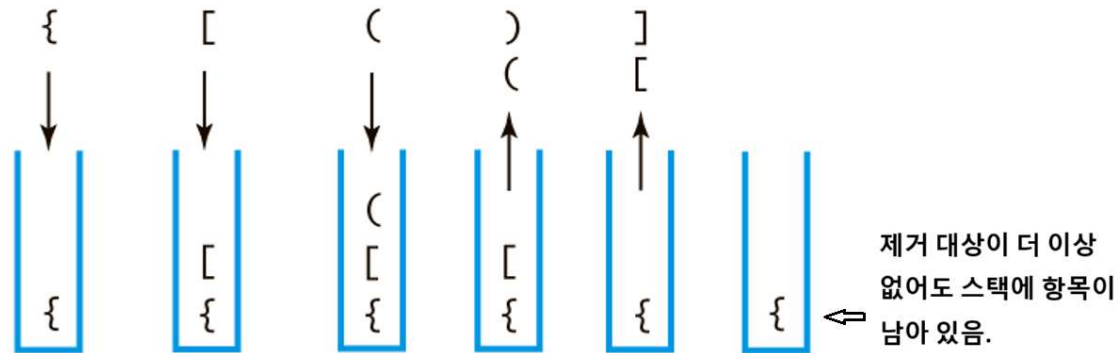
괄호 매칭이 거짓인 경우 2

`[()]]`의 경우: 아직 확인해야 할 닫는 괄호가 있지만 스택이 먼저 비워질 수 있음.



괄호 매칭이 거짓인 경우 3

{[(())]의 경우: 문자열에 포함된 모든 괄호를 확인했음에도 불구하고 스택에 항목이 남아 있을 수 있음.



두 괄호의 매칭 여부 판정

```
In [11]: def matches(par_left, par_right):
          all_lefts = "([{"
          all_rights = ")]}"

          if (par_left not in all_lefts) or (par_right not in all_rights):
              return False

          return all_lefts.index(par_left) == all_rights.index(par_right)
```

```
In [12]: print(matches('(', ')'))
          print(matches('[', ']'))
          print(matches('{', '}'))
          print(matches('(', ']'))
          print(matches('[', '}'))
          print(matches('(', '('))
```

```
True
True
True
False
False
False
```

괄호 매칭 여부 판정 함수

```
In [13]: def balance_checker(par_string):  
          s = Stack()  
  
          for symbol in par_string:  
              if symbol in "([{":  
                  s.put(symbol)  
              elif s.empty():  
                  return False  
              elif not matches(s.get(), symbol):  
                  return False  
  
          return s.empty()
```

```
In [14]: print(balance_checker('{{([][])}()}'))  
print(balance_checker('[[{({()}})]'))  
print(balance_checker('[] [] () {}'))  
print(balance_checker(' ( ) '))  
print(balance_checker(' ( ( ( ) ) ) '))  
print(balance_checker(' [ { ( ) ] '))
```

```
True  
True  
True  
False  
False  
False
```

표현식 대상 괄호 매칭 판정

표현식에 포함된 문자 중에서 괄호가 아니면 무시하도록 하는 기능 추가

```
In [15]: def balance_checker(exp_string):
          s = Stack()

          for symbol in exp_string:
              if symbol not in "()[]{}":    # 괄호가 아니면 무시
                  continue
              elif symbol in "([{":
                  s.put(symbol)
              elif s.empty():
                  return False
              elif not matches(s.get(), symbol):
                  return False

          return s.empty()
```

```
In [16]: print(balance_checker('{{{([[]])}}(())}}'))  
print(balance_checker('[{()}]'))  
print(balance_checker('(A + B) * C - (D - E) * (F + G)'))  
print(balance_checker('(5 + 6) * (7 + 8) / (4 + 3)'))  
print(balance_checker("[1, 2, (5, 3), {'a':7, 'b':9}]"))  
print(balance_checker("[1, 2, (5, 3), ('a':7, 'b':9}]"))
```

```
True  
False  
True  
False  
True  
False
```