실습 2-2-1.스택 자료구조 구현

- 스택 ADT를 참고하여 스택 자료구조를 구현해본다
- 리스트로 스택을 생성하고 각 함수를 정의한다

스택 ADT

- is_empty(STACK): 스택이 비었는지 아닌지를 검사 (returns True or False)
- push(STACK, item): 스택의 top 위치에 요소를 추가한다
- pop(STACK): 스택의 top 위치에 있는 요소를 제거한다
- top_value(STACK): 스택의 top 위치에 있는 요소를 반환한다
- get_size(STACK): 스택의 사이즈(요소의 개수)를 반환한다

In [1]:

```
def is_empty(stack):
    return bool(len(stack))
def push(stack, item):
    stack.append(item)
def pop(stack):
    return stack.pop()
def top_value(stack):
    return stack[len(stack) - 1]
def get_size(stack):
    return len(stack)
```

In [4]:

```
stack = [1, 2, 3, 4, 5]
print(is_empty(stack))
push(stack, 6)
print(stack)
pop(stack)
print(stack)
print(top_value(stack))
print(get_size(stack))
```

```
True
[1, 2, 3, 4, 5, 6]
[1, 2, 3, 4, 5]
5
5
```

실습 2-2-2. 큐 자료구조 구현

- 큐 ADT를 참고하여 큐 자료구조를 구현해본다
- 리스트로 큐를 생성하고 각 함수를 정의한다
- front와 rear를 리스트의 어느 끝으로 정의해야 할 것인가?!

큐 ADT

- is_empty(QUEUE): 큐가 비었는지 아닌지를 검사
- enqueue(QUEUE, item): 큐의 rear 위치에 요소를 추가한다
- dequeue(QUEUE): 큐의 front 위치에 있는 요소를 제거한다
- get_size(QUEUE): 스택의 사이즈(요소의 개수)를 반환한다

In [15]:

```
# front는 리스트의 끝, rear는 리스트의 시작
def is_empty(queue):
    return bool(len(queue))
def enqueue(queue, item):
    queue.append(item)
def dequeue(queue):
    temp = queue[0]
    for i in range(len(queue)-1):
        queue[i] = queue[i+1]
    queue.pop()
    return temp
def get_size(queue):
    return len(queue)
```

In [16]:

```
queue = [1, 2, 3, 4, 5]
print(is_empty(queue))
enqueue(queue, 6)
print(queue)
dequeue(queue)
print(queue)
print(get_size(queue))
```

```
True
[1, 2, 3, 4, 5, 6]
[2, 3, 4, 5, 6]
5
```

In [23]:

```
# front는 리스트의 시작, rear는 리스트의 끝
def is_empty(queue):
    return bool(len(queue))
def enqueue(queue, item):
    queue.append(None)
    for i in range(len(queue) - 1, 0, -1):
        queue[i] = queue[i-1]
    queue[0] = item
def dequeue(queue):
    return queue.pop()
def get_size(queue):
    return len(queue)
```

In [24]:

```
queue = [5, 4, 3, 2, 1]
print(is_empty(queue))
enqueue(queue, 6)
print(queue)
dequeue(queue)
print(queue)
print(get_size(queue))
```

```
True
[6, 5, 4, 3, 2, 1]
[6, 5, 4, 3, 2]
5
```

실습 2-2-3. 삽입정렬 구현

- 리스트를 입력으로 받아 정렬된 리스트를 리턴해주는 insertion_sort() 함수를 구현한다
- 삽입정렬을 구현하는 데에는 수많은 방법이 있다 – 가장 효율적인 방법을 창의적으로 생각해 본다

In [27]:

```
def insertion_sort(seq):
    for i in range(1,len(seq)):
        j = i
        while j > 0 and seq[j] < seq[j-1]:
            seq[j], seq[j-1] = seq[j-1], seq[j]
            j=j-1
    return seq
```

In [28]:

```
insertion_sort([9, 4, 8, 2, 3, 5, 1])
```

Out[28]:

```
[1, 2, 3, 4, 5, 8, 9]
```

실습 2-2-4. 거품정렬 구현

- 리스트를 입력으로 받아 정렬된 리스트를 리턴해주는 bubble_sort() 함수를 구현한다
- 거품정렬을 구현하는 데에는 수많은 방법이 있다 – 가장 효율적인 방법을 창의적으로 생각해 본다

In [32]:

```
def bubble_sort(seq):
    length = len(seq)
    for a in range(length):
        for b in range(length-1):
            if (seq[a] < seq[b]):
                seq[a], seq[b] = seq[b], seq[a]
    return seq
```

In [33]:

```
bubble_sort([9, 4, 8, 2, 3, 5, 1])
```

Out[33]:

```
[1, 2, 3, 4, 5, 8, 9]
```

실습 2-2-5. 이진탐색 구현

- 정렬된 리스트와 특정 값을 입력 받아 그 값이 리스트에서 위치한 인덱스를 반환해주는 binary_search() 함수를 구현한다
- 그 값이 리스트에 없을 경우 None을 반환한다

In [1]:

```
def binary_search(alist, value):
    lower = 0
    upper = len(alist)
    while lower < upper:
        x = lower + (upper - lower) // 2
        temp = alist[x]
        if value == temp:
            return x
        elif value > temp:
            if lower == x:
                break
            lower = x
        else:
            upper = x
```

In [6]:

```
# 이진 탐색의 인풋은 정렬된 리스트여야 함
print(binary_search([1,5,8,10], 5))
print(binary_search([1,5,8,10], 0))
print(binary_search([1,2,9,10,12,15,19], 9))
print(binary_search([1,2,9,10,12,15,19], 20))
```

```
1
None
2
None
```