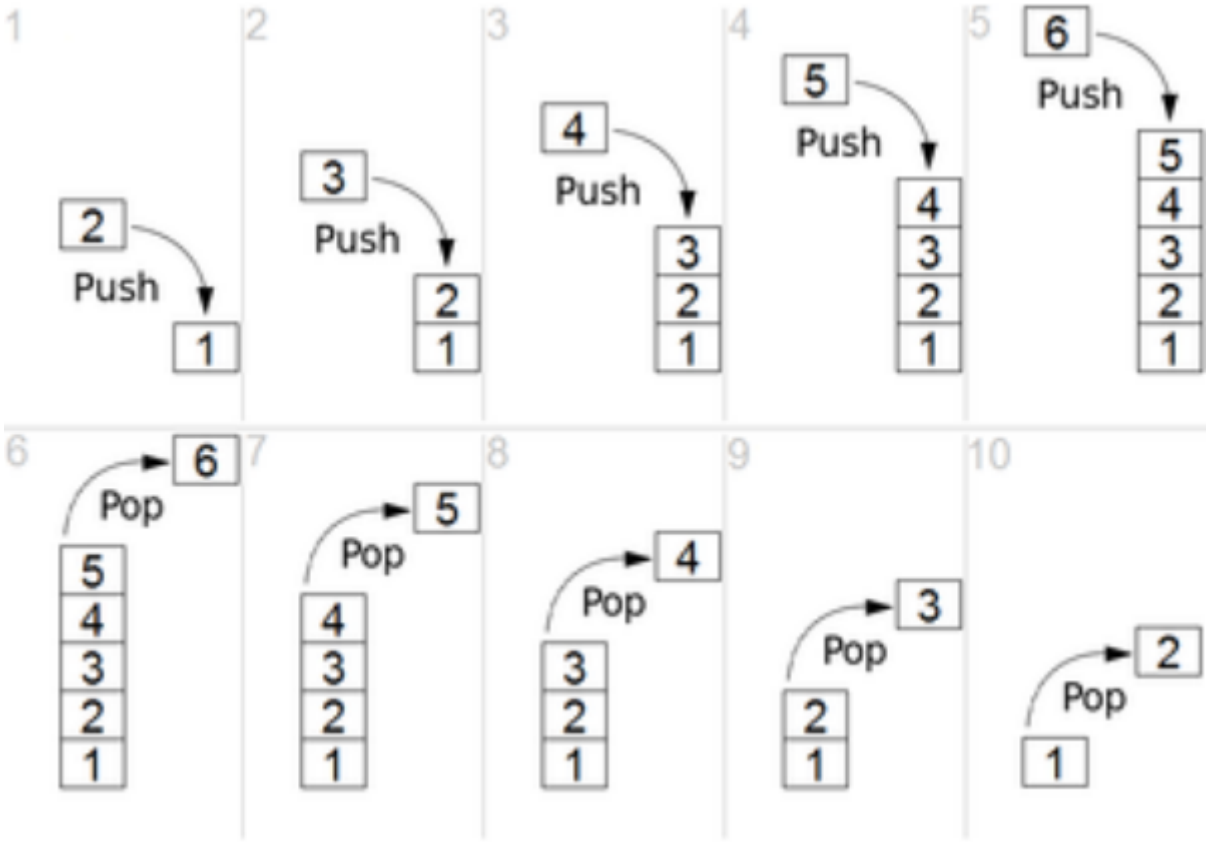
스택(Stack)

- 데이터를 제한적으로 접근할 수 있는 구조
  - 한쪽 끝에서만 자료를 넣거나 뺄 수 있는 구조
- **가장 나중에 쌓은 데이터를 가장 먼저 빼낼 수 있는 데이터 구조**
  - 큐 : FIFO 정책
  - **스택 : LIFO(Last in First Out) 정책**

1. 스택 구조

- 스택은 LIFO(Last In, First Out)또는 FILO(First In, Last Out)데이터 관리 방식을 따름
  - LIFO : 마지막에 넣은 데이터를 가장 먼저 추출하는 데이터 관리 정책
  - FILO : 처음에 넣은 데이터를 가장 마지막에 추출하는 데이터 관리 정책
- 대표적인 스택의 활용
  - **컴퓨터 내부의 프로세스 구조**의 함수 동작 방식
- 주요 기능
  - push() : 데이터를 스택에 넣기
  - pop() : 데이터를 스택에서 꺼내기



2. 스택 구조와 프로세스 스택

- 스택 구조는 프로세스 실행 구조의 가장 기본
  - 함수 호출시 프로세스 실행 구조를 스택과 비교해서 이해 필요

```
#재귀 함수
def recursive(data):
    if data < 0:
       print('ended')
    else:
       print(data)
       recursive(data-1)
       print('returned', data)
```

```
recursive(4)
>>4
  3
  2
  1
  0
  ended
  returned 0
  returned 1
  returned 2
```

```
  returned 3
  returned 4
```

3. 자료 구조 스택의 장단점

- 장점
  - 구조가 단순해서, 구현이 쉽다.
  - 데이터 저장/읽기 속도가 빠르다.
- 단점(일반적인 스택 구현시)
  - 데이터 최대 개수를 미리 정해야 한다.
    - 파이썬의 경우 재귀 함수는 1000번까지만 호출이 가능함
  - 저장 공간의 낭비가 발생할 수 있음
    - 미리 최대 개수만큼 저장공간을 확보해야 함

※ 스택은 단순하고 빠른 성능을 위해 사용되므로, **보통 배열 구조를 활용해서 구현**하는 것이 일반적임. 이 경우, 위에서 열거한 단점이 있을 수 있음

4. 파이썬 리스트 기능에서 제공하는 메서드로 스택 사용해보기

- append(push), pop 메서드 제공

```
data_stack = list()

data_stack.append(1)
data_stack.append(2)
```

```
data_stack
>>[1,2]
```

```
data_stack.pop()
>>2
```

5. 프로그래밍 연습

리스트 변수로 스택을 다루는 pop,push기능 구현해보기

```
stack_list=list()

def push(data):
  stack_list.append(data)

def pop():
  data = stack_list[-1]
  del stack_list[-1]
  return data

for i in range(10):
  push(i)

print(stack_list)
>>[0,1,2,3,4,5,6,7,8,9]

pop()
>>9
```