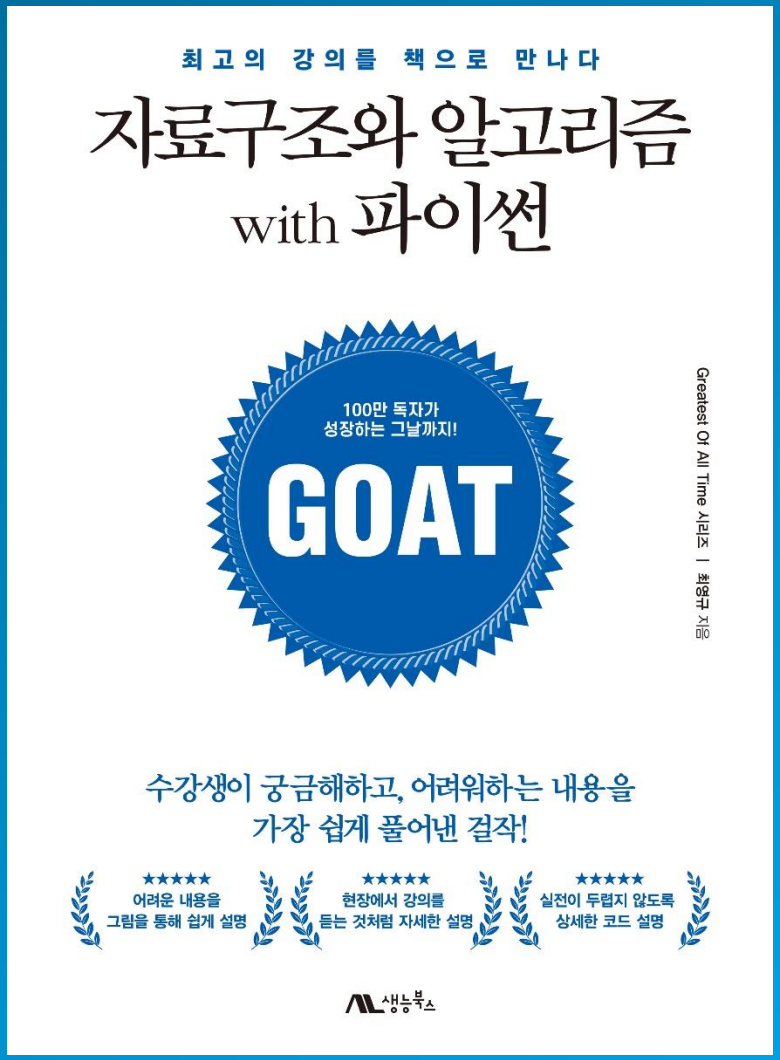
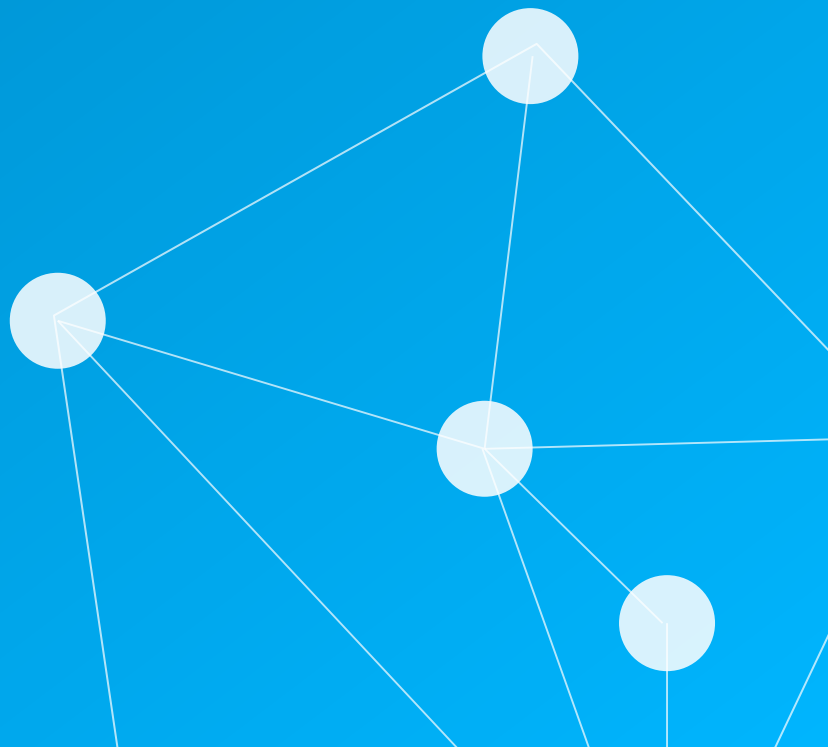
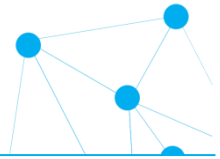

최고의 강의를 책으로 만나다
자료구조와 알고리즘
with 파이썬
100만 독자가
성장하는 그날까지!
GOAT
Greatest Of All Time 시리즈 | 최영규 지음
수강생이 궁금해하고, 어려워하는 내용을
가장 쉽게 풀어낸 걸작!
★★★★★
어려운 내용을
그림을 통해 쉽게 설명
★★★★★
현장에서 강의를
듣는 것처럼 자세한 설명
★★★★★
실전이 두렵지 않도록
상세한 코드 설명
생능북스

SW알고리즘개발



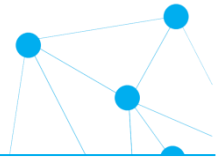
자료구조(Data Structure)



- **자료구조**
  - 데이터를 효율적으로 저장하고 처리하는 형식
  - 특정한 종류의 작업을 효율적으로 수행하기 위해 설계

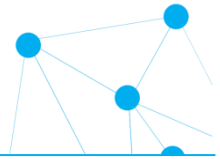
- **주요 자료구조의 종류**:
  - **배열**: 고정된 크기의 순차적인 데이터 저장
  - **연결 리스트**: 노드들이 포인터를 통해 연결된 동적 크기의 데이터 저장
  - **스택**: 후입선출(LIFO) 방식으로 데이터를 저장
  - **큐**: 선입선출(FIFO) 방식으로 데이터를 저장
  - **트리**: 각 노드가 자식 노드로 연결(계층적)을 통해 데이터저장
  - **그래프**:정점과 정점 간의 간선(관계)을 통해 데이터 저장
  - **해시 테이블**: 키-값 쌍을 저장, 해시 함수를 사용하여 검색

SW 알고리즘(Algorithm)



- **정의:** 특정한 문제를 해결하기 위해 입력 데이터를 통해 출력(문제의 해) 데이터을 반환하기 위한 단계별로 수행해야 할 일련의 규칙(절차)들의 집합
- **주요 알고리즘의 종류**:
  - **정렬 알고리즘**: 데이터를 특정 순서로 정렬.
    - Bubble Sort, Quick Sort, Merge Sort, Radix Sort, Heap Sort
  - **탐색 알고리즘**: 데이터 집합에서 특정 값을 찾기.
    - Linear Search, Binary Search, Binary Search Tree
  - **그래프 알고리즘**: 그래프 내의 노드를 탐색 또는 최단 경로 찾기.
    - DFS, BFS, Dijkstra's Algorithm, Prim
- **주요 알고리즘 설계 기법**
  - **탐욕(Greedy) 기법**: 매 단계에서 최선의 선택을 하여 문제 해결
  - **분할과 정복(D&C) 기법 :** 재귀적 호출(Recursion)
  - **동적 계획법(DP)**: 문제를 더 작은 하위 문제로 나누고, 이 하위 문제들의 결과를 저장하여 중복 계산을 피함

자료구조와 알고리즘의 관계

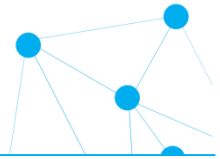


- **특정 알고리즘은 특정 자료구조와 함께 사용할 때 더 효율적으로 동작**
  - 좋은 자료구조는 특정 알고리즘의 성능을 최적화할 수 있음
  - 특정한 문제에 적합한 자료구조와 알고리즘을 선택 중요

- **예: 탐색 문제**
  - 알고리즘 선택 : 이진 탐색
  - 자료구조 선택: 정렬된 배열 구조에 데이터 저장

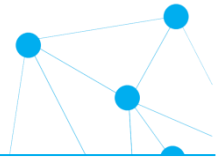
- **예: 정렬 문제**
  - 알고리즘 선택 : 병합 정렬
  - 자료구조 선택: 배열(또는 리스트)
    - 병합 정렬은 분할과 정복 기법을 사용하여 배열을 분할하여 재귀적 정렬함
    - 이 과정에서 재귀 호출을 사용하여 분할된 배열 부분을 정렬

SW 알고리즘 설계



- **문제 해결 절차**: 주어진 문제를 해결하기 위한 절차를 명확하게 정의
- **정확성(Correctness)**:
  - 알고리즘이 **모든 가능한 입력에** 대해 올바른 출력을 생성
- **유한성(Finiteness)**:
  - 모든 입력에 대해 유한한 시간 내에 종료
- **명확성(Definiteness)**:
  - 알고리즘의 각 단계의 실행 결과가 예측 가능
- **효과성(Effectiveness)**:
  - 각 단계는 기계적 실행 가능, 실행 가능한 연산으로 구성
- **단순성(Simplicity)**:
  - 간결하고 이해하기 쉬운 알고리즘은 유지보수하기 용이
- **확장성(Scalability)**:
  - **큰 입력 데이터 세트에서 얼마나 잘 수행되는지를** 평가

알고리즘 성능의 평가 방법

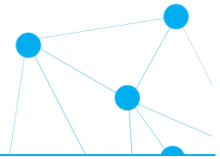


- **알고리즘 효율성**:
  - **더 적은 시간과 메모리를 사용**하여 문제를 해결

- **시간 복잡도(Time Complexity)**:
  - 알고리즘이 실행되는 데 필요한 시간의 양을 입력 크기에 따라 Big-O 표기법을 사용
  - 예: O(n), O(log n), O(n^2) 등

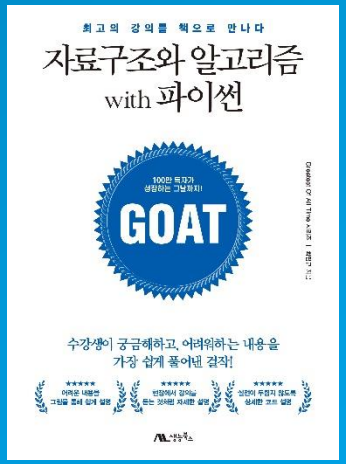
- **공간 복잡도(Space Complexity)**:
  - 알고리즘이 실행되는 데 필요한 메모리의 양을 입력 크기에 따라 Big-O 표기법으로 표현

알고리즘 구현



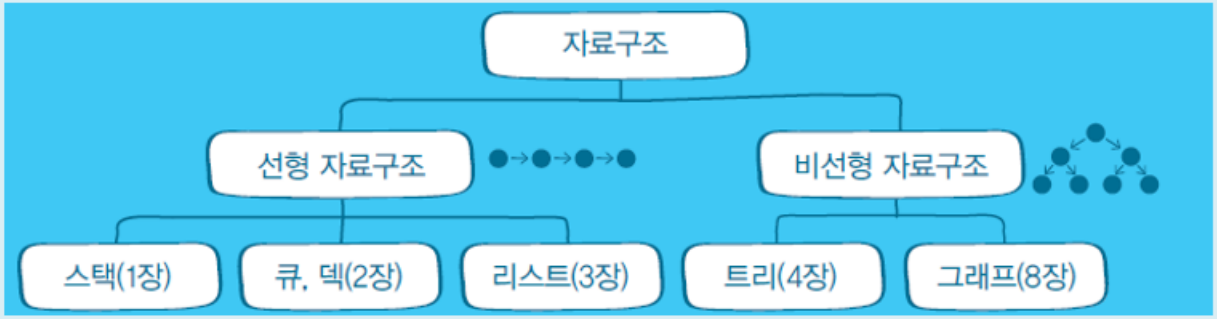
- 코드의 작성, 구조화, 유지 보수 및 성능 최적화와 같은 여러 측면을 고려하며, 특정 문제나 작업을 효율적으로 해결하기 위해 선택
  - Python, C, C++, java 등

- **구조적 프로그래밍 (Structured Programming)**
  - 코드을 순차적, 조건적, 반복적 'if', 'else', 'for', 'while 사용
  - 함수와 절차를 명확히 정의
  - 코드의 가독성, 유지 보수와 디버깅이 용이, 코드가 재사용 가능하고 모듈화
- **객체 지향 프로그래밍 (Object-Oriented Programming, OOP)**
  - 데이터와 함수를 하나의 객체로 묶어(캡슐화) 문제를 해결 기법.
  - 클래스와 객체를 통해 유지 보수 비용이 감소, 코드의 재사용성과 확장성을 높임



01 CHAPTER

스택



1장 스택 | 2장 큐 | 3장 리스트 | 4장 트리

1장. 스택

01-1 스택이란?

01-2 배열 구조로 스택 구현하기

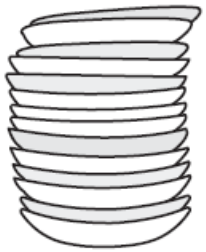
01-3 스택의 응용: 괄호 검사

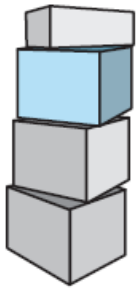
01-4 파이썬에서 스택 사용하기

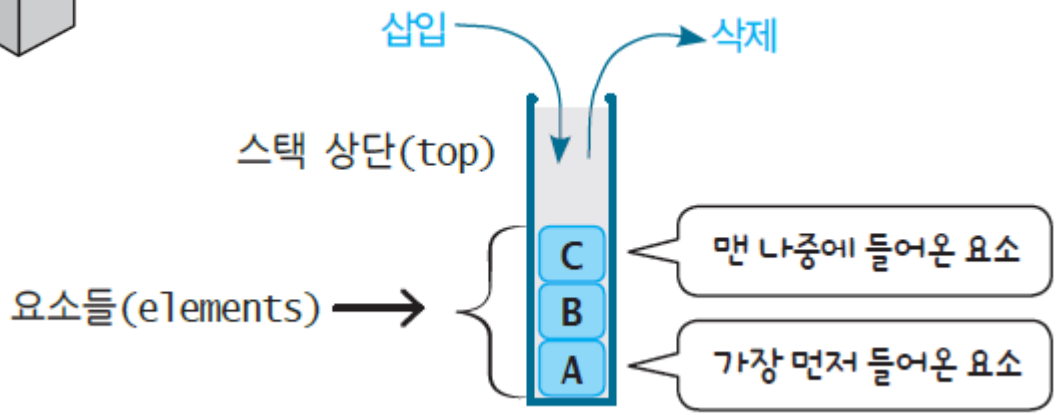
01-5 시스템 스택과 순환 호출

1.1 스택이란?

- 스택의 구조
  - 다른 통로들은 모두 막고 한쪽만을 열어둔 구조
  - 요소의 삽입과 삭제가 상단에서만 가능한 자료구조
  - 후입선출(LIFO:Last-In First-Out)의 자료구조
  - 가장 최근에 들어온 데이터가 가장 먼저 나감

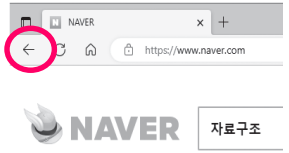




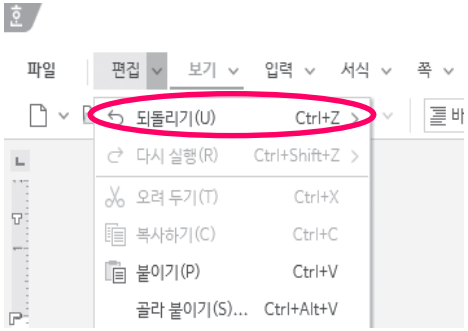


스택의 활용 예

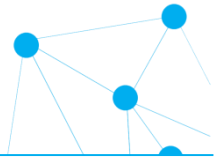
- 웹 브라우저의 이전페이지로 이동 기능



- 편집기의 되돌리기
- 괄호 검사, 계산기, 미로 탐색 등



스택의 추상 자료형

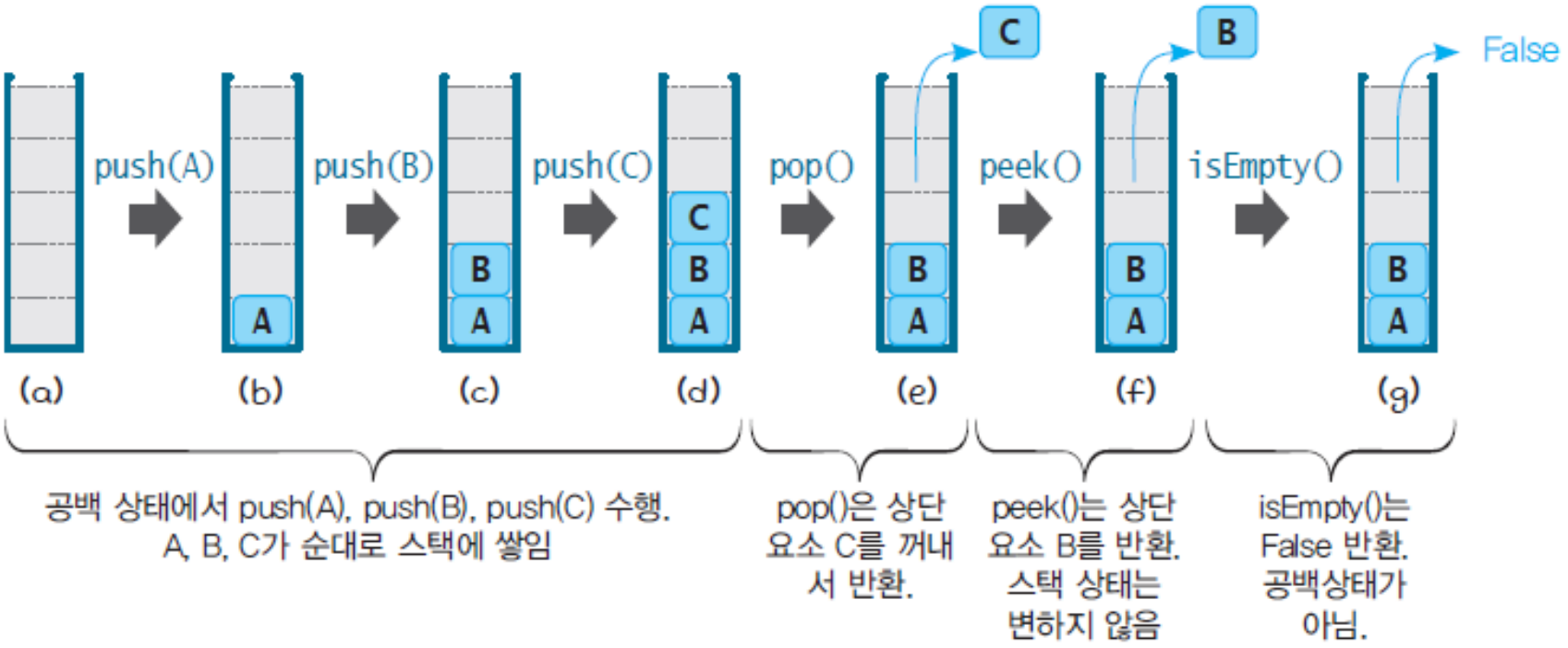


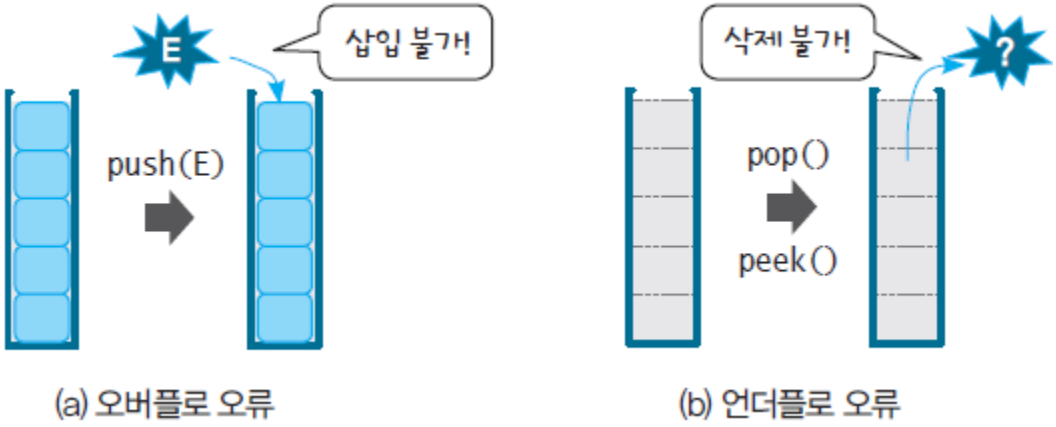
- 새로운 자료형이 필요하면?
  - 그 자료형의 자세하고 복잡한 내용 대신에 필수적이고 중요한 특징만 골라서 단순화시키는 작업이 필요
- 추상 자료형(ADT: Abstract Data Type)
  - 그 자료형이 어떤 자료를 다루고, 어떤 연산이 필요한지를 정의
- 스택이 다루는 데이터 & 연산
  - 숫자와 문자열을 포함한 어떤 자료든 저장할 수 있음

스택의 연산

- push(e) : 새로운 요소 e를 스택의 맨 위에 추가
- pop() : 스택의 맨 위에 있는 요소를 꺼내서 반환
- isEmpty() : 스택이 비어 있으면 True를 아니면 False를 반환
- isFull() : 스택이 가득 차 있으면 True를 아니면 False를 반환
- peek() : 스택의 맨 위에 있는 항목을 삭제하지 않고 반환
- size() : 스택에 들어 있는 전체 요소의 수를 반환

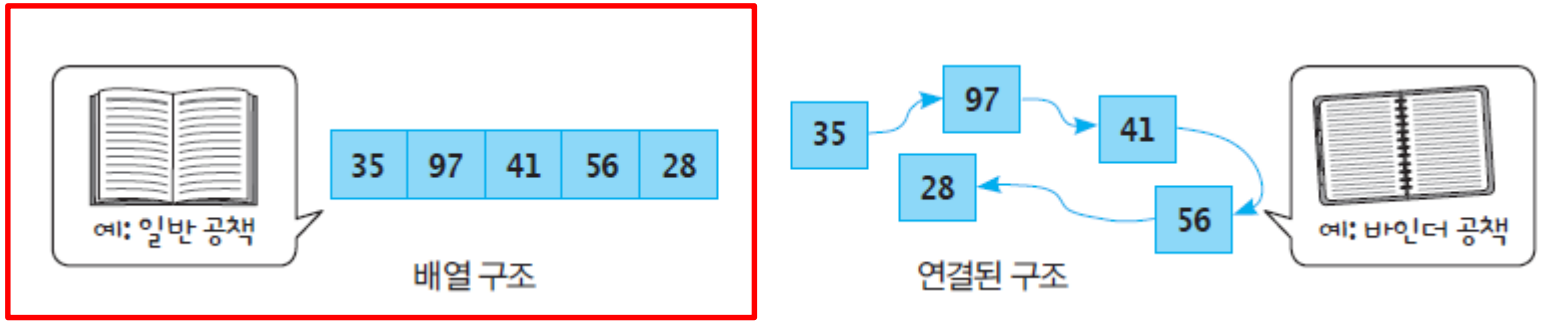
스택의 일련의 연산 예


push(A)
push(B)
push(C)
pop()
peek()
isEmpty()
C
B
False
A
B
A
C
B
A
B
A
B
A
B
A
(a)
(b)
(c)
(d)
(e)
(f)
(g)
공백 상태에서 push(A), push(B), push(C) 수행.
A, B, C가 순대로 스택에 쌓임
pop()은 상단
요소 C를 꺼내
서 반환.
peek()는 상단
요소 B를 반환.
스택 상태는
변하지 않음
isEmpty()는
False 반환.
공백상태가
아님.


E
삽입 불가!
push(E)
삭제 불가!
?
pop()
peek()
(a) 오버플로 오류
(b) 언더플로 오류

1.2 배열 구조로 스택 구현하기

- 스택은 배열 구조와 연결된 구조로 구현할 수 있음



- 배열 구조
  - 자료를 배열에 모아 저장
  - (예) 일반 공책: 편리하지만 공책이 가득 차면 저장 불가
- 연결된 구조
  - 요소들을 연결하여 하나로 관리
  - (예) 바인더 공책: 페이지의 위치를 바꾸거나 페이지를 쉽게 추가, 삭제 가능. 관리하기 복잡.

배열 구조의 스택을 위한 데이터

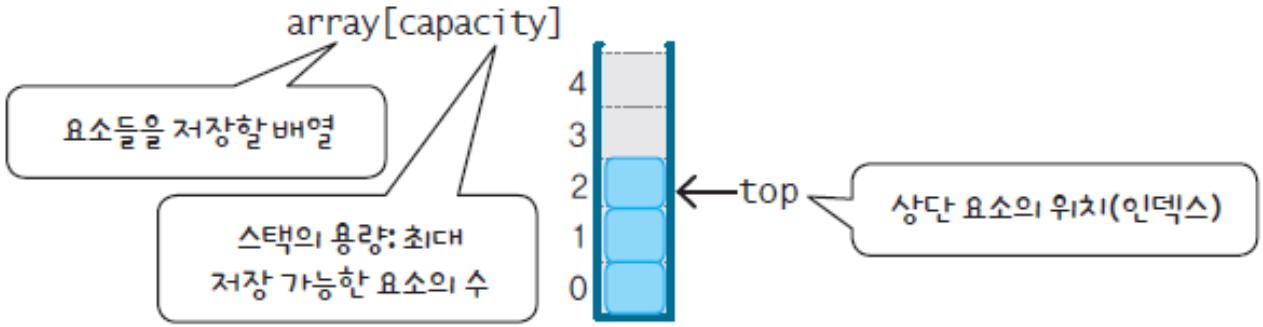


그림 1.7 | 배열을 이용한 스택의 구조

- array[] : 스택 요소들을 저장할 배열
- capacity : 스택의 최대 용량. 저장할 수 있는 요소의 최대 개수(상수)
- top : 상단 요소의 위치(변수, 인덱스)

- 스택(고정된 용량)의 구현 방법
  (1) 전역 변수와 함수로 구현
  (2) 클래스로 구현

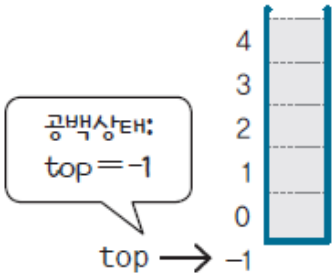
(1) 전역 변수와 함수로 구현

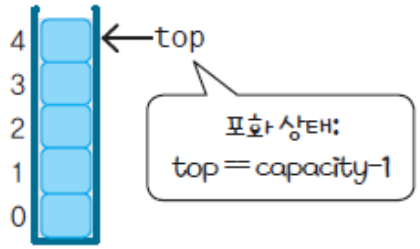
- 데이터 → 전역 변수
- 연산 → 일반 함수로 구현
- 용량이 고정된 스택의 데이터

```
capacity = 10               # 스택 용량: 예) 용량을 10으로 지정
array = [None]*capacity     # 요소 배열: [None, .., None] (길이가 capacity)
top = -1                    # 상단의 인덱스: 공백 상태(-1)로 초기화함
```

- 공백 상태와 포화 상태를 검사하는 isEmpty( )와 isFull( )

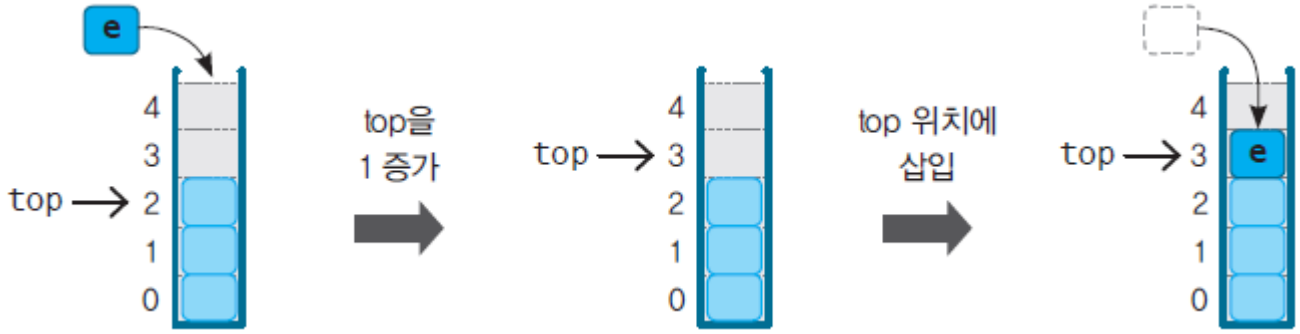
```
def isEmpty( ) :
    if top == -1 : return True
    else : return False

def isFull( ) : return top == capacity
```





삽입 연산: push(e)

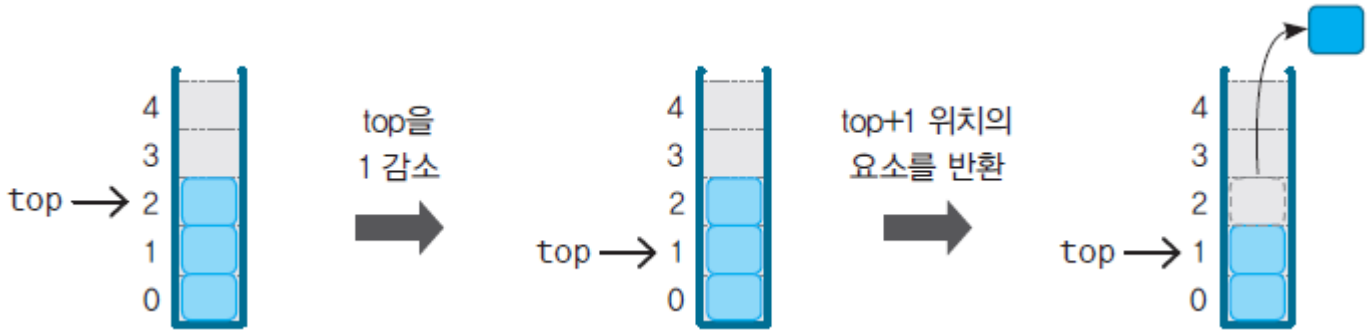
- 삽입 과정




```
def push( e ) :
    # global top
    if not isFull() :              # 포화 상태가 아닌 경우
        top += 1                   # top 증가(global top 선언 필요)
        array[top] = e             # top 위치에 e 복사
    else :                         # 포화 상태: overflow
        print("stack overflow")
        exit()
```

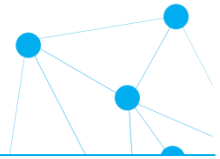
삭제 연산: pop( )

- 삭제 과정




```
def pop( ) :
    # global   top
    if not isEmpty():                # 공백 상태가 아닌 경우
        top -= 1                     # top 감소(global top 선언 필요)
        return array[top+1]          # 이전(top+1) 위치의 요소 반환
    else:                            # 공백 상태: underflow
        print("stack underflow")
        exit()
```

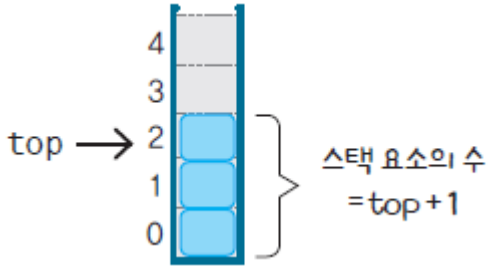
기타 연산들



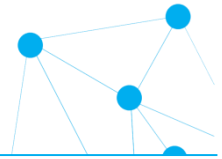
- 상단 요소를 들여다보는 peek( ) 연산

```
def peek( ) :
    if not isEmpty():              # 공백 상태가 아닌 경우
        return array[top]
    else: pass                     # underflow 예외는 처리하지 않았음
```

- 현재 스택 요소의 수를 반환하는 size( ) 연산



(2) 배열 구조 스택의 클래스 구현



- 전역 변수와 함수의 클래스 변환
  - 전역 변수(데이터) → 클래스의 멤버 변수 : 생성자에서 처리
  - 함수(연산) → 클래스의 멤버 함수(메서드)

클래스 이름

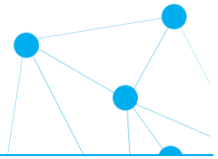
스택의 생성자

클래스를 만드는 예약어

```
class ArrayStack :
    def __init__( self, capacity ):
        self.capacity = capacity                # 스택 용량
        self.array = [None]*self.capacity       # 요소 배열
        self.top = -1                           # 상단의 인덱스
```

- 주의할 점
  - 파이썬에서 들어쓰기가 매우 중요
  - 생성자: __init__ ()
  - 멤버 함수의 첫 번째 매개변수 self

1.4 파이썬에서 스택 사용하기



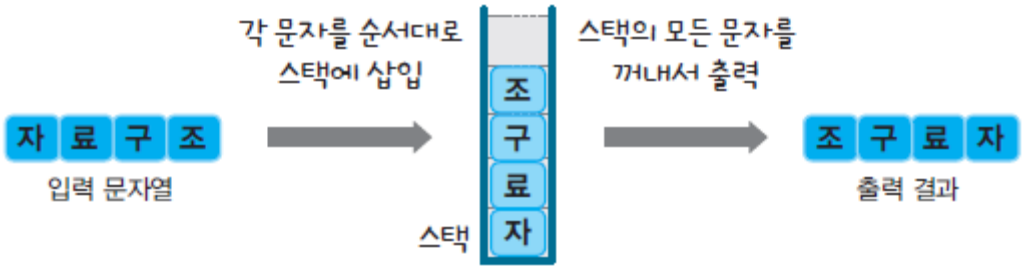
- 방법 1) Stack 클래스를 구현해서 사용하기
- 방법 2) 파이썬 리스트를 스택으로 사용하기
- 방법 3) queue 모듈의 LifoQueue 사용하기
  - import queue # 파이썬의 큐 모듈 포함
  - s = queue.LifoQueue(maxsize=20) # 스택 객체 생성(크기 20)

연산	ArrayStack	파이썬 list	사용 예
삽입	push( )	메서드 append( ) 사용	L.append(e)
삭제	pop( )	메서드 pop( ) 사용	L.pop( )
요소의 수	size( )	내장 함수 len( )을 사용	len(L)
공백 상태 검사	isEmpty( )	요소의 수가 0인지 검사	len(L)==0
포화 상태 검사	isFull( )	list는 용량이 무한대라 포화 상태는 의미가 없음. 항상 False	False
상단 들여다보기	peek( )	맨 마지막 요소 참조	L[len(L)-1] 또는 L[-1]

연산	ArrayStack	queue.LifoQueue
삽입/삭제	push( ), pop( )	put( ), get( )
공백/포화 상태 검사	isEmpty( ), isFull( )	empty( ), full( )
상단 들여다보기	peek( )	제공하지 않음

스택의 클래스 사용 예

- 말을 거꾸로 뒤집는 프로그램




```
s = ArrayStack(100)
```

새로운 스택 객체 생성. ArrayStack의 매개변수 capacity에 100이 전달되어 용량이 100인 스택 객체를 만듦

```
msg = input("문자열 입력: ")
for c in msg :
    s.push(c)
```

문자열을 입력받아 msg에 저장하고, msg의 각 문자 c를 순서대로 스택에 삽입

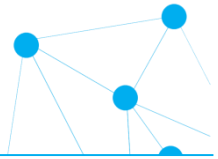
```
print("문자열 출력: ", end='')
while not s.isEmpty():
    print(s.pop(), end='')
print()
```

스택이 공백이 아니면 상단의 문자를 꺼내서 화면에 출력. 이 과정을 공백 상태가 될 때까지 반복

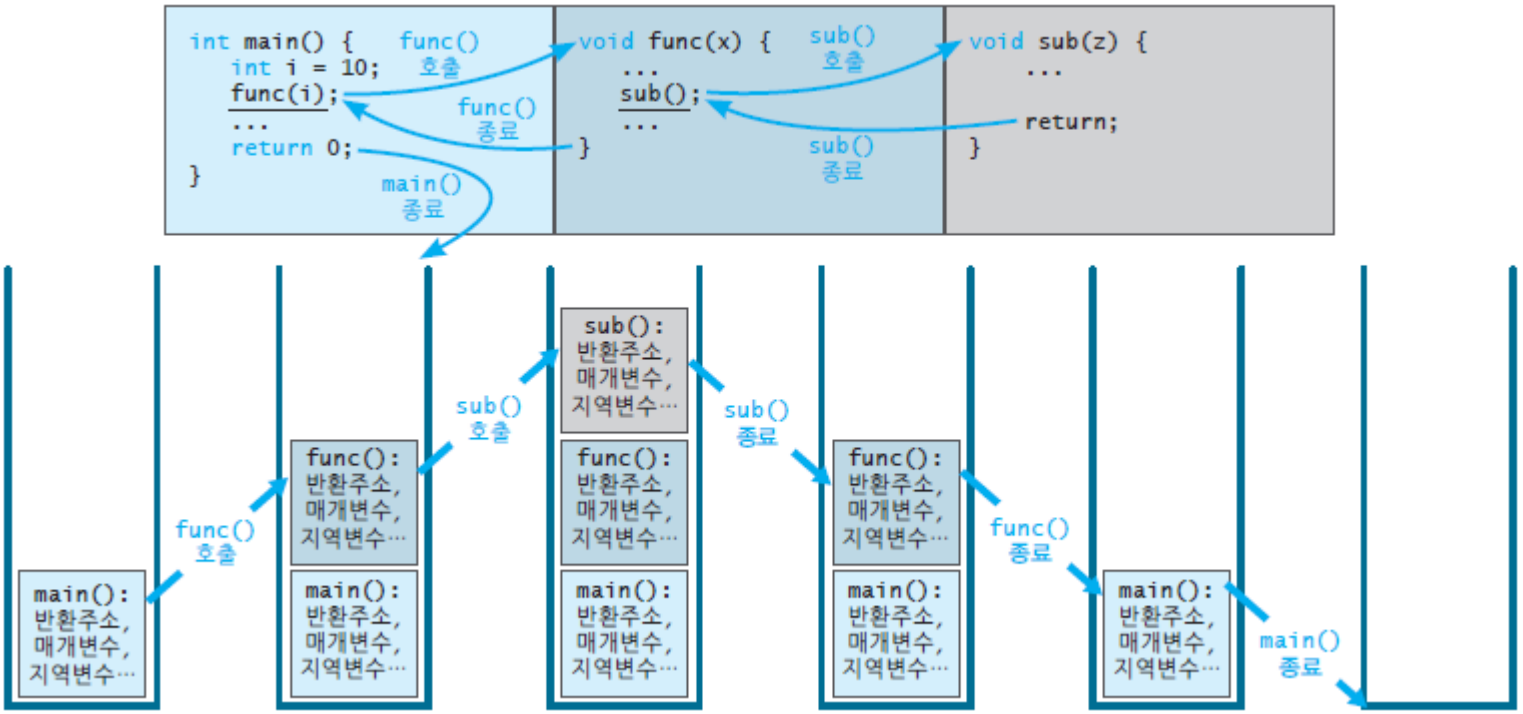
실행 결과

```
문자열 입력: 안녕하세요. 반갑습니다.
문자열 출력: .다니습갑반 .요세하녕안
```

1.5 시스템 스택과 순환 호출

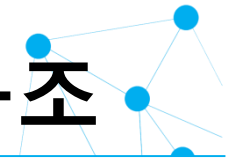


- 함수의 호출과 반환을 위해 시스템 스택 예



- 시스템 스택을 적극적으로 사용하는 프로그래밍 기법?
  - 같은 일을 되풀이하는 방법: (1) 반복, (2) 순환

프로그래밍 기법: 반복 구조 vs. 순환 구조



- 모두 어떤 작업을 반복적으로 수행하는데 사용
- 두 방식 모두 같은 결과를 반환
- 상황에 따라 적합한 방법 선택 필요
- 반복 구조
  - 'for', 'while' 과 같은 반복문을 통해 구현
  - 메모리 효율이 높고 큰 입력에 대해 성능이 좋다.
- 순환 구조
  - 함수가 자기 자신을 다시 호출하여 문제를 해결하는 프로그래밍
  - 재귀함수을 통해 구현
  - 코드가 더 간결, 재귀 깊이가 커지면 성능 저하가 발생
  - 문제 자체가 순환적인 구조 : 팩토리얼 계산, 하노이 탑 등
  - 순환적으로 정의되는 자료구조 : 이진 트리

예: n! 팩토리얼 함수

반복

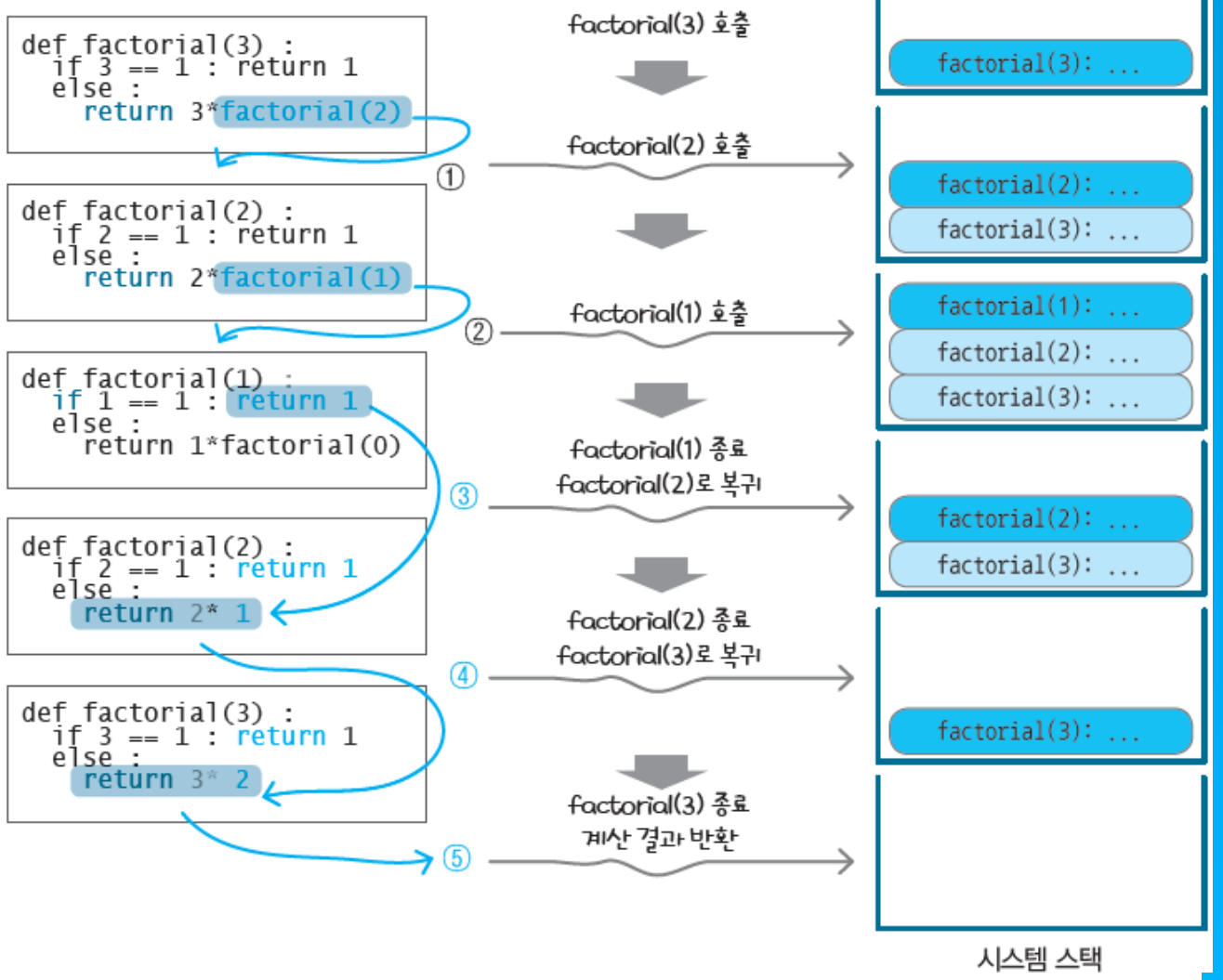
$$n! = 1 \times 2 \times 3 \times \cdots \times (n-1) \times n$$

```
def factorial_iter(n) :
    result = 1
    for k in range(2, n+1) :
        result = result * k
    return result
```

순환

$$n! = \begin{cases} 1 & n=1 \\ n \times (n-1)! & n>1 \end{cases}$$

```
def factorial(n) :
    if n == 1 :
        return 1
    else :
        return n * factorial(n-1)
```



예: 피보나치 수열(Fibonacci Sequence)

- 정의: 수학에서 매우 유명한 수열 중 하나
  - 첫 번째 항과 두 번째 항은 각각 1이다.
  - 세 번째 항부터는 바로 앞의 두 항의 합으로 이루어진다.
  - 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

이를 수식으로 표현하면, 피보나치 수열의 n번째 항 $F(n)$

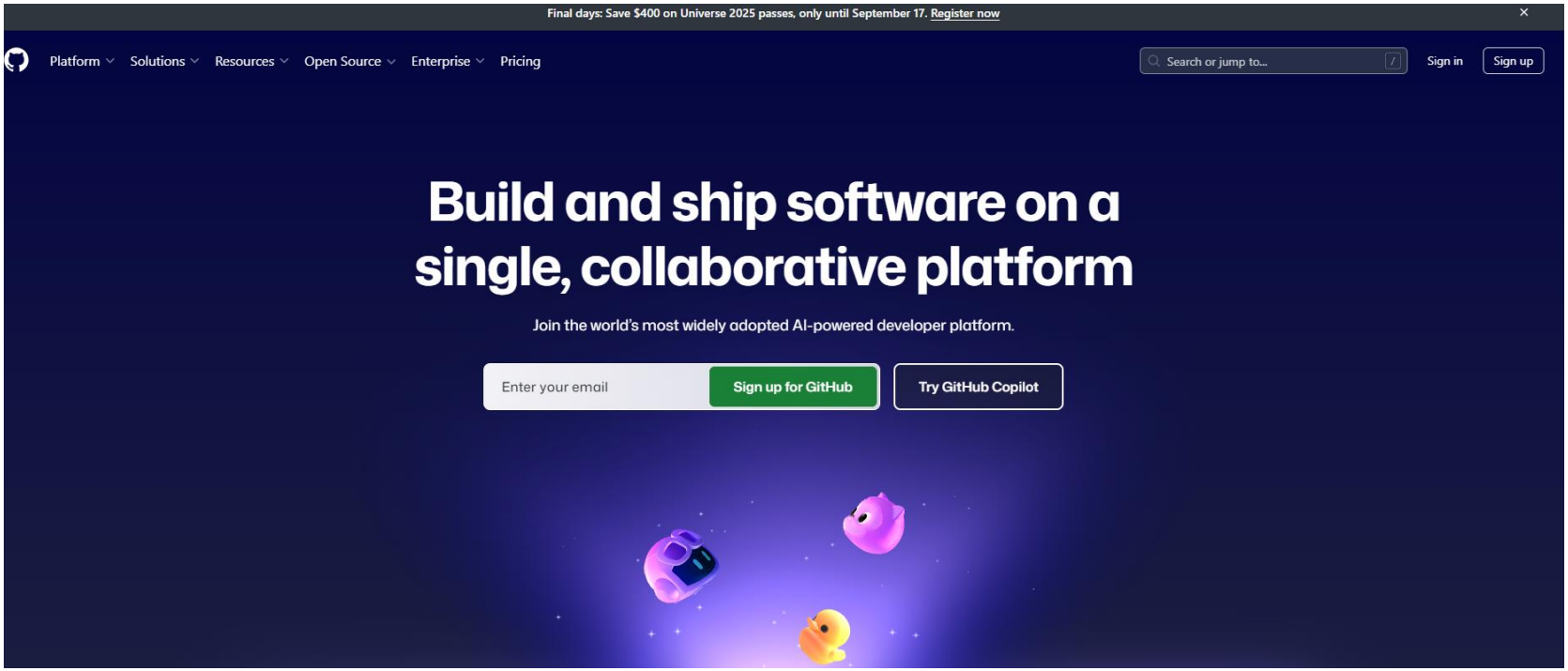
- $F(1) = 1$
- $F(2) = 1$
- $F(n) = F(n-1) + F(n-2)$ (n ≥ 3)

```python
def fibonacci_recursive(n):
    if n <= 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2)
```

```python
def fibonacci_iterative(n):
    if n <= 0:
        return 0
    elif n == 1:
        return 1

    prev, current = 0, 1
    for _ in range(2, n + 1):
        prev, current = current, prev + current
    return current
```

GitHub Desktop 초보자 가이드 (강의실 다중 사용자용)

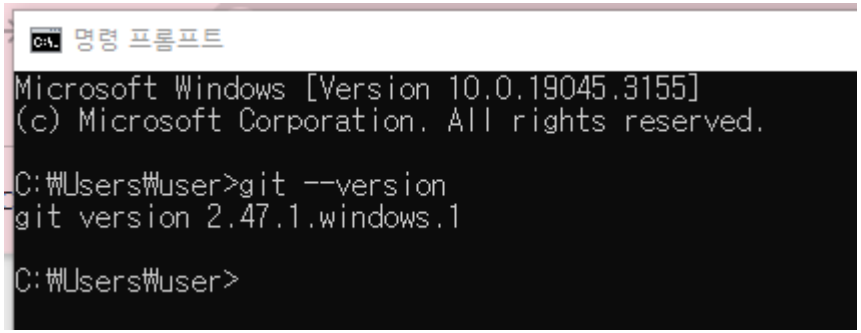
1. **GitHub 계정 만들기**
    1. GitHub 가입 페이지 접속
        1. https://github.com/?utm_source=chatgpt.com
    2. 이메일, 비밀번호, 사용자 이름 입력 후 계정 생성
    3. 이메일 인증 완료

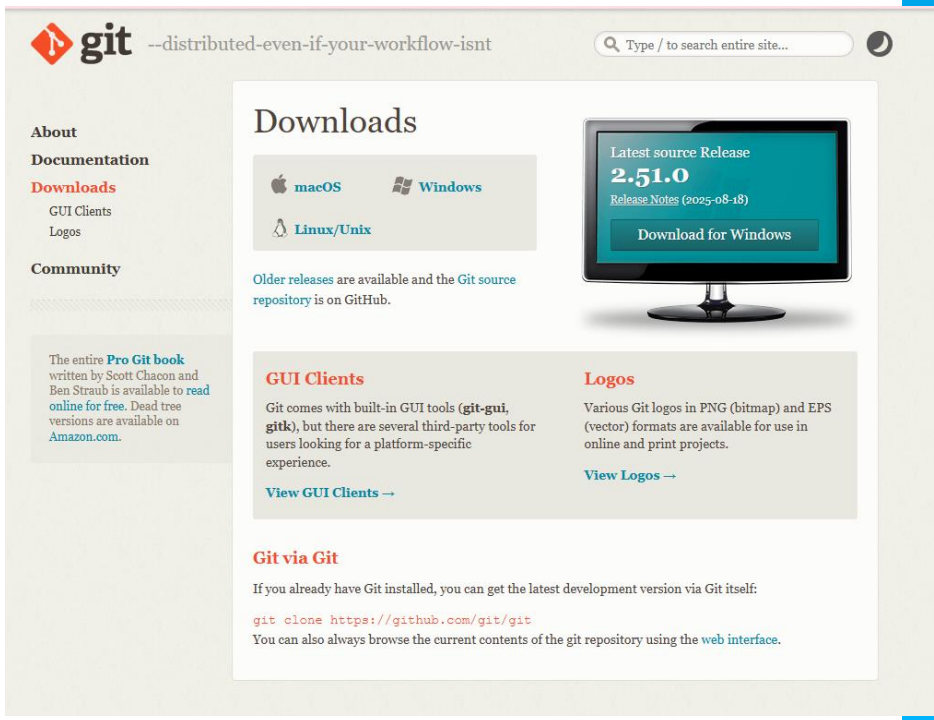


GitHub Desktop 초보자 가이드 (강의실 다중 사용자용)

2. Git 설치하기

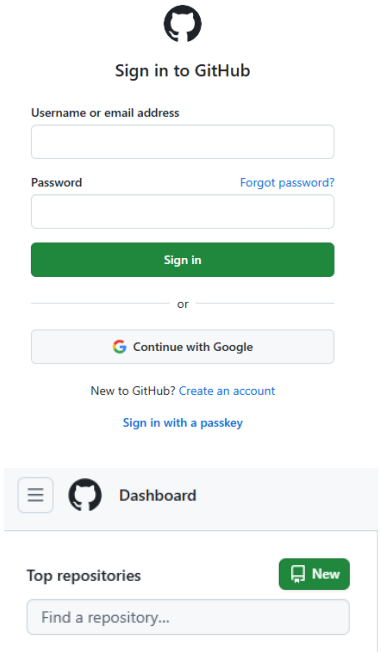
- Git 공식 다운로드 에서 운영체제에 맞는 버전 설치
  - https://git-scm.com/downloads?utm_source=chatgpt.com
- 설치 확인:

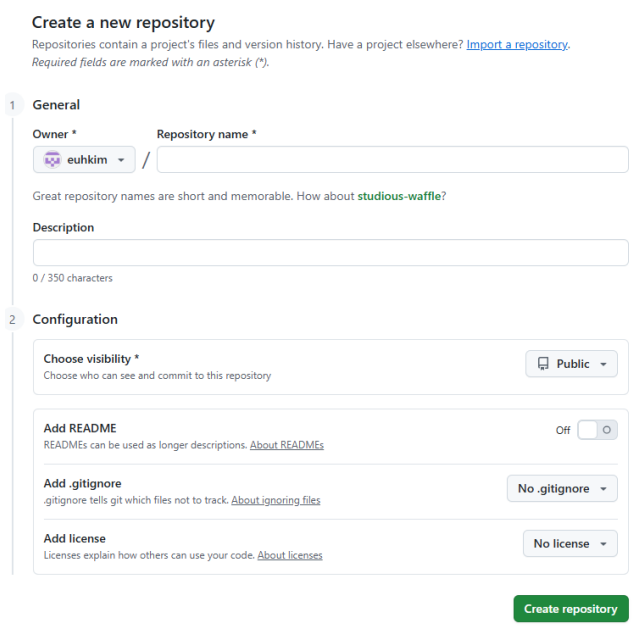


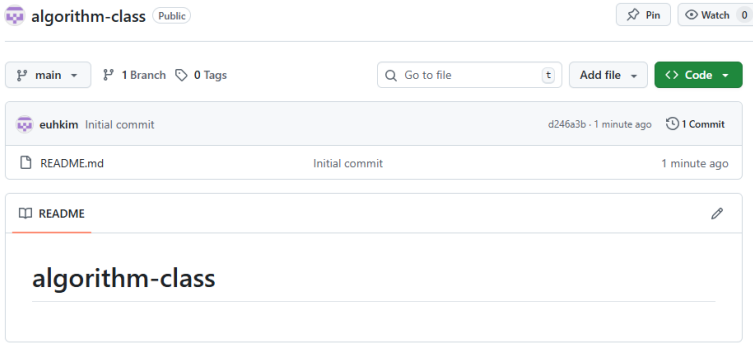


GitHub Desktop 초보자 가이드 (강의실 다중 사용자용)

3. GitHub 저장소(Repository) 만들기
   1. GitHub 로그인 → Dashboard 왼쪽 상단 + New 버튼 → New repository 선택
   2. 저장소 이름 입력 (예: algorithm-class)
   3. Public (공개) 또는 Private (비공개) 선택
   4. README.md 추가 체크 (설명 파일 자동 생성)
   5. Create repository 클릭







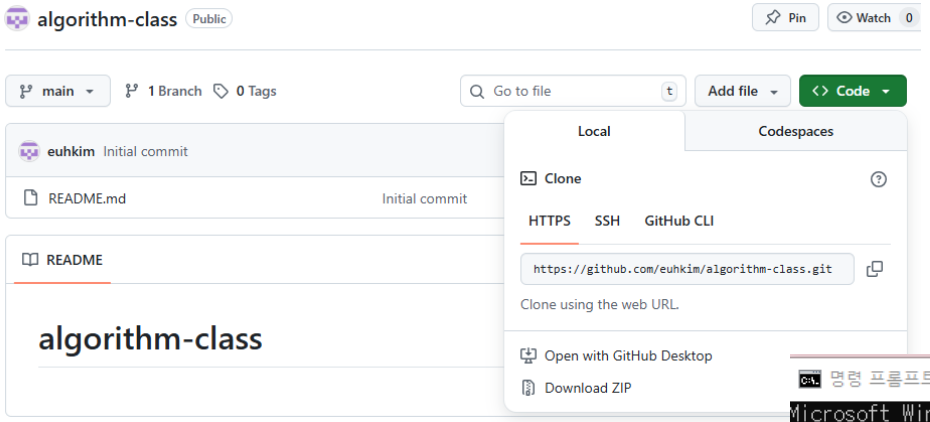
GitHub Desktop 초보자 가이드 (강의실 다중 사용자용)

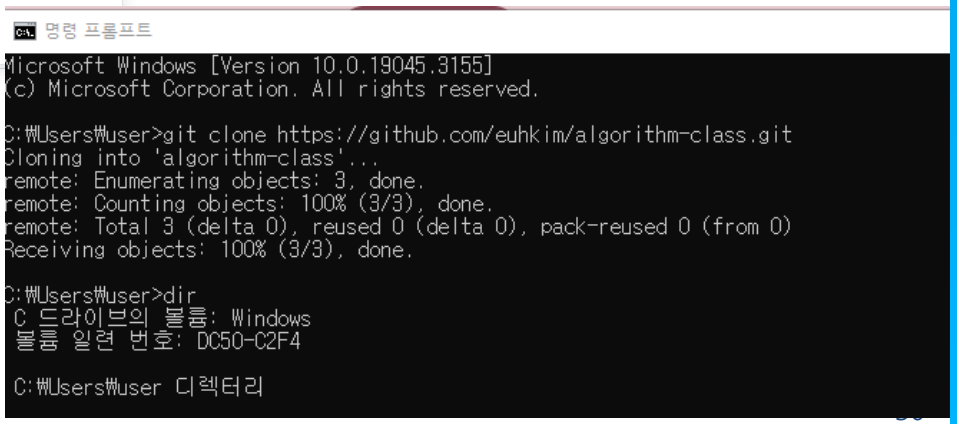
4. 로컬 PC에 프로젝트 연결 (Clone)

– GitHub에 만든 저장소를 내 PC로 복사(clone)

– **터미널에서 명령 실행**

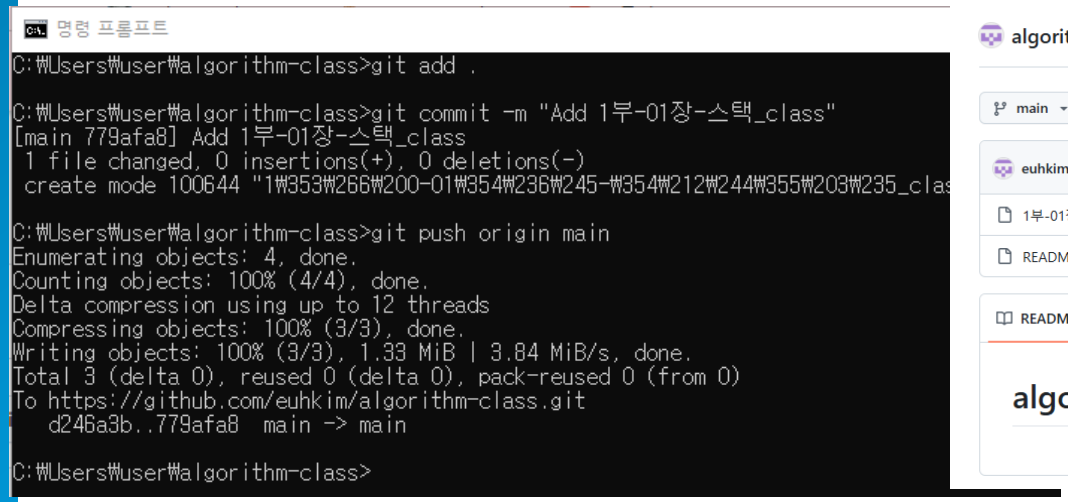
- **# 원하는 폴더에서 실행 : algorithm-class 폴더가 생김**
- **git clone https://github.com/계정이름/algorithm-class.git**

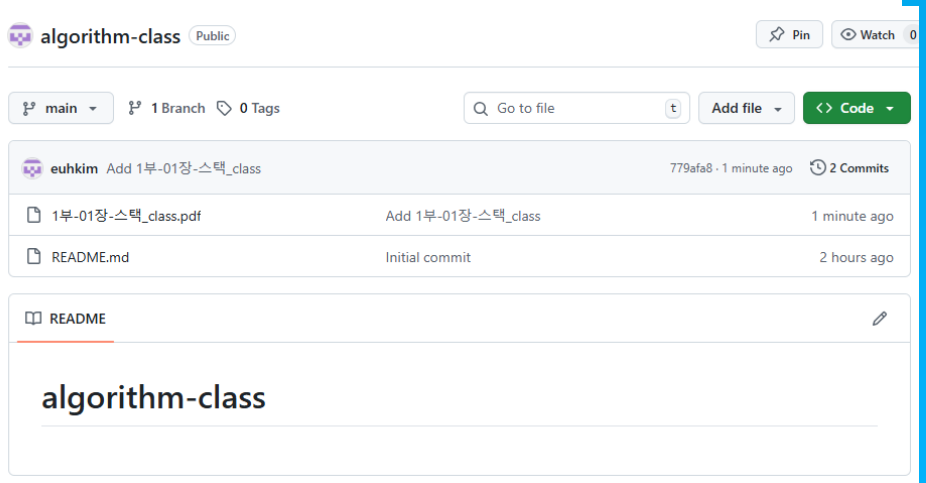




GitHub Desktop 초보자 가이드 (강의실 다중 사용자용)

5. **파일 업로드 (수업 자료 추가)**
   - 수업 자료 (예: lecture.pdf, example.py 등)를 algorithm-class 폴더 안에 넣기
   - 터미널에서 명령 실행
     1. **cd algorithm-class**
     2. **git add . # 새 파일 추가**
     3. **git commit –m "Add lecture1 and example code" # 변경 기록 저장**
     4. **git push origin main # Github에 업로드**
   - **Github 사이트에서 파일이 업데이트된 것을 확인하기**





GitHub Desktop 초보자 가이드 (강의실 다중 사용자용)

6. **수정하기 (Update)**
   - 수업 자료 (예: example.py 등)를 수정한 후 저장해서 algorithm-class 폴더 안에 넣기
   - 터미널에서 명령 실행
     1. **Cd algorithm-class**
     2. **Git add example.py # 수정 파일 추가**
     3. **git commit –m "Fix bug in example code" # 변경 기록 저장**
     4. **git push origin main # Github에 업로드**
   - **Github 사이트에서 파일이 업데이트된 것을 확인하기**