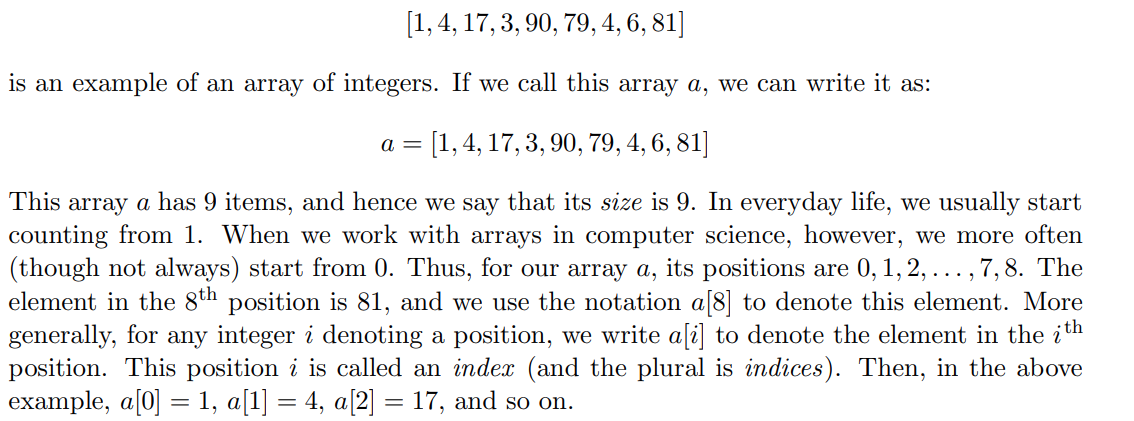
**배열(Arrays)**

컴퓨터 과학에서 정렬된 항목들의 집합을 저장하는 가장 일반적인 방법 중 하나가 배열(array)입니다. 배열은 컴퓨터 메모리 상의 연속적인 위치에 항목들이 저장됩니다. 배열을 논의하거나 표현할 때는 종이에 쓰기 쉽도록 대괄호로 묶고, 항목들을 쉼표로 구분하여 나타냅니다.

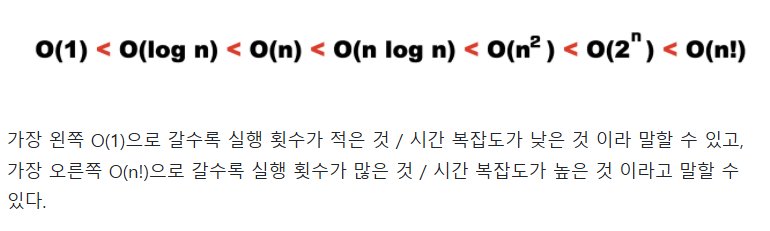
****

수학에서는 =가 두 값이 같음을 나타내는 기호(Equality)로 사용됩니다. (두 값이 같은지를 확인할 때는 일반적으로 ==)

그러나 현대 프로그래밍 언어에서는 =가 값을 할당(Assignment) 하는 의미로 쓰이는 경우가 많습니다.

##인덱스(i)를 이용한 빠른 접근과 연속적 데이터 저장이 가능하며, 알고리즘 설계 및 데이터 처리에 널리 사용됩니다.

##접근의 개념으로 O(1)의 시간복잡도를 가지지만 해당 인덱스를 찾아야한다면 검색의 시간복잡도인 O(n)에 해당한다.



**선형리스트 [Linear List]:**

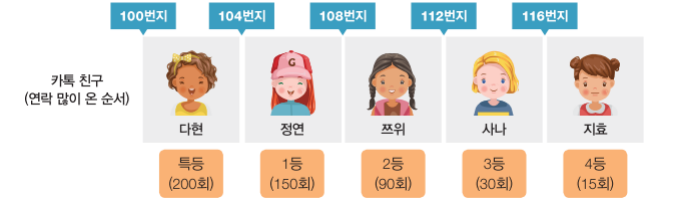
데이터 구조와 알고리즘의 기초를 이해하고 이를 활용해 효율적인 프로그램을 설계하기 위해서입니다. 선형 리스트는 데이터 구조 중에서도 가장 기본적이면서도 다양한 분야에서 활용도가 높기 때문에 배우는 것이 중요합니다. 스택(Stack), 큐(Queue)와 같은 자료 구조는 선형 리스트의 개념을 기반으로 동작합니다. 데이터를 일정한 순서로 나열한 자료구조로, 입력 순서대로 저장하는 데이터에 적당하다. 배열은 선형 리스트를 구현하는 하나의 방식

****

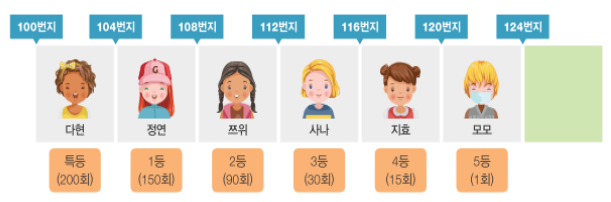
데이터를 일정한 순서로 나열한 자료구조

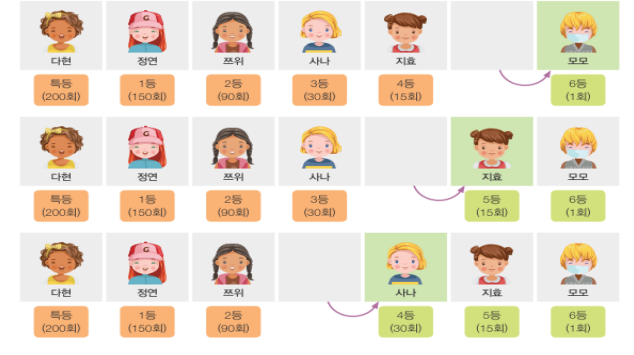
순차 리스트(Ordered List)라고도 함

선형 리스트는 입력 순서대로 저장하는 데이터에 적당

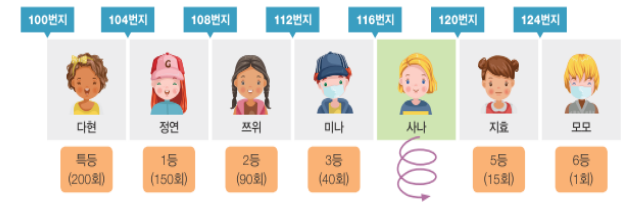


선형 리스트에서 데이터 삽입 과정 [미나]



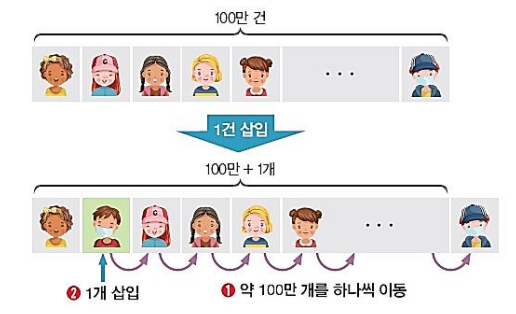


선형 리스트에서 데이터 삭제 과정







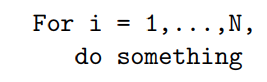
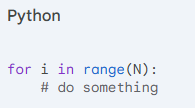


**표로 정리**

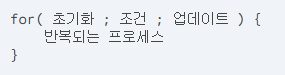
|  |  |  |
| --- | --- | --- |
| **특성** | **선형 리스트 (Linear List)** | **배열 (Array)** |
| **정의** | 데이터를 순차적으로 저장하는 자료 구조 | 데이터를 연속된 메모리 공간에 저장 |
| **구현 방식** | 배열 또는 연결 리스트로 구현 가능 | 배열로만 구현 |
| **메모리** | 동적 또는 고정적 (구현에 따라 다름) | 고정 크기 |
| **접근 속도** | 구현 방식에 따라 다름 | 빠름 (O(1)) |
| **삽입/삭제** | 연결 리스트일 경우 빠름 | 느림 (O(n)) |
| **유연성** | 배열과 연결 리스트의 장점 모두 포함 | 고정된 구조 |

**반복문(Loops and Iteration)**

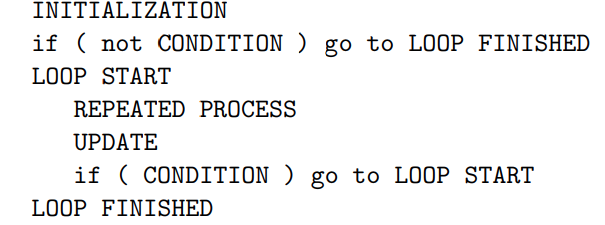
프로그래밍 언어에서 특정 횟수만큼 프로세스를 반복하는 표준적인 접근 방식은 루프를 사용하는 것입니다.

(의사코드:pseudocode) 

일반적인 for 루프 구조는 다음과 같습니다:



여기서 네 가지 부분 중 어느 것이든 선택적으로 생략될 수 있습니다. 이를 명시적으로 작성하는 한 가지 방법은 다음과 같습니다:



초기화  
조건이 거짓이면 루프 종료로 이동  
루프 시작  
반복되는 프로세스  
업데이트  
조건이 참이면 루프 시작으로 이동  
루프 종료

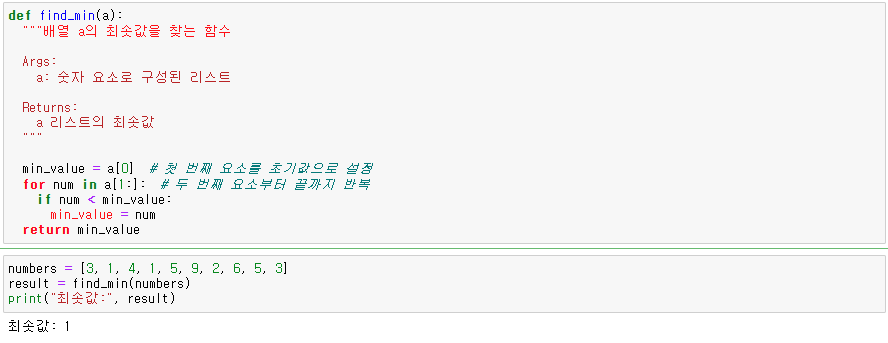
다른 프로그래밍 언어는 다른 구문을 사용하며, 반복을 종료하기 위해 조건을 확인하는 다양한 변형이 있다는 것을 기억하는 것이 중요합니다.

**불변식(Invariant)**

어떤 객체가 정상적으로 작동하기 위해 절때 허무러지지 않아야 하는 값, 식, 상태의 일관성을 보장하기 위해 항상 참이 되는 조건(condition)을 말한다.

주어진 프로그램이나 알고리즘의 실행 중에 변하지 않는 조건을 의미합니다

##불변식은 데이터 구조와 알고리즘에서 중요한 역할을 합니다. 왜냐하면 이를 통해 **정확성 증명(correctness proofs)**과 **검증(verification)**을 가능하게 하기 때문입니다.



각 반복의 시작과 끝에서 불변식 " min\_value은 a[0], ..., a[i-1] 중 최솟값과 같다"는 참입니다.

 ,

이는 처음에 참으로 시작하고, 반복적인 과정과 업데이트를 통해 계속해서 참으로 유지됩니다. 따라서 루프가 종료될 때(i == n// n번째), "min\_value은 a[0], ..., a[n-1] 중 최솟값과 같다"를 알 수 있으므로, min을 필요한 최솟값으로 반환할 수 있습니다. 이는 일종의 수학적 귀납법 증명입니다:

**기저 단계:** 초기 상태에서 불변식이 참입니다.

**귀납 단계:** 만약 k번째 반복에서 불변식이 참이라면, k+1번째 반복에서도 불변식이 참입니다.

불변식은 루프의 시작 시에 참이고, 각 반복에 의해 보존되므로, 루프의 끝에서도 참이어야 합니다.

좋은 루프 불변식은 프로그램의 품질을 향상시키는 데 매우 중요합니다. 루프 불변식을 명확하게 정의하고 유지함으로써, 더욱 안정적이고 신뢰할 수 있는 프로그램을 개발할 수 있습니다.

**루프 불변조건 (Loop invariant)**

**(좋은) loop invariant는 세 가지 조건을 만족해야 한다.**

**초기화 (Initialization):** 루프가 시작되기 전에 불변식이 반드시 참이어야 합니다. 즉, 루프에 들어가기 전에 불변식을 만족하는 상태로 초기화해야 합니다.

**유지 (Maintenance):** 루프의 한 번의 반복이 끝난 후에도 불변식이 계속 참이어야 합니다. 즉, 루프의 각 반복이 불변식을 깨뜨리지 않아야 합니다. 이를 통해 루프가 진행되는 동안 항상 불변식이 유지된다는 것을 보장할 수 있습니다.

**종료 (Termination):** 루프가 종료될 때, 불변식은 문제의 해결 조건과 일치해야 합니다. 즉, 불변식이 참이라는 것은 문제가 해결되었다는 것을 의미해야 합니다. 이를 통해 알고리즘의 정확성을 보장할 수 있습니다.

ex)

**초기화:** min\_value 변수에 리스트의 첫 번째 값을 할당합니다. 이는 min\_value가 항상 현재까지 확인한 값 중 가장 작은 값이라는 가정을 설정하는 것과 같습니다.

**반복:** 리스트의 두 번째 요소부터 끝까지 반복하며, 현재 min\_value와 비교합니다. 만약 현재 요소가 min\_value보다 작다면 min\_value를 현재 요소로 업데이트합니다.

**종료:** 루프가 끝나면 min\_value에는 리스트의 최솟값이 저장됩니다.

문제-

주어진 배열의 모든 요소의 합을 계산하는 아래 알고리즘에 대해 **Loop Invariant**를 설계하고 증명하시오.

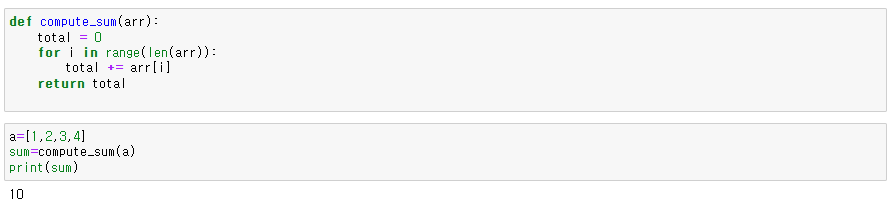
다음은 배열에서 가장 큰 요소를 찾는 알고리즘입니다. 이 알고리즘의 **Loop Invariant**를 정의하고 증명하시오.

**답안 작성 요령**

각 문제에서:

1. Loop Invariant를 정의합니다.
2. 초기 조건, 유지 조건, 종료 조건을 명확히 설명합니다.
3. 증명 과정을 통해 알고리즘의 올바름을 보장합니다.

주어진 배열의 모든 요소의 합을 계산하는 아래 알고리즘에 대해 **Loop Invariant**를 설계하고 증명하시오.



다음은 배열에서 가장 큰 요소를 찾는 알고리즘입니다. 이 알고리즘의 **Loop Invariant**를 정의하고 증명하시오.

