배열 자료 구조 (Array)

- 영상 정보 : https://www.youtube.com/watch?v=NFETSCJON2M

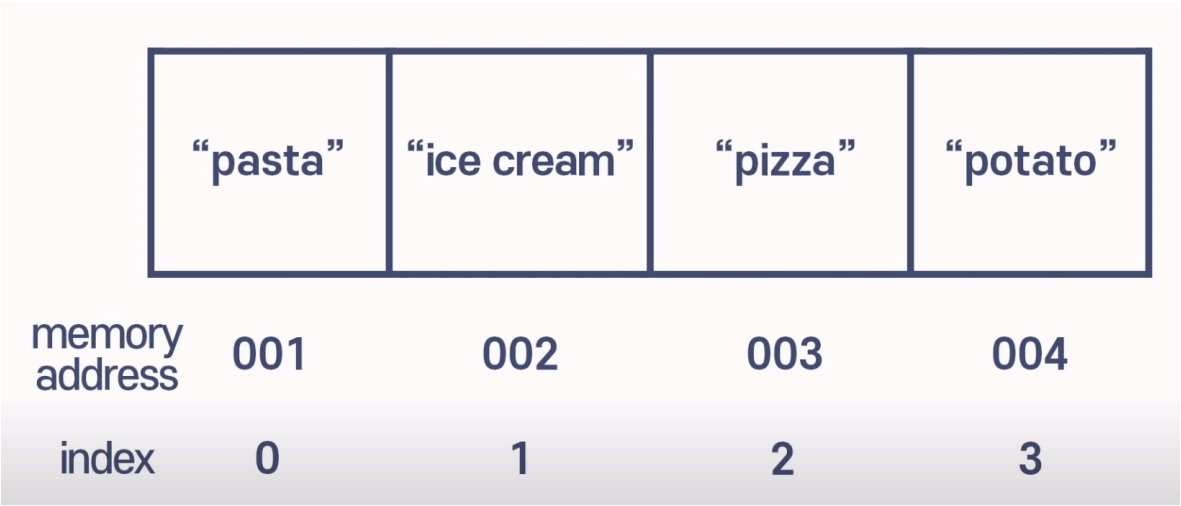
- 배열(Array)
- 시간복잡도(Time Complexity)이란?
  - 데이터 구조의 오퍼레이션 혹은 알고리즘이 얼마나 빠르고 느린지 측정하는 방법
  - 실제 시간을 측정하는것이 아닌 얼마나 많은 단계(steps)로 이루어져 있는가로 측정하며 단계가 적을 수록 좋다고 보면 된다. (5 steps > 20 steps)

메모리 관점에서 배열이 어떻게 보이는가?

- 휘발성(volatile) 메모리
  - RAM같은 개념 - 컴퓨터를 껐다 키면 데이터가 사라짐
  - 컴퓨터를 껐다 키면 다시 0부터 시작
  - 프로그램이 실행 -> 변수 생성은 모두 RAM에 저장이 된다. -> Random Access Memory
  - 하드드라이브에서 읽는 것 보다 RAM에서 데이터를 읽는 것이 더 빠르다.
    - why? 데이터 접속을 랜덤으로 할 수 있기 때문
      - ex. RAM = 박스그룹, 각각의 박스들마다 데이터들을 보관, 박스명 = Memory Address
      - -> 프로그램이 RAM에게 원하는 박스명을 부를 시 랜덤하게 접근이 가능하여 빠르다.

- 비휘발성(non-volateile) 메모리
  - 하드드라이브 같은 개념 - 컴퓨터를 껐다 켜도 데이터가 그대로 존재
  -
- 메모리 관점에서 배열을 만들 때 컴퓨터에게 길이를 선언해주어야 한다. -> 컴퓨터는 이 배열의 길이를 기억한다.
  - JS와 파이썬은 배열의 길이를 정해주지 않아도 자동으로 핸들링 해준다. -> 속도가 느려질 수 있음

1. Reading - 배열 읽기




- 배열은 0부터 indexing하여 위치만 알고 있다면 배열의 데이터에 접속이 가능하다.
- 배열 내에서 원하는 값의 인덱스를 호출하면 해당 인덱스 내에 존재하는 값을 받아올 수 있다.
- 인덱스에서 요소를 읽어내는 속도는 동일하기 때문에 배열의 길이는 상관없다. (step 동일)

2. Searching - 배열 검색
- 검색 =/= 읽기
- 해당 값이 배열에 있는지, 어디에 있는지 모르기 때문에 하나하나씩 0부터 순서대로 다 검색할 수 밖에 없다.(선형검색(Linear Search)) -> 그로 인해 시간이 많이 소요되는 단점이 있고, 원하는 데이터가 존재하지 않을 수 있다.
- 배열 안에서의 검색은 그다지 빠르지 않다고 보면 된다.

3. Insert (or 배열에 쓰기) - 배열 추가(adding)
- 배열을 만들 때에는 메모리 공간을 미리 확보해야 한다.
- 만약 5개의 아이템이 들어있는 배열에 4개의 아이템이 존재하고, 추가적으로 아이템을 넣어주고 싶다면 미리 확보해놓은 메모리 공간에 넣어주면 된다.
  - 이 아이템을 어디에 넣어줄 것인지가 관건이 되는 것인데
    - 1. 맨 끝에 넣어주고 싶을 경우 : 컴퓨터가 배열의 시작과 끝을 기억하고 있기 때문에 자동으로 끝에 입력될 수 있다.
    - 2. 배열 중간에 넣어주고 싶을 경우 : 원하는 위치 옆의 아이템들을 이동 해주어야 원하는 아이템을 추가할 공간이 생긴다.
    - 3. 배열 맨 앞에 넣어주고 싶을 경우 : 배열 내의 모든 아이탬들을 움직여줘야 한다. (배열의 크기가 클 경우 매우 비효율적)
    - 4. 꽉찬 배열의 추가적으로 넣어주고 싶을 경우 : 더 큰 배열 생성 -> 이전 배열 복사 -> 그다음 새로 값을 추가 ( 시간이 오래걸림.)

4. Delete - 배열 삭제
  - 1. 배열의 마지막 요소를 삭제하고 싶을 경우 : 그냥 삭제가 된다. (컴퓨터가 배열의 길이를 기억하고 있기 때문)
  - 2. 배열의 중간 요소를 삭제하고 싶을 경우 : 아이템 삭제 -> 공백 발생 -> 모든 배열의

값들을 이동시킴으로써 공백 메꿔줘야 함

- 3. 배열의 첫 번째 요소를 삭제하고 싶을 경우 : 2번과 같은 원리로 모든 값들이 한칸씩 앞으로 이동되어야함 (값이 많을 경우 비효율적)