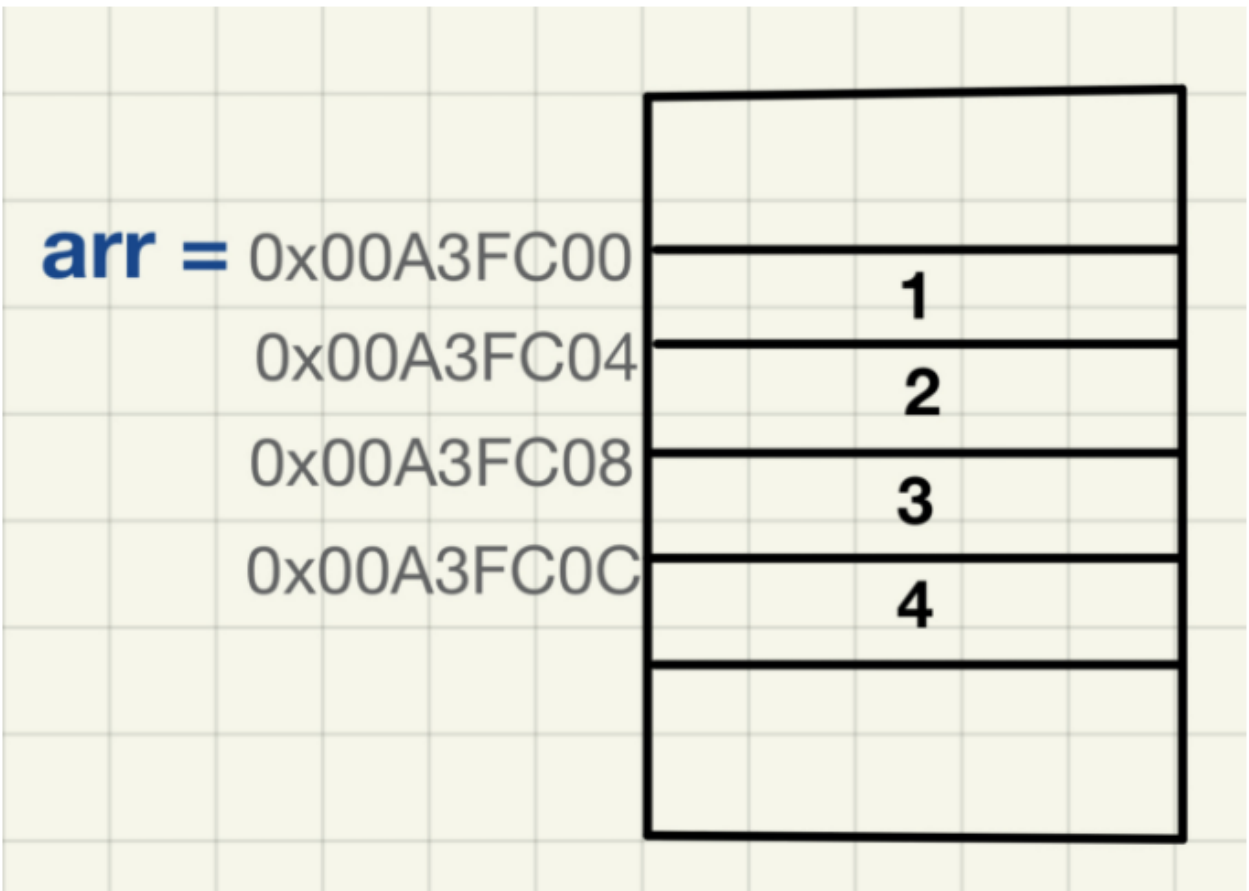
C언어의 배열

실제 데이터 값이 메모리에 할당됨

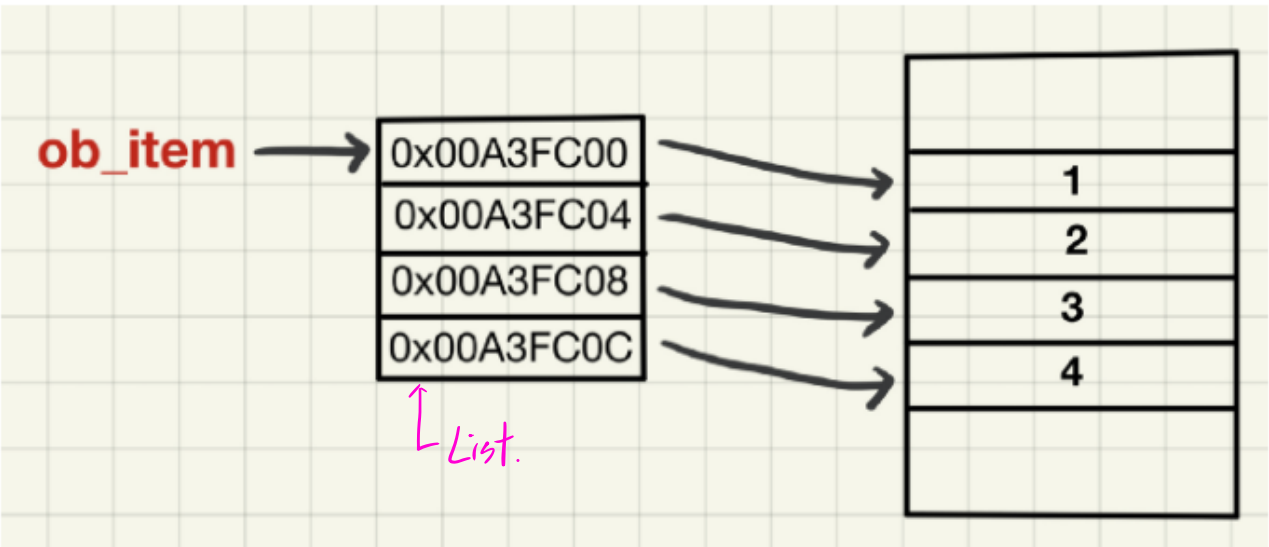
`int arr[4] = {1, 2, 3, 4}` 와 같은 배열이 있을때, 해당 배열은 4바이트 * 4 만큼 메인 메모리를 할당받아 연속적으로 저장된다. 변수 arr은 배열의 시작 주소를 가지고 있기 때문에 index를 통해 배열의 원소에 접근할 수 있다.



c언어의 배열 저장 구조

파이썬의 리스트

동일한 배열에 대해 파이썬 리스트의 저장 구조는 위와 다르다. ob_item 라는 이중 포인터가 있고, ob_item은 리스트 원소들의 주소 값을 저장한 C언어의 배열을 가리킨다. 그리고 이 배열에 저장된 주소값을 통해 실제 원소 값에 접근할 수 있다.



파이썬의 리스트 저장 구조

이와 같이 파이썬의 리스트는 원소의 주소값을 저장하기 때문에 각 원소의 자료형이 무엇이든 상관없다. 따라서 초기 선언시 자료형을 지정하지 않아도 되고, 하나의 리스트 안에 서로 다른 자료형을 저장할 수 있는 것이다.

하지만 이중 포인터를 사용하기 때문에 특정 값을 찾기 위해 두 번의 탐색 과정을 거치게 되므로 C언어의 배열보다 속도가 느리다.

메모리 할당

(C언어의 배열과 달리)**파이썬의 리스트는 동적 배열**이기 때문에 초기 선언 시 리스트의 크기를 지정하지 않아도 된다. 동적 배열은 초깃값을 작게 잡아 배열을 생성하고, 리스트가 꽉 채워지면 크기를 늘려주는 방식(더블링) 으로 동작한다. 파이썬 리스트의 메모리 할당은 list_resize 메소드를 통해 이뤄진다.

append

파이썬 리스트는 동적 배열 자료형이므로, 아이템을 삽입할때 사용하는 append 메소드에서 배열의 용량이 꽉 차면 알아서 리스트의 크기를 증가시킨다. 더 자세하게 말하면 append는 cpython에서 PyList_Append 함수를 통해 구현되어있고, PyList_Append 함수는 내부에서 app1 함수를 통해 list_resize를 호출하여 리스트의 크기를 증가시키는 방식으로 동작된다.