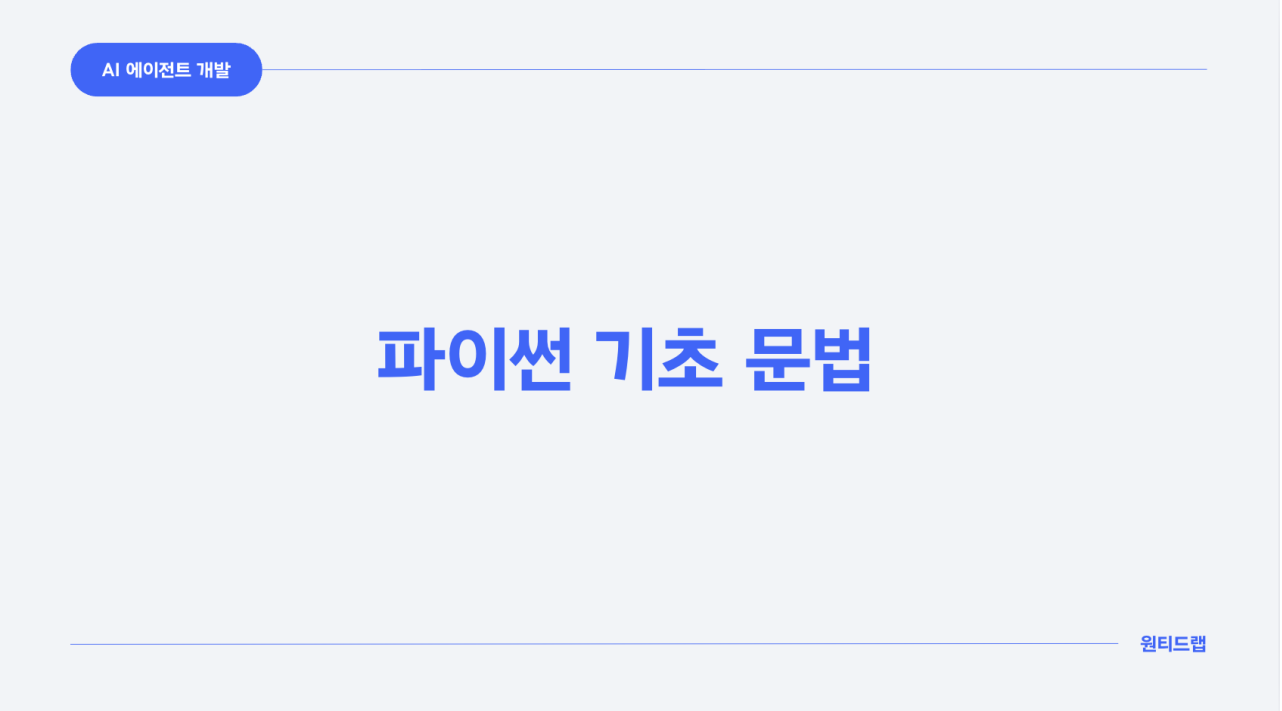
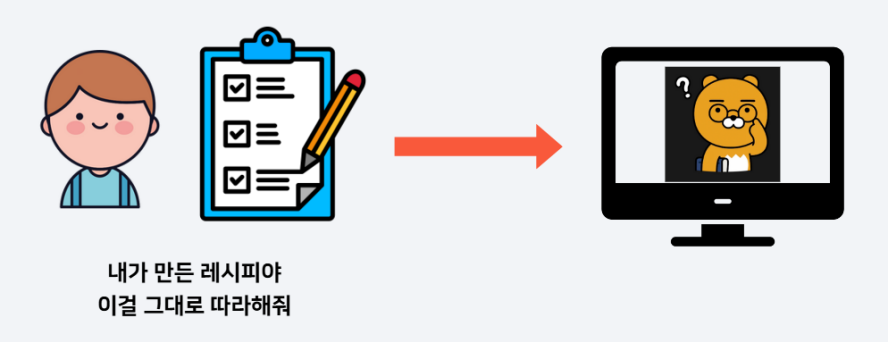
01 파이썬 기초 문법 I (변수, 자료형)



- 코딩
- 변수
- 자료형
  - 1) 숫자형
  - 2) 문자열
  - 3) 리스트(List)
  - 4) 튜플(Tuple)
  - 5) 딕셔너리(Dictionary)
  - 6) 집합(Set)
  - 7) 불(Bool)

코딩

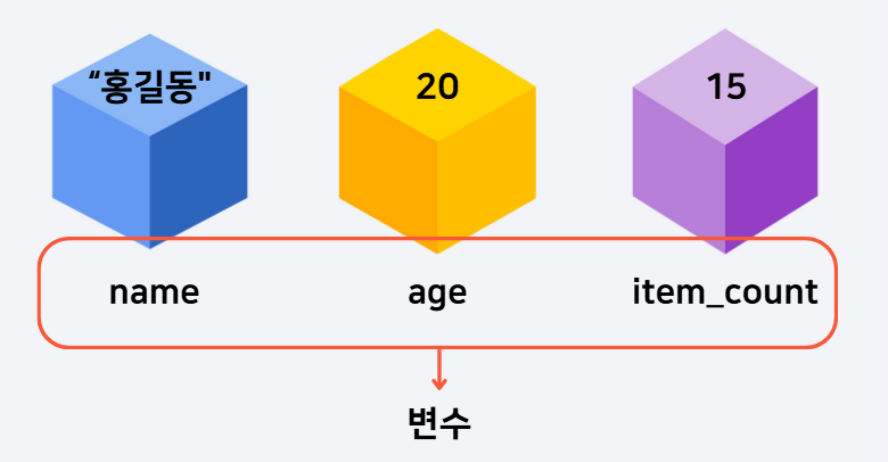


코딩이란 컴퓨터에게 컴퓨터가 알아들을 수 있는 언어로 레시피를 작성하는 것을 말한다.
그리고 이 레시피를 **1) 만들고 2) 테스트해보고 3) 간을 맞추는 과정**을 프로그래밍이라고 한다.

🔗 참고자료: 전과자, 포항공대 컴퓨터공학과 편

변수

변수의 의미



- 변수는 데이터를 저장하는 **메모리 공간(상자)** 의 이름
- '변수 = 데이터' 형태로 값을 대입시킨다. 즉, '데이터를 변수라는 이름의 상자에 넣는다'라고 표현할 수 있다.
- ex. name 상자에 '홍길동'을 넣고 싶으면 `name = '홍길동'` 이라고 작성한다.

대입 연산자⭐

- '='을 대입연산자 라고 하며, '같다'라는 의미가 아니다.
- a = a + 1 의 의미
  - 'a는 a+1과 같다'(❌)
  - 'a라는 상자에 (a+1)을 넣는다.(✅)

> 파이썬은 동적 타이핑 언어이므로, 변수를 선언할 때 데이터 타입을 지정하지 않아도 된다.

변수 생성 규칙

- 영어 + 숫자 등을 사용하여 생성
- 대소문자 구분
- 문자부터 시작해야 하며, 숫자로 시작 불가능
- 특수기호는 언더바(`_`)만 사용 가능
- 키워드(if, for, while, def, end, or 등) 사용 불가능

> **키워드(Keywords)**
>
> - Python 키워드는 Python 자체 사용 목적으로 이미 예약된 문자
> - Python 키워드를 변수나 함수 이름, 즉 식별자로 사용할 수 없음
>
> ```
> True, False, None, and, as, assert, async, await, break, class, ....
> ```

자료형

1) 숫자형

- 정수(Integer): 양의 정수, 음의 정수, 0 (ex. 123, -456, 0)
- 실수(float): 소수점이 포함된 숫자 (ex. 123.4, -456.789)
- 8진수(octal): 0부터 7까지의 숫자만 사용. 숫자 앞에 0o를 붙임(ex. 0o11, 0o323)
- 16진수(hexadecimal): 09, AF까지의 숫자 사용. 숫자 앞에 0x를 붙임(ex. 0x2F, 0x3D)

산술 연산

연산	의미	연산	의미
`a + b`	a 더하기 b	`a // b`	a를 b로 나눈 몫
`a - b`	a 빼기 b	`a % b`	a를 b로 나눈 나머지
`a * b`	a 곱하기 b	`a ** b`	a의 b제곱 (a^b)
`a / b`	a 나누기 b		

복합 연산자

- 연산 수행과 동시에 변수에 값 대입하는 연산자
- 구문은 짧으나 가독성이 다소 나빠짐
- 너무 많이 사용하면 오히려 복잡해질 수 있음

연산	의미	연산	의미
`a += b`	`a = a + b`	`a //= b`	`a = a // b`
`a -= b`	`a = a - b`	`a %= b`	`a = a % b`
`a *= b`	`a = a * b`	`a **= b`	`a = a ** b`
`a /= b`	`a = a / b`		

2) 문자열

- 문자나 단어 등으로 이루어진 문자들의 집합
- 작은 따옴표(' ') 또는 큰 따옴표(" ")로 묶어서 표현

산술 연산

- 파이썬에서는 문자열끼리 더하기(+)와 곱하기(*) 연산이 가능

- **숫자형과 문자열 연산은 불가능**
- 데이터 타입 확인 후 연산을 진행
- 타입이 다를 경우 문자열로 변환하여 문자열 연산을 진행

인덱싱(Indexing)

- 인덱스(Index)는 어떤 배열 안에 원소가 위치하고 있는 순서
- 인덱싱이란 문자열 내에서 **특정 위치의 문자에 접근**하는 방법
- 문자열 내 특정 위치의 내용을 가져올 때 사용
- 파이썬은 **0부터 시작**하는 인덱스 값을 통해 문자열에 접근

index

9	0	1	2	1	2	-	1	3	2	5	2	2	1
0	1	2	3	4	5	6	7	8	9	10	11	12	13
-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

슬라이싱(Slicing)

- 인덱스(Index) **범위를 지정하여 여러 개의 원소를 추출**하는 방법
- **end 번째 요소는 결과에 포함되지 않음**
  - `data[start:end]`: 인덱스 n에서 end-1까지 추출
  - `data[start:]`: 인덱스 m에서 끝까지 추출
  - `data[:end]`: 처음부터 end-1까지 추출

start end

9	0	1	2	1	2	-	1	3	2	5	2	2	1
0	1	2	3	4	5	6	7	8	9	10	11	12	13
-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

표시 형식 지정

- 문자열 일부분만 바꾸면서 해당 문자열을 재사용할 경우 사용
- 문자열 포맷팅(Formatting)
  - % 포맷팅: % 연산자를 이용하여 문자열에 변수나 값을 포함시킬 수 있음. (%s는 문자, %는 정수)
  - 문자열.format() 메서드: 중괄호(`{}`)를 사용하여 변수를 표시하고 값을 전달할 수 있음.
  - f-문자열: 문자열 앞에 `f`를 붙이고 중괄호(`{}`)안에 변수나 표현식을 직접 넣을 수 있음.

관련 주요 메서드

메서드	기능
`문자열.split`	지정한 문자열을 구분자로 하여 나눈 문자열들을 요소로 갖는 리스트 반환
`문자열.replace`	지정한 문자열을 다른 문자열로 만든 문자열 반환
`문자열.lower`	소문자로 바꾼 문자열 반환
`문자열.upper`	대문자로 바꾼 문자열 반환

메서드	기능
`문자열.strip`	양쪽에서 공백이나 지정한 문자열을 제거한 문자열 반환 (왼쪽- `lstrip`, 오른쪽- `rstrip`)
`문자열.count`	특정 문자열이 나타나는 횟수를 반환

3) 리스트(List)

리스트가 필요한 이유

- 정수나 실수 등을 갖는 변수는 그 값 하나 → 많은 데이터 처리가 어려움
- 리스트는 여러 값을 요소로 가짐 → 많은 데이터 처리가 쉬움

리스트 특징

- 대괄호(`[]`) 안에 콤마로 구분해 요소를 나열
- 정수형, 문자열, 딕셔너리 등 여러 데이터 타입을 혼합해서 포함 가능
- 요소 값이 중복될 수도 있음
- 순서를 가지므로 특정 위치의 요소나 범위의 요소 확인 가능
- 또 다른 리스트를 요소로 포함 가능
- 문자열과 마찬가지로 인덱싱, 슬라이싱이 가능
- 인덱싱, 슬라이싱으로 값을 찾은 후 **새로운 값 대입 가능**

리스트 연산

- 리스트끼리 더하기(=결합), 숫자와 곱하기(=반복) 가능
- 빼기와 나누기 연산 불가능
- 문자열과 연산 기능과 동일

리스트 요소 추가/삭제

추가

- `리스트 + [1, 2, 3]` : 리스트 뒤에 요소들 추가
- `리스트.extend([1, 2, 3]` : 리스트 뒤에 요소 1, 2, 3 추가
- `리스트.append(값)` : 리스트 뒤에 요소 하나 추가
- `리스트.insert(index, value)` : 특정 위치에 요소(value) 하나 삽입

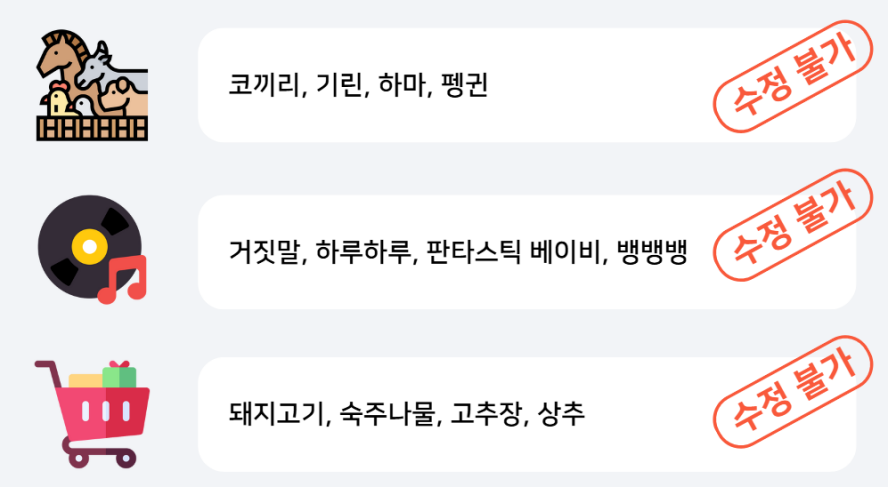
삭제

- `del 리스트[index]` : 리스트에서 특정 위치 요소 삭제
- `리스트[start:end] = []` : 리스트에서 특정 범위 지우기
- `리스트.remove(값)` : 리스트에서 첫번째로 등장하는 값 삭제(없으면 오류 발생)
- `리스트.pop()` : 리스트에서 마지막 요소 삭제 및 반환

관련 주요 메서드

메서드	기능
`리스트.count(값)`	리스트 안에 "값"이 몇 개 있는지 개수 반환(없으면 0 반환)
`리스트.index(값)`	지정한 "값"이 있는 요소의 위치 반환(없으면 오류 발생)
`문자열.sort()`	리스트 안의 요소 정렬

4) 튜플(Tuple)



- 소괄호(`()`)안에 콤마로 구분해 요소를 나열
- **읽기 전용**으로 한 번 만들면 요소의 값을 바꿀 수 없음(Immutable)
- 데이터 추가, 삭제, 수정이 불가
- 인덱싱, 슬라이싱, 튜플 연산 가능

5) 딕셔너리(Dictionary)

- 대괄호(`{}`)를 사용해서 Key와 Value를 콜론(`:`)으로 구분하여 쌍으로 구성된 자료형
- `{key1: value1, key2: value2, ...}` 의 형태
- 요소의 순서가 의미 없으며, Key를 사용해 Value를 확인

딕셔너리 다루기

- 딕셔너리는 인덱싱, 슬라이싱을 할 수 없음
- `딕셔너리[Key]` 를 사용하여 Value를 조회, 변경, 추가 가능
- `del 딕셔너리[Key]` 를 사용하여 딕셔너리에서 Key 삭제 가능
- `in` 연산자를 사용해 해당 Key가 있는지 확인 가능(ex. `"홍길동" in members`)

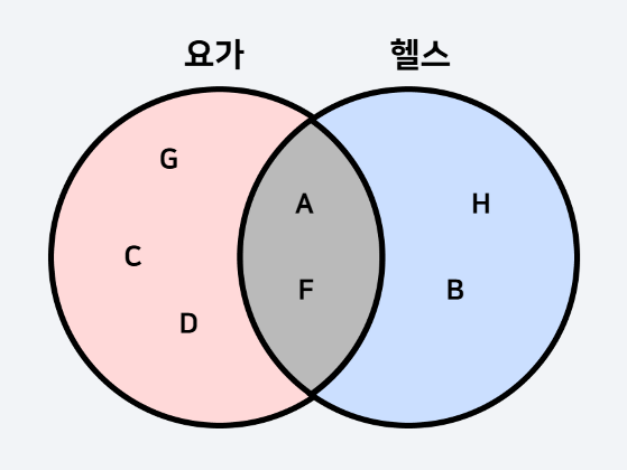
관련 주요 메서드

메서드	기능
`딕셔너리.keys()`	모든 key를 요소로 갖는 dict_keys 객체 반환
`딕셔너리.values()`	모든 value를 요소로 갖는 dict_values 객체 반환
`딕셔너리.items()`	모든 key와 value를 (key, value) 형태로 갖는 dict_items 객체 반환
`딕셔너리.clear()`	모든 요소 삭제
`딕셔너리.get(key, "대신할 값")`	key로 value를 찾을 때 없는 경우 "대신할 값"을 지정

6) 집합(Set)

- 중괄호(`{}`)를 사용해서 선언
- 집합연산(교집합, 합집합, 차집합, 대칭 차집합)을 위한 자료형
- **중복을 허용하지 않아** 중복된 원소는 하나만 제외하고 모두 무시됨
- 원소의 **순서가 의미 없으므로** 인덱싱과 슬라이싱을 할 수 없음

집합 연산



- 합집합: `|` 기호를 이용하거나, `union` 메서드 사용
- 교집합: `&` 기호를 이용하거나, `intersection` 메서드 사용
- 차집합: `-` 기호를 이용하거나, `difference` 메서드 사용
- 대칭 차집합: `^` 기호를 이용하거나, `symmetric_difference` 메서드 사용

7) 불(Bool)

- 참(True)과 거짓(False)의 두 가지 값을 가지고 있는 자료형
- 주로 조건문에서 사용된다.