

Python Language 교육

2차시. 조건문과 반복문

카이스트 문화기술대학원

박사과정 김종수

Review

프로그래밍이란?

프로그래밍이란 프로그램을 개발하는 행위를 일컫는다.

여기서 프로그래밍 언어는 컴퓨터가 이해할 수 있는 언어로 컴퓨터에게 특정 명령이나 연산을 수행할 수 있도록 하기 위해 만들어진 언어이다.

Python은 이 중에서도 매우 쉽고 데이터 처리에 능한 언어이다.

C와 C++에 비해서 데이터를 쉽게 처리할 수 있으며 문법이 매우 쉬워 프로그래밍을 처음 입문하는 사람들에게 매우 적합한 언어이다.

연산 속도가 C와 C++에 비해 느림에도 불구하고 데이터를 매우 쉽게 처리할 수 있어 딥러닝과 머신러닝에서 필수적으로 사용하는 언어가 되었다.

개발자에게 “솔직히” 프로그래밍이란?

주어진 문제의 해답을 제시하는 프로그램을 코딩을 통해서 만들어내는 것

그러나 프로그래밍을 ”잘 한다”라는 것은 문법에 능한 것이 아니라

- 1) 모르는 내용을 구글링을 통해 잘 찾아내고 이를 자신의 코드에 잘 적용하는 것
- 2) 문제가 주어졌을 때 컴퓨터적 사고 능력으로 코드를 잘 구성할 수 있는 것

→ 기본 문법을 익힌 후, 순서도와 실전 코딩 연습을 통해 컴퓨터적 사고 능력을 향상

→ 이를 진행하면서 구글링 검색 방식도 함께 익숙해지도록 할 예정

프로그래밍은 수학?!

프로그래밍은 완벽히 수학이라 할 수 없지만 수학 문제 풀이와 비슷한 구조입니다.

종수가 사과 5개를 가지고 있다. 종수는 어머니에게 사과를 3개 더 추가적으로 받게 되었다. 그렇다면 종수가 가지게 될 사과의 개수는?


Solution)

종수가 가진 사과의 개수를 a 라고 하면 $a=5$ 이다.

사과를 3개 더 받았으므로 종수가 가지게 될 사과의 개수는 $a+3$ 이다.

따라서, $a+3 = 5+3 = 8$ 개 이다.

Programming



```
a = 5
# a는 5라는 정수를 담는 변수

answer = a + 3
# answer는 a+3의 결과값을 담는 변수

print(answer)
# print는 괄호 안의 값을 콘솔에 출력하는 함수
```

8

변수(Variable)는 데이터를 담는 상자

자료형은 변수나 함수의 형태

int : 정수형 (-1, -100, 0, 2, 30 등)

float : 실수형 (소수점이 있는 값)

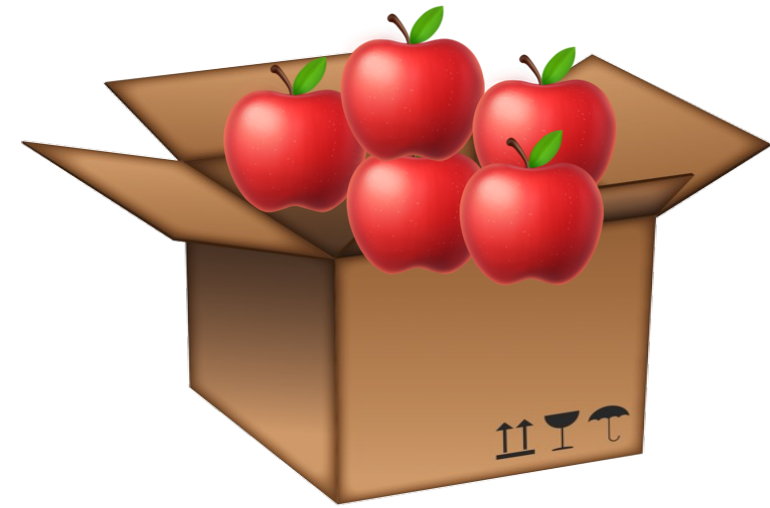
string: 문자열

함수는 연산들의 단축키

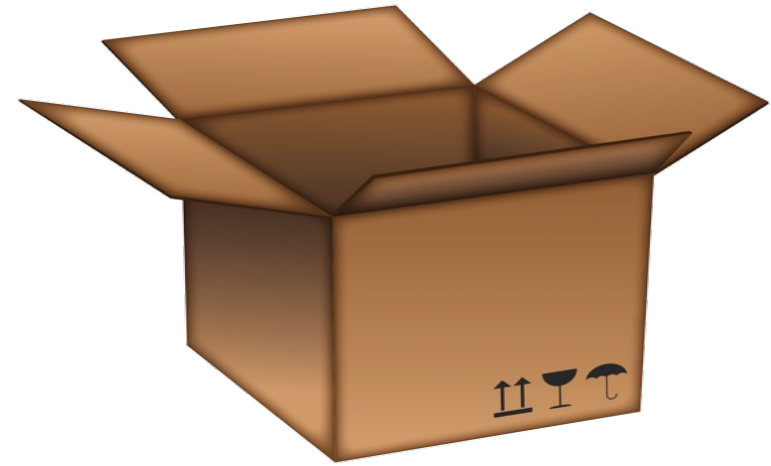
print 등

주석은 코드가 아닌 메모

Variable

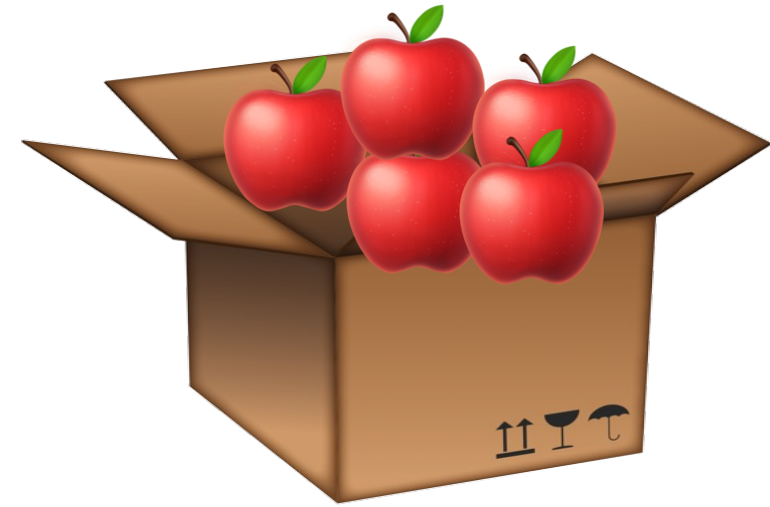


a

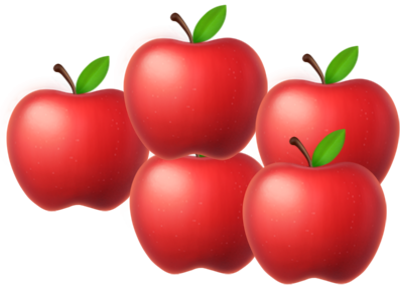


answer

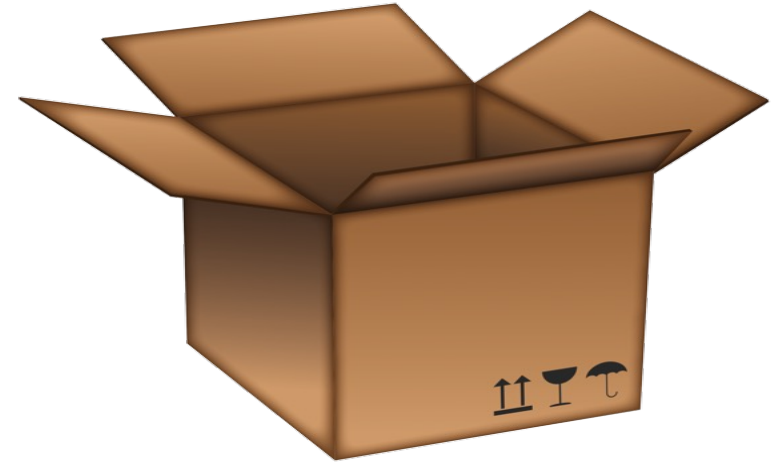
Variable



a

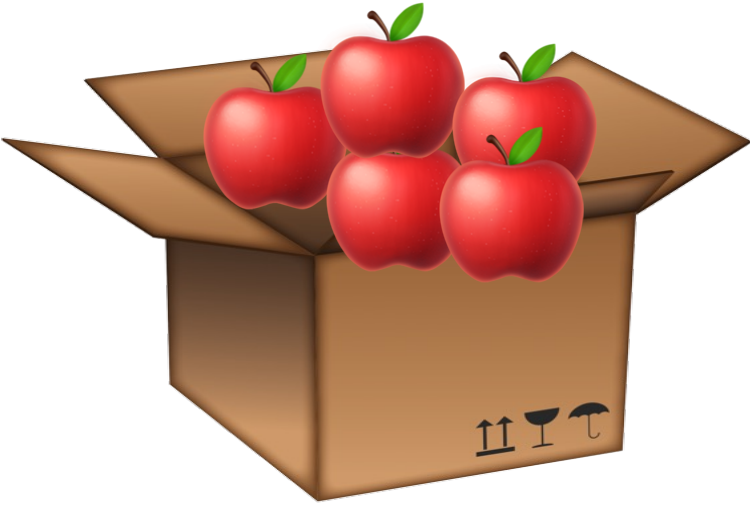


a 상자에 든 사과 개수만큼
똑같이 사과를 준비

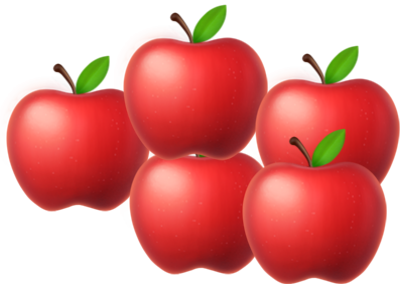


answer

Variable

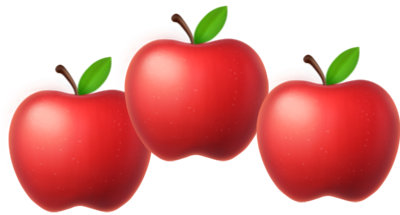


a



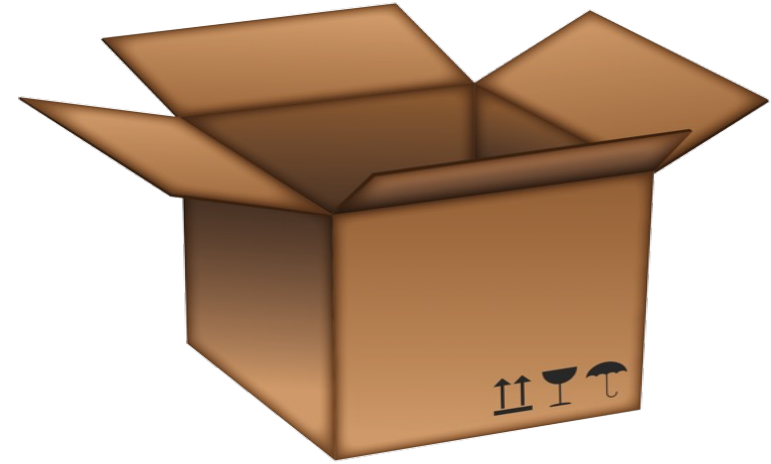
a

+



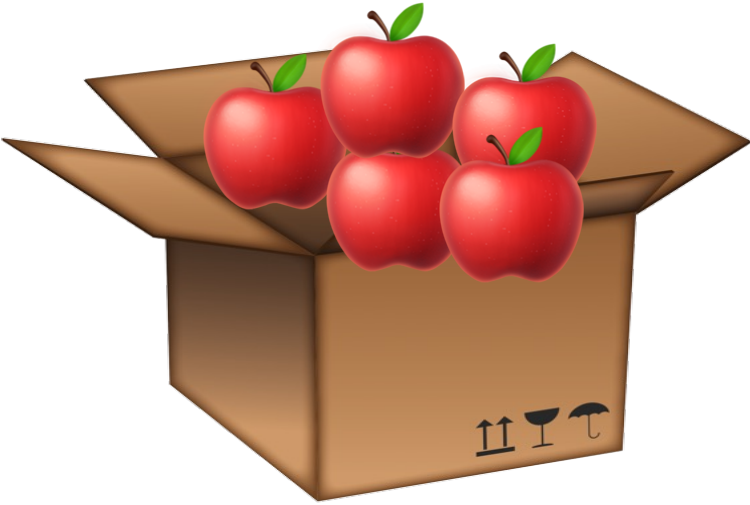
3

+3 연산을 위해
사과 3개 더 준비

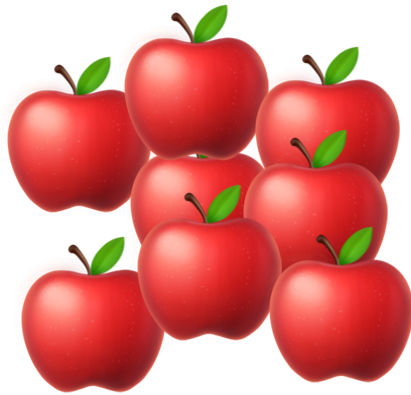


answer

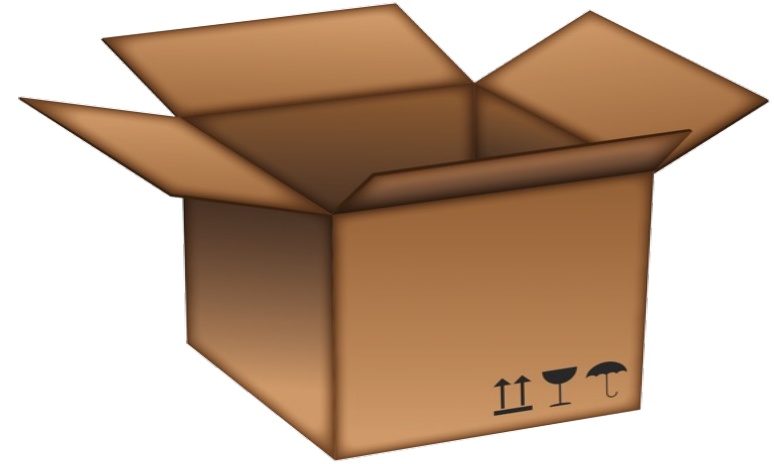
Variable



a

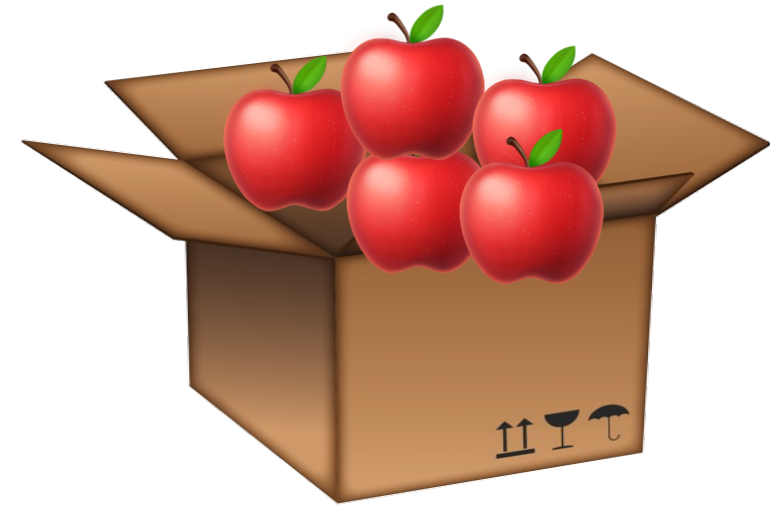


총 8개의 사과



answer

Variable



a

$$\text{answer} = a + 3$$



answer

answer라는 상자에
사과 8개를 담기

자료형 (Data Type)

Python에는 다음과 같은 기본 자료형이 있음.

수와 관련된 자료형

- 정수형 (`int`) (-100, 200, 5, 0, -3 등과 같이 소수점 없이 딱 떨어지는 수)
- 실수형 (`float`) (-100.03, 30.23, 0.0, -1.0 등과 같이 소수점이 존재하는 수)

문자열과 관련된 자료형

- 문자열 (`str`)
"String", '안녕' 과 같이 큰따옴표 `"` 혹은 작은따옴표 `'` 안에 문자로 구성된 집합을 의미)

수와 관련된 연산 (Operation)

Python에서 수와 관련된 기본 연산은 다음과 같다.

- 대입 연산자
- 덧셈, 뺄셈
- 곱셈, 나눗셈, 몫, 나머지, 거듭제곱 연산
- 할당 연산자

대입 연산자

변수의 특정 값을 대입하는 연산자 (=)

수학에서 '='은 equal의 의미를 가지고 있지만 프로그래밍 언어에서는 조금은 다른 의미

`a=5` 는 a라는 변수에 5라는 값을 대입한다는 의미

덧셈, 뺄셈

수를 더하거나(+) 빼는(-) 연산자

`a=5+3;` 은 a라는 변수에 5+3 연산을 수행한 결과값을 대입한다는 의미

곱셈, 나눗셈, 몫, 나머지, 거듭제곱

곱셈과 나눗셈 연산은 '*', '/'을 이용해 덧셈, 뺄셈과 동일하게 사용

몫 연산(//)은 몫을 반환하고 나머지 연산(%)는 나머지 값을 반환함

거듭제곱 연산(**)은 앞에 나오는 수를 뒤에 나오는 수만큼 거듭제곱 연산함

예시)

$5 + 2 \triangleright 7$	$5 - 2 \triangleright 3$	$5 * 2 \triangleright 10$	$5 / 2 \triangleright 2.5$
$5 // 2 \triangleright 2$	$5 \% 2 \triangleright 1$	$5 ** 2 \triangleright 25$	

할당 연산자

`+=`, `-=`, `*=`, `/=`, `%=` 등 다양하게 존재

해당 변수에 할당된 값에 특정 산술 연산을 시행한 후 결과값을 다시 해당 변수에 재할당

예를 들어,

- `a += 5;`
- `a = a + 5;`

위 두 식은 동일한 의미를 가진다

문자열과 관련된 연산 (Operation)

Python에서 문자열과 관련된 기본 연산은 다음과 같다.

- 대입 연산자
- 덧셈, 곱셈
- 할당 연산자
- 인덱싱
- 슬라이싱

대입 연산자

수의 대입 연산자와 동일한 의미로 `a="hello"` 는 a라는 변수에 hello라는 값을 대입한다는 의미

덧셈, 곱셈

두 문자열을 연결하거나 혹은 문자열을 반복시키는 연산

예시)

<code>"Hello" + "World" ➤ "HelloWorld"</code>	<code>"Hello" * 3 ➤ "HelloHelloHello"</code>
---	--

할당 연산자

`+=`, `*=` 등이 존재하며 수에서의 할당 연산자와 방식은 동일

인덱싱 (Indexing)

문자열에서 인덱스는 문자의 위치 정보를 의미한다. 인덱스라는 것을 이용해 우리는 문자열이 특정 위치에서 어떤 값을 가지고 있는지 알 수 있다. 문자열의 가장 첫 번째 문자는 인덱스 값이 0이며 마지막 값은 문자열의 길이에서 1을 뺀 값이다.

<code>a = "Hello, World"</code>			
<code>a[0] ➤ "H"</code>	<code>a[6] ➤ " "</code>	<code>a[-1] ➤ "d"</code>	<code>a[-3] ➤ "r"</code>

슬라이싱 (Slicing)

문자열의 인덱스 정보를 이용해 문자열의 일부분만 추출할 수 있다. 시작 인덱스 값과 끝 인덱스 값만 안다면 해당 인덱스 값들에 맞는 문자열의 일부를 추출할 수 있다. 인덱싱과 유사함.

예시)

<code>a = "Hello, World"</code>			
<code>a[1:4] ➤ "ell"</code>	<code>a[5:] ➤ ", World"</code>	<code>a[:6] ➤ "Hello,"</code>	<code>a[10:-1] ➤ "l"</code>

문자열과 관련된 함수 (Function)

Python에서 문자열과 관련된 함수는 다음과 같다.

`len (문자열)`

: 소괄호 내에 입력된 문자열의 길이를 반환한다.

예시)

<code>a = "Hello, World"</code>			
<code>len(a) ➤ 12</code>	<code>len(a[1:8]) ➤ 7</code>	<code>len(a[8:]) ➤ 4</code>	<code>len("hello") ➤ 5</code>

배열 자료형 (Array Data Type)

Python에는 다음과 같은 배열 관련 자료형이 있음.

- 리스트 (`list`)
 - `[0, 2, 3, 1]` 과 같이 대괄호를 이용해 작성
 - 리스트 내 원소들 간에는 순서가 존재 (원소 중복 가능)
 - 원소 수정, 삭제, 변경 가능 (가변)
- 튜플 (`tuple`)
 - `(0, 2, 3, 1, 0)` 과 같이 소괄호를 이용해 작성
 - 튜플 내 원소들 간에는 순서가 존재 (원소 중복 가능)
 - 원소 수정, 삭제, 변경 불가능 (불변)

배열 자료형 (Array Data Type)

Python에는 다음과 같은 배열 관련 자료형이 있음.

- 집합 (`set`)
 - `{0, 2, 3, 1}` 과 같이 중괄호를 이용해 작성
 - 리스트 내 원소들 간에는 순서가 존재하지 않음 (원소 중복 불가능)
 - 원소 수정, 삭제, 변경 가능 (가변)
- 딕셔너리 (`dict`)
 - `{1:"a", 2:"b", 3:"c", 4:"d"}` 과 같이 중괄호를 이용해 작성
 - `key:value` 형태로 원소가 구성되어 있음
 - 원소 수정, 삭제, 변경 가능 (가변)
 - `key`를 통해 `value` 값 접근 가능

리스트와 관련된 연산 (Operation) 및 함수(Function)

Python에서 리스트와 관련된 기본 연산과 함수는 다음과 같다.

- 연산
 - 대입 연산자, 덧셈, 곱셈, 할당 연산자, 인덱싱, 슬라이싱
- 함수
 - `.append()` : 리스트 맨 뒤에 새로운 값을 추가
 - `.sort()` & `.reverse()` : 리스트를 정렬 & 역순으로 정렬
 - `.insert()` & `.remove()` & `.pop()` : 리스트 특정 인덱스에 값 추가 & 리스트 특정 값 제거 & 리스트 특정 인덱스의 값 추출
 - `.index()` & `.count()` : 리스트 내 특정 값의 인덱스 값 반환 & 리스트 내 특정 값 개수 세기
 - `len()` : 리스트 길이 반환
 - `sorted()` : 리스트의 정렬된 버전을 반환
 - `del` : 값을 제거
 - `.clear()` : 모든 원소 제거

튜플과 관련된 연산 (Operation) 및 함수(Function)

Python에서 튜플과 관련된 기본 연산과 함수는 다음과 같다.

- 연산
 - 대입 연산자, 덧셈, 곱셈, 인덱싱, 슬라이싱
- 함수
 - len() : 튜플의 길이 반환
 - list() : 튜플을 리스트로 변환
 - .index() & .count() : 리스트 내 특정 값의 인덱스 값 반환 & 리스트 내 특정 값 개수 세기
 - del : 값을 제거
 - sorted() : 튜플의 정렬된 버전을 리스트로 반환

집합과 관련된 연산 (Operation) 및 함수(Function)

Python에서 집합과 관련된 기본 연산과 함수는 다음과 같다.

- 연산
 - 대입 연산자, 할당 연산자, 합집합 (|), 교집합 (&), 차집합 (-), 대차집합 (^)
- 함수
 - .union() & .intersection & .difference() & .symmetric_difference() : 합집합 & 교집합 & 차집합 & 대차집합
 - len() : 집합 원소 개수 반환
 - .add() : 특정 원소 추가
 - .pop() : 임의의 원소 추출
 - .remove() & .discard() : 제거(없으면 에러) & 제거(없으면 패스)
 - del : 값을 제거
 - .clear() : 모든 원소 제거

딕셔너리와 관련된 연산 (Operation) 및 함수(Function)

Python에서 딕셔너리와 관련된 기본 연산과 함수는 다음과 같다.

- 연산
 - 대입 연산자
- 함수
 - .keys() & .values() : key 값들 반환 & value 값들 반환
 - .items() : key & value의 pair들을 반환
 - .get() : key 값을 통해 사전 내 일치하는 원소 반환
 - del : 값을 제거
 - .clear() : 사전 내 모든 원소들 삭제

Examples

이름	교통 수단	출발 시각	도착 시각
김민지	SRT	09:05	09:21
홍길동	버스	08:00	09:23
오픈하이머	자차	08:50	09:53

LIST: ["이름", "교통 수단", "출발 시각", "도착 시각"]

[["김민지", "SRT", [9,5], [9,21]], ["홍길동", "버스", [8,0], [9,23]], ["오픈하이머", "자차", [8,50], [9,53]]]

TUPLE: ("이름", "교통 수단", "출발 시각", "도착 시각")

((("김민지", "SRT", (9,5), (9,21))), ("홍길동", "버스", (8,0), (9,23))), ("오픈하이머", "자차", (8,50), (9,53)))

Examples

이름	교통 수단	출발 시각	도착 시각
김민지	SRT	09:05	09:21
홍길동	버스	08:00	09:23
오픈하이머	자차	08:50	09:53

DICTIONARY + LIST:

```
[{"이름": "김민지", "교통 수단": "SRT", "출발 시각": [9, 5], "도착 시각": [9, 21]},  
 {"이름": "홍길동", "교통 수단": "버스", "출발 시각": [8, 0], "도착 시각": [9, 23]},  
 {"이름": "오픈하이머", "교통 수단": "자차", "출발 시각": [8, 50], "도착 시각": [9, 53]}]
```

오늘 배울 내용!

Conditional Statement

조건문

조건문은 특정 조건을 만족할 때, 정해진 연산을 수행하도록 할 때 사용한다

대표적으로 if 문이 있다

if 문: A라는 조건이 만족할 때, B라는 내용을 수행한다.

조건문

```
if condition1:
    # condition 1을 만족할 때 해당 Task들을 수행하고 아래의 elif, else는 무시
    Task_1_1
    Task_1_2
    ...
elif condition2:
    # condition 1을 만족하지 않으나 condition 2를 만족할 때 해당 Task들을 수행하고
    # 아래의 elif, else는 무시
    # elif는 여러개 가능하며 필요에 따라 생략해도 무방
    Task_2_1
    Task_2_2
    ...
elif condition3:
    # condition 1,2를 만족하지 않으나 condition 3를 만족할 때 해당 Task들을 수행하고
    # 아래의 elif, else는 무시
    Task_3_1
    Task_3_2
    ...
.
.
.
else: # else는 필요에 따라 생략 가능
    TaskA
    TaskB
    ...
```


Conditional Statement

if 문

if A: B: A라는 조건이 만족할 때, B라는 내용을 수행한다.

elif C: D: 위 조건을 만족하지 않을 경우, C라는 조건을 만족할 때, D라는 내용을 수행한다.

else: E: 위 조건들을 모두 만족하지 않을 경우, E라는 내용을 수행한다.

※ ‘ if ’ 다음에는 조건(비교 연산)이, ‘ : ’ 다음에는 실행문이 들어감

※ 상황에 따라 올바르게 조건문 안에 조건문을 넣거나 두 조건문을 분리할 수 있어야 함

Boolean

Boolean (참/거짓): 보통 프로그래밍 언어에는 **True**와 **False** 둘 중 하나의 값만 가질 수 있는 Boolean이라는 데이터 자료형이 존재한다. 또한, 0은 거짓을 의미하고 이외의 수는 참을 의미한다. 보편적으로는 1을 참으로 사용 (**0=False, 1=True**)

비교 연산자

연산자 기호	의미	사용 예
==	x와 y가 같은가?	x == y
!=	x와 y가 다른가?	x != y
>	x가 y보다 큰가?	x > y
<	x가 y보다 작은가?	x < y
>=	x가 y보다 크거나 같은가?	x >= y
<=	x가 y보다 작거나 같은가?	x <= y

조건을 만족하면 1을 반환
1=True

Conditional Statement

if 문

if A: B: A라는 조건이 만족할 때, B라는 내용을 수행한다.

elif C: D: 위 조건을 만족하지 않을 경우, C라는 조건을 만족할 때, D라는 내용을 수행한다.

else: E: 위 조건들을 모두 만족하지 않을 경우, E라는 내용을 수행한다.

※ 'if' 다음에는 조건(비교 연산)이, ':' 다음에는 실행문이 들어감

→ 즉, 비교 연산자를 이용한 비교 연산이 반환하는 값 (0 or 1) 을 통해 실행문을 수행

※ 상황에 따라 올바르게 조건문 안에 조건문을 넣거나 두 조건문을 분리할 수 있어야 함

논리 연산자

A and B (and 연산자): and 연산자는 두 연산식 A와 B가 모두 참일 경우에만 참을 반환한다. 하나라도 거짓일 경우, 거짓을 반환

A or B (or 연산자): or 연산자는 두 연산식 A와 B가 모두 거짓일 경우에만 거짓을 반환한다. 하나라도 참일 경우, 참을 반환

not A (not 연산자): not 연산자는 연산식 A가 참일 경우에 거짓을 반환하고 거짓일 경우에는 참을 반환

A	B	not A	A and B	A or B
False	False	True	False	False
False	True	True	False	True
True	False	False	False	True
True	True	False	True	True

실습 활동 (Practice Time)

실습 활동을 통해 조건문을 연습해보자

구글 Colab을 이용해 코드 작성 시작!

반복문

반복문은 특정 내용을 반복해서 수행하기 위해 사용된다

대표적으로 for 문과 while 문이 존재한다

for 문: 배열 B에서 원소를 하나씩 순서대로 꺼내 변수 A에 할당하고 꺼낼 때 마다 실행문 C를 수행

while 문: 조건 A가 만족되지 않을 때까지 실행문 B를 수행한다

Loops

for 문

```
for x in array:  
    Task1  
    Task2  
    ...  
  
Task3
```

while 문

```
while condition:  
    Task1  
    Task2  
    Task3  
    ...
```

while 문

0을 제외한 특정 값이 들어가면 항상 참이기 때문에 별도의 장치가 없으면 무한루프가 발생

무한루프는 컴퓨터가 멈추지 않고 연산을 계속 수행하기 때문에 문제가 되지만

이를 역이용해 유용하게 사용할 수 있음

이러한 반복문을 제어하기 위해서 필요한 것이 바로 break와 continue

break

무조건 해당 반복문을 탈출한다.

continue

해당 반복문의 실행문에서 continue 아래에 있는 코드들은 실행되지 않고 넘어간다.

실습 활동 (Practice Time)

실습 활동을 통해 반복문을 연습해보자

구글 Colab을 이용해 코드 작성 시작!