CODINGON X posco

K-Digital Training 스마트 팩토리 3기

리스트와 조건문

리스트

순서가 있는 리스트

- 리스트(list) : 자료 목록, 하나의 변수에 여러 값을 할당하는 자료형
- 리스트는 대괄호([])로 표현, 쉼표(,) 로 구분하여 작성
- 왜 필요할까?
  - 전교생을 1번부터 끝까지 나열하면 너무 많고 찾기가 어려움!

  => 일정 학생 수를 한 반으로 묶어서 관리 = 여러 데이터를 하나로 묶어서 보관하는 역할

```
print( [] ) # 비어있는 리스트
print( [ 1, 2, 3 ] ) # 숫자가 들어있는 리스트
print( [ 'a', 'b', 'c' ] ) # 문자가 들어있는 리스트
```

리스트 특징

- 새로운 항목 추가/삭제 가능
- 항목에 대한 순서가 있음 – 모든 요소가 번호(인덱스)를 갖는다
- 항목 검색 가능 – 인덱스 번호로 특정 가능
- 같은 자료형일 필요 없음! – 숫자형, 문자열 모두 가능

```
number=[ 1,5,3,4,8,2,3 ]
print(number[3]) #결과: 4
```

리스트 슬라이싱

- 문자열과 동일하게 인덱스로 슬라이싱 가능

```
shop = ["반팔", "청바지", "이어폰", "키보드"]

# 인덱싱
print(shop[0]) # 반팔

# 슬라이싱
print(shop[0:2]) # ['반팔', '청바지']


# 리스트 안에 리스트에서 인덱싱
my_shop = ["반팔", "청바지", "이어폰", ["무선 키보드", "유선 키보드", "기계식 키보드"]]
print(my_shop[3]) # ['무선 키보드', '유선 키보드', '기계식 키보드']
print(my_shop[-1]) # ['무선 키보드', '유선 키보드', '기계식 키보드']
print(my_shop[2:4]) # ['이어폰', ['무선 키보드', '유선 키보드', '기계식 키보드']]
```

리스트 연산

- 더하기 +
- 반복 *
- 길이 len()

```
a = [1, 2, 3]
b = [4, 5]

print(a + b) # [1, 2, 3, 4, 5]
print(a * 2) # [1, 2, 3, 1, 2, 3]
print(len(a)) # 3
print(len(b)) # 2
```

리스트 값 수정

- 인덱스로 접근해 값 수정하기

* 주의) 리스트 길이를 넘는 인덱스로 요소에 접근하면 IndexError 발생

```
shop = ["반팔", "청바지", "이어폰", "키보드"]

shop[0] = "무지 반팔"
print(shop) # ['무지 반팔', '청바지', '이어폰', '키보드']

shop[100] = "신발"
print(shop) # IndexError : 리스트 길이를 넘는 인덱스로 요소에 접근하려할 때
```

리스트 함수

리스트 함수 – 정렬

- sort()
  - 숫자 정렬

```
num = [3, 1, 5, 2]
num.sort()
print(num) # [1, 2, 3, 5]
```

  - 문자 정렬 (알파벳순, 자음순)

```
alphabet = ['b', 'c', 'a', 'd']
alphabet.sort()
print(alphabet) # ['a', 'b', 'c', 'd']


korean = ['강', '이', '박', '최']
korean.sort()
print(korean) # ['강', '박', '이', '최']
```

리스트 함수 - 정렬

- reverse()
  - 뒤집기

```
alphabet = ['b', 'c', 'a', 'd']
alphabet.reverse()
print(alphabet) # ['d', 'a', 'c', 'b']


korean = ['강', '이', '박', '최']
korean.reverse()
print(korean) # ['최', '박', '이', '강']
```

리스트 함수 – 추가/삭제

- 요소 이름으로 추가/삭제
- append() : 맨 마지막에 요소 추가
- remove() : 리스트에서 첫 번째로 나오는 요소 삭제

```
shop = ["a", "b", "c", "d"]


# 추가
shop.append("e")
print(shop) # ['a', 'b', 'c', 'd', 'e'] : 마지막 위치에 추가


# 삭제
shop.remove("b")
print(shop) # ['a', 'c', 'd', 'e']
```

리스트 함수 – 추가/삭제

- 요소 번호로 추가/삭제 – 원하는 위치에 추가, 삭제 가능!
- insert(idx, item) : idx번째 위치에 item을 삽입

```
shop = ["a", "c", "d", "e"]


# 추가
shop.insert(1, "b") # ['a', 'b', 'c', 'd', 'e']
print(shop)
```

리스트 함수 – 추가/삭제

- 요소 번호로 추가/삭제 – 원하는 위치에 추가, 삭제 가능!
- pop(): 맨 마지막 요소 삭제
  - pop(idx): idx번째 인덱스 위치의 요소 삭제

```
shop = ["반팔", "청바지", "이어폰", "키보드"]

shop.pop(1)
print(shop) # ['반팔', '이어폰', '키보드']
shop.pop()
print(shop) # ['반팔', '이어폰']
```

- del list_name[idx] : idx번째 위치의 값을 삭제
  - del list_name[idx:idx] : 슬라이싱으로 여러 요소 한 번에 삭제 가능

```
shop = ["a", "b", "c", "d", "e"]

# 삭제
del shop[1] # ['a', 'c', 'd', 'e']
print(shop)
```

리스트 함수 – 위치 찾기

- 만약에 리스트 개수가 100개가 넘는다면,, 중간 정도에 있는 아이템을 찾기 위해 다 셀 수 없음!
- index(item): item이 몇 번째 위치에 있는지 값을 반환
  - 중복된 값이 있으면 제일 앞에 위치 반환

```
shop = ["a", "b", "c", "d", "e"]
print(shop.index("c")) # 2
print(shop.index("없는값")) # 존재 하지 않는 값: value error
```

리스트 함수 – 개수 세기

- 중복된 상품명, 동명이인도 상품코드나 출석 번호가 다르면 다른 것으로 판단
- count(item) : 같은 이름 가진 요소 개수 세기

```
a = ["a", "b", "c", "b"]
a.count("b") # 2
```

조건문

Boolean 자료형

- Boolean (불, 불린, 불리언 등으로 불림) 자료형
  - True
  - False
  - 대소문자 주의! (소문자 X)
- True/False 결과가 나오는 연산에서 볼 수 있음

=> 비교 = 부등호 사용하는 수식 등,
비교 연산자를 통해 Boolean 결과를 만듦

Boolean 자료형

- False만이 False를 의미하진 않는다!

-> False 데이터 타입만 거짓을 의미하지 않는다는 소리

- None, 숫자 0 (0.0 포함), 빈 문자열/리스트 등 모두 False 로 변환 됨

-> 그 외에는 모두 True값 의미

```
print(bool(1)) # True
print(bool(-2)) # True
print(bool(0)) # False
print(bool(None)) # False
print(bool("")) # False
print(bool("hi")) # True
```

* bool() : 다른 자료형을 Boolean 자료형으로 변환

비교 연산자

- `==` 같다
- `!=` 다르다
- `<` 작다
- `>` 크다
- `<=` 작거나 같다
- `>=` 크거나 같다

```
print(1 == 2)
print(1 != 2)
print(1 < 2)
print(1 > 2)
print(1 <= 2)
print(1 >= 2)

'''
False
True
True
False
True
False
'''
```

비교 연산자

- 동시에 비교 연산 가능! (다른 언어에서는 지원 안하는 경우가 많다)
- 문자 데이터에서의 비교 연산자
  - 사전 순서, 정렬 순서로 비교!

```
print("가" == "하")
print("가" != "하")
print("가" < "하")
print("가" > "하")

'''
False
True
True
False
'''
```

```
x = 3
print(1 < x < 5)
print(3 < x < 10)

# True
# Flase
```

논리 연산자

- Boolean 데이터 끼리 논리 연산자 사용 가능!
- not : 부정
- and : 그리고 -> 둘다 참이어야 참, 하나라도 거짓이면 거짓
- or : 또는 -> 둘 중 하나라도 참이면 참, 둘다 거짓이면 거짓

```
# not
print(not True)
print(not False)


age = 18
under_20 = age < 20
print("under_20: ", under_20)
print("not under_20: ", not under_20)
```

```
print(True and True)
print(True and False)
print(True or False)
print(False or False)
```

if 조건문

- '만약에' 라는 if의 의미처럼 조건을 걸 때 사용
- 컴퓨터에게 만약 OO 라면, OO 해줘 라는 명령을 내리는 것!

```
# 조건문 기본 형태
if 조건:
    실행할 코드
    실행할 코드
    ...
```

*주의) 조건 뒤에 콜론 (:) 반드시 붙이기,
들여쓰기(space 4칸- 권장사항) 하기!

- 들여쓰기로 if문 블록을 구분하기 때문!

if-else문

- 조건이 참일 때 실행하는 문장과 거짓일 때 실행하는 문장이 다를 때
- 두 가지의 경우의 수로만 나뉘는 경우 사용

```
# if-else문 기본 형태

if 조건:
    조건이 참일 때 실행하는 문장
    조건이 참일 때 실행하는 문장
    ...
else:
    조건이 거짓일 때 실행하는 문장
    조건이 거짓일 때 실행하는 문장
    ...
```

*세 가지 이상의 케이스로 나뉘는 경우에는??

-> if문 여러 개 사용?? X

-> elif 등장

if-elif-else문

- 조건이 2개 이상일 때 사용

```
# if-elif-else문 기본 형태

if 조건1:
    조건1이 참일 때 실행할 코드
    ...
elif 조건2:
    조건2이 참일 때 실행할 코드
    ...
elif 조건3:
    조건3이 참일 때 실행할 코드
    ...
...
else:
    모든 조건이 거짓일 때 실행할 코드
    ...
```

```
age = int(input("나이: "))
if age < 20:
    print("미성년자 입니다")
elif age < 30:
    print("20대 입니다!")
elif age < 40:
    print("30대 입니다~")
else:
    print("이제는 중년,,")
```

중첩 if문

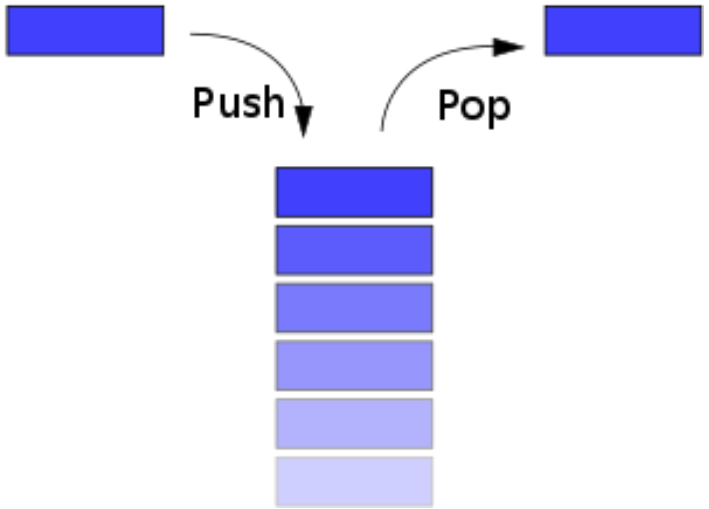
- if문 안에 if문 사용하는 경우

```
# 중첩 if문 기본 형태

if 조건1:
    조건1이 참일 때 실행할 코드
    if 조건1-1:
        조건1-1이 참일 때 실행할 코드
    ...
elif 조건2:
    조건2이 참일 때 실행할 코드
    if 조건2-1:
        조건2-1이 참일 때 실행할 코드
    ...
...
else:
    모든 조건이 거짓일 때 실행할 코드
    ...
```

```
age = int(input("나이: "))
gender = input("여 or 남: ")

if age < 20:
    print("미성년자 입니다")
elif age < 30:
    if gender == "여":
        print("20대 여성입니다~!")
    else:
        print("20대 남성입니다!")
elif age < 40:
    print("30대 입니다~")
else:
    print("이제는 중년,,")
```

추가) 스택

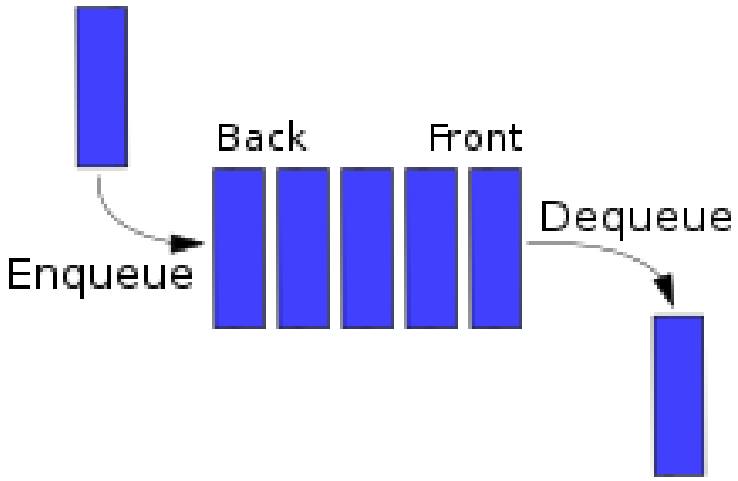
- LIFO(Last In First Out)
- Ex. 접시 쌓기 (가장 마지막에 쌓은 접시를 가장 먼저 꺼내야 한다.)




```python
stk = [1, 2, 3, 4, 5]

# 스택에 추가
stk.append(6)
print(stk) # [1, 2, 3, 4, 5, 6]

# 스택에서 제거
stk.pop()
print(stk) # [1, 2, 3, 4, 5]
```

추가) 큐

- FIFO(First In First Out)
- Ex. 버스 줄 서기 (먼저 온 사람이 먼저 버스에 탄다.)




```python
q = [1, 2, 3, 4, 5]

# 큐에 추가
q.append(6)
print(q) # [1, 2, 3, 4, 5, 6]

# 큐에서 제거
q.pop(0)
print(q) # [2, 3, 4, 5, 6]
```

```python
from collections import deque

q = deque([1, 2, 3, 4, 5])
print(q) # deque([1, 2, 3, 4, 5])

q.append(6)
print(q) # deque([1, 2, 3, 4, 5, 6])

q.popleft()
print(q) # deque([2, 3, 4, 5, 6])
```