# input 함수 사용하기

input()     # input 함수는 사용자가 입력한 값을 가져오는 함수

# input 함수의 결과를 변수에 할당하기

# 변수 = input()

# 변수 = input('문자열')  이 문자열은 사용자에게 입력받는 값의 용도를 미리 알려줄 때 사용하면 유용하다. 프롬프트(prompt)라고도 함

# input에서 입력받은 값은 항상 문자열(str)이다. 그리서 입력 값를 원하는 type로 변환해야 한다

# 변수 = 형변환타입(input())

# 변수 = int(input())          입력받은 문자열(str)를 정수형으로 형변환

# 변수 = float(input())        입력받은 문자열(str)를 실수형으로 형변환

# 입력 값을 변수 여러개에 저장하기

# 변수1, 변수2 = input().split()                  입력받은 값을 공백을 기준으로 분리하여 변수에 차례대로 할당.

# 변수1, 변수2 = input().split('기준문자열')      기준문자열(, . / 등)를 기준으로 분리하여 변수에 차례대로 할당.

# map을 사용하여 정수로 변환하기

# split의 결과를 매번 형변환해야 할 때 map을 함께 사용하면 됩니다.  map(int, input().split())    split의 결과를 모두 int로 변환해줌

# 변수1, 변수2 = map(형변환타입, input().split())

# 변수1, 변수2 = map(형변환타입, input().split('기준문자열'))

# 언패킹(unpacking) 응용

x = input().split()  # 10 20

a, b = x             # a, b = input().split()과 같음 - 언패킹(unpacking)

print(a, b)          # 10 20

##불(boolean) : 참(True), 거짓(False)을 나타내는

참, 거짓을 판단해야 할 때 사용

비교, 논리 연산자의 결과 판단에 자주 사용

- if, while 조건 판단할 때, 값이나 객체을 비교 할때, 부등호 결과 때,

a and     두 값이 모두 True라야 True입니다. 하나라도 False이면 False

a or b    두 값 중 하나라도 True이면 True입니다. 두 값이 모두 False라야 False

not x     논리값을 반대로 출력. not True는 False가 되고, not False는 True가 된다.

bool(정수, 실수, 문자열)

숫자는 0외 모든 숫자는 True. 빈 문자열 '', ""를 제외한 모든 문자열은 True입니다.

단락 평가(short-circuit evalution)

첫 번째 값만으로 결과가 확실할 때 두 번째 값은 확인(평가)하지 않는 방법을 말합니다.

and 연산자의 경우 두 값이 모두 참이라야 참이므로 첫 번째 값이 거짓이면 두 번째 값은 확인하지 않고 바로 거짓으로 결정합니다.

파이썬에서 논리 연산자는 마지막으로 단락 평가를 실시한 값을 그대로 반환한다.

따라서 논리 연산자는 무조건 불을 반환하지 않습니다.

A(T) and B(T)  =>  B값을 반환

A(T) and B(F)  =>  B값을 반환

A(F) and B(T)  =>  A값을 반환

A(F) and B(F)  =>  A값을 반환

A(T) or B(T)  =>  A값을 반환

A(T) or B(F)  =>  A값을 반환

A(F) or B(T)  =>  B값을 반환

A(F) or B(F)  =>  B값을 반환

True and 'Python'의 결과는 무엇이 나올까요?

>>> True and 'Python'  #'Python'

문자열 'Python'도 불로 따지면 True라서 True and True가 되어 True가 나올 것 같지만 'Python'이 나왔습니다.

여기서는 문자열 'Python'을 True로 쳐서 and 연산자가 두 번째 값까지 확인하므로 두 번째 값이 반환됩니다.

>>> 'Python' and True    # True

>>> 'Python' and False   # False

만약 다음과 같이 and 연산자 앞에 False나 False로 치는 값이 와서 첫 번째 값 만으로 결과가 결정나는 경우에는 첫 번째 값이 반환됩니다.

>>> False and 'Python'   #False

>>> 0 and 'Python'       # 0은 False이므로 and 연산자는 두 번째 값을 평가하지 않음

or 연산자에서 첫 번째 값만으로 결과가 결정되므로 첫 번째 값이 반환됩니다.

>>> True or 'Python'    # True

>>> 'Python' or True    # 'Python'

만약 두 번째 값까지 판단해야 한다면 두 번째 값이 반환됩니다.

>>> False or 'Python'   # 'Python'

>>> 0 or False          # False

시퀀스(sequence) 자료형

시퀀스 자료형(sequence types) : 값이 연속적(sequence)으로 이어져 있는 자료형.

list, tuple, range, str에 주로 사용하며 bytes, bytearray라는 자료형도 있습니다

시퀀스 자료형의 공통 기능

시퀀스 자료형으로 만든 객체를 시퀀스 객체

시퀀스 객체에 들어있는 각 값을 요소(element)라고 부릅니다.

특정 값이 있는지 확인하기(튜플, range, 문자열도 동일함)

값 in 시퀀스객체 : 시퀀스 객체 안에 특정 값이 있는지 확인(특정 값이 있으면 True, 없으면 False가 나옴)

a = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90]

30 in a   # True

100 in a  # False

값 not in 시퀀스객체 : 반대로 in 앞에 not을 붙이면 특정 값이 없는지 확인(특정 값이 없으면 True, 있으면 False가 나옴.)

100 not in a  # True

43 in (38, 76, 43, 62, 19)    # 튜풀

True

1 in range(10)                # range

True

'P' in 'Hello, Python'        # 문자열

True

시퀀스 객체 연결하기

시퀀스 객체는 + 연산자를 사용하여 객체를 서로 연결하여 새 객체를 만들 수 있습니다.

시퀀스객체1 + 시퀀스객체2

a = [0, 10, 20, 30]

b = [9, 8, 7, 6]

a + b                 # [0, 10, 20, 30, 9, 8, 7, 6]

리스트 a와 b를 더하니 두 리스트가 연결되었습니다. 물론 변수를 만들지 않고 리스트 여러 개를 직접 연결해도 상관없습니다.

range는 + 연산자로 객체를 연결할 수 없다.

이때는 range를 리스트 또는 튜플로 만들어서 연결하면 됩니다.

list(range(0, 10)) + list(range(10, 20))

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]

문자열은 + 연산자로 여러 문자열을 연결할 수 있습니다.

'Hello, ' + 'world!'

'Hello, world!'

참고 | 문자열에 숫자 연결하기 - str()을 사용하여 숫자를 문자열로 변환.

'문자열' + str(숫자)

'Hello, ' + str(10)      # str을 사용하여 정수를 문자열로 변환

시퀀스 객체 반복하기

\* 연산자는 시퀀스 객체를 특정 횟수만큼 반복하여 새 시퀀스 객체를 만듭니다

(0 또는 음수를 곱하면 빈 객체가 나오며 실수는 곱할 수 없습니다).

시퀀스 객체 반복하기

시퀀스객체 \* 정수

range는 리스트 또는 튜플로 만들어서 반복하면 됩니다.

list(range(0, 5, 2)) \* 3

len(시퀀스객체) : 시퀀스 객체에는 요소의 개수(길이)를 구하는 함수 - len(length)

range에 len 함수를 사용하면 숫자가 생성되는 개수를 구합니다.

len(range(0, 10, 2))    # 5

문자열에 len 함수를 사용하면 길이(문자의 개수)를 구함

hello = 'Hello, world!'

len(hello)    # 13(문자열의 길이는 공백까지 포함)

참고 | UTF-8 문자열의 바이트 수 구하기

hello = '안녕하세요'

len(hello.encode('utf-8'))    # 15

UTF-8에서 한글 글자 하나는 3바이트로 표현하므로 '안녕하세요'가 차지하는 실제 바이트 수는 15바이트입니다.

참고로 파이썬 3에서 len은 문자의 개수를 구해주지만 파이썬 2.7에서는 실제 바이트 수를 구해주는 차이점이 있습니다.

즉, 한글 문자열의 길이를 구할 때 파이썬 버전에 따라 결과가 달라지므로 주의해야 합니다.

인덱스 : 시퀀스 객체 요소들의 순서(인덱스은 0번 부터 시작한다)

시퀀스객체[인덱스] : [ ] 안에 접근할 요소의 인덱스를 지정하면 해당 요소에 접근

인덱스(index)는 위치 값을 뜻하며 인덱스는 항상 0부터 시작한다

a = [38, 21, 53, 62, 19]

a[0]    # 리스트의 첫 번째(인덱스 0) 요소 38출력

a[2]    # 리스트의 세 번째(인덱스 2) 요소 53출력

시퀀스 객체에 인덱스를 지정하지 않으면 해당 객체 전체를 뜻한다.

a    # 시퀀스 객체에 인덱스를 지정하지 않으면 전체 [38, 21, 53, 62, 19]를 호출

음수로 인덱스 지정하기 : 시퀀스 객체에 인덱스를 음수로 지정하면 뒤에서부터 요소에 접근

(-1은 뒤에서 첫 번째(마지막 요소), -5는 뒤에서 다섯 번째 요소)

len(a) - 1로 시퀀스 객체의 마지막 인덱스의 위치를 구할 수 있다

요소에 값 할당하기 : 시퀀스 객체는 [ ]로 요소에 접근한 뒤 =로 값을 할당.

시퀀스객체[인덱스] = 값

del로 요소 삭제하기 : 시퀀스 객체의 [ ]로 요소에 접근한 뒤 요소를 삭제

del 시퀀스객체[인덱스]

시퀀스 자료형 중에서 튜플, range, 문자열은 읽기 전용으로 요소에 값을 할당, 삭제 할수 없다.

참고 | \_\_getitem\_\_ 메서드

시퀀스 객체에서 [ ]를 사용는 실제로 \_\_getitem\_\_ 메서드를 호출하는 기능으로 객체의 요소를 가져옴

시퀀스객체.\_\_getitem\_\_(인덱스) : 인덱스로 접근할 수 있는 이터레이터 이다

a.\_\_getitem\_\_(1)    # a시퀀스 객체의 1번 인덱스 21호출

a[1]                # a시퀀스 객체의 1번 인덱스 21호출

슬라이스(slice) : 시퀀스 객체의 일부를 잘라내기.

[ ] 안에 시작 인덱스와 끝 인덱스를 지정하면 해당 범위의 리스트를 잘라서 가져올 수 있습니다.

시퀀스객체[시작인덱스:끝인덱스:인덱스증가폭]

인덱스 생략 : 시작 인덱스(0인텍스) 끝 인덱스(len(a)) 증가폭(1)와 같다

시퀀스객체[:] - a[:],(a[::]) 인덱스를 생략하면 a객체 전체가 설정됨.

len() 응용하기 - 인덱스의 범위를 알수 없을 때 유용함.

a[:len(a)]     # 시작 인덱스 생략, 끝 인덱스에 len(a) 지정하여 리스트 전체를 가져옴

인덱스 증가폭을 음수로 지정하면 요소를 뒤에서부터 가져올 수 있습니다.

a[::-1]     # 인덱스 증가폭만 -1로 지정하면 객체 전체를 반대로 가져온다

            # 주의 - 시작인덱스가 증가폭만큼(-1) 감소하므로 끝 인덱스보다 시작 인덱스를 더 크게 지정해야 한다.

시퀀스 객체는 슬라이스로 범위를 지정하여 여러 요소에 값을 할당할 수 있다.

시퀀스객체[시작인덱스:끝인덱스] = 시퀀스객체

a = list(range(0,100,10))

a[2:5] = ['a', 'b', 'c']    # 인덱스 2, 3, 4에 'a', 'b', 'c'값을 할당

print(a)                    # [0, 10, 'a', 'b', 'c', 50, 60, 70, 80, 90]

슬라이스 범위와 할당할 리스트의 요소 개수 다른 경우

할당할 요소 개수가 적으면 그만큼 리스트의 요소 개수도 줄어듭니다.

a[2:5] = ['a']    # 인덱스 2부터 4까지에 값 1개를 할당하여 요소의 개수가 줄어듦

a                 # [0, 10, 'a', 50, 60, 70, 80, 90]

할당할 요소 개수가 많으면 그만큼 리스트의 요소 개수도 늘어납니다.

a[2:5] = ['a', 'b', 'c', 'd', 'e'] # 인덱스 2부터 4까지 값 5개를 할당하여 요소의 개수가 늘어남

a                                  # [0, 10, 'a', 'b', 'c', 'd', 'e', 50, 60, 70, 80, 90]

인덱스 증가폭을 지정하여 인덱스를 건너뛰면서 할당해보겠습니다.

시퀀스객체[시작인덱스:끝인덱스:인덱스증가폭] = 시퀀스객체 (범위의 요소와 할당할 요소 개수가 정확히 일치해야 한다)

a[2:8:2] = ['a', 'b', 'c']    # 인덱스 2부터 2씩 증가시키면서 인덱스 7까지 값 할당

a                             # [0, 10, 'a', 30, 'b', 50, 'c', 70, 80, 90]

del로 슬라이스 삭제하기(원래 있던 리스트가 변경)

del 시퀀스객체[시작인덱스:끝인덱스]

a = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90]

del a[2:5]    # 인덱스 2부터 4까지 요소를 삭제

a             # [0, 10, 50, 60, 70, 80, 90]

인덱스 증가폭을 지정하면 인덱스를 건너뛰면서 삭제

a = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90]

del a[2:8:2]    # 인덱스 2부터 2씩 증가시키면서 인덱스 6까지 삭제

a               # [0, 10, 30, 50, 70, 80, 90]

------------  이런게 사용도 할 수 있음 ----------------

range()에 슬라이스를 사용하면 지정된 범위의 숫자를 생성하는 range 객체를 새로 만듭니다.

range객체[시작인덱스:끝인덱스:인덱스증가폭]

r1 = range(10)           # r1에 range(10) 객체를 생성 (이후 r1은 range(10)와 동일한 기능을 한다)

r2 = list(r1[::2])       # r2에 r1객체에서 [::2] 범위만큼 리스트로 만들어서 r2에 저장

print(r2)                # [0, 2, 4, 6, 8] 객체 생성. r2는 list(range(0, 10, 2))와 같다.

slice() 객체를 사용하여 시퀀스 객체(시퀀스 자료형으로 만든 변수)를 잘라낼 수도 있다.

슬라이스객체 = slice(시작인덱스, 끝인덱스, 인덱스증가폭)

시퀀스객체[슬라이스객체]

시퀀스 객체의 [ ] 또는 \_\_getitem\_\_ 메서드에 slice 객체를 넣어주면 지정된 범위만큼 잘라내서 새 객체를 만듭니다.

시퀀스객체.\_\_getitem\_\_(슬라이스객체)

range(4, 7, 2) == range(10)[slice(4, 7, 2)] == range(10).\_\_getitem\_\_(slice(4, 7, 2)) 같은 기능

range() 슬라이스와 slice() 객체를 만든 뒤 여러 시퀀스 객체에 사용하는 방법도 가능.

a = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90]

s = slice(4, 7)    # 인덱스 4부터 6까지 자르는 slice 객체 생성

print(a[s])        # [40, 50, 60] - s(변수)에 slice()기능을 저장 후 a객체에 적용하는 방식.

-------------- ps : 별루 인거 같음 ---------------------

Variable, 연산자

산술 연산자

연산자 / 기능 / 문법 / 설명

+ 덧셈 a + b 두 값을 더함

- 뺄셈 a - b a에서 b를 뺌

\* 곱셈 a \* b 두 값을 곱함

/ 나눗셈 a / b a에서 b를 나누며 결과는 실수

// 버림 나눗셈 (floor division) a // b a에서 b를 나누며 소수점 이하는 버림

% 나머지 a % b a에서 b를 나누었을 때 나머지를 구함

\*\* 거듭제곱 a \*\* b a를 b번 곱함

@ 행렬 곱셈 a @ b 행렬 a와 b를 곱함

+ 양수 부호 +a a에 양수 부호를 붙임

- 음수 부호 -a a에 음수 부호를 붙임

+= 덧셈 후 할당 a += b a와 b를 더한 후 결과를 a에 할당

-= 뺄셈 후 할당 a -= b a에서 b를 뺀 후 결과를 a에 할당

\*= 곱셈 후 할당 a \*= b a와 b를 곱한 후 결과를 a에 할당

/= 나눗셈 후 할당 a /= b a에서 b를 나눈 후 결과를 a에 할당(결과는 실수)

//= 버림 나눗셈 후 할당 a //= b a에서 b를 나눈 후 결과를 a에 할당(소수점 이하는 버림)

%= 나머지 연산 후 할당 a %= b a에서 b를 나누었을 때 나머지를 구하여 a에 할당

\*\*= 거듭제곱 후 할당 a \*\*= b a를 b번 곱한 후 결과를 a에 할당

@= 행렬 곱셈 후 할당 a @= b 행렬 a와 b를 곱한 후 결과를 a에 할당

핵심 정리

숫자의 덧셈은 +, 뺄셈은 -, 곱셈은 \*, 나눗셈은 / 연산자를 사용합니다.

계산 결과를 정수, 실수로 만들기

int(5 / 2)      # 실수 2.5를 정수 2로 만듦

float(1 + 2)    # 정수 3을 실수 3.0으로 만듦

괄호 사용하기

덧셈, 뺄셈, 곱셈, 나눗셈이 함께 있을 때는 곱셈과 나눗셈부터 계산합니다.

만약 곱셈보다 덧셈을 먼저 계산하고 싶다면 덧셈 부분을 ( )로 묶어줍니다.

변수 만들기

변수는 변수이름 = 값 형식으로 만듭니다.

x = 10

변수 삭제하기

변수를 삭제할 때는 del 변수 형식으로 삭제합니다.

x = 10

del x

변수 여러 개 만들기

변수이름1, 변수이름2, 변수이름3 = 값1, 값2, 값3 (변수를 ,(콤마)로 구분한 뒤 각 변수에 할당될 값을 지정)

a, b, c = 10, 20, 30

변수1 = 변수2 = 변수3 = 값 (변수 여러 개를 =로 연결하고 값을 할당하면 같은 값을 가진 변수가 만들어짐)

x = y = z = 10

산술 연산 후 할당 연산자 사용하기(미리 변수가 있어야함)

산술 연산자 앞에 =(할당 연산자)를 붙이면 연산 결과를 변수에 저장합니다.

a = 10

a += 20

객체의 자료형 알아내기

객체(변수)가 어떤 타입(자료형)인지 알고 싶을 때는 type 함수를 사용합니다.

type(3.3)

<class 'float'>

x = 10

type(x)

<class 'int'>

파이썬에서는 숫자도 객체(object)이며, 객체는 클래스(class)로 표현합니다.

# Variable

# 변수이름 = 값형식  (변수에 값을 할당(assignment)), 수학의 등호와 같은 역할을 하는 연산자는 ==입니다

# 변수이름 만들 때 주의 사항

# 영문(대소문자를 구분), 숫자(숫자부터 시작하면 안 됨), \_(밑줄 문자)로 시작할 수 있음

# 특수 문자(+, -, \*, /, $, @, &, % 등), 파이썬의 키워드(if, for, while, and, or 등)는 사용할 수 없습니다.

# 변수의 자료형 알아내기

# type(변수)

# 변수 여러 개를 한 번에 만들기

# 변수이름1, 변수이름2, 변수이름3 = 값1, 값2, 값3  (변수와 값을 ,(콤마)로 구분하면 각 변수에 값이 할당됨)

# 변수와 값의 개수는 동일하게 맞춰주어야 하며 나열된 순서대로 값이 할당됨

x, y, z = 10, 20, 30

# 변수와 값의 개수는 동일하게 맞춰주어야 하며 나열된 순서대로 값이 할당 가능

# 변수1 = 변수2 = 변수3 = 값 형식

x = y = z = 10

# 변수1, 변수2 = 변수2, 변수1 형식으로 두 변수의 값을 바꿀 수 있다

x, y = y, x

# 변수 삭제하기

# del 변수

# 빈 변수 만들기

# 변수 = None

# 파이썬에서 None은 아무것도 없는 상태를 나타내는 자료형. 다른 언어에서는 널(null)이라고 표현

# 산술 연산 후 값 할당

# 변수 a의 값을 20 증가시키 경우

a = 10

a + 20                # a은 10의 값 그대로 이다(a + 20의 결과 값을 어느곳에도 할당 하지 않아서)

a = a + 20            # a와 20을 더한 후 결과를 다시 a에 저장

a += 20               # a와 20을 더한 후 결과를 다시 a에 저장

# a = a + 20과 같이 a에 20을 더한 값을 다시 a에 다시 할당(변수를 두 번 입력하지 않고 산술 연산 후 할당 연산자를 제공)

# 덧셈(+=), 뺄셈(-=), 곱셈(\*=), 나눗셈(/=, //=), 나머지(%=)

# d += 10 처렴 만들지 않은 변수 d에 10을 더한 후 다시 할당하면 에러가 발생

# 부호 붙이기

# 계산을 하다 부호를 붙여야 하는 경우 값이나 변수 앞에 양수, 음수 부호를 붙이면 됩니다

x = -10

+x     # (-10에 + 부호를 붙이면 부호의 변화 없음)

# -10

-x     # (-10에 - 부호를 붙이면 양수 10이 됨)

# 10

tuple : 여러 데이터를 저장 할 수 있는 컨테이너형 변수

- 특징 immutable, indexing, slicing, 시퀀스형

- 튜플은 요소를 일렬로 저장하지만, 요소를 변경, 추가, 삭제를 할 수 없습니다.

튜플 생성 : ( )에 각 값을  ,(콤마)로 구분, 또는 괄호로 묶지 않고 값만 콤마로 구분. 각 값을 요소(element)라함

튜플은 문자열, 정수, 실수, 불 등 모든 자료형을 저장할 수 있으며 자료형을 사용가능

보통 튜플은 요소가 절대 변경되지 않고 유지되어야 할 때 사용. 요소를 변경하게 되면 에러가 발생

튜플 = (값, 값, 값), 튜플 = 값, 값, 값

요소가 한 개인 튜플을 만들 때

( ) 안에 값 한 개를 넣고 ,(콤마)를 붙임. 또는, 괄호로 묶지 않고 값 한 개에 ,를 붙임.

튜플 = (값, ), 튜플 = 값,

range를 사용하여 튜플 만들기

튜플 = tuple(range(횟수))

튜플을 리스트로 만들고 리스트를 튜플로 만들기

튜플과 리스트는 요소를 변경, 추가, 삭제할 수 없는거만 다를 뿐 기능과 형태는 같다.

따라서 튜플을 리스트로 만들거나 리스트를 튜플로 만들 수도 있습니다.

tuple 안에 리스트를 넣으면 새 튜플이 생성.

a = [1, 2, 3]

tuple(a)            # (1, 2, 3) 튜플생성

반대로 list 안에 튜플을 넣으면 새 리스트가 생성.

b = (4, 5, 6)

list(b)             # [4, 5, 6] 리스트생성

참고  tuple(list)안에 문자열을 넣으면? 문자 튜플(문자 리스트)이 생성.

list('Hello')     # ['H', 'e', 'l', 'l', 'o']

tuple('Hello')    # ('H', 'e', 'l', 'l', 'o')

참고  튜플(리스트)을 사용헤 변수 여러 개를 한 번에 만들 수 있음. 변수의 개수와 튜플(리스트))의 요소 개수는 같아함.

a, b, c = [1, 2, 3]

print(a, b, c)        # 1 2 3

d, e, f = (4, 5, 6)

print(d, e, f)        # 4 5 6

튜플(리스트) 변수로도 변수 여러 개를 만들 수 있다. - 튜플 언패킹(tuple unpacking),

y = (4, 5, 6)        # 튜플 패킹

d, e, f = y          # 튜플 언패킹(tuple unpacking)

print(d, e, f)       # 4 5 6

x = [1, 2, 3]        # # 리스트 패킹

a, b, c = x          # 리스트 언패킹(list unpacking)

print(a, b, c)       # 1 2 3

튜플 패킹(tuple packing), 리스트 패킹(list packing)은 변수에 튜플(리스트)을 할당하는 과정을 뜻함.

a = [1, 2, 3]    # 리스트 패킹

b = (1, 2, 3)    # 튜플 패킹

c = 1, 2, 3      # 튜플 패킹

리스트와 튜플 응용하기

'Unit 10 리스트와 튜플 사용하기'에서 리스트의 기본적인 사용 방법을 알아보았습니다.

파이썬의 리스트는 생각보다 기능이 많은데, 요소를 추가/삭제하거나, 정보를 조회하는 메서드(함수)도 제공합니다.

특히 for 반복문과 결합하면 연속적이고 반복되는 값을 손쉽게 처리할 수 있습니다.

22.1 리스트 조작하기

먼저 리스트를 조작하는 메서드(method)입니다(메서드는 객체에 속한 함수를 뜻하는데,

                            자세한 내용은 '34.1 클래스와 메서드 만들기'에서 설명하겠습니다).

                            파이썬에서 제공하는 리스트 메서드는 여러 가지가 있지만 여기서는 자주 쓰는 메서드를 다루겠습니다.

22.1.1  리스트에 요소 추가하기

리스트의 대표적인 기능이 바로 요소 추가입니다. 다음과 같이 리스트에 요소를 추가하는 메서드는 3가지가 있습니다.

append: 요소 하나를 추가

extend: 리스트를 연결하여 확장

insert: 특정 인덱스에 요소 추가

22.1.2  리스트에 요소 하나 추가하기

append(요소)는 리스트 끝에 요소 하나를 추가합니다.

다음은 리스트 [10, 20, 30]에 500을 추가하여 리스트는 [10, 20, 30, 500]이 됩니다(메서드를 호출한 리스트가 변경되며 새 리스트는 생성되지 않음).

>>> a = [10, 20, 30]

>>> a.append(500)

>>> a

[10, 20, 30, 500]

>>> len(a)

4

▼ 그림 22-1 append로 리스트 끝에 요소 하나 추가

물론 빈 리스트에 값을 추가할 수도 있습니다.

>>> a = []

>>> a.append(10)

>>> a

[10]

22.1.3  리스트 안에 리스트 추가하기

append는 append(리스트)처럼 리스트를 넣으면 리스트 안에 리스트가 들어갑니다.

다음은 리스트 a 안에 [500, 600]이 들어가서 중첩 리스트가 됩니다.

>>> a = [10, 20, 30]

>>> a.append([500, 600])

>>> a

[10, 20, 30, [500, 600]]

>>> len(a)

4

a.append([500, 600])은 [500, 600]이라는 요소 하나를 리스트 a 끝에 추가합니다.

따라서 리스트 a를 len으로 길이를 구해보면 5가 아닌 4가 나옵니다.

▼ 그림 22-2 append로 리스트 안에 리스트 추가하기

즉, append는 항상 리스트의 길이가 1씩 증가합니다.

22.1.4  리스트 확장하기

그러면 리스트에 요소를 여러 개 추가하려면 어떻게 해야 할까요? append를 여러 번 사용하는 방법도 있지만,

추가할 요소가 많은 경우에는 상당히 번거롭습니다. 이때는 extend를 사용합니다.

extend(리스트)는 리스트 끝에 다른 리스트를 연결하여 리스트를 확장합니다.

다음은 리스트 [10, 20, 30]에 다른 리스트 [500, 600]을 연결하여 [10, 20, 30, 500, 600]이 됩니다

(메서드를 호출한 리스트가 변경되며 새 리스트는 생성되지 않음).

>>> a = [10, 20, 30]

>>> a.extend([500, 600])

>>> a

[10, 20, 30, 500, 600]

>>> len(a)

5

extend를 사용하면 리스트의 길이는 extend에 전달된 리스트의 길이만큼 증가합니다.

따라서 길이가 3인 리스트 [10, 20, 30]에 길이가 2인 [500, 600]을 연결했으므로 길이는 5가 나옵니다.

▼ 그림 22-3 extend로 리스트 확장하기

extend의 동작을 좀 더 정확하게 이야기하면 extend에 전달된 [500, 600]의 요소를 반복하면서 각 요소를 리스트 a에 추가하는 것입니다.

따라서 리스트와 리스트를 연결한 모양이 됩니다.

22.1.5  리스트의 특정 인덱스에 요소 추가하기

append, extend는 리스트 끝에 요소를 추가합니다. 그러면 원하는 위치에 요소를 추가하는 방법은 없을까요? 이때는 insert를 사용합니다.

insert(인덱스, 요소)는 리스트의 특정 인덱스에 요소 하나를 추가합니다.

다음은 리스트 [10, 20, 30]의 인덱스 2에 500을 추가하여 [10, 20, 500, 30]이 됩니다.

>>> a = [10, 20, 30]

>>> a.insert(2, 500)

>>> a

[10, 20, 500, 30]

>>> len(a)

4

▼ 그림 22-4 insert로 특정 인덱스에 요소 추가

insert에서 자주 사용하는 패턴은 다음 2가지입니다.

insert(0, 요소): 리스트의 맨 처음에 요소를 추가

insert(len(리스트), 요소): 리스트 끝에 요소를 추가

다음은 리스트 [10, 20, 30]의 맨 처음에 500을 추가합니다.

>>> a = [10, 20, 30]

>>> a.insert(0, 500)

>>> a

[500, 10, 20, 30]

그리고 insert에 마지막 인덱스보다 큰 값을 지정하면 리스트 끝에 요소 하나를 추가할 수 있습니다.

다음은 리스트 [10, 20, 30] 끝에 500을 추가합니다.

>>> a = [10, 20, 30]

>>> a.insert(len(a), 500)

>>> a

[10, 20, 30, 500]

len(리스트)는 마지막 인덱스보다 1이 더 크기 때문에 리스트 끝에 값을 추가할 때 자주 활용합니다.

사실 이 방법은 a.append(500)과 같습니다.

특히 insert는 요소 하나를 추가하므로 insert에 리스트를 넣으면 append처럼 리스트 안에 리스트가 들어갑니다.

다음은 리스트 [10, 20, 30]의 인덱스 1에 리스트 [500, 600]을 추가하여 중첩 리스트가 됩니다.

>>> a = [10, 20, 30]

>>> a.insert(1, [500, 600])

>>> a

[10, [500, 600], 20, 30]

만약 리스트 중간에 요소 여러 개를 추가하고 싶다면 '11.4 슬라이스 사용하기'에서 배운 슬라이스에 요소 할당하기를 활용하면 됩니다.

다음은 리스트 [10, 20, 30]의 인덱스 1부터 500, 600을 추가하여 [10, 500, 600, 20, 30]이 됩니다.

>>> a = [10, 20, 30]

>>> a[1:1] = [500, 600]

>>> a

[10, 500, 600, 20, 30]

a[1:1] = [500, 600]과 같이 시작 인덱스와 끝 인덱스를 같게 지정하면 해당 인덱스의 요소를 덮어쓰지 않으면서 요소 여러 개를 중간에 추가할 수 있습니다.

22.1.6  리스트에서 요소 삭제하기

이번에는 리스트에서 요소를 삭제하는 방법입니다. 다음과 같이 요소를 삭제하는 메서드는 2가지가 있습니다.

pop: 마지막 요소 또는 특정 인덱스의 요소를 삭제

remove: 특정 값을 찾아서 삭제

22.1.7  리스트에서 특정 인덱스의 요소를 삭제하기

pop()은 리스트의 마지막 요소를 삭제한 뒤 삭제한 요소를 반환합니다.

다음은 리스트 [10, 20, 30]에서 pop으로 마지막 요소를 삭제한 뒤 30을 반환합니다. 따라서 리스트는 [10, 20]이 됩니다.

>>> a = [10, 20, 30]

>>> a.pop()

30

>>> a

[10, 20]

▼ 그림 22-5 pop으로 리스트의 마지막 요소 삭제

그러면 원하는 인덱스의 요소를 삭제할 수는 없을까요? 이때는 pop에 인덱스를 지정하면 됩니다.

pop(인덱스)는 해당 인덱스의 요소를 삭제한 뒤 삭제한 요소를 반환합니다.

다음은 리스트 [10, 20, 30]에서 인덱스 1을 삭제합니다.

>>> a = [10, 20, 30]

>>> a.pop(1)

20

>>> a

[10, 30]

사실 pop 대신 del을 사용해도 상관없습니다.

>>> a = [10, 20, 30]

>>> del a[1]

>>> a

[10, 30]

22.1.8  리스트에서 특정 값을 찾아서 삭제하기

pop이나 del은 인덱스로 요소를 삭제했는데, 리스트에서 원하는 값을 찾아서 삭제하고 싶을 수도 있습니다.

이런 경우에는 remove를 사용합니다.

remove(값)은 리스트에서 특정 값을 찾아서 삭제합니다. 다음은 리스트 [10, 20, 30]에서 20을 삭제하여 [10, 30]이 됩니다.

>>> a = [10, 20, 30]

>>> a.remove(20)

>>> a

[10, 30]

만약 리스트에 같은 값이 여러 개 있을 경우 처음 찾은 값을 삭제합니다.

>>> a = [10, 20, 30, 20]

>>> a.remove(20)

>>> a

[10, 30, 20]

리스트 a에 20이 2개 있지만 가장 처음 찾은 인덱스 1의 20만 삭제합니다.

▼ 그림 22-6 remove로 특정 값을 찾아서 삭제

참고 | 리스트로 스택과 큐 만들기

지금까지 알아본 리스트의 메서드로 스택(stack)과 큐(queue)를 만들 수 있습니다.

다음과 같이 append와 pop을 호출하는 그림을 90도 돌리면 스택의 모습이 됩니다.

여기서 pop() 대신 pop(0)을 사용하면 큐가 됩니다.

물론 append(), pop(0)이 아닌 insert(0, 요소), pop()을 사용해서 추가/삭제 방향을 반대로 해도 큐가 됩니다.

파이썬에서 스택은 리스트를 그대로 활용해도 되지만, 큐는 좀 더 효율적으로 사용할 수 있도록 덱(deque, double ended queue)이라는 자료형을 제공합니다.

덱은 양쪽 끝에서 추가/삭제가 가능한 자료 구조입니다.

deque(반복가능한객체)

>>> from collections import deque    # collections 모듈에서 deque를 가져옴

>>> a = deque([10, 20, 30])

>>> a

deque([10, 20, 30])

>>> a.append(500)    # 덱의 오른쪽에 500 추가

>>> a

deque([10, 20, 30, 500])

>>> a.popleft()     # 덱의 왼쪽 요소 하나 삭제

10

>>> a

deque([20, 30, 500])

deque의 append는 덱의 오른쪽에 요소를 추가하고, popleft는 덱의 왼쪽 요소를 삭제합니다.

반대로 appendleft는 덱의 왼쪽에 요소를 추가하고, pop으로 덱의 오른쪽 요소를 삭제할 수도 있습니다.

▼ 그림 22-7 리스트로 스택 만들기

▼ 그림 22-8 리스트로 큐 만들기

22.1.9  리스트에서 특정 값의 인덱스 구하기

index(값)은 리스트에서 특정 값의 인덱스를 구합니다. 이때 같은 값이 여러 개일 경우 처음 찾은 인덱스를 구합니다(가장 작은 인덱스).

 다음은 20이 두 번째에 있으므로 인덱스 1이 나옵니다.

>>> a = [10, 20, 30, 15, 20, 40]

>>> a.index(20)

1

▼ 그림 22-9 index로 특정 값의 인덱스 구하기

22.1.10                특정 값의 개수 구하기

count(값)은 리스트에서 특정 값의 개수를 구합니다. 다음은 리스트 [10, 20, 30, 15, 20, 40]에서 20의 개수를 구합니다.

>>> a = [10, 20, 30, 15, 20, 40]

>>> a.count(20)

2

리스트 a에는 20이 2개 들어있으므로 2가 나옵니다.

▼ 그림 22-10 count로 특정 값의 개수 구하기

22.1.11                리스트의 순서를 뒤집기

reverse()는 리스트에서 요소의 순서를 반대로 뒤집습니다.

다음은 리스트 [10, 20, 30, 15, 20, 40]의 순서를 반대로 뒤집어서 [40, 20, 15, 30, 20, 10]이 됩니다.

>>> a = [10, 20, 30, 15, 20, 40]

>>> a.reverse()

>>> a

[40, 20, 15, 30, 20, 10]

22.1.12                리스트의 요소를 정렬하기

sort()는 리스트의 요소을 작은 순서대로 정렬합니다(오름차순).

다음은 리스트 [10, 20, 30, 15, 20, 40]의 값을 작은 순서대로 정렬하여 [10, 15, 20, 20, 30, 40]이 됩니다.

sort() 또는 sort(reverse=False): 리스트의 값을 작은 순서대로 정렬(오름차순)

sort(reverse=True): 리스트의 값을 큰 순서대로 정렬(내림차순)

>>> a = [10, 20, 30, 15, 20, 40]

>>> a.sort()

>>> a

[10, 15, 20, 20, 30, 40]

참고 | sort 메서드와 sorted 함수

파이썬은 리스트의 sort 메서드뿐만 아니라 내장 함수 sorted도 제공합니다.

sort와 sorted 모두 정렬을 해주는 함수지만, 약간의 차이점이 있습니다.

 sort는 메서드를 사용한 리스트를 변경하지만, sorted 함수는 정렬된 새 리스트를 생성합니다.

>>> a = [10, 20, 30, 15, 20, 40]

>>> a.sort()    # a의 내용을 변경하여 정렬

>>> a

[10, 15, 20, 20, 30, 40]

>>> b = [10, 20, 30, 15, 20, 40]

>>> sorted(b)    # 정렬된 새 리스트를 생성

[10, 15, 20, 20, 30, 40]

22.1.13                리스트의 모든 요소를 삭제하기

clear()는 리스트의 모든 요소를 삭제합니다. 다음은 리스트 [10, 20, 30]의 모든 요소를 삭제하여 빈 리스트 []가 됩니다.

>>> a = [10, 20, 30]

>>> a.clear()

>>> a

[]

clear 대신 del a[:]와 같이 시작, 끝 인덱스를 생략하여 리스트의 모든 요소를 삭제할 수도 있습니다.

>>> a = [10, 20, 30]

>>> del a[:]

>>> a

[]

22.1.14                리스트를 슬라이스로 조작하기

리스트는 메서드를 사용하지 않고, 슬라이스로 조작할 수도 있습니다.

다음은 리스트 끝에 값이 한 개 들어있는 리스트를 추가합니다.

>>> a = [10, 20, 30]

>>> a[len(a):] = [500]

>>> a

[10, 20, 30, 500]

a[len(a):]는 시작 인덱스를 len(a)로 지정해서 리스트의 마지막 인덱스보다 1이 더 큰 상태입니다.

 즉, 그림과 같이 리스트 끝에서부터 시작하겠다는 뜻입니다(이때는 리스트의 범위를 벗어난 인덱스를 사용할 수 있습니다).

▼ 그림 22-11 a[len(a):]의 뜻

a[len(a):] = [500]과 같이 값이 한 개 들어있는 리스트를 할당하면 리스트 a 끝에 값을 한 개 추가하며 a.append(500)과 같습니다.

그리고 a[len(a):] = [500, 600]과 같이 요소가 여러 개 들어있는 리스트를 할당하면 리스트 a 끝에 다른 리스트를 연결하며 a.extend([500, 600])과 같습니다.

>>> a = [10, 20, 30]

>>> a[len(a):] = [500, 600]

>>> a

[10, 20, 30, 500, 600]

참고 | 리스트가 비어 있는지 확인하기

리스트(시퀀스 객체)가 비어 있는지 확인하려면 어떻게 해야 할까요? 방법은 간단합니다.

 리스트는 len 함수로 길이를 구할 수 있죠? 이걸 if 조건문으로 판단하면 리스트가 비어 있는지 확인할 수 있습니다.

if not len(seq):    # 리스트가 비어 있으면 True

if len(seq):        # 리스트에 요소가 있으면 True

하지만 파이썬에서는 이 방법보다 리스트(시퀀스 객체)를 바로 if 조건문으로 판단하는 방법을 권장합니다(PEP 8).

if not seq:    # 리스트가 비어 있으면 True

if seq:        # 리스트에 내용이 있으면 True

특히 리스트가 비어 있는지 확인하는 방법은 리스트의 마지막 요소에 접근할 때 유용하게 사용할 수 있습니다.

 리스트의 마지막 요소에 접근할 때는 인덱스를 -1로 지정하면 되죠?

>>> seq = [10, 20, 30]

>>> seq[-1]

30

만약 리스트가 비어 있을 경우에는 인덱스를 -1로 지정하면 에러가 발생합니다.

>>> a = []

>>> a[-1]

Traceback (most recent call last):

  File "<pyshell#3>", line 1, in <module>

    a[-1]

IndexError: list index out of range

이때는 if 조건문을 활용하여 리스트에 요소가 있을 때만 마지막 요소를 가져오면 됩니다.

seq = []

if seq:               # 리스트에 요소가 있는지 확인

    print(seq[-1])    # 요소가 있을 때만 마지막 요소를 가져옴