데이터 자료형 (시퀀스)

04강

리스트

- 리스트 = 목록 : 값을 일렬로 늘어놓은 형태
- 변수에 값을 저장할 때 **[ ]**(대괄호)로 묶어 줌
- 각 값은 **,**(콤마)로 구분
- **리스트 = [값, 값, 값]**
- 리스트에 저장된 각 값 = 요소(element)
- 문자열, 정수, 실수, 불 등 모든 자료형 저장 가능
- 자료형을 섞어서 저장 가능
- **리스트 = []**
- **리스트 = list()**
- **리스트 = list(range(횟수))**
- **리스트 = list(range(시작, 끝))**
- **리스트 = list(range(시작, 끝, 증가폭))**

```
a = [38, 21, 53, 62, 19]

person = ['james', 17, 175.3, True]

a = []

b = list()

a = list(range(10))
b = list(range(5, 12))
c = list(range(-4, 10, 2))
d = list(range(10, 0, -1))
```

튜플

- 리스트처럼 요소를 일렬로 저장
- 저장된 **요소를 변경, 추가, 삭제를 할 수 없음**
- 읽기 전용 리스트
- **( )**(괄호)로 묶어 줌
- **,**(콤마)로 구분
- 괄호로 묶지 않고 값만 콤마로 구분해도 튜플이 됨
- **튜플 = (값, 값, 값)**
- **튜플 = 값, 값, 값**
- 요소가 절대 변경되지 않고 유지되어야 할 때 사용
- **튜플 = (값,)**
- **튜플 = 값,**
- 튜플 ↔ 리스트 변환 가능

```
a = (38, 21, 53, 62, 19)
a = 38, 21, 53, 62, 19
person = ('james', 17, 175.3, True)

(38,)
38,

a = [1, 2, 3]
tuple(a)

b = (4, 5, 6)
list(b)

list('Hello')
tuple('Hello')
```

딕셔너리

- 연관된 값을 묶어서 저장하는 용도
- 값 마다 이름을 붙여서 저장하는 방식
- 사전(dictionary)에서 단어를 찾듯 값을 참조
- **{ }**(중괄호) 안에 **키: 값** 형식으로 저장
- 키를 먼저 지정하고 :(콜론)을 붙여서 값 표현
- 각 키와 값은 ,(콤마)로 구분
- **딕셔너리 = {키1: 값1, 키2: 값2}**
- 키-값 쌍(key-value pair) 1:1 대응
- 키가 중복되면 가장 뒤에 있는 값만 사용
- 키는 문자열뿐만 아니라 정수, 실수, 불, 복합 자료형도 사용 가능
- 값에는 리스트, 딕셔너리 등을 포함 모든 자료형 사용
- 키에는 리스트와 딕셔너리를 사용할 수 없음
- **딕셔너리 = {}**
- **딕셔너리 = dict()**

```
lux = [490, 334, 550, 18.72]

lux = {'health': 490, 'mana': 334, 'melee':
550, 'armor': 18.72}

lux = {'health': 490, 'health': 800, 'mana':
334, 'melee': 550, 'armor': 18.72}

x = {100: 'hundred', False: 0, 3.5: [3.5, 3.5]}
```

```
x = {[10, 20]: 100}
x = {{'a': 10}: 100}
```

딕셔너리 만들기

- **딕셔너리 = dict(키1=값1, 키2=값2)**
- dict에서 **키=값** 형식으로 딕셔너리 생성. 키에 ' '(작은따옴표)나 " "(큰따옴표)를 사용하지 않음. 키는 딕셔너리를 만들고 나면 문자열로 변경됨.
- **딕셔너리 = dict(zip([키1, 키2], [값1, 값2]))**
- dict에서 zip 함수를 이용. 키가 들어있는 리스트와 값이 들어있는 리스트를 차례대로 zip에 넣은 뒤 다시 dict에 입력함.
- **딕셔너리 = dict([(키1, 값1), (키2, 값2)])**
- 리스트 안에 **(키, 값)** 형식의 튜플을 나열하는 방법
- **딕셔너리 = dict({키1: 값1, 키2: 값2})**
- dict 안에서 중괄호로 딕셔너리를 생성하는 방법

```
lux1 = dict(health=490, mana=334, melee=550, armor=18.72)
lux2 = dict(zip(['health', 'mana', 'melee', 'armor'], [490, 334, 550, 18.72]))
lux3 = dict([('health', 490), ('mana', 334), ('melee', 550), ('armor', 18.72)])
lux4 = dict({'health': 490, 'mana': 334, 'melee': 550, 'armor': 18.72})
```

딕셔너리 참조

- 딕셔너리의 키에 접근할 때는 딕셔너리 뒤에 [ ](대괄호)를 사용하며 [ ] 안에 키를 지정
- **새로운 값 할당: 딕셔너리[키] = 값**
- 딕셔너리는 없는 키에 값을 할당하면 해당 키가 추가되고 값이 할당
- 딕셔너리는 없는 키에서 값을 가져오려고 하면 에러 발생
- 딕셔너리 키 여부 확인
  - **키 in 딕셔너리**
  - **키 not in 딕셔너리**
- 딕셔너리의 키 개수
  - **len(딕셔너리)**

```
lux = {'health': 490, 'mana': 334, 'melee': 550, 'armor': 18.72}
print(lux['health'])

lux['health'] = 2037
lux['mana'] = 1184
print(lux)

lux['mana_regen'] = 3.28

'health' in lux
'attack_speed' not in lux

len(lux)
```

시퀀스

- 시퀀스 자료형은 공통된 동작과 기능을 제공
- 시퀀스 자료형으로 만든 객체 = 시퀀스 객체
- 시퀀스 객체에 들어있는 각 값 = 요소(element)
- **값 in 시퀀스객체**
- **값 not in 시퀀스객체**

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

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

- 시퀀스 객체는 + 연산자를 사용하여 객체를 서로 연결하여 새 객체 생성
- 단, 시퀀스 자료형 중에서 range는 + 연산자로 객체를 연결할 수 없음
- 시퀀스 객체 반복
- **시퀀스객체 * 정수**
- **정수 * 시퀀스객체**

```
range(0, 10) + range(10, 20)
list(range(0, 10)) + list(range(10, 20))
tuple(range(0, 10)) + tuple(range(10, 20))
'Hello, ' + 'world!'
```

```
[0, 10, 20, 30] * 3
range(0, 5, 2) * 3
```

```
list(range(0, 5, 2)) * 3
tuple(range(0, 5, 2)) * 3
'Hello, ' * 3
```

시퀀스 활용

- 시퀀스 객체의 길이
- **len(시퀀스객체)**
- 시퀀스 객체의 요소 접근 방법
- **시퀀스객체[인덱스]**
- **시퀀스 객체의 인덱스는 항상 0부터 시작**
- 인덱스를 음수로 지정하면 뒤에서부터 요소에 접근
- 요소에 값 할당 : **시퀀스객체[인덱스] = 값**
- 튜플, range, 문자열 할당 불가 (읽기 전용)
- del로 요소 삭제 : **del 시퀀스객체[인덱스]**

```
a = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
len(a)
b = (38, 76, 43, 62, 19)
len(b)
len(range(0, 10, 2))
hello = 'Hello, world!'
len(hello)
```

```
a = [38, 21, 53, 62, 19]
a[0]
b = (38, 21, 53, 62, 19)
b[0]
r = range(0, 10, 2)
r[2]

a[-1]
a[5]
a[len(a) - 1]
del a[2]
```

시퀀스 슬라이스

- 슬라이스(slice)는 무엇인가의 일부를 잘라낸다는 뜻 (시퀀스 객체의 일부를 잘라 냄)
- **시퀀스객체[시작인덱스:끝인덱스]**
- 끝 인덱스는 실제로 가져오려는 인덱스보다 1을 더 크게 지정
- 실제로 가져오는 요소는 **시작 인덱스**부터 **끝 인덱스 - 1**까지
- 인덱스 증가폭 사용
- **시퀀스객체[시작인덱스:끝인덱스:인덱스증가폭]**
- 인덱스 생략
- **시퀀스객체[:끝인덱스]**
- **시퀀스객체[시작인덱스:]**
- **시퀀스객체[:]**
- **슬라이스객체 = slice(끝인덱스)**
- **슬라이스객체 = slice(시작인덱스, 끝인덱스)**
- **슬라이스객체 = slice(시작인덱스, 끝인덱스, 인덱스증가폭)**
- **시퀀스객체[슬라이스객체]**

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

```
a[0:4]
a[1:1]
a[1:2]
a[4:-1]
a[2:8:3]
a[:7]
a[7:]
a[:]
```

```
a[:7:2]
a[7::2]
a[::2]
a[::]
a[5:1:-1]
a[0:len(a)]
```

```
a[2:5] = ['a', 'b', 'c']
a[2:5] = ['a']
a[2:5] = ['a', 'b', 'c', 'd', 'e']
a[2:8:2] = ['a', 'b', 'c']
del a[2:5]
del a[2:8:2]
```

이스케이프 시퀀스

이스케이프 시퀀스	설명
₩₩	백슬래시, ₩
₩'	작은따옴표, Single quote, '
₩"	큰따옴표, Double quote, "
\a	벨, ASCII Bell, BEL
\b	백스페이스, ASCII Backspace, BS
\f	폼피드, ASCII Formfeed, FF
\n	새 줄, 개행 문자, ASCII Linefeed, LF
\r	캐리지 리턴, ASCII Carriage Return, CR
\t	탭 문자, ASCII Horizontal Tab, TAB
\v	수직 탭, ASCII Vertical Tab, VT
\ooo	₩ 뒤에 8진수 숫자를 지정하여 ASCII 코드의 문자 표현 예) '₩141'은 'a'를 표현
\xhh	₩ 뒤에 16진수 숫자를 지정하여 ASCII 코드의 문자 표현 예) '₩x61'은 'a'를 표현 ASCII 코드는 다음 URL 참조 ascii Table https://www.asciitable.com
\N{name}	{ } 안에 문자 이름을 지정하여 유니코드의 문자 표현(파이썬 3.3이상) 예) '₩N{LINE FEED}'는 '₩n'을 표현. 다음 URL 참조 formal name aliases for Unicode characters http://www.unicode.org/Public/8.0.0/ucd/NameAliases.txt
\uxxxx	₩ 뒤에 16비트 16진수 숫자를 지정하여 유니코드의 문자 표현 예) '₩u0061'은 'a'를 표현 유니코드는 다음 URL 참조 List of Unicode characters(유니코드 문자 목록) https://en.wikipedia.org/wiki/List_of_Unicode_characters Hangul Syllables(한글 음절) https://en.wikipedia.org/wiki/Hangul_Syllables
\Uxxxxxxxx	₩ 뒤에 32비트 16진수 숫자를 지정하여 유니코드의 문자 표현 예) '₩U00000061'은 'a'를 표현 유니코드는 위 URL과 동일

EOD