파이썬 프로그래밍

리스트, 튜플, 사전 및 내장 자료형 특성

한국기술교육대학교
온라인평생교육원

▣ 리스트, 튜플, 사전

1. 리스트의 정의와 리스트 기본 연산

•리스트: 임의의 객체를 순차적으로 저장하는 집합적 자료형
•문자열이 지닌 대부분의 연산들은 리스트도 지원

```
L = [1,2,3]
print type(L)
print
print len(L)
print
print L[1]
print L[-1]
print L[1:3]
print
print L + L
print L * 3
```

```
<type 'list'>

3

2
3
[2, 3]

[1, 2, 3, 1, 2, 3]
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

- 리스트: 임의의 객체를 순차적으로 저장하는 집합적 자료형
- 리스트는 객체를 순차적으로 저장하므로 인덱스가 존재
- L=[1, 2, 3] → 1은 인덱스 0, 2는 인덱스 1, 3은 인덱스 2
- type(L) → L의 타입은 무엇인가?
- L에 할당된 개체를 통해 list로 판단
- len(L) → L의 길이(원소의 갯수)는 얼마인가?
- L[1] → 인덱스가 1인 값
- L[-1] → 인덱스가 뒤에서 첫 번째인 값

▣ 리스트, 튜플, 사전

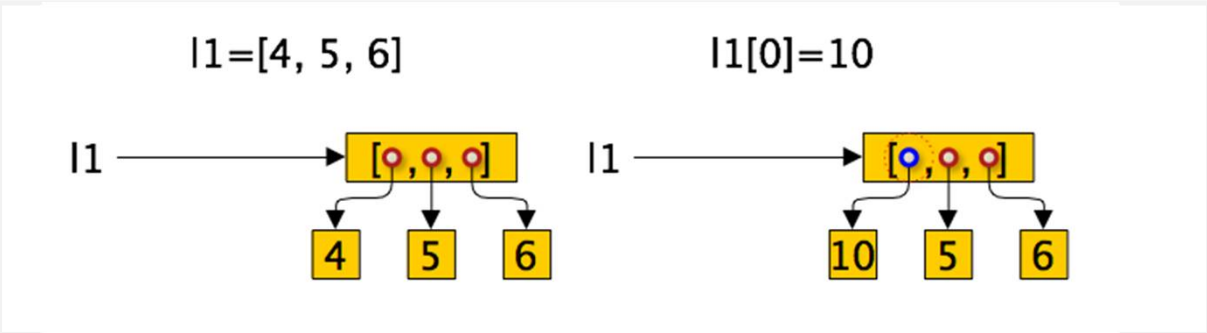
1. 리스트의 정의와 리스트 기본 연산

- L[1:3] → 인덱스 1(2)부터 인덱스 3 앞(3)까지 슬라이싱
- 리스트 + 리스트 → 두 리스트의 원소를 합쳐 하나의 리스트로
- 리스트 * 숫자 → 리스트를 숫자만큼 반복하여 하나의 리스트로
- +(더하기), *(곱하기), 슬라이싱, 인덱싱은 문자열과 동일하게 행동

• 리스트는 변경 가능

```
l1 = [4,5,6]
l1[0] = 10
print l1
```

```
[10, 5, 6]
```



- 리스트 → 변경 가능 (문자열 → 변경 불가능, 에러 발생)
- l1[0] = 10 → l1의 인덱스 0(현재 4)를 10으로 하겠다는 의미
- l1은 4, 5, 6을 담고 있는 리스트를 가리키는 레퍼런스를 가짐
- 1[0] = 10 → 기존의 4는 쓰레기가 되고 새로운 레퍼런스로 수정

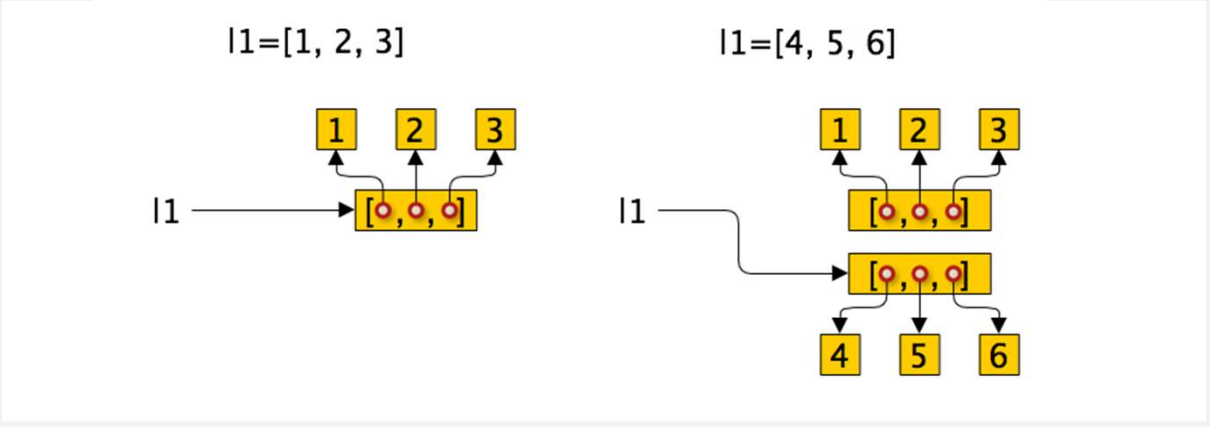
▣ 리스트, 튜플, 사전

1. 리스트의 정의와 리스트 기본 연산

• 동일한 변수에 다른 리스트를 할당하는 것은 해당 변수의 레퍼런스를 변경함

```
l1 = [1,2,3]
l1 = [4,5,6]
print l1
```

```
[4, 5, 6]
```



- l1을 [1, 2, 3] 리스트에서 [4, 5, 6] 리스트로 다시 할당
- l1 = [4, 5, 6] → 기존의 리스트는 쓰레기, 새로운 레퍼런스로 수정

▣ 리스트, 튜플, 사전

2. range() 함수를 통한 인덱스 리스트 생성

• range(k): 0부터 k-1까지의 숫자의 리스트를 반환함

```
L = range(10)
print L
print L[::2]
print L[::-1]
print 4 in L
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 2, 4, 6, 8]
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
True
```

- range(k) → [0 , 1, ..., k-1]로 리스트 구성하여 반환
- range(10) → 0부터 10-1인 9까지 있는 리스트 반환
- L[::2] → 전체 리스트에서 인덱스가 2씩 차이나게 슬라이싱
- L[::-1] → 전체 리스트를 거꾸로 슬라이싱
- 4 in L (멤버쉽 테스트) → L 안에 4가 있다(true)

▣ 리스트, 튜플, 사전

3. 튜플의 정의와 기본 연산

•튜플: 리스트와 유사하지만 튜플 내의 값을 변경할 수 없음
•적합한 사용 예
- months = ('January','February','March','April','May','June','July','August','September',
'October','November',' December')
- 각 값에 대해 인덱스가 부여됨
•문자열이 지닌 대부분의 연산들은 튜플도 지원

- 튜플: 리스트와 유사하지만 값 변경 불가능
- 튜플은 둥근 괄호()를 사용
- months라는 튜플에 January부터 December까지 있음
- 튜플도 인덱스가 존재, January는 인덱스 0, December는 인덱스 11
- months의 길이는 12, 마지막 인덱스(December)는 11
- 튜플은 리스트와 달리 수정 불가능
- 수정이 불가능하므로 상수와 비슷한 속성

▣ 리스트, 튜플, 사전

3. 튜플의 정의와 기본 연산

```
t = (1,2,3)

print len(t)
print
print t[0]
print t[-1]
print t[0:2]
print t[::2]
print
print t + t + t
print t * 3
print
print 3 in t
```

```
3

1
3
(1, 2)
(1, 3)

(1, 2, 3, 1, 2, 3, 1, 2, 3)
(1, 2, 3, 1, 2, 3, 1, 2, 3)

True
```

- 문자열 연산은 가능(인덱싱, 슬라이싱, +, *, 멤버쉽 테스트)

▣ 리스트, 튜플, 사전

4. 튜플의 상수적 성격

• 튜플은 내용 변경 불가

```
t = (1,2,3)


t[0] = 100
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-105-b050fc22719c> in <module>()
      1 t = (1,2,3)
      2
----> 3 t[0] = 100

TypeError: 'tuple' object does not support item assignment
```

- 튜플 → 상수와 같이 변경 불가능
- t[0] = 100 → t의 인덱스 0인 1을 100으로 변경(불가)

• 반면에 리스트는 내용 변경 가능

```
L = [1,2,3]
L[0] = 100


print L
```

```
[100, 2, 3]
```

- 리스트 → 변경 가능

▣ 리스트, 튜플, 사전

5. 사전의 정의와 기본 사용법

•정수형 인덱스가 아닌 키를 이용하여 값을 저장하는 자료 구조
  - 저장된 각 자료에 대한 순서는 의미 없음
•매핑(Mapping) 함수와 비슷한 역할을 함
  - x라는 키값을 넣으면 값 y를 반환함

- 사전(dictionary)도 활용 빈도가 높은 자료구조
- 사전 → 인덱스가 없음, 키를 이용해 값 저장(매핑)

```
d = {'one': 'hana', 'two': 'dul', 'three': 'set'}
print d['one']
```

```
hana
```

- 사전 → 중괄호({}) 사용
- 콤마(,)를 기준으로 아이템 구분
- d = {키:벨류, 키:벨류, 키:벨류} → 사전 d의 원소 3개
- 사전의 원소 → 키:벨류(쌍으로 이루어짐)
- print d['one'] → 사전 d 중 one이라는 키의 벨류 출력

▣ 리스트, 튜플, 사전

5. 사전의 정의와 기본 사용법

```
d = {'one': 'hana', 'two': 'dul', 'three': 'set'}
d['four'] = 'net' # 새 항목의 삽입
print d
d['one'] = 1       # 기존 항목의 값 변경
print d
print 'one' in d  # 키에 대한 멤버쉽 테스트
```

```
{'four': 'net', 'three': 'set', 'two': 'dul', 'one': 'hana'}
{'four': 'net', 'three': 'set', 'two': 'dul', 'one': 1}
True
```

- 사전에 새 항목 넣기 → 사전['키']='벨류'
- 사전은 순차적이지 않고, 랜덤한 순서로 존재함
- 실제로는 키에 대한 해시값으로 순차 정렬이 됨
- 자세한 내용은 "사전, 해시"로 검색하여 참고
- one의 벨류를 hana에서 1로 변경하기
- d['one'] = 1 → 키 one에 매핑된 기존 벨류 hana를 1로 변경
- 'one' in d → one이라는 키가 d에 존재하는지 확인
- 사전에서 기본적인 연산자는 키값을 사용
- 'one' in d → 4개의 키값 중 one이 존재하는지 확인

```
d = {'one': 1, 'two': 'dul', 'three': 'set', 'four': 'net'}
print d.keys()    # 키만 리스트로 추출함
print d.values()  # 값만 리스트로 추출함
print d.items()   # 키와 값의 튜플을 리스트로 반환함
```

```
['four', 'three', 'two', 'one']
['net', 'set', 'dul', 1]
[('four', 'net'), ('three', 'set'), ('two', 'dul'), ('one', 1)]
```

- 사전.keys( ) → 키만 추출(임의의 순서)
- 사전.values( ) → 벨류만 추출(임의의 순서)
- 사전.items( ) → 키와 값을 튜플로 추출(임의의 순서)
- Keys, values, items가 반환하고 있는 자료형 → 리스트

파이썬 프로그래밍

리스트, 튜플, 사전 및 내장 자료형 특성

한국기술교육대학교
온라인평생교육원

▣ 내장 자료형의 정리와 객체 신원 파악

1. 내장 자료형의 특성 정리

자료형	저장/접근 방법	변경 가능성	저장 모델
수치형	직접(Direct)	변경불가능(Immutable)	리터럴 (Literal)
문자열	시퀀스 (Sequence)	변경불가능(Immutable)	리터럴 (Literal)
리스트	시퀀스 (Sequence)	변경가능(Mutable)	컨테이너 (Container)
튜플	시퀀스 (Sequence)	변경불가능(Immutable)	컨테이너 (Container)
사전	매핑 (Mapping)	변경가능(Mutable)	컨테이너 (Container)

- 내장 자료형 → 수치형, 문자열, 리스트, 튜플, 사전
- 문자열, 리스트, 튜플 → 시퀀스(인덱스 존재)
- 사전 → 매핑(인덱스 없음)
- 리스트, 사전 → 변경 가능 / 나머지는 불가능
- 수치형, 문자열 → 리터럴
- 리터럴은 정수나 숫자, 표기법 등이 하나씩 저장됨
- 0x→ 16진법을 나타내는 리터럴(표기법)
- 표기법을 사용하여 숫자(수치형 자료)를 나타냄
- -0.2e-4도 하나의 표기법(리터럴)
- 단일/이중 따옴표, 연속된 단일/이중 따옴표 세개도 표기법(리터럴)
- 리스트, 튜플, 사전 → 컨테이너(집합체 형태)

▣ 내장 자료형의 정리와 객체 신원 파악

2. 내장 자료형 알아보기

```
print type(3)       #정수
print type(3.3)     #실수
print type('abc')   #문자열
```

```
<type 'int'>
<type 'float'>
<type 'str'>
```

- type(리터럴) → 리터럴의 자료형

```
print type([])      #리스트
print type(())      #튜플
print type({})      #사전(dict)
```

```
<type 'list'>
<type 'tuple'>
<type 'dict'>
```

- 내용이 없는 리스트, 튜플, 사전도 타입 조사 가능

▣ 내장 자료형의 정리와 객체 신원 파악

2. 내장 자료형 알아보기

• 자료형의 비교

```
a = 0
L = [1,2,3]
print type(a) == type(0)
print type(L) == type([])
print type(L[0]) == type(0)
```

```
True
True
True
```

- type(A)==type(B) → A와 B의 타입이 같은지 확인
- type(L)==type([]) → L과 리스트의 타입이 같은지 확인
- type(L[0]) → L의 0번 인덱스(1)의 타입 → 정수(int)
- type(0) → 0의 타입 → 정수(int)
- type(L[0]) == type(0) → True

```
print type(None)    #None 객체, 아무 값도 없다(혹은 아니다)를 나타내는 객체
print
a = None
print a
print type(a)
```

```
<type 'NoneType'>

None
<type 'NoneType'>
```

- none 객체는 none 타입
- a = none → a는 아무 것도 아닌 객체
- print a → none 출력됨
- print type(a) → a의 타입을 출력 → none타입
- none 객체는 none 타입, none 타입은 none 객체

▣ 내장 자료형의 정리와 객체 신원 파악

3. 객체의 신원 식별하기

• id(): 객체의 식별자를 반환한다.

```
a = 500
b = a
print id(a)
print id(b)
print
x = 1
y = 1
print id(x)
print id(y)
```

```
4478569376
4478569376

4298182776
4298182776
```

- id(a) → a 객체의 식별자 반환
- print id(a) → 파이썬 인터프리터가 관리하는 식별자 출력
- print id(b) → b의 식별자 출력
- b = a 문장으로 인해 b도 500을 가리킴
- a = 500 → 500을 가리키는 레퍼런스가 a에 할당
- b = a → b에도 동일한 레퍼런스가 할당됨
- 따라서 a, b가 동일한 객체(500)를 가리키게 됨
- 때문에 a와 b의 식별자가 동일하게 나타남
- 파이썬에서는 숫자를 따로 객체로 만들지 않음(이미 존재)
- x, y에 1을 가리키는 레퍼런스 할당
- 따라서 x, y는 동일한 객체(1)를 가리킴

▣ 내장 자료형의 정리와 객체 신원 파악

3. 객체의 신원 식별하기

• is 키워드: 두 객체의 식별자가 동일한지 테스트한다.

```
c = [1,2,3]
d = [1,2,3]
print c is d

a = 500
b = a
print a is b

x = 1
y = 1
print x is y

e = f = [4,5,6]
print e is f
```

```
False
True
True
True
```

- c와 d가 동일한 리스트를 가질 경우
- c is d→ c, d가 가리키는 객체의 식별자가 동일한지 확인
- 결과는 False(식별자가 다름)
- id 함수로 c, d의 식별자를 확인해보면
- 식별자가 다른 c, d → is 함수(id가 동일한가?) → 결과 False 출력
- 내용이 같은 리스트라도 서로 다른 객체
- 수치는 새로 만들어지지 않고 이미 존재
- 이미 존재하는 수치를 c, d가 같이 참조
- 때문에 c, d 식별자가 동일함
- 이미 존재하는 500을 a가 가리킴

▣ 내장 자료형의 정리와 객체 신원 파악

3. 객체의 신원 식별하기

- b가 a와 같은 레퍼런스를 가짐 → b도 동일한 500을 가리킴
- f=[4, 5, 6] → [4, 5, 6]을 가리키는 레퍼런스 값이 f에 할당
- e = f → f에 할당된 레퍼런스 값이 e에도 할당
- 따라서 e와 f는 동일한 식별자를 가짐

• == 연산자: 두 객체의 값이 동일한지를 테스트한다.

```
c = [1,2,3]
d = [1,2,3]
c == d
```

```
True
```

- c==d → c와 d의 내용이 같은가를 확인
- c와 d는 서로 다른 객체를 가지지만 내용이 같으므로 True
- ==는 두 객체의 내용이 같은지 확인, is는 식별자가 같은지 확인