파이썬 프로그래밍

문자열 정의 및 기초 연산

한국기술교육대학교
온라인평생교육원

▣ 시퀀스 자료형의 지원 연산

1. 시퀀스 자료형이란?

- 저장된 각 요소를 정수 Index를 이용하여 참조가 가능한 자료형
- 시퀀스(Sequence) 자료형: 문자열, 리스트, 튜플

▪문자열: 시퀀스 자료형의 대표적인 자료형
▪시퀀스 자료형이 가지고 있는 공통적인 연산 존재

```
s = 'abcdef'
L = [100,200,300]
t = ('tuple', 'object', 1, 2)
```

- 시퀀스 자료형이 가지는 공통적인 연산⊠인덱싱 (Indexing)
  - 슬라이싱 (Slicing)
  - 확장 슬라이싱 (Extended Slicing)
  - 연결 (Concatenation)
  - 반복 (Repitition)
  - 멤버쉽 테스트 (Membership Test)
  - 길이 정보 (Length)
  - for ~ in 문

▣ 시퀀스 자료형의 지원 연산

2. 인덱싱

```
s = 'abcdef'
l = [100, 200, 300]
print s[0]
print s[1]
print s[-1]
print
print l[1]
l[1] = 900
print l[1]
```

```
a
b
f

200
900
```

- s= 문자열, l = 리스트가 할당
- 숫자는 반드시 하나의 정수
- [하나의 정수] = 인덱싱 연산
- s[0] = 0번째 해당하는 문자 = a
- s[1] = 1에 해당하는 인덱스 = b
- s[-1] = 맨 마지막 인덱스 = f
- Print (아무것도 없음): 한 줄 띄우기
- l[1] = 두 번째 해당하는 인덱스 = 200
- list(리스트): 변경이 가능한 자료형

▣ 시퀀스 자료형의 지원 연산

2. 인덱싱

```
print l[100]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-2-dd85adc9a089> in <module>()
----> 1 print l[100]

IndexError: list index out of range
```

- l [100] → list l에 존재하지 않음 → error 발생
- error의 종류: Index Error

▣ 시퀀스 자료형의 지원 연산

3. 슬라이싱

• L[start:end]: start는 inclusive, end는 exclusive

```
s = 'abcdef'
L = [100, 200, 300]

print s[1:3]
print s[1:]
print s[:]
print s[-100:100]
print
print L[:-1]      # L[:2] 와 동일
print L[:2]
```

```
bc
bcdef
abcdef
abcdef

[100, 200]
[100, 200]
```

▪슬라이싱: '[ ]'가 쓰이고 안쪽에 반드시 ':(콜론)'이 쓰임
▪시퀀스 자료형의 일부분을 가져옴
▪s[1:3] = 1은 start에 해당→ b
▪s[1:3] = 3은 마지막 인덱스로 포함하지 않음
▪c → '2'에 해당하는 인덱스
▪s[1:3] = 1과 2에 해당하는 문자열만 반환
▪start 값 O stop값 X → 해당 문자열의 마지막까지 슬라이싱 됨
▪start 값 X stop값 X → 전체를 다 슬라이싱
▪-100 → 처음 값, 100 → 마지막 값
▪L[:-1] → 끝 인덱스 → 300 → '2'에 해당
▪L[:-1] = L[:2]

▣ 시퀀스 자료형의 지원 연산

4. 확장 슬라이싱

• L[start:end:step]: 인덱싱되어지는 각 원소들 사이의 거리가 인덱스 기준으로 step 만큼 떨어짐

```
s = 'abcd'
print s[::2]    #step:2 - 각 원소들 사이의 거리가 인덱스 기준으로 2가 됨
print s[::-1]   #step:-1 - 왼쪽 방향으로 1칸씩
```

```
ac
dcba
```

▪확장 슬라이싱: 콜론(:)이 2개가 쓰임
▪s[::2] → '2'는 step에 해당
▪2[::] → 전체 내용을 다 들고 오는 것
▪s[::2] → 들고 나오는 문자에 해당하는 인덱스의 차이가 2가 됨
▪a의 인덱스: 0, c의 인덱스: 2 → 2-0=2
▪스탭에 해당하는 숫자만큼 인덱스에 차이를 두어 반환
▪-1 → 역순으로 값을 가지고 옴
▪d의 인덱스: 3, c의 인덱스: 2 → 2-3=-1
▪c의 인덱스: 2, b의 인덱스: 1 → 1-2=-1

▣ 시퀀스 자료형의 지원 연산

5. 연결하기

```
s = 'abc' + 'def'
print s

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

```
abcdef
[1, 2, 3, 4, 5, 6]
```

▪시퀀스 + 시퀀스 → 2개의 시퀀스를 연결해서 하나로

▪리스트 + 리스트 → 하나의 리스트로 반환

▣ 시퀀스 자료형의 지원 연산

6. 반복하기

```
s = 'abc'
print s * 4

L = [1,2,3]
print L * 2
```

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

▪반복하기: '곱하기' 연산을 보는 것

▪s * 4= s의 내용을 4번 반복하여 반환

▣ 시퀀스 자료형의 지원 연산

7. 멤버십 테스트

```
s = 'abcde'
print 'c' in s

t = (1,2,3,4,5)
print 2 in t
print 10 in t
print 10 not in t
```

```
True
True
False
True
```

▪'c' in s : c라고 하는 문자열이 s 안에 존재하는지 확인
▪True 아니면 False로 반환
▪10 not in t : t 안에 10이 존재하지 않는지 확인

```
print 'ab' in 'abcd'
print 'ad' in 'abcd'
print ' ' in 'abcd'
print ' ' in 'abcd '
```

```
True
False
False
True
```

▪ad(연속된 문자)는 'abcd'에 존재하지 않음
▪'(공백)' 은 'abcd'에 존재하지 않음
▪'abcd "는 'abcd(공백)'으로 false가 아닌 true

▣ 시퀀스 자료형의 지원 연산

8. 길이 정보

```
s = 'abcde'
l = [1,2,3]
t = (1, 2, 3, 4)
print len(s)
print len(l)
print len(t)
```

```
5
3
4
```

▪문자열, 리스트, 튜플 모두 len 함수의 인자가 될 수 있음

▣ 시퀀스 자료형의 지원 연산

9. for~in 문

```
for c in 'abcd':
    print c,
```

```
a b c d
```

▪컨테이너 종류: 시퀀스 자료형, 시퀀스 자료형이 아닌 것
▪시퀀스 자료형 = 문자열, 리스트, 튜플
▪시퀀스 자료형이 아닌 것 = 사전

파이썬 프로그래밍

문자열 정의 및 기초 연산

한국기술교육대학교
온라인평생교육원

▣ 문자열 정의 및 기초 연산

1. 한 줄 문자열

```
s = ''
str1 = 'Python is great!'
str2 = "Yes, it is."
str3 = "It's not like any other languages"
```

▪s = ' ' → 아무런 내용이 없는 문자열
▪str1 → 단일 따옴표 사용
▪str2 → 이중 따옴표 사용
▪str3 → 이중 따옴표 사용시 단일따옴표 문자 사용 OK

```
str4 = 'Don \ 't walk. "Run"'
print str4
```

```
Don't walk. "Run"
```

▪str4 → 단일 따옴표 사용시 이중따옴표 문자 사용 OK
▪단일 따옴표 사용 시 단일 따옴표 문자 사용: \ 와 같이 사용

- \ : 다음 라인이 현재 라인의 뒤에 이어짐을 나타냄

```
long_str = "This is a rather long string \
containing back slash and new line.  \ nGood!"
print long_str
```

```
This is a rather long string containing back slash and new line.
Good!
```

▪ \ 사용 → 두 줄로 나눠진 부분을 한 줄로 연결
▪ \ n : 개행

▣ 문자열 정의 및 기초 연산

2. 여러 줄 문자열

```
multiline = """ While the rest of the world has been catching on to
the Perl scripting language, the Linux commnunity,
long since past the pleasing shock of Perl's power,
has been catching on to a different scripting animal -- Python."""
print multiline
print
ml = ''' While the rest of the world has been catching on to
the Perl scripting language, the Linux commnunity,
long since past the pleasing shock of Perl's power,
has been catching on to a different scripting animal -- Python.'''
print ml
```

```
While the rest of the world has been catching on to
the Perl scripting language, the Linux commnunity,
long since past the pleasing shock of Perl's power,
has been catching on to a different scripting animal -- Python.

 While the rest of the world has been catching on to
the Perl scripting language, the Linux commnunity,
long since past the pleasing shock of Perl's power,
has been catching on to a different scripting animal -- Python.
```

▪multiline = """ (문장)""" : 긴 문장이 자동 개행되어 여러 줄이 됨

▪ml = ' ' '(문장)' ' ' : 긴 문장이 자동 개행되어 여러 줄이 됨

▪단일 따옴표 3개로 시작 시 단일 따옴표 단독 사용 가능

▣ 문자열 정의 및 기초 연산

3. 이스케이프 문자 (Escape Characters)

- 문자열 내부의 이스케이프 문자

이스케이프 문자	의미
\ \	\
\'	'
\"	"
\b	백스페이스
\n	개행
\t	탭
\0nn	8진법 수 nn
\xnn	16진법 수 nn

▣ 문자열 정의 및 기초 연산

3. 이스케이프 문자 (Escape Characters)

```
print ' \  \ abc \  \ '
print
print 'abc \ tdef \ tghi'
print
print 'a \ nb \ nc'
```

```
 \ abc \

abc             def             ghi

a
b
c
```

- \ 를 문자로 사용 원할 시 \ 두 번 사용
- 앞 \ : '일반적인 문자열의 형태가 아닌 잠깐 탈출하겠다'는 의미
- 뒤 \ : ' \ 를 하나의 문자로써 인식하라'는 의미
- \ n : \ → 일반적인 문자열에서 탈출, n→ 탭에 관한 것
- \ n → '탭'이라는 문자가 들어감
- 탭 : 여덟 개의 문자가 차지하는 공간
- 여덟 개의 공간을 모두 차지 X , 여덟 개의 column으로 포멧 맞춤
- 곱하기는 반복 연산자

▣ 문자열 정의 및 기초 연산

4. 문자열 연산

```
str1 = 'First String'
str2 = 'Second String'
str3 = str1 + ' ' + str2
print str3
print str1 * 3
print
print str1[2]
print str1[1:-1]
print len(str1)
print
print str1[0:len(str1)]
```

```
First String Second String
First StringFirst StringFirst String

r
irst Strin
12

First String
```

▪strl[1:-1] → 슬라이싱으로 i와 g앞의 n까지의 문자 출력

▪str1[0:len(str1)] → 0은 F, len(str1)은 12이므로 11까지의 문자

▪str1[:] → 전체 문자 반환 및 출력

▣ 문자열 정의 및 기초 연산

4. 문자열 연산

• 문자열 자료 - Immutable (변경불가능)

• str1[0] = 'f'

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-47-7213ba3e679f> in <module>()
----> 1 str1[0] = 'f'

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

▪str1[0] = 'f' → 0번의 문자를 f로 변환 → error 발생!

▪문자열 자료는 변경 불가능

```
str1[0:3] = 'abc'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-48-08d42bad7f22> in <module>()
----> 1 str1[0:3] = 'abc'

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

▣ 문자열 정의 및 기초 연산

4. 문자열 연산

• 문자열 변경을 위해서는 여러 Slicing 연결 활용
  - [주의] 문자열 자체가 변경되는 것이 아니라 새로운 문자열을 생성하여 재 할당하는 것임

```
s = 'spam and egg'
s = s[:4] + ', cheese, ' + s[5:]
print s'
```

```
spam, cheese, and egg
```

▪s[:4] → 0에서 3까지 슬라이싱 → spam
▪s[5:] → 5에서 끝까지 → egg
▪연결되어 새로운 문자열이 된 것이지 수정된 것이 아님

▣ 문자열 정의 및 기초 연산

5. 유니코드

•다국어 문자의 올바른 표현을 위하여 유니코드 타입 지원이 됨
•유니코드 타입의 문자열 리터럴: u'Hello'

```
print u'Spam and Egg'
print
a = 'a'
b = u'bc'
print type(a)
print a
print type(b)
print b
print
c = a + b          # 일반 문자열과 유니코드를 합치면 유니코드로 변환
print type(c)
print c
```

```
Spam and Egg

<type 'str'>
a
<type 'unicode'>
bc

<type 'unicode'>
abc
```

▪한글을 사용하기 위해선 유니코드 활용 필요
▪유니코드 사용: 문자열 정의 시 앞에 'u' 사용
▪전세계의 모든 문자가 표준화된 코드값을 가지고 있음
▪a b c 처럼 가 나 다 글씨가 하나하나 코드화 됨
▪a + b = 문자열 + 문자열(유니코드)
▪문자열 + 유니코드 = 유니코드

▣ 문자열 정의 및 기초 연산

5. 유니코드

```
print u'Spam \ uB610 Egg'    # 문자열 내에 유티코드 이스케이프 문자인 \ uHHHH
                               사용가능, HHHH는 4자리 16진수 (unicode 포맷)
```

```
Spam 또 Egg
```

▪ \ u → escape u → u뒤에 16진수 4자리 적음

▪b610 → '또'라는 문자로 코드화 되어 있음

```
a = unicode('한글', 'utf-8')    # '한글' 문자열의 인코딩 방식을 'utf-8'형태로
                                  인식시키면서 해당 문자열을 unicode로 변환
print type(a)
print a
```

```
<type 'unicode'>
한글
```

▪unicode내장함수: unicode(문자열, 문자열 인식 방

```
print len('한글과 세종대왕')
print len(unicode('한글과 세종대왕', 'utf-8'))   #유니코드 타입의 문자열은 한글 문자열
                                                   길이를 올바르게 반환함
print len(u'한글과 세종대왕')
```

```
22
8
8
```

▪len(한글) X --> len(u'한글') O

▣ 문자열 정의 및 기초 연산

5. 유니코드

```
u = unicode('한글과 세종대왕', 'utf-8')    #유니코드 타입의 한글 문자열에 대해서는
                                            인덱싱 및 슬라이싱이 올바르게 수행됨
print u[0]
print u[1]
print u[:3]
print u[4:]
print u[::-1]
print
u2 = u'한글과 세종대왕'
print u2[0]
print u2[1]
print u2[:3]
print u2[4:]
print u2[::-1]
print
u3 = '한글과 세종대왕'
print u3[0]
print u3[1]
print u3[:3]
print u3[4:]
print u3[::-1]
print
```

▣ 문자열 정의 및 기초 연산

5. 유니코드

```
한
글
한글과
세종대왕
왕대종세 과글한

한
글
한글과
세종대왕
왕대종세 과글한

□
□
한
□□과 세종대왕
□□뼬녢츄□ □□☝□□
```

▪유니코드 사용 시 인덱스 연산도 가능

▪u2[:3] → 0,1,2 → 한글과