

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 나뉜 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 community,
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 community,
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 community,  
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 community,  
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 포맷 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
```

- `str1[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 재할당하는 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  
□□벨넝츄□ □□ᄅ□□

- 유니코드 사용 시 인덱스 연산도 가능
- `u2[:3] → 0,1,2 → 한글과`