LECTURE 5

Strings

STRINGS

Strings 은 sequence data 타입의 sub 타입.

Strings 은 single 이나 double quotes로 표시 가능

```
s1 = 'This is a string!'
s2 = "Python is so awesome."
```

파이썬에는 character data type은 없음.
character 하나는 한글자 스트링으로 표시

ACCESSING STRINGS

스트링은 the sequence data type 임으로 element별로 접근 가능.

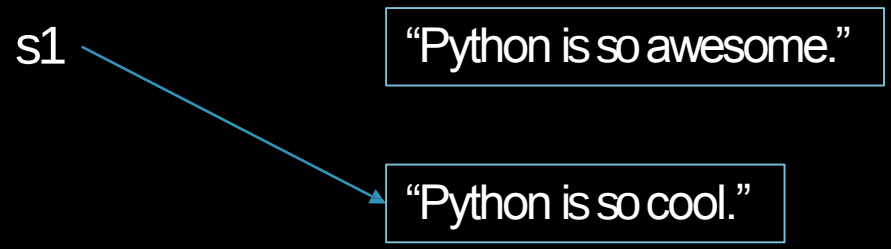
array indexin과 마찬가지로 [ ] 을 이용하여 접근. slicing도 가능

```
>>> s1 = "This is a string!"
>>> s2 = "Python is so awesome."
>>> print s1[3]
s
>>> print s2[5:15]
n is so aw
```

MODIFYING STRINGS

스트링은 불변(*immutable*)– 스트링 안의 값을 변경시킬 수는 없고, 다시 assign을 해서 새로운 스트링을 지칭하게는 할 수 있음


```
>>> s1 = "Python is so awesome."
>>> s1 = "Python is so cool."
```



MODIFYING STRINGS

그래서 우회적으로 다음과 같이 할 수 있음

```
>>> s1 = "Python is so awesome."
>>> s1 = s1[:13] + "cool."
```

This will create a substring “Python is so”, which is concatenated with “cool.”, stored in memory and associated with the name s1.

The “+” operato는 두 개의 string을 합침
The “*” operato는 하나의 string을 여러 번 반복하여 붙임

스트링 안에 어떤 char가 있는지 `in` 과 `not in` 을 가지고 체크 가능

ESCAPE CHARACTERS

다수의 escape characters 지원 (C처럼) :

- '\n' – newline
- '\s' – space
- '\t' – tab

BUILT-IN STRING METHODS

스트링과 관련된 함수 제공. 하지만 스트링은 immutable(불변)이기 때문에, 변환된 값을 retur e한다.

- `s.upper()` and `s.lower()`

```
>>> s1 = "Python is so awesome."
>>> print s1.upper()
PYTHON IS SO AWESOME.
>>> print s1.lower()
python is so awesome.
```

BUILT-IN STRING METHODS

- `s.isalpha(), s.isdigit(), s.isalnum(), s.isspace()`
  - 리턴값 : True / False
- `s.islower(), s.isupper()`
  - islower() 모든 string의 element가 lowercase이면 True

```
>>> "WHOA".isupper()
True
>>> "12345".isdigit()
True
>>> " \n ".isspace()
True
>>> "hello!".isalpha()
False
```

BUILT-IN STRING METHODS

- `str.split([sep[, maxsplit]])`
  - sep 를 구분자(delimiter)로 하여 스트링을 나눔 (기본값은 연속된 whitespace).
  - *maxsplit 은 몇 개까지 split 할지를 제한 가능* (default is -1, 제한 없음).
- `str.rsplit([sep[, maxsplit]])`
  - split과 동일하나 오른쪽에서 부터 시작
- `str.strip([chars])`
  - 맨 앞과 맨 뒤쪽에 있는 chars를 삭제 (default is whitespace).
- `str.rstrip([chars])`
  - 맨 뒤쪽에 있는 chars만 삭제.

BUILT-IN STRING METHODS

```
>>> "Python programming is fun!".split()
['Python', 'programming', 'is', 'fun!']
>>> "555-867-5309".split('-')
['555', '867', '5309']
>>> "***Python programming is fun***".strip('*')
'Python programming is fun'
```

BUILT-IN STRING METHODS

- `str.capitalize()` – 맨 첫 char는 대문자로 나머지는 소문자로.
- `str.center(width[, fillchar])` – 주어진 스트링이 중간에 놓이도록 앞 뒤에 *fillchar* (defaults to a blank space)를 padding.

  * See also `str.ljust()` and `str.rjust()`.
- `str.count(sub[, start[, end]])` – str안에서 주어진 sub 스트링하고 매칭되는 substring의 개수 리턴(non- overlapping occurrences).
- `str.endswith(suffix[, start[, end]])`- suffix와 일치하는지 여부를 판단
- See also `str.startswith()`.

BUILT-IN STRING METHODS

```
>>> "i LoVe pYtHoN".capitalize()
'I love python'
>>> "centered".center(20,'*')
'******centered******'
>>> "mississippi".count("iss")
2
>>> "mississippi".count("iss", 4, -1)
1
>>> "mississippi".endswith("ssi")
False
>>> "mississippi".endswith("ssi", 0, 8)
True
```

BUILT-IN STRING METHODS

- `str.find(sub[, start[, end]])` – 맨처음 만나는 sub의 위치index를 return. 발견하지 못하는 경우는 -1을 retur함. (See also `str.rfind()`)
- `str.index(sub[, start[, end]])` –find()와 동일하나 발견하지 못했을 경우 raises *ValueError* exception (See also `str.rindex()`)
- `str.join(iterable)` – 주어진 *iterable*의 모든 elemen를 concat 함. 여기서 *str* objec는 element 사이를 구분하는 구분자이다. (예: ‘ “.joing(iterable))
- `str.replace(old, new[, count])` – 모든 old를 new로 변환 (count를 이용하여 변경 개수 제한 가능)

BUILT-IN STRING METHODS

```
>>> "whenever".find("never")
3
>>> "whenever".find("what")
-1
>>> "whenever".index("what")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: substring not found
>>> "-".join(['555','867','5309'])
'555-867-5309'
>>> " ".join(['Python', 'is', 'awesome'])
'Python is awesome'
>>> "whenever".replace("ever", "ce")
'whence'
```

THE STRING MODULE

기본적으로 built-in함수에서 대부분의 string연산을 지원하지만, string모듈을 통해서 추가적인 함수를 지원함.

string module은 유용한 스트링 constants나 string formatting 등을 지원.

STRING CONSTANTS

```
>>> import string
>>> string.ascii_letters  'abcdefghijklmnopqrstuvwxyzA
BCDEFGHIJKLMNOPQRSTUVWXYZ'
>>> string.ascii_lowercase
'abcdefghijklmnopqrstuvwxyz'
>>> string.ascii_uppercase
'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
>>> string.digits
'0123456789'
>>> string.hexdigits
'0123456789abcdefABCDEF'
```

STRING CONSTANTS

```
>>> import string
>>> string.lowercase #locale-dependent
'abcdefghijklmnopqrstuvwxyz'
>>> string.uppercase #locale-dependent
'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
>>> string.letters # lowercase+uppercase
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
>>> string.octdigits
'01234567'
>>> print string.punctuation
!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
```

STRING CONSTANTS

- `string.whitespace` – whitespae관련된 모든 char들을 포함 (space, tab, linefeed, return, formfeed, and vertical tab)
- `string.printable` – 모든 출력 가능한 char를 포함(digits, letters, punctuation, and whitespace)

STRING FORMATTING

String 포맷팅(formatting) :

```
str.format(*args, **kwargs)
```

*args는 포맷을 포함한 variable들을 표시
 **kwargs는 keyword arguments를 허용 (dictionary)

str은 일반 text와 replacement field(변경 가능한 필드, { } 로 표시)로 구성. 변경 가능한 필드는 위치에 해당하는 숫자로 표시하거나, argument의 이름으로 접근 가능

STRING FORMATTING

```
>>> '{0}, {1}, {2}'.format('a', 'b', 'c')
'a, b, c'
>>> '{}, {}, {}'.format('a', 'b', 'c')
'a, b, c'
>>> '{2}, {1}, {0}'.format('a', 'b', 'c')
'c, b, a'
>>> '{2}, {1}, {0}'.format(*'abc')
'c, b, a'
>>> '{0}{1}{0}'.format('abra', 'cad')
'abracadabra'
```

STRING FORMATTING

keyword arguments를 format 함수에 적용 가능

```
>>> 'Coords: {lat}, {long}'.format(lat='37.24N', long='-115.81W')
'Coords: 37.24N, -115.81W'
>>> coord = {'lat': '37.24N', 'long': '-115.81W'}
>>> 'Coords: {lat}, {long}'.format(**coord)
'Coords: 37.24N, -115.81W'
```

STRING FORMATTING

replacement field에 각 변수의 attributes나 methods접근 가능.

```
>>> c = 2+3j
>>> '{0} has real part {0.real} and imaginary part {0.imag}.'.format(c)
'(2+3j) has real part 2.0 and imaginary part 3.0.'

>>> coord = (3, 5)
>>> 'X: {0[0]}; Y: {0[1]}'.format(coord)
'X: 3; Y: 5'
```

STRING FORMATTING

reserved sequences 의 예(alignment 등에 활용).

```
>>> '{:<30}'.format('left aligned')
'left aligned                  '
>>> '{:>30}'.format('right aligned')
'                 right aligned'
>>> '{:^30}'.format('centered')
'           centered           '
>>> '{:*^30}'.format('centered') # use '*' as a fill char
'***********centered***********'
```

STRING FORMATTING

부동소숫점(floating-point) 포맷

```
>>> '{:+f}; {:+f}'.format(3.14, -3.14) #항상 부호 표시
'+3.140000; -3.140000'
>>> '{: f}; {: f}'.format(3.14, -3.14) # +시 스페이스로 표시
' 3.140000; -3.140000'
>>> '{:-f}; {:-f}'.format(3.14, -3.14) # -만 표시
'3.140000; -3.140000'
>>> '{:.3f}'.format(3.14159) # 소수점 3째 자리까지만 표시
'3.142'
```