



▶

데이터 사이언스와 파이썬 데이터 처리

2020년 2학기
데이터사이언스융합전공

남세진
jordse@gmail.com

유니코드(<http://www.unicode.org/charts/>)

- 유니코드는 전 세계 언어의 문자를 정의하기 위한 국제 표준 코드임.
 - 유니코드는 수학 및 기타 분야의 기호들도 포함하고 있음.
 - 유니코드는 플랫폼, 프로그램, 언어에 상관없이 문자마다 고유한 코드 값을 제공함.
 - 유니코드 코드 차트 페이지는 현재 정의된 모든 문자 집합과 이미지에 대한 링크를 제공
- 파이썬 3 유니코드 문자열
 - 파이썬 3 문자열은 바이트 배열이 아닌 유니코드 문자열임.
 - 파이썬 3의 유니코드 문자열은 파이썬 2로부터의 가장 큰 변화임.
 - 파이썬 3은 일반적인 바이트 문자열과 유니코드 문자를 구별함.
 - 유니코드 식별자(ID) 혹은 문자에 대한 이름name을 안다면, 이 문자를 파이썬 문자열에 사용할 수 있음
 - 4자리 16진수와 그 앞에 `u`는 유니코드의 기본 평면 256개 중 하나의 문자를 지정함.
 - 높은 평면의 문자일수록 비트수가 더 필요함.
 - 모든 문자는 표준 이름 `UN{name}`으로 지정할 수 있음.

Unicode Basic multilingual plane

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F
70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F
80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF
B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF
C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF
D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF
E0	E1	E2	E3	E4	E5	E6	E7	E8	E9	EA	EB	EC	ED	EE	EF
F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF

- 로마자, 로마자권 기호
- 기타 유럽 문자
- 아프리카 문자
- 중동·서남아시아 문자
- 남부와 중앙 아시아 문자
- 동남아시아 문자
- 동아시아 문자
- CJK 문자
- 인도네시아, 오세아니아 문자
- 북미 및 남미 문자
- Notational systems
- 기호
- 사용자 정의 영역
- UTF-16 상·하위 대체 영역
- 쓰이지 않음

유니 코드 버전 13.0

텍스트와 문자열 : unicodedata 모듈

- 파이썬의 unicodedata 모듈은 유니코드 식별자와 이름으로 검색할 수 있는 함수를 제공함.
 - lookup() : 대/소문자를 구분하지 않는 인자를 취하고, 유니코드 문자를 반환한다.
 - name() : 인자로 유니코드 문자를 취하고, 대문자 이름을 반환한다.

```
import unicodedata

value = '한'
name = unicodedata.name(value)
value2 = unicodedata.lookup(name)
print('value = "%s", name = "%s", value2="%s"' % (value, name, value2))
```

```
value = "한", name = "HANGUL SYLLABLE HAN", value2="한"
```

```
import unicodedata

value = 'A'
name = unicodedata.name(value)
value2 = unicodedata.lookup(name)
print('value = "%s", name = "%s", value2="%s"' % (value, name, value2))
```

```
value = "A", name = "LATIN CAPITAL LETTER A", value2="A"
```

텍스트와 문자열 : unicodedata 모듈

Cyrillic Supplement	Coptic Epact Numbers	Devanagari Extended	Makasar
Cyrillic Extended-A	Egyptian Hieroglyphs (1MB)	Dives Akuru	Rejang
Cyrillic Extended-B	Egyptian Hieroglyph Format Controls	Dogra	Sundanese
Cyrillic Extended-C	Ethiopic	Grantha	Sundanese Supplement
Elbasan	Ethiopic Supplement	Gujarati	Tagalog
Georgian	Ethiopic Extended	Gunjala Gondi	Tagbanwa
Georgian Extended	Ethiopic Extended-A	Gurmukhi	East Asian Scripts
Georgian Supplement	Medefaidrin	Kaithi	Bopomofo
Glagolitic	Mende Kikakui	Kannada	Bopomofo Extended
Glagolitic Supplement	Meroitic	Kharoshthi	CJK Unified Ideographs (Han) (35MB)
Gothic	Meroitic Cursive	Khojki	CJK Extension A (6MB)
Greek	Meroitic Hieroglyphs	Khudawadi	CJK Extension B (40MB)
Greek Extended	N'Ko	Lepcha	CJK Extension C (3MB)
Ancient Greek Numbers	Osmanya	Limbu	CJK Extension D
Latin	Tifinagh	Mahajani	CJK Extension E (3.5MB)
Basic Latin (ASCII)	Vai	Malayalam	CJK Extension F (4MB)
Latin-1 Supplement	Middle Eastern Scripts	Masaram Gondi	CJK Extension G (2MB)
Latin Extended-A	Anatolian Hieroglyphs	Meetei Mayek	(see also Unihan Database)
Latin Extended-B	Arabic	Meetei Mayek Extensions	CJK Compatibility Ideographs
Latin Extended-C	Arabic Supplement	Modi	CJK Compatibility Ideographs Supplement
Latin Extended-D	Arabic Extended-A	Mro	CJK Radicals / Kangxi Radicals
Latin Extended-E	Arabic Presentation Forms-A	Multani	CJK Radicals Supplement
Latin Extended Additional	Arabic Presentation Forms-B	Nandinagari	CJK Strokes
Latin Ligatures	Aramaic, Imperial	Newa	Ideographic Description Characters
Fullwidth Latin Letters	Avestan	Oi Chiki	Hangul Jamo
IPA Extensions	Chorasmian	Oriya (Odia)	Hangul Jamo Extended-A
Phonetic Extensions	Cuneiform (1MB)	Saurashtra	Hangul Jamo Extended-B
Phonetic Extensions Supplement	Cuneiform Numbers and Punctuation	Sharada	Hangul Compatibility Jamo
Linear A	Early Dynastic Cuneiform	Siddham	Halfwidth Jamo
Linear B		Sinhala	

<http://www.unicode.org/charts/>

































































텍스트와 문자열 : unicodedata 모듈

```
import unicodedata

def unicode_test(value):
    name = unicodedata.name(value)
    value2 = unicodedata.lookup(name)
    print('value = "%s", name = "%s", value2="%s"' %(value, name, value2))

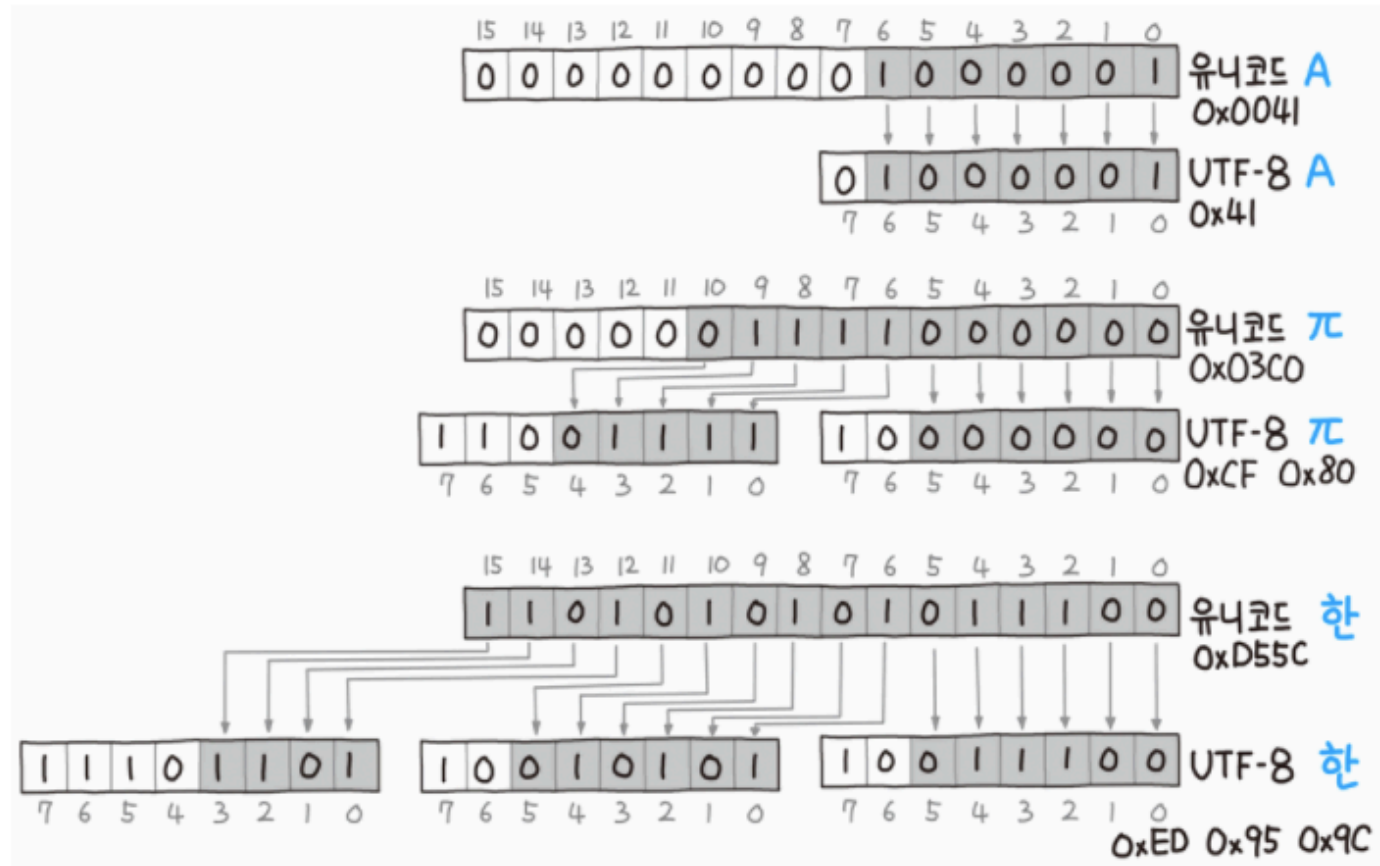
unicode_test('\u2603')
```

```
value = "☃", name = "SNOWMAN", value2="☃"
```

	260	261	262	263	264	265	266	267	268	269	26A	26B	26C	26D	26E	26F
0	 2600	 2610	 2620	 2630	 2640	 2650	 2660	 2670	 2680	 2690	 26A0	 26B0	 26C0	 26D0	 26E0	 26F0
1	 2601	 2611	 2621	 2631	 2641	 2651	 2661	 2671	 2681	 2691	 26A1	 26B1	 26C1	 26D1	 26E1	 26F1
2	 2602	 2612	 2622	 2632	 2642	 2652	 2662	 2672	 2682	 2692	 26A2	 26B2	 26C2	 26D2	 26E2	 26F2
3	 2603	 2613	 2623	 2633	 2643	 2653	 2663	 2673	 2683	 2693	 26A3	 26B3	 26C3	 26D3	 26E3	 26F3

- UTF-8은 가장 많이 사용되는 가변 길이 유니코드 인코딩
 - 켄 톰슨과 롭 파이크(Go 언어를 만든 사람)가 개발
- UTF-8 인코딩과 디코딩
 - UTF-8은 파이썬, 리눅스, HTML의 표준 텍스트 인코딩임.
 - UTF-8은 빠르고 완전하고 잘 동작함.
 - 외부 데이터를 교환할 때는 다음 과정이 필요함.
 - 문자열을 바이트로 인코딩
 - 바이트를 문자열로 디코딩
 - UTF-8 동적 인코딩 형식
 - 1바이트: 아스키코드
 - 2바이트: 키릴Cyrillic 문자를 제외한 대부분의 파생된 라틴어
 - 3바이트: 기본 다국어 평면의 나머지
 - 4바이트: 아시아 언어 및 기호를 포함한 나머지

UTF-8의 인코딩



예시 1: 문자 **A**는 아스키 문자이며 유니코드 값은 65로, 이는 16진수 0x41(0100 0001)인데, 7비트 이내로 표현 가능하므로 UTF-8로도 0x41로 표현

예시 2: 문자 **π**는 유니코드 값이 7비트를 벗어난다. 그러나 11비트 이내에 표현이 가능한 비교적 앞쪽에 위치한 문자며, 따라서 그림과 같이 2바이트에 표현이 가능 하다.

예시 3: 문자 **한**은 한글 문자로 16비트를 모두 사용한다. 마지막 16비트가 1이며 따라서 이를 표현하기 위해서는 그림과 같이 3바이트를 사용해야 한다. 참고로 유니코드에는 완성형 한글 11,172자뿐만 아니라 조합형 자모가 모두 포함되어 있으며, 이처럼 한글의 UTF-8 인코딩 값은 모두 각 문자당 3바이트를 차지한다.

UTF-8의 인코딩

코드 범위(십육진법)	UTF-16BE 표현(이진법)	UTF-8 표현(이진법)	설명
000000- 00007F	00000000 0xxxxxxx	0xxxxxxx	ASCII와 동일한 범위
000080- 0007FF	00000xxx xxxxxxxx	110xxxxx 10xxxxxx	첫 바이트는 110 또는 1110 으로 시작하고, 나머지 바이트들은 10 으로 시작함
000800- 00FFFF	xxxxxxxx xxxxxxxx	1110xxxx 10xxxxxx 10xxxxxx	
010000- 10FFFF	110110ZZ ZZxxxxxx 110111xx xxxxxxxx	11110zzz 10zzxxxx 10xxxxxx 10xxxxxx	UTF-16 서러게이트 쌍 영역 (ZZZZ = zzzzz - 1). UTF-8로 표시된 비트 패턴은 실제 코드 포인트와 동일하 다.

```
place = 'caf\u00e9'
print(place)
print(type(place))

place_bytes = place.encode('utf-8')
print(place_bytes)
print(type(place_bytes))
```

```
café
<class 'str'>
b'caf\xc3\xa9'
<class 'bytes'>
```

UTF-8 인코딩

```
place = 'caf\u00e9'  
print(place)  
print(type(place))  
  
place_bytes = place.encode('utf-8')  
print(place_bytes)  
print(type(place_bytes))  
  
place = place_bytes.decode('latin-1')  
print(place)
```

```
café  
<class 'str'>  
b'caf\xc3\xa9'  
<class 'bytes'>  
cafÃ©
```

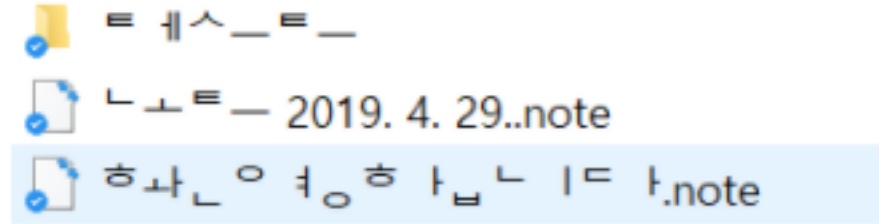
유니코드 등가성

- 유니코드 등가성(Unicode equivalence)은 특정한 일련의 코드포인트들이 반드시 동일 문자를 대표해야 하는 유니코드 문자 인코딩 표준의 사양
 - 유니코드는 2가지 개념을 제공하는데, 하나는 표준 형식의 등가성이고 나머지 하나는 호환성임
 - **유니코드 정규화**(Unicode normalization)는 모양이 같은 여러 문자들이 있을 경우 이를 기준에 따라 하나로 통합해 주는 일을 가리킨다. 그 기준으로는 아래 표와 같이 NFD, NFC, NFKD, NFKC가 있음

제목	묘사
정규화 방식 D (NFD)	정준 분해
정규화 방식 C (NFC)	정준 분해한 뒤에 다시 정준 결합
정규화 방식 KD (NFKD)	호환 분해
정규화 방식 KC (NFKC)	호환 분해한 뒤에 다시 정준 결합

NFD와 NFC

- NFD로의 정규화: 코드를 정준 분해한다.
 - 발음 구별 기호가 붙은 글자가 하나로 처리되어 있을 경우, 이를 기호별로 나누어 처리하기
 - Å (U+00C0) → A (U+0041) + ` (U+0300)
 - ê (U+1EC7) → e (U+0065) + ^ (U+0302) + . (U+0323)
 - が (U+304C) → か (U+304B) + (U+3099)
 - 𐄎 (U+30DD) → 𐄎 (U+30DB) + (U+309A)
 - Ъ (U+0419) → И (U+0418) + ~ (U+0306)
 - 한글을 한글 소리마디 영역(U+AC00~U+D7A3)으로 썼을 경우, 이를 첫가끝 코드로 처리하기
 - 위 (U+C704) → ㄱ (U+110B) + ㅞ (U+1171)
 - 한 (U+D55C) → ㅎ (U+1112) + ㅓ (U+1161) + ㄴ (U+11AB)
 - 표준과 다른 조합 순서를 제대로 맞추기
 - e (U+0071) + ` (U+0307) + . (U+0323) → e (U+0071) + . (U+0323) + ` (U+0307)
- NFC로의 정규화: 코드를 정준 분해한 뒤에 다시 정준 결합한다.
 - 발음 구별 기호(조합 분음 기호: U+0300~U+036F)가 잇따라 붙었을 경우, 이를 코드 하나로 처리하기
 - A (U+0041) + ` (U+0300) → Å (U+00C0)
 - e (U+0065) + ^ (U+0302) + . (U+0323) → ê (U+1EC7)
 - か (U+304B) + (U+3099) → が (U+304C)
 - 𐄎 (U+30DB) + (U+309A) → 𐄎 (U+30DD)
 - И (U+0418) + ~ (U+0306) → Ъ (U+0419)
 - 한글을 첫가끝 코드로 썼을 경우, 이를 한글 소리마디 영역(U+AC00~U+D7A3)으로 처리하기
 - ㄱ (U+110B) + ㅞ (U+1171) → 위 (U+C704)
 - ㅎ (U+1112) + ㅓ (U+1161) + ㄴ (U+11AB) → 한 (U+D55C)
 - ㅛ (U+1103) + ㅛ (U+1172) + ㅛ (U+11F0) → 뽀 (U+B4C0) + ㅛ (U+11F0) - 종성을 뺀 초성과 중성까지만 변환할 수 있는 경우, 알고리즘에 따라 거기까지만 변환한다^[1]



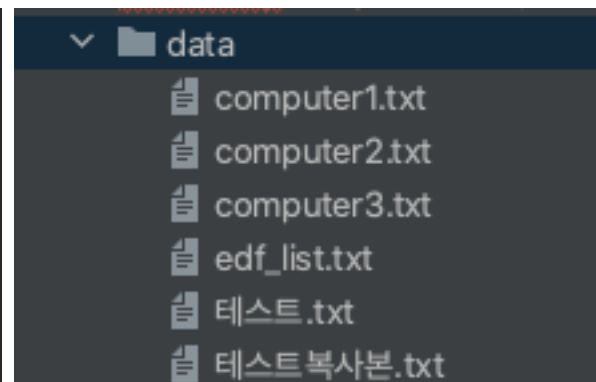
프로그램 예

```
import os
from unicodedata import normalize

file_list = []
for filename in os.listdir("./data"):
    file_list.append(filename)
    print(filename, len(filename))
    filename_str = [(filename[index], len(filename[index])) for index in range(len(filename))]
    print(filename_str)

for filename in file_list:
    print(filename, len(filename))

for filename in file_list:
    new_filename = normalize('NFC', filename)
    print(new_filename, len(new_filename))
```



```

테스트복사본.txt 18
[('ㄷ', 1), ('ㄱ', 1), ('ㅅ', 1), ('ㅡ', 1), ('ㅓ', 1), ('ㅡ', 1), ('ㅓ', 1), ('ㅓ', 1), ('ㅓ', 1), ('ㅓ', 1), ('ㅓ', 1), ('ㅓ', 1), ('ㅓ', 1), ('ㅓ', 1),
edf_list.txt 12
[('e', 1), ('d', 1), ('f', 1), ('_', 1), ('l', 1), ('i', 1), ('s', 1), ('t', 1), ('.', 1), ('t', 1), ('x', 1), ('t', 1)]
테스트.txt 10
[('ㄷ', 1), ('ㄱ', 1), ('ㅅ', 1), ('ㅡ', 1), ('ㅓ', 1), ('ㅡ', 1), ('.', 1), ('t', 1), ('x', 1), ('t', 1)]
computer3.txt 13
[('c', 1), ('o', 1), ('m', 1), ('p', 1), ('u', 1), ('t', 1), ('e', 1), ('r', 1), ('3', 1), ('.', 1), ('t', 1), ('x', 1), ('t', 1)]
computer2.txt 13
[('c', 1), ('o', 1), ('m', 1), ('p', 1), ('u', 1), ('t', 1), ('e', 1), ('r', 1), ('2', 1), ('.', 1), ('t', 1), ('x', 1), ('t', 1)]
computer1.txt 13
[('c', 1), ('o', 1), ('m', 1), ('p', 1), ('u', 1), ('t', 1), ('e', 1), ('r', 1), ('1', 1), ('.', 1), ('t', 1), ('x', 1), ('t', 1)]
테스트복사본.txt 18
edf_list.txt 12
테스트.txt 10
computer3.txt 13
computer2.txt 13
computer1.txt 13
테스트복사본.txt 10
edf_list.txt 12
테스트.txt 7
computer3.txt 13
computer2.txt 13
computer1.txt 13

```



Text Formatting

포맷

- 포맷은 보고서와 그 외의 정리된 출력물을 만들기 위해 사용함.
 - 파이썬에는 옛 스타일과 새로운 스타일의 포매팅 문자열이 있음.
- 옛 스타일: %
 - 문자열 포매팅의 옛 스타일은 string % data 형식임.
 - 문자열 안에 끼워 넣을 데이터를 표시하는 형식은

표 7-2 변환 타입

%s	문자열
%d	10진 정수
%x	16진 정수
%o	8진 정수
%f	10진 부동소수점수
%e	지수로 나타낸 부동소수점수
%g	10진 부동소수점수 혹은 지수로 나타낸 부동소수점수
%%	리터럴 %

포맷

```
print_('%s' % 42)
print_('%d' % 42)
print_('%x' % 42)
print_('%o' % 42)

print_('%s' % 7.03)
print_('%f' % 7.03)
print_('%e' % 7.03)
print_('%g' % 7.03)
print_('%d%%' % 100)
```

```
42
42
2a
52
7.03
7.030000
7.030000e+00
7.03
100%
```

```
name = '홍길동'
fet = '고양이'
weight = 5

print_('내 이름은 %s' % name)
print_("우리 강아지 %s의 몸무게는 %d kg이다" % (fet, weight))
```

```
내 이름은 홍길동
우리 강아지 고양이의 몸무게는 5 kg이다
```

- 문자열 내의 %s는 다른 문자열을 끼워 넣는 것을 의미함
- 문자열 안의 %수는 %뒤의 데이터 항목의 수와 일치해야 함

포맷

```
print_('%10d %10s %10f' % (42, "홍길동", 3.14))
print_('%-10d %-10s %-10f' % (42, "홍길동", 3.14))
print_('%10.4d %10.1s %10.1f' % (42, "홍길동", 3.14))
print_('%10.1d %10.0s %10.4f' % (42, "홍길동", 3.14))
print_('%1d %0s %.4f' % (42, "홍길동", 3.14))
print_('%*.*d %*.*s %*.*f' % (10, 4, 42, 10, 2, "홍길동", 10, 1, 3.14))
```

```
      42      홍길동    3.140000
42      홍길동    3.140000
    0042      홍      3.1
      42      3.1400
42  3.1400
    0042      홍길      3.1
```

새로운 스타일의 포매팅 : {}와 format

```
n = 42
f = 7.03
s = "kimchi"

print_({'{} {} {}'.format(n, f, s)})
```

```
42 7.03 kimchi
```

새로운 스타일의 포매팅 : {}와 format

```
print_('{ } { } { }'.format(n, f, s))

print_('{2} {0} {1}'.format(n, f, s))

print_('{n} {f} {s}'.format(n=42, f=7.03, s="nam"))

values = {'name': 'nam se jin', 'age': 20, 'sex': 'male'}
print_('{0[name]} {0[age]} {0[sex]}'.format(values))
```

```
42 7.03 kimchi
kimchi 42 7.03
42 7.03 nam
nam se jin 20 male
```

새로운 스타일의 포매팅 : {}와 format

- 새로운 스타일에서는 : 다음에 타입 지정자를 입력할 수 있음

```
n = 42
f = 7.03
s = "kimchi"

print_('{0:d} {1:f} {2:s}'.format(n, f, s))
print_('{0:10d} {1:10f} {2:10s}'.format(n, f, s))
print_('{0:>10d} {1:>10f} {2:>10s}'.format(n, f, s))
print_('{0:<10d} {1:<10f} {2:<10s}'.format(n, f, s))
print_('{0:^10d} {1:^10f} {2:^10s}'.format(n, f, s))
```

```
42 7.030000 kimchi
      42    7.030000 kimchi
      42    7.030000    kimchi
42          7.030000    kimchi
      42          7.030000    kimchi
```

새로운 스타일의 포매팅 : {}와 format

- (소수점 이후의) 정밀값은 옛 스타일과 같이 소수부 숫자의 자릿수와 문자열의 최대 문자수를 의미함.
 - 단 정수에서는 사용할 수 없음

```
n = 42
f = 7.03
s = "kimchi"

print('{0:>10d} {1:>10.4f} {2:>10.4s}'.format(n, f, s))
```

```
42      7.0300    kimc
```

```
n = 42
f = 7.03
s = "kimchi"

print('{0:>10.4d} {1:>10.4f} {2:>10.4s}'.format(n, f, s))
```

```
Traceback (most recent call last):
  File "/Users/rt datum/PycharmProjects/lecture07/code04.py", line 35, in <module>
    print('{0:>10.4d} {1:>10.4f} {2:>10.4s}'.format(n, f, s))
ValueError: Precision not allowed in integer format specifier
```



정규표현식

Python에서 정규표현식을 사용할 때

- python에서 정규표현식을 사용하기 위해서는 re 모듈을 импорт
 - **"import re"**
- 시작부터 일치하는 패턴 찾기 : match()
- 첫 번째 일치하는 패턴 찾기 : search()
- 일치하는 모든 패턴 찾기 : findall()
- 패턴으로 나누기 : split()
- 일치하는 패턴 대체하기 : sub()

정규 표현식의 사용

```
import re
# 정규식 사용 예 #1
result = re.match("You", "Young Frankenstein")
if result:
    print(result)
    print(result.group())

# 정규식 사용 예 #2
mypattern = re.compile('You')
result = mypattern.match('Young Frankenstein')
if result:
    print(result)
    print(result.group())
```

```
<re.Match object; span=(0, 3), match='You'>
You
<re.Match object; span=(0, 3), match='You'>
You
```

- "You"는 패턴
- "Young Frankenstein"은 확인하고자 하는 문자열
- match()는 패턴이 소스의 처음에 있는 경우에만 작동
- 반면, search()는 패턴이 아무데나 있어도 작동

Quick Guide

- `^` Matches the beginning of a line
- `$` Matches the end of the line
- `.` Matches any character
- `\s` Matches whitespace
- `\S` Matches any non-whitespace character
- `*` Repeats a character zero or more times
- `*?` Repeats a character zero or more times (non-greedy)
- `+` Repeats a character one or more times
- `+?` Repeats a character one or more times (non-greedy)
- `[aeiou]` Matches a single character in the listed set
- `[^XYZ]` Matches a single character not in the listed set
- `[a-z0-9]` The set of characters can include a range
- `(` Indicates where string extraction is to start
- `)` Indicates where string extraction is to end

match(), search()

re.match(*pattern*, *string*, *flags=0*)

If zero or more characters at the beginning of *string* match the regular expression *pattern*, return a corresponding **match object**. Return `None` if the string does not match the pattern; note that this is different from a zero-length match.

re.search(*pattern*, *string*, *flags=0*)

Scan through *string* looking for the first location where the regular expression *pattern* produces a match, and return a corresponding **match object**. Return `None` if no position in the string matches the pattern; note that this is different from finding a zero-length match at some point in the string.

match() search()

```
x = "computer apple, computer intel, fruit apple, smartphone apple"
print(re.match("smartphone", x))
print(re.search("smartphone", x))
```

None

<re.Match object; span=(45, 55), match='smartphone'>

```
x = "computer apple, computer intel, fruit apple, smartphone apple"
print(re.match(".*fruit", x))
print(re.search("fruit", x))
```

<re.Match object; span=(0, 37), match='computer apple, computer intel, fruit'>
<re.Match object; span=(32, 37), match='fruit'>

findall(), finditer()

re.findall(*pattern, string, flags=0*)

Return all non-overlapping matches of *pattern* in *string*, as a list of strings. The *string* is scanned left-to-right, and matches are returned in the order found. If one or more groups are present in the pattern, return a list of groups; this will be a list of tuples if the pattern has more than one group. Empty matches are included in the result.

re.finditer(*pattern, string, flags=0*)

Return an **iterator** yielding **match objects** over all non-overlapping matches for the RE *pattern* in *string*. The *string* is scanned left-to-right, and matches are returned in the order found. Empty matches are included in the result.

findall(), finditer()

```
x = "computer apple, computer intel, fruit apple, smartphone apple"
m = re.findall('apple', x)
print(m)
for item in m:
    print(item)

m = re.finditer('apple', x)
print(m)
for item in m:
    print(item)
```

```
['apple', 'apple', 'apple']
apple
apple
apple
<callable_iterator object at 0x7fb19a24e430>
<re.Match object; span=(9, 14), match='apple'>
<re.Match object; span=(38, 43), match='apple'>
<re.Match object; span=(56, 61), match='apple'>
```

```
import re
x = 'My 3 favorite numbers are 19, 42, and 30'
y = re.findall('[0-9]+', x)
print(y)
```

```
['3', '19', '42', '30']
```

search()

```
with open('./data/mbox-short.txt', 'r') as txt_file:
    for line in txt_file.readlines():
        line = line.rstrip()
        result = re.search('^X.*:', line)
        if result is not None:
            print(line)
            print(result.group(0))
```

```
X-Content-Type-Message-Body: text/plain; charset=UTF-8
X-Content-Type-Message-Body:
X-DSPAM-Result: Innocent
X-DSPAM-Result:
X-DSPAM-Processed: Fri Jan  4 06:08:27 2008
X-DSPAM-Processed: Fri Jan  4 06:08:
X-DSPAM-Confidence: 0.6948
X-DSPAM-Confidence:
```

split(), sub()

re.split()(*pattern*, *string*, *maxsplit*=0, *flags*=0)

Split *string* by the occurrences of *pattern*. If capturing parentheses are used in *pattern*, then the text of all groups in the pattern are also returned as part of the resulting list. If *maxsplit* is nonzero, at most *maxsplit* splits occur, and the remainder of the string is returned as the final element of the list.

re.sub()(*pattern*, *repl*, *string*, *count*=0, *flags*=0)

Return the string obtained by replacing the leftmost non-overlapping occurrences of *pattern* in *string* by the replacement *repl*. If the pattern isn't found, *string* is returned unchanged. *repl* can be a string or a function; if it is a string, any backslash escapes in it are processed. That is, `\n` is converted to a single newline character, `\r` is converted to a carriage return, and so forth. Unknown escapes of ASCII letters are reserved for future use and treated as errors. Other unknown escapes such as `\&` are left alone. Backreferences, such as `\6`, are replaced with the substring matched by group 6 in the pattern. For example:

split()

- 간단한 문자열 대신 패턴으로 문자열을 리스트로 나눌 때 사용

```
import re

e = "test@www.example.org"
m = re.split('@', e)
print(m)

x = "abcd vs efgh"
y = re.split("vs ", x)
print(y)
```

```
['test', 'www.example.org']
['abcd', 'efgh']
```

```
text = """Ross McFluff: 834.345.1254 155 Elm Street
Ronald Heathmore: 892.345.3428 436 Finley Avenue
Frank Burger: 925.541.7625 662 South Dogwood Way"""

entries = re.split("\n+", text)
lines = [re.split(":? ", entry, 3) for entry in entries]
for line in lines:
    print(line)
```

```
['Ross', 'McFluff', '834.345.1254', '155 Elm Street']
['Ronald', 'Heathmore', '892.345.3428', '436 Finley Avenue']
['Frank', 'Burger', '925.541.7625', '662 South Dogwood Way']
```

sub()

- 패턴을 사용하여 문자열 치환

```
x = "computer apple, computer intel, fruit apple, smartphone apple"  
print(re.sub('computer|smartphone', 'smartcomputer', x))
```

```
smartcomputer apple, smartcomputer intel, fruit apple, smartcomputer apple
```

패턴 : 특수 문자

- 리터럴은 모든 비특수 문자와 일치
- `\n`을 제외한 하나의 문자: `.`
- 0회 이상 : `*`
- 0 또는 1회 : `?`

[표 7-3]에 특수 문자를 나타냈다.

표 7-3 특수 문자

패턴	일치
<code>\d</code>	숫자
<code>\D</code>	비숫자
<code>\w</code>	알파벳 문자
<code>\W</code>	비알파벳 문자
<code>\s</code>	공백 문자
<code>\S</code>	비공백 문자
<code>\b</code>	단어 경계(<code>\w</code> 와 <code>\W</code> 또는 <code>\W</code> 와 <code>\w</code> 사이의 경계)
<code>\B</code>	비단어 경계

패턴 : 특수 문자

```
import re
import string

printable = string.printable
print(printable)

print(re.findall('\d', printable))
print(re.findall('\w', printable))
print(re.findall('\W', printable))
print(re.findall('\s', printable))
```

```
0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~
```

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

```
['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o',
```

```
['!', '"', '#', '$', '%', '&', "'", '(', ')', '*', '+', ',', '-', '.', '/', ':', ';', '<', '=', '>', '?', '@', '[', '\\', ']',
```

```
[' ', '\t', '\n', '\r', '\x0b', '\x0c']
```

패턴: 지정자

표 7-4 패턴 지정자

패턴	일치
<i>abc</i>	리터럴 <i>abc</i>
<i>(expr)</i>	<i>expr</i>
<i>expr1 expr2</i>	<i>expr1</i> 또는 <i>expr2</i>
<i>.</i>	\n을 제외한 모든 문자
<i>^</i>	소스 문자열의 시작
<i>\$</i>	소스 문자열의 끝
<i>prev ?</i>	0 또는 1회의 <i>prev</i>
<i>prev*</i>	0회 이상의 최대 <i>prev</i>
<i>prev*?</i>	0회 이상의 최소 <i>prev</i>
<i>prev+</i>	1회 이상의 최대 <i>prev</i>
<i>prev+?</i>	1회 이상의 최소 <i>prev</i>
<i>prev {m}</i>	<i>m</i> 회의 <i>prev</i>
<i>prev {m, n}</i>	<i>m</i> 에서 <i>n</i> 회의 최대 <i>prev</i>
<i>prev {m, n}?</i>	<i>m</i> 에서 <i>n</i> 회의 최소 <i>prev</i>
<i>[abc]</i>	<i>a</i> 또는 <i>b</i> 또는 <i>c</i> (<i>a b c</i> 와 같음)
<i>[^abc]</i>	(<i>a</i> 또는 <i>b</i> 또는 <i>c</i>)가 아님
<i>prev (?:next)</i>	뒤에 <i>next</i> 가 오면 <i>prev</i>
<i>prev (?!next)</i>	뒤에 <i>next</i> 가 오지 않으면 <i>prev</i>
<i>(?<=prev) next</i>	전에 <i>prev</i> 가 오면 <i>next</i>
<i>(?<!prev) next</i>	전에 <i>prev</i> 가 오지 않으면 <i>next</i>

- *expr*은 표현식(expression)
- *prev*는 이전 토큰(previous token)
- *next*는 다음 토큰(next token)

패턴 지정자의 사용 예

```
import re
x = "computer apple, computing power, compute, computer intel, fruit apple, smartphone apple, compute!!!"

print(re.findall('comput[ering]+', x))
print(re.findall('computer|compute', x))
print(re.findall('^computer [a-z]*', x))
print(re.findall('smartphone.*$', x))
print(re.findall('compute\\W', x))
```

```
['computer', 'computing', 'compute', 'computer', 'compute']
['computer', 'compute', 'computer', 'compute']
['computer apple']
['smartphone apple, compute!!!']
['compute,', 'compute!']
```

더 알아보기 : Lookahead, Lookbehind

- 전방 탐색(lookahead)
 - 작성한 패턴에 일치하는 영역이 존재하여도 그 값이 제외되어서 나오는 패턴
 - 전방 탐색 기호는 `?=` 이며, `=` 다음에 오는 문자가 일치하는 영역에서 제외
- 후방 탐색(lookbehind)
 - 전방 탐색이 앞에 있는 문자열을 탐색하는 것이라면, 후방 탐색은 뒤에 있는 문자열을 탐색
 - 후방 탐색의 기호는 `?<=` 입니다. 전방 탐색 기호의 `?`와 `=` 사이에 `<` 기호가 추가된 것입니다

탐색 기호	설명
<code>(?=)</code>	긍정형 전방탐색
<code>(?!)</code>	부정형 전방탐색
<code>(?<=)</code>	긍정형 후방탐색
<code>(?<!)</code>	부정형 후방탐색

더 알아보기 : Lookahead, Lookbehind

```
x = "computer apple, computer intel, fruit apple, smartphone apple"
print(re.search('computer (?=intel)', x))

m = re.finditer('(?!fruit )apple', x)
for item in m:
    print(item)
```

```
<re.Match object; span=(16, 25), match='computer '>
<re.Match object; span=(9, 14), match='apple'>
<re.Match object; span=(56, 61), match='apple'>
```


mbox-short.txt

```
From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008
Return-Path: <postmaster@collab.sakaiproject.org>
Received: from murder (mail.umich.edu [141.211.14.90])
    by frankenstein.mail.umich.edu (Cyrus v2.3.8) with LMTPA;
    Sat, 05 Jan 2008 09:14:16 -0500
X-Sieve: CMU Sieve 2.3
Received: from murder ([unix socket])
    by mail.umich.edu (Cyrus v2.2.12) with LMTPA;
    Sat, 05 Jan 2008 09:14:16 -0500
Received: from holes.mr.itd.umich.edu (holes.mr.itd.umich.edu [141.211.14.79])
    by flawless.mail.umich.edu () with ESMTP id m05EEFR1013674;
    Sat, 5 Jan 2008 09:14:15 -0500
Received: FROM paploo.uhi.ac.uk (app1.prod.collab.uhi.ac.uk [194.35.219.184])
    BY holes.mr.itd.umich.edu ID 477F90B0.2DB2F.12494 ;
    5 Jan 2008 09:14:10 -0500
Received: from paploo.uhi.ac.uk (localhost [127.0.0.1])
    by paploo.uhi.ac.uk (Postfix) with ESMTP id 5F919BC2F2;
    Sat, 5 Jan 2008 14:10:05 +0000 (GMT)
Message-ID: <200801051412.m05ECIaH010327@nakamura.uits.iupui.edu>
Mime-Version: 1.0
Content-Transfer-Encoding: 7bit
Received: from prod.collab.uhi.ac.uk ([194.35.219.182])
```

예를 통한 정규식 이해하기 : search()

```
import re

with open('./data/mbox-short.txt', 'r') as txt_file:
    for line in txt_file.readlines():
        line = line.rstrip()
        if re.search('From:', line):
            print(line)
```

```
From: stephen.marquard@uct.ac.za
From: louis@media.berkeley.edu
From: zqian@umich.edu
From: rjlowe@iupui.edu
From: zqian@umich.edu
From: rjlowe@iupui.edu
From: cwen@iupui.edu
From: cwen@iupui.edu
From: gsilver@umich.edu
```

```
From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008
Return-Path: <postmaster@collab.sakaiproject.org>
Received: from murder (mail.umich.edu [141.211.14.90])
    by frankenstein.mail.umich.edu (Cyrus v2.3.8) with LMTPA;
    Sat, 05 Jan 2008 09:14:16 -0500
X-Sieve: CMU Sieve 2.3
Received: from murder ([unix socket])
    by mail.umich.edu (Cyrus v2.2.12) with LMTPA;
    Sat, 05 Jan 2008 09:14:16 -0500
Received: from holes.mr.itd.umich.edu (holes.mr.itd.umich.edu [141.211.14.79])
    by flawless.mail.umich.edu () with ESMTP id m05EEFR1013674;
    Sat, 5 Jan 2008 09:14:15 -0500
Received: FROM paploo.uhi.ac.uk (app1.prod.collab.uhi.ac.uk [194.35.219.184])
    BY holes.mr.itd.umich.edu ID 477F90B0.2DB2F.12494 ;
    5 Jan 2008 09:14:10 -0500
Received: from paploo.uhi.ac.uk (localhost [127.0.0.1])
    by paploo.uhi.ac.uk (Postfix) with ESMTP id 5F919BC2F2;
    Sat, 5 Jan 2008 14:10:05 +0000 (GMT)
Message-ID: <200801051412.m05ECIaH010327@nakamura.uits.iupui.edu>
Mime-Version: 1.0
Content-Transfer-Encoding: 7bit
Received: from prod.collab.uhi.ac.uk ([194.35.219.182])
```

예를 통한 정규식 이해하기 : search()

```
with open('./data/mbox-short.txt', 'r') as txt_file:
    for line in txt_file.readlines():
        line = line.rstrip()
        if re.search('^From:', line):
            print(line)
```

- ^는 문장의 시작

```
with open('./data/mbox-short.txt', 'r') as txt_file:
    for line in txt_file.readlines():
        line = line.rstrip()
        if re.search('^X.*:', line):
            print(line)
```

```
X-DSPAM-Confidence: 0.6178
X-DSPAM-Probability: 0.0000
X-Sieve: CMU Sieve 2.3
X-Plane is behind schedule: two weeks
X-Authentication-Warning: nakamura.vits.iupui.edu: apache set sender to zqian@umich.edu using -f
X-Content-Type-Outer-Envelope: text/plain; charset=UTF-8
X-Content-Type-Message-Body: text/plain; charset=UTF-8
```

예를 통한 정규식 이해하기 : 정규식

- The dot character matches any character
- If you add the asterisk character, the character is "any number of times"

X-Sieve: CMU Sieve 2.3

X-DSPAM-Result: Innocent

X-DSPAM-Confidence: 0.8475

X-Content-Type-Message-Body: text/plain

Match the start of the line

Many times

$\wedge X.*:$

Match any character

예를 통한 정규식 이해하기 : 정규식

```
with open('./data/mbox-short.txt', 'r') as txt_file:
    for line in txt_file.readlines():
        line = line.rstrip()
        if re.search('^X-\S+:', line):
            print(line)
```

Match the start of the line

One or more times



The diagram shows the regular expression `^X-\S+:` with three arrows pointing to its parts: a purple arrow points to the caret `^`, a green arrow points to the `\S` part, and an orange arrow points to the plus sign `+`.

Match any non-whitespace character

예를 통한 정규식 이해하기 : re.findall()을 정교하게 사용하기

From `stephen.marquard@uct.ac.za` Sat Jan 5 09:14:16 2008

`\S+@ \S+`



At least one non-whitespace character

예를 통한 정규식 이해하기 : re.findall()을 정교하게 사용하기

- **Parenthesis** are not part of the match - but they tell where to **start** and **stop** what string to extract

```
import re
x = "From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008, To jordse@gmail.com"
y = re.findall('\s+\s+', x)
print(y)
```

```
['stephen.marquard@uct.ac.za', 'jordse@gmail.com']
```

```
import re
x = "From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008, To jordse@gmail.com"
y = re.findall('^From (\s+\s+)', x)
print(y)
```

```
['stephen.marquard@uct.ac.za']
```

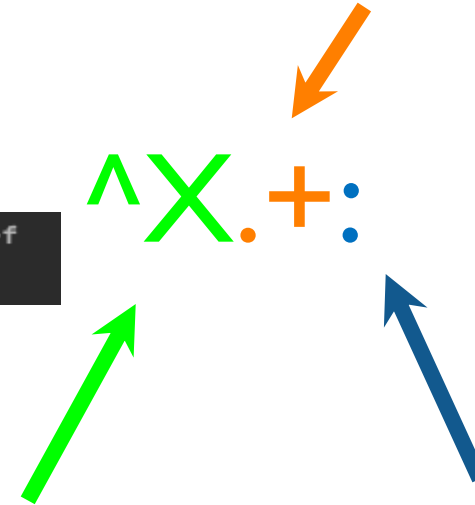
Greedy Matching

- The repeat characters (* and +) push outward in both directions (greedy) to match the largest possible string

```
with open('./data/mbox-short.txt', 'r') as txt_file:
    for line in txt_file.readlines():
        line = line.rstrip()
        result = re.search('^X.+:', line)
        if result is not None:
            print(line)
            print(result.group(0))
```

```
X-Authentication-Warning: nakamura.uits.iupui.edu: apache set sender to cwen@iupui.edu using -f
X-Authentication-Warning: nakamura.uits.iupui.edu:
```

One or more characters



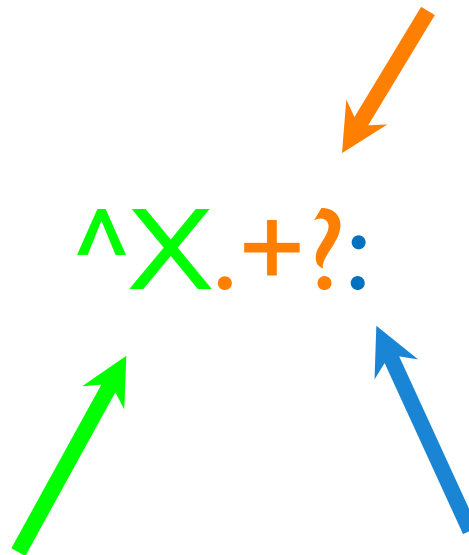
First character in the match is an F

Last character
in the match is a :

Non-Greedy Matching

- Not all regular expression repeat codes are greedy! If you add a ? character - the + and * chill out a bit...

One or more characters but not greedily



First character in the match is an X

Last character in the match is a :

Non-Greedy Matching의 예

```
with open('./data/mbox-short.txt', 'r') as txt_file:
    for line in txt_file.readlines():
        line = line.rstrip()
        result = re.search('^X.+?:', line)
        if result is not None:
            print(line)
            print(result.group(0))
```

```
X-Authentication-Warning: nakamura.uits.iupui.edu: apache set sender to cwen@iupui.edu using -f
X-Authentication-Warning:
```

예를 통한 정규식 이해하기 : find()

```
x = "From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008, To jordse@gmail.com"
xstart = x.find('@')
print(xstart)

xend = x.find(' ', xstart)
print(xend)

print(x[xstart+1:xend])
```

```
21
31
uct.ac.za
```

Quiz

- X-DSPAM-Confidence의 값이 0.7이상인 메일의 개수를 계산하라

```
X-DSPAM-Result: Innocent
X-DSPAM-Processed: Fri Jan  4 11:35:08 2008
X-DSPAM-Confidence: 0.7615
X-DSPAM-Probability: 0.0000
```