파이썬 스터디

1주차

Ch01. 사용 중인 파이썬의 버전을 알자

파이썬의 버전은 --version 플래그를 통해서 알 수 있습니다.

```
>python --version
Python 3.5.1
```

sys모듈을 통해서 런타임에 버전을 확인할 수 있습니다.

```
import sys
print(sys.version_info)
print(sys.version)

>>>
sys.version_info(major=3, minor=5, micro=1, releaselevel='final', serial=0)
3.5.1 (v3.5.1:37a07cee5969, Dec  6 2015, 01:38:48) [MSC v.1900 32 bit (Intel)]
```

사용 중인 파이썬의 버전을 알자

현재 파이썬 버전은 Python3와 Python2가 있습니다.

Python2

버그 수정,
보안 강화,
Python3로의 쉽게 포팅하는 기능
이외에는 개발 중지 상태

Python3

새기능과 향상이 계속 적용중

Ch02. 스타일 가이드를 따르자

파이썬 개선 제안서(Python Enhancement Proposal) 8

PEP 8은 파이썬 코드에 대한 스타일 가이드입니다.

전체 가이드는 여기서 확인 가능합니다.

https://www.python.org/dev/peps/pep-0008/

몇 가지 규칙을 손꼽아 보면…

- Tab이 아닌 스페이스로 들여쓴다.
- 문법적으로 의미 있는 들여쓰기는 각 수준 마다 스페이스 네 개를 사용한다.
- 한 줄의 문자 길이가 79자 이하여야 한다.
- 파일에서 함수와 클래스는 빈 줄 두 개로 구분해야 한다.
- 변수 할당 앞뒤에 스페이스를 하나만 사용한다.

몇 가지 규칙을 손꼽아 보면…

- 함수, 변수, 속성은 lowercase_underscore 형식을 따른다.
- 보호 인스턴스 속성은 _leading_underscore 형식을 따른다.
- 비공개 인스턴스 속성은 __double_leading_underscore 형식을 따른다.
- 클래스와 예외는 CapitalizedWord 형식을 따른다.
- 모듈 수준 상수는 ALL_CAPS 형식을 따른다.

몇 가지 규칙을 손꼽아 보면…

- 긍정 표현식의 부정(if not a is b) 대신 인라인 부정(if a is not b)을 사용한다.
- 길이를 확인하여 빈값을 확인하지 않는다. 빈 값은 암시적으로 False가 된다고 가정한다.
- 비어 있지 않은 값은 암시적으로 True가 된다고 가정한다.
- 한 줄로 된 if, for와 while 루프, except 복합문을 쓰지 않는다.
- 항상 파일의 맨 위에 import 문을 놓는다.

너무 많다…

개인적으로는 Style Convention Checker의 사용이 좋다고 생각됩니다.

온라인 PEP8 Checker http://pep8online.com/

Ch03. bytes, str, unicode의 차이점을 알자

bytes, str, unicode의 차이점을 알자

C++에서 문자열을 다룰때…

char[], wchar_t[], std::string, std::wstring etc…

문자열은 결국 바이트의 집합

문자열을 바이트로 어떻게 표현할 것인가 -> 인코딩

수 많은 인코딩

CP949 EUC-KR ASCII Windows-1250 Something else...

이건 65인가? 'A'인가?

0	1	0	0	0	0	0	1

인코딩에 따라 해석이 달라질 수 있습니다.

이럴 때는 역시 표준…

bytes, str, unicode의 차이점을 알자

파이썬은 문자열을 바이트와 유니코드로 나눠서 다룹니다.

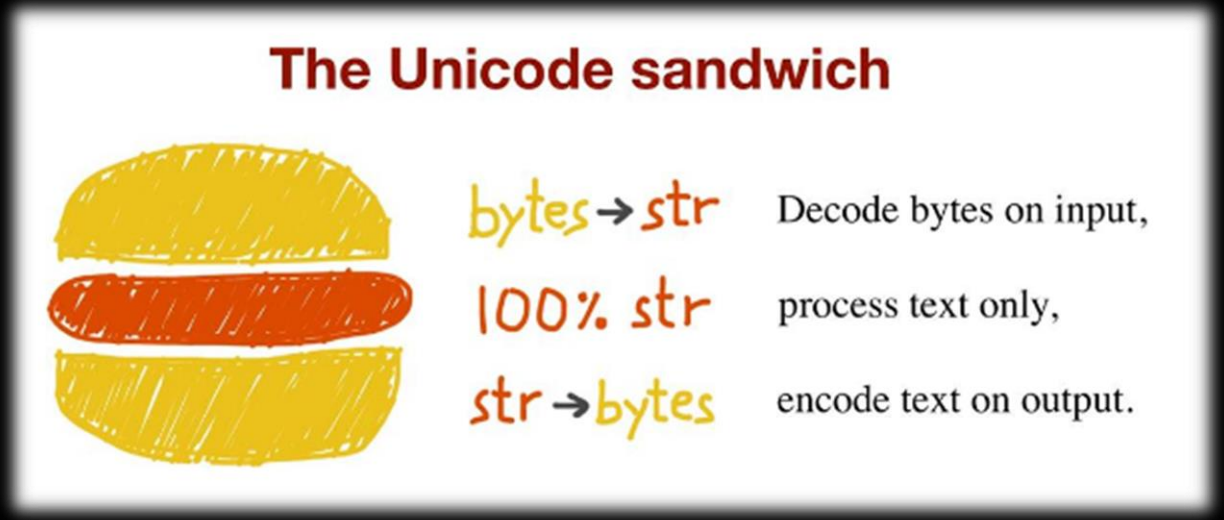
	Python3	Python2
Unicode	str	unicode
byte	bytes	str

유니코드 값을 표현하기 위해서는 코드 포인트를 사용합니다.

U+0041
U-00000041

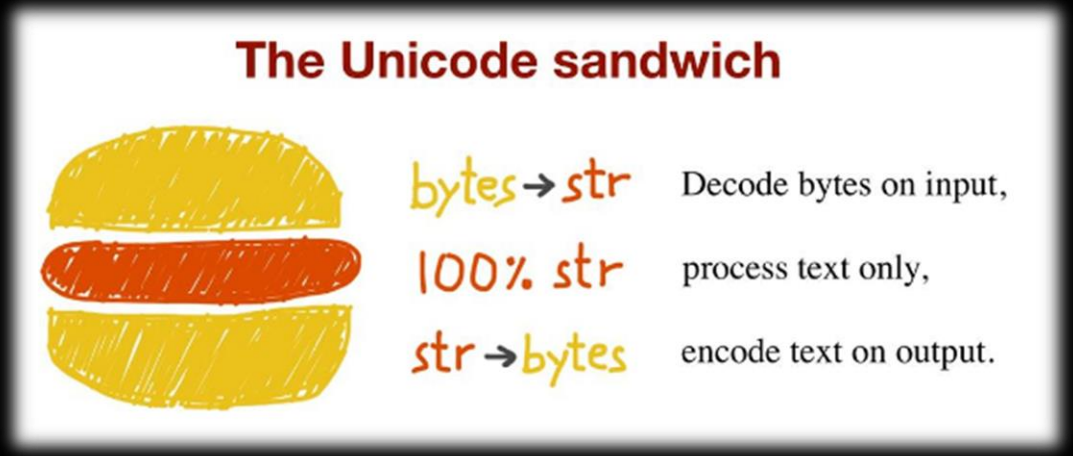
파이썬에서 다양한 인코딩의 문자열을 일관되게 다루려면…

유니코드 샌드위치


The Unicode sandwich
bytes → str
Decode bytes on input,
100% str
process text only,
str → bytes
encode text on output.

유니코드 샌드위치

1. 입력시 bytes를 디코딩.
2. 텍스트만 다룹니다.
3. 출력시 bytes로 인코딩.



인코딩, 디코딩

Encoding **Decoding**

무슨 소리인가?

1. 입력시 bytes를 디코딩.
   -> bytes.decode()로 str객체로 입력받습니다.

2. 문자열을 텍스트로만 다룹니다.
   -> str객체로 문자열을 다룹니다.

3. 출력시 bytes로 인코딩.
   -> str.encode()로 bytes 객체로 출력합니다.

bytes 에서 str 로…

```
def to_str(bytes_or_str):
    if isinstance(bytes_or_str, bytes):
        value = bytes_or_str.decode('utf-8')
    else:
        value = bytes_or_str
    return value
```

str 에서 bytes 로…

```
def to_bytes(bytes_or_str):
    if isinstance(bytes_or_str, str):
        value = bytes_or_str.encode('utf-8')
    else:
        value = bytes_or_str
    return value
```

파일을 다룰 때는 인코딩을 지정할 수 있습니다.

```
with open('test.txt', 'w', encoding='utf-8') as f:
    f.write('파이썬 코딩의 기술')
```

인코딩을 지정하지 않으면 시스템 기본 인코딩을 따릅니다.

```
with open('test.txt', 'r') as f:
    print(f.read())
```

-> 한글 윈도우의 경우 CP949

바이너리 파일은 바이너리 모드로 열어야 합니다.

```
with open('binary.txt', 'wb') as f:
    f.write(os.urandom(10))
```

Ch04. 복잡한 표현식 대신 헬퍼함수를 작성하자

흔한 파이썬의 URL 쿼리 문자열 디코드 코드

```
my_values = parse_qs('red=5&blue=0&green=', keep_blank_values=True)
```

쿼리 문자열 파라미터에 따라서 값이 있을 수도 없을 수도…

```
print('Red          ', red)
print('Green        ', green)
print('Opacity      ', opacity)

>>>
Red          ['5']
Green        ['']
Opacity      None
```

dict의 get함수에 초깃값을 설정할 수 있는것을 이용.

값이 없다면 기본값 0을 반환

빈 dict는 암시적으로 False로 평가됩니다.

```
red = my_values.get('red', [''])[0] or 0
green = my_values.get('green', [''])[0] or 0
opacity = my_values.get('opacity', [''])[0] or 0
```

삼항연산자를 사용할 수 있습니다.

```
red = my_values.get('red', [''])
red = int(red[0]) if red[0] else 0

green = my_values.get('green')
green = int(green[0]) if green[0] else 0

opacity = my_values.get('opacity', [''])
opacity = int(opacity[0]) if opacity[0] else 0
```

복잡하고 재사용성이 떨어집니다.

재사용성을 높이는 방법?

Extract Method

From refactoring

헬퍼함수를 작성합니다.

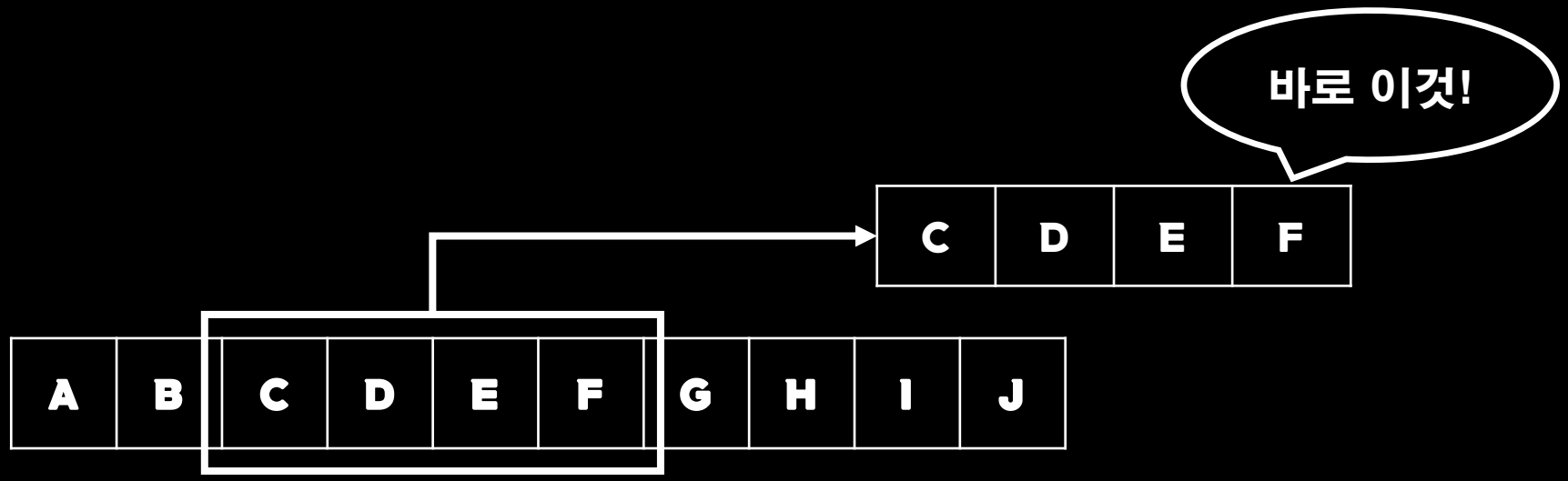
```
def get_first_int(values, key, default=0):
    found = values.get(key, [''])
    if found[0]:
        found = int(found[0])
    else:
        found = default
    return found

red = get_first_int(my_values, 'red', 0)
green = get_first_int(my_values, 'green', 0)
opacity = get_first_int(my_values, 'opacity', 0)
```

Ch05. 시퀀스를 슬라이스하는 방법을 알자

슬라이스

시퀀스(list, str 등...)나 특수 메서드(__getitem__, __setitem__) 을 제공하는 클래스를 조각으로 만드는 문법

조각?



슬라이스는 어떻게?

```
sequence[start:end:stride]
```

코드로 보면?

```
a = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
a[:4]
a[-4:]
a[3:-3]

>>>
['a', 'b', 'c', 'd']
['e', 'f', 'g', 'h']
['d', 'e']
```

처음부터 혹은 끝까지 슬라이스할 때는 생략이 가능합니다.

```
# 처음부터 슬라이스
assert a[:5] == a[0:5]
# 끝까지 슬라이스
assert a[0:] == a[0:len(a)]
```

끝을 기준으로 계산할 때는 음수로 슬라이스 할 수 있습니다.

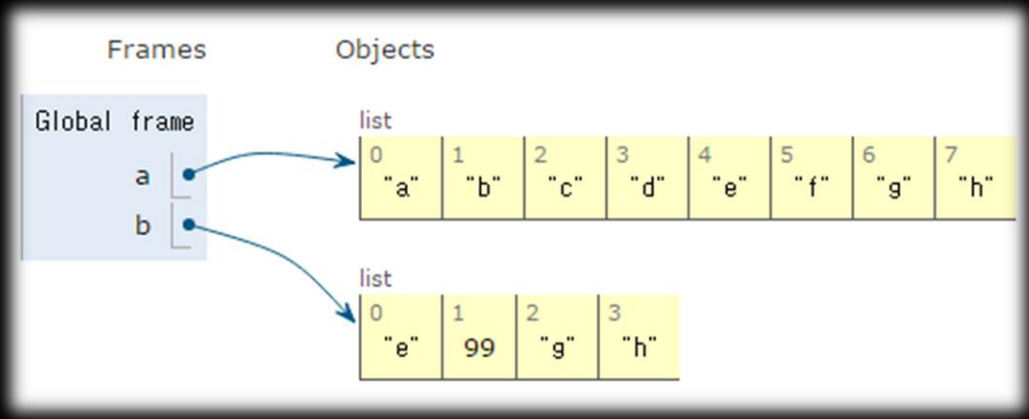
```
a[-3:]
a[2:-1]
a[-3:-1]

>>>
['f', 'g', 'h']
['c', 'd', 'e', 'f', 'g']
['f', 'g']
```

인덱스가 경계를 벗어나도 OK

```
first_twenty_items = a[:20]
last_twenty_items = a[-20:]

>>>
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
```

슬라이스 하면 원본은?
슬라이스 한 결과는 새로운 시퀀스

```
b = a[4:]
print('Before   ', b)
b[1] = 99
print('After    ', b)
print('No change', a)

>>>
Before    ['e', 'f', 'g', 'h']
After     ['e', 99, 'g', 'h']
No change ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
```



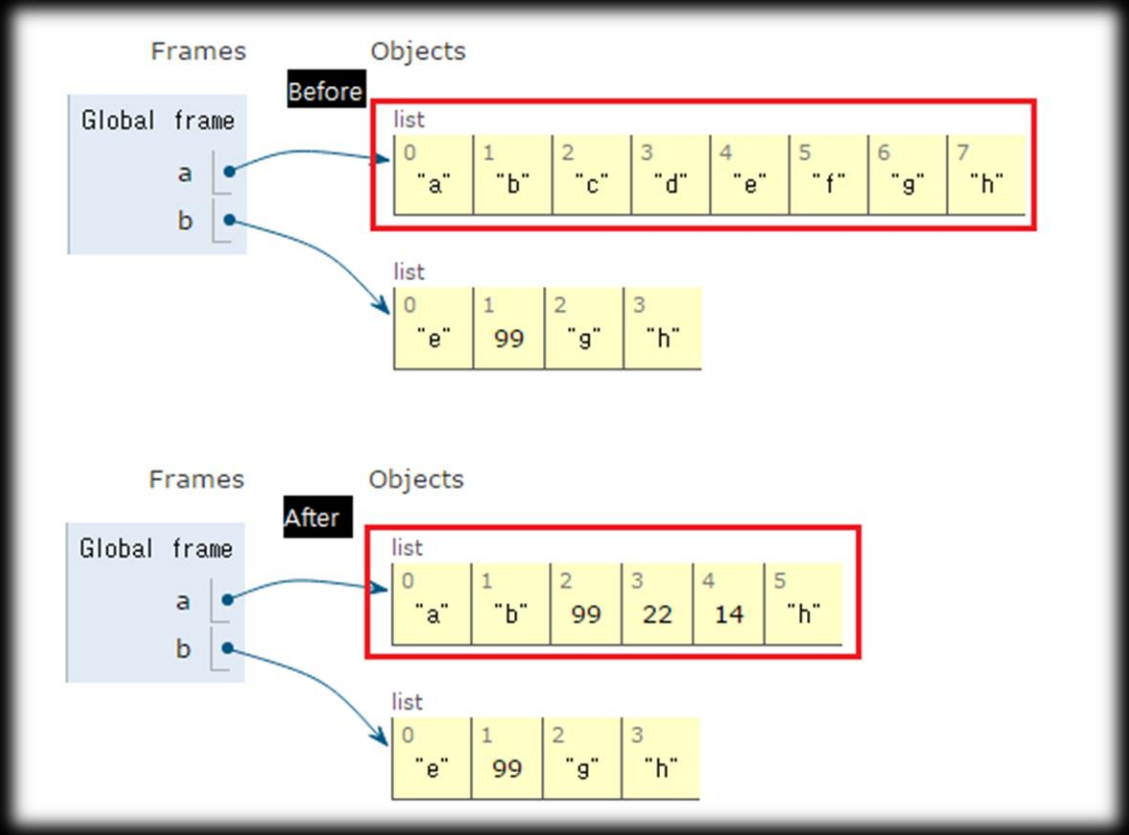
따라서 시작과 끝 인덱스를 모두 생략하면?
복사본을 얻을 수 있습니다.

```
b = a[:]
assert b == a and b is not a
```

슬라이스는 할당에도 사용할 수 있습니다.

```
print('Before   ', a)
a[2:7] = [99, 22, 14]
print('After    ', a)
>>>
Before    ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
After     ['a', 'b', 99, 22, 14, 'h']
```

슬라이스 통한 할당은 지정한 범위를 대체합니다.



Reference

파이썬 코딩의 기술 – 저자 브렛 슬라킨 | 김형철옮김 | 길벗

전문가를 위한 파이썬 – 저자 루시아누 하말류 | 강권학옮김 | 한빛

http://www.pythontutor.com/

Thanks!