

Python

프로그래밍 언어마다 특징도 다양하고 장단점도 다양하다.



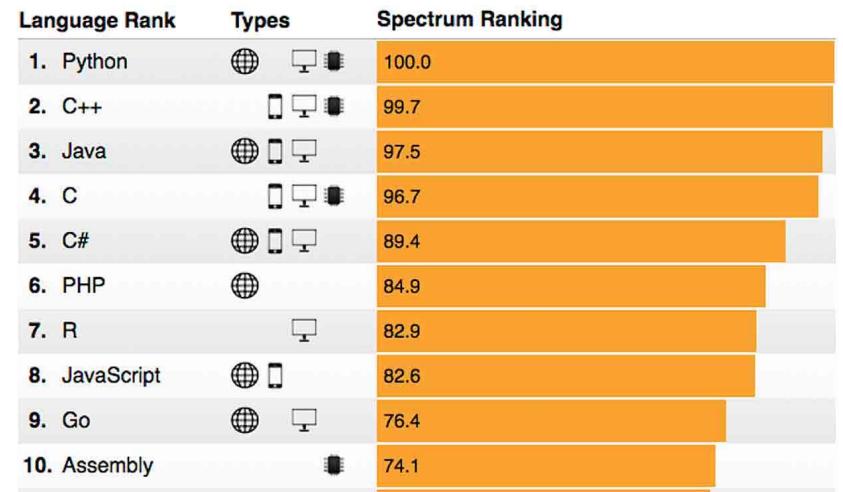
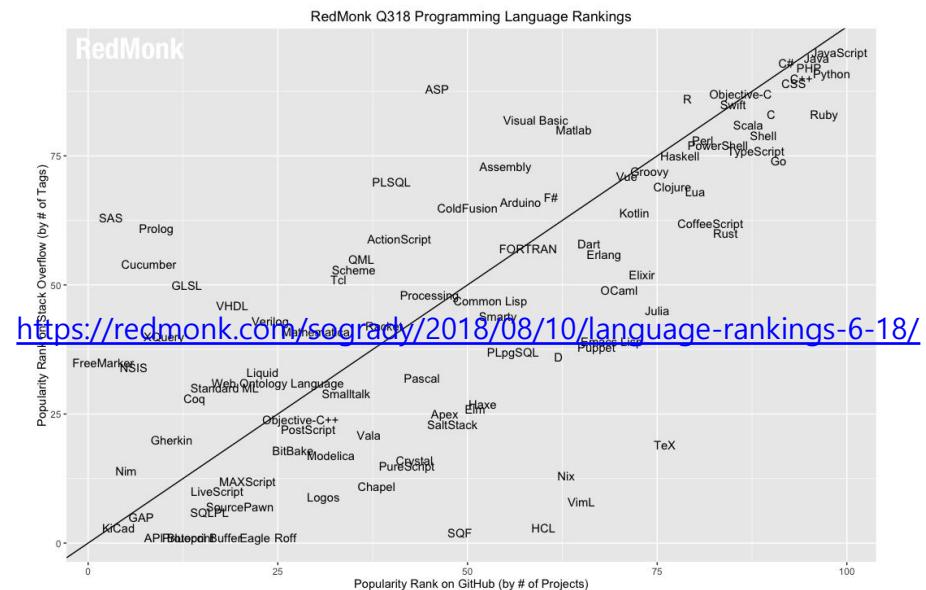
[http://www.inven.co.kr/mobile/board/powerbbs.php?come_idx=2097&category=%EF%BF%BD%EA%B3%97%EF%BF%BD%EF%BF%BD%3E%C2%A0%C2%A0%EF%BF%BD%EF%BF%BD%EF%BF%BD%EF%BF%BD%EF%BF%BD%2Fa%3E%20%7C%3Ca%20href%3D&I=832573](http://www.inven.co.kr/mobile/board/powerbbs.php?come_idx=2097&category=%EF%BF%BD%EA%B3%97%EF%BF%BD%EF%BF%BD%3E%C2%A0%C2%A0%EF%BF%BD%EF%BF%BD%EF%BF%BD%EF%BF%BD%2Fa%3E%20%7C%3Ca%20href%3D&I=832573)
<http://kstatic.inven.co.kr/upload/2017/08/03/bbs/i14621578868.jpg>
<http://redmist.tistory.com/21>
<http://redmist.tistory.com/30>

Worldwide, Sept 2018 compared to a year ago:				
Rank	Change	Language	Share	Trend
1	↑	Python	24.58 %	+5.7 %
2	↓	Java	22.14 %	-0.6 %
3	↑	Javascript	8.41 %	+0.0 %
4	↓	PHP	7.77 %	-1.4 %
5		C#	7.74 %	-0.4 %
6		C/C++	6.22 %	-0.8 %
7		R	4.04 %	-0.2 %
8		Objective-C	3.33 %	-0.9 %
9		Swift	2.65 %	-0.9 %
10		Matlab	2.1 %	-0.3 %

<http://pypl.github.io/PYPL.html>

Sep 2018	Sep 2017	Change	Programming Language	Ratings	Change
1	1		Java	17.436%	+4.75%
2	2		C	15.447%	+8.06%
3	5	▲	Python	7.653%	+4.67%
4	3	▼	C++	7.394%	+1.83%
5	8	▲	Visual Basic .NET	5.308%	+3.33%
6	4	▼	C#	3.295%	-1.48%
7	6	▼	PHP	2.775%	+0.57%
8	7	▼	JavaScript	2.131%	+0.11%
9	-	▲	SQL	2.062%	+2.06%
10	18	▲	Objective-C	1.509%	+0.00%

<https://www.tiobe.com/tiobe-index/>



<https://spectrum.ieee.org/at-work/innovation/the-2018-top-programming-languages>



파이썬은 배우기 쉽고, 강력한 프로그래밍 언어입니다. 효율적인 자료 구조들과 객체 지향 프로그래밍에 대해 간단하고도 효과적인 접근법을 제공합니다. 우아한 문법과 동적 타이핑(typing)은, 인터프리터 적인 특징들과 더불어, 대부분 플랫폼과 다양한 문제 영역에서 스크립트 작성과 빠른 응용 프로그램 개발에 이상적인 환경을 제공합니다.

파이썬 인터프리터와 풍부한 표준 라이브러리는 소스나 바이너리 형태로 파이썬 웹 사이트, <https://www.python.org/>, 에서 무료로 제공되고, 자유롭게 배포할 수 있습니다. 같은 사이트는 제삼자들이 무료로 제공하는 확장 모듈, 프로그램, 도구, 문서들의 배포판이나 링크를 포함합니다.

파이썬 인터프리터는 C 나 C++ (또는 C에서 호출 가능한 다른 언어들)로 구현된 새 함수나 자료 구조를 쉽게 추가할 수 있습니다. 파이썬은 고객화 가능한 응용 프로그램을 위한 확장 언어로도 적합합니다.

<https://docs.python.org/ko/3/tutorial/index.html>

<https://docs.python.org/ko/3/tutorial/appetite.html>

Python

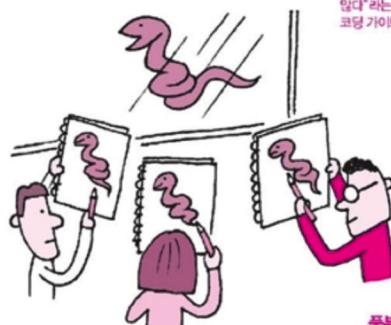
파이썬

웹 앱에서 인공지능까지
인기가 급상승하며 주목받는 언어

이란 언어

데이터 분석에 장점을 가진 스크립트 언어로 데이터 사이언티스
트러는 직종에 인기가 있다. 해외에서는 웹 애플리케이션의 개
발 언어로도 많이 사용되고 있으며, Python 프로그래머의 평균
연봉이 높은 것이 화제가 되기도 한다.

최근에는 기계학습 등 인공지능의 개발에도 많이 사용되고 있다.
버전 2x와 3x은 일부 호환이 되지 않지만 두 버전 모두 이용자
가 많다.



다양한 구현이 있다

원래의 처리계인 'CPython'뿐만 아니라
'Jython'이나 'PyPy', 'Cython'이나 'IronPython'
등 다양한 구현이 있으며, 각각 특징이 있다.

Column

The Zen of Python

Python 프로그래머가 가치야 할 마음가짐이 정리된 말.
Python으로 소스코드를 작성할 때 '단순'과 '가독성'을 실현하기
위한 19개의 문장으로 구성되어 있다.



탄생
1991년
만든 사람
Guido van Rossum
주요 용도
웹 애플리케이션,
인공지능
분류
설치형·할수형·格外지
향형/인터프리터

인엔트가 중요

많은 언어가 'if' 등을 통해 구조를 표현하는 반면, Python에서는 인엔트들이 쓰기로 표현한다.
"코드는 쓰는 것이다"라는 것이 더
많다"라는 의미에서 PEP8이라는
코딩 가이드도 제공되고 있다.

중부한 통계 라이브러리

데이터 분석과 기계학습에 사용
할 수 있는 라이브러리가 중부하
게 갖추어져 있으며 무료로 사용
할 수 있다. 'NumPy'나 'Pandas',
'matplotlib' 등이 특히 유명하다.

기억해야 할 키워드



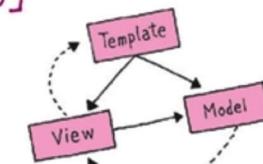
pip

Python 패키지 관리 시스템.
최신 Python이 기본으로 제공되어 검색
및 설치, 업데이트 등도 가능하다.
Git 저장소에서 직접 설치할 수도 있다.

[`x for x in range(10) if is_prime(x)`]

리스트 컴프리헨션

리스트를 생성할 때 사용되는 표기 방법.
수학에서 집합을 나타낼 때 사용되는 `{x |
x < 10 이하의 소수}`라는 표현에 기깝다.
for 루프보다 빠르게 처리할 수 있다.



Django

Python으로 사용되는 웹 애플리케이션
프레임워크 중에서도 가장 인기가 있다.
Instagram과 Pinterest의 개발에도 사용
되고 있는 것으로 알려져 있다.

프로그래밍 예제 히노이의 탑(hanoi.py)

```

# 히노이의 탑
def hanoi(n, from_, to, via):
    if n > 1:
        hanoi(n - 1, from_, via, to)
        print from_ + " -> " + to
        hanoi(n - 1, via, to, from_)
    else:
        print from_ + " -> " + to

# 표준 입력에서 단수를 받아서 실행
n = input()
hanoi(n, "a", "b", "c")
  
```

인엔트가 중요

탭 문자 또는 4 문자의 공백
이 사용되는 경우가 많다.
하나의 블록은 동일한 들여
쓰기로 동일이 필요하다.

Humorous



<https://www.youtube.com/watch?v=jiu0IYQIPqE>



<https://gvanrossum.github.io/>

Life is too short, You need

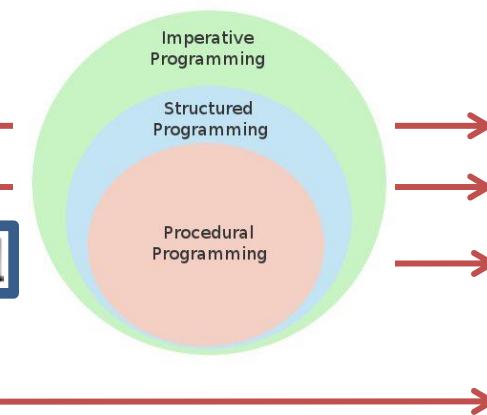
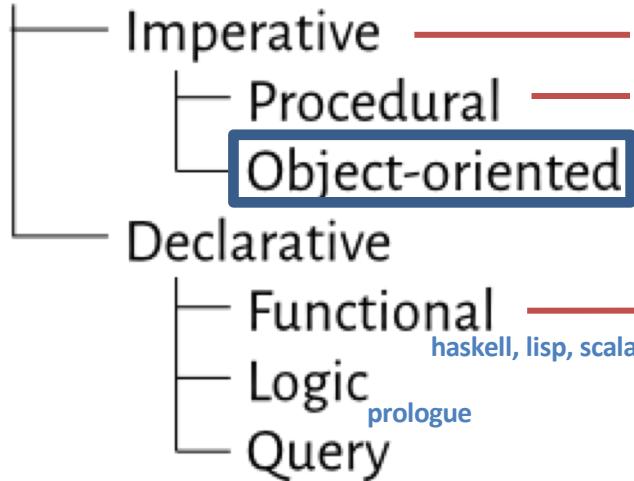


생산성

**Effective
Efficient**

Multi Paradigm

Programming languages



Everything is an object

The same problem can be solved and expressed in **different ways**

Glue Language



■ CPython (c.f : cython)

- De facto
- Python Software Foundation 관리

■ 대체, 플랫폼 특정 구현체

- PyPy, Stackless
- IronPython, PythonNet, Jython
- Skulpt, Brython
- MicroPython

<https://wiki.python.org/moin/AdvocacyWritingTasks/GlueLanguage>
<https://www.python.org/doc/essays/omg-darpa-mcc-position/>

Library & Tool

■ 다양한 종류의 수많은 **standard library** 기본 탑재

플랫폼에 상관없음

■ 다양한 종류의 수많은 **open-source libraries**

the Python Package Index: <https://pypi.python.org>

pip

the de facto

default package manager

버전과 라이브러리에 대한 의존성 문제?

General Purpose



C.f)
Domain-specific



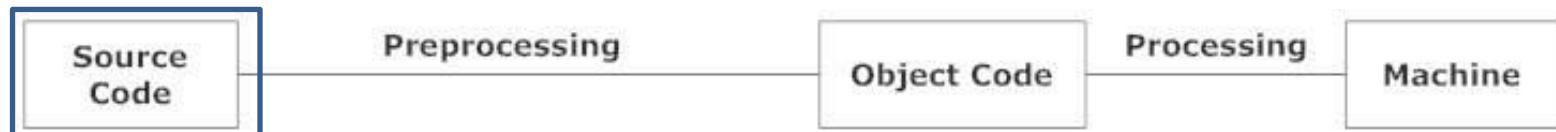
https://en.wikipedia.org/wiki/List_of_Python_software
<https://github.com/vinta/awesome-python>

c.f) 모바일 지원에 대한 약점?

Dynamic Language

<https://wiki.python.org/moin/Why%20is%20Python%20a%20dynamic%20language%20and%20also%20a%20strongly%20typed%20language>

Interpreted Language script

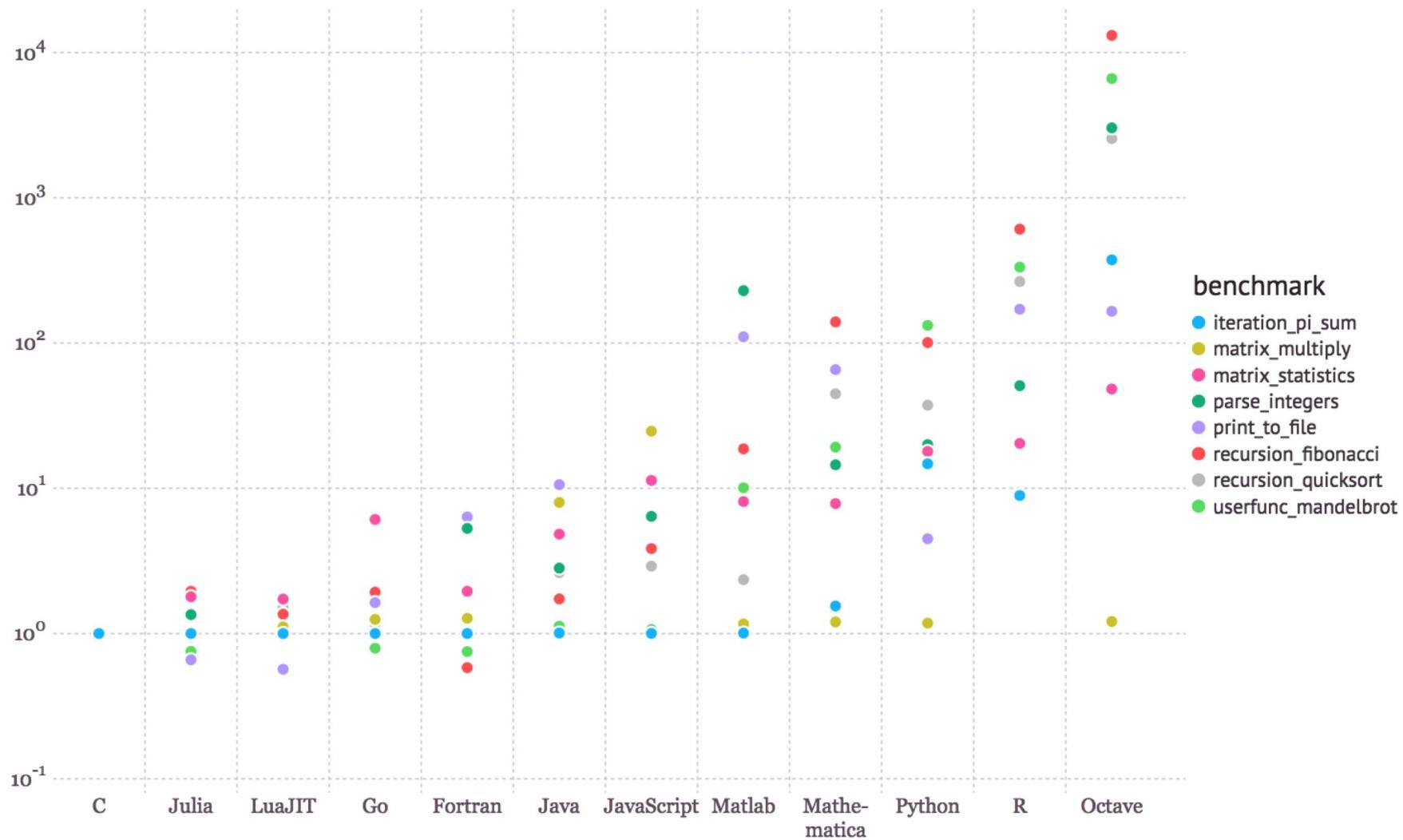


Text 형태

Interpreter vs Compiler



JIT 없는 dynamic 인터프리터 언어



Python 개발 환경



2020년까지 bugfix만, 지원 종료 예정

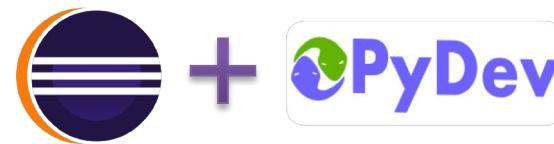
<https://www.python.org/dev/peps/pep-0373/>

REPL vs IDE vs Text Editor

don't need
to use
the `print()` function

Interactive Mode
playground

https://en.wikipedia.org/wiki/Read–eval–print_loop

REPL**IDE****Text Editor**

vi, emacs...

■ Online

- Python.org Online Console: www.python.org/shell
- Python Fiddle: pythonfiddle.com
- Repl.it: repl.it
- Trinket: trinket.io
- Python Anywhere: www.pythonanywhere.com
- codeskulptor : <http://www.codeskulptor.org/viz/index.html>, <http://py3.codeskulptor.org/>
- <http://www.pythontutor.com/>

■ iOS (iPhone / iPad)

- [Pythonista app](#)

■ Android (Phones & Tablets)

- [Pydroid 3](#)

■ pip

- 모듈이나 패키지를 쉽게 설치할 수 있도록 도와주는 도구 (De Facto)

- 설치된 패키지 확인
pip list / pip freeze
pip show '패키지 이름'
- 패키지 검색
pip search '패키지 이름'
- 패키지 설치
pip install '패키지 이름'
- 패키지 업그레이드
pip intall --upgrade '패키지 이름' pip install -U '패키지 이름'
- 패키지 삭제
pip uninstall '패키지 이름'

■ conda

■ 윈도우 라이브러리

- <https://www.lfd.uci.edu/~gohlke/pythonlibs/>

■ Distribution = a software bundle

- 용량 문제, 관리 문제, 충돌 문제, 버전 호환성 등의 문제를 해결 가능
- Python interpreter (Python standard library) + package managers



[Anaconda](#)



[ActivePython](#)



[Enthought Canopy](#)



[python\(x,y\)](#)

■ 가상환경

○ Application-level isolation of Python environments

- virtualenv / venv
 - VirtualenvWrapper
- buildout
 - <http://www.buildout.org/en/latest/>

○ System-level environment isolation

- docker
 - <https://djangostars.com/blog/what-is-docker-and-how-to-use-it-with-python/>
 - <https://www.pycon.kr/2015/program/71>

■ Packing isolation

- pipenv
 - dependencies를 간단하게 관리

■ REPL 실행

- 대화형 모드 - 인터프리터
- \$ python3

■ file.py 파일 실행

- \$ python3 file.py

■ "print('hi')" 문 실행

- \$ python3 -c "print('hi')"

■ Execute the file.py file, and drop into REPL with namespace of file.py:

- \$ python3 -i file.py

■ json/tool.py 모듈 실행

- \$ python3 -m json.tool

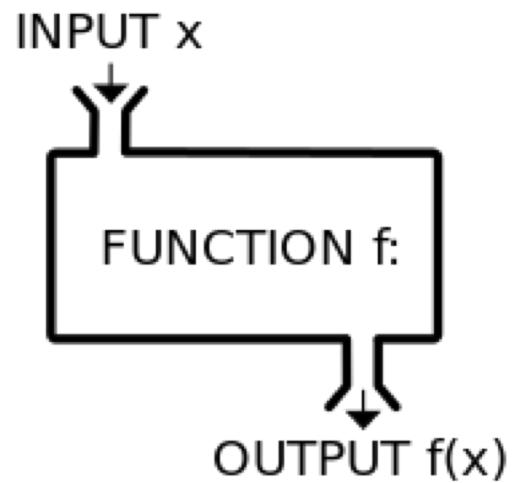
■ 대화형 환경에서는, 마지막에 인쇄된 표현식은 변수 _로 표현 가능

- tax = 12.5 / 100
- price = 100.50
- price * tax 12.5625
- price + _ 113.0625
- round(_, 2) 113.06

프로그래밍의 구성 요소

프로그래밍

문법(키워드, 식, 문)을 이용해서
값을 입력받고, 계산/변환하고, 출력하는 흐름을 만드는 일



The Zen of Python (PEP 20)

import this

프로그래밍 언어의 문법

- 생각을 표현해내는 도구인 동시에, 생각이 구체화되는 틀
- 언어가 지향하고자 하는 철학에 따라 고안

문법에 대한 올바른 이해는 프로그래밍을 위한 필수적인 과정

<https://legacy.python.org/doc/essays/>

BDFL(Benevolent Dictator For Life!)

자비로운 종신 독재자 : [Guido van Rossum](#), 파이썬의 창시자

https://en.wikipedia.org/wiki/Benevolent_dictator_for_life

- 아름다운 것이 보기 싫은 것보다 좋다.
- 명시적인 것이 암묵적인 것보다 좋다.
- 간단한 것이 복합적인 것보다 좋다.
- 복합적인 것이 복잡한 것보다 좋다.
- 수평한 것이 중첩된 것보다 좋다.
- 희소한 것이 밀집된 것보다 좋다.
- 가독성이 중요하다.
- 규칙을 무시할 만큼 특별한 경우는 없다.
- 하지만 실용성이 순수함보다 우선한다.
- 에러가 조용히 넘어가서는 안된다.
- 명시적으로 조용히 만든 경우는 제외한다.
- 모호함을 만났을 때 추측의 유혹을 거부해라.
- 하나의 — 가급적 딱 하나의 — 확실한 방법이 있어야 한다.
- 하지만 네덜란드 사람(귀도)이 아니라면 처음에는 그 방법이 명확하게 보이지 않을 수 있다.
- 지금 하는 것이 안하는 것보다 좋다.
- 하지만 안하는 것이 이따금 지금 당장 하는 것보다 좋을 때가 있다.
- 설명하기 어려운 구현이라면 좋은 아이디어가 아니다.
- 설명하기 쉬운 구현이라면 좋은 아이디어다.
- 네임스페이스는 아주 좋으므로 더 많이 사용하자!

False	elif	lambda
None	else	nonlocal
True	except	not
and	finally	or
as	for	pass
assert	from	raise
break	global	return
class	if	try
continue	import	while
def	in	with
del	is	yield

import keyword
identifier로 사용하지 않음

Expression vs Statement

표현식 vs 구문

Declaration vs Assignment

선언 vs 할당(대입)

■ 표현식(expression) = 평가식 = 식

○ 값

○ 값들과 연산자를 함께 사용해서 표현한 것

○ 이후 “평가”되면서 하나의 특정한 결과값으로 축약

➤ 수

– $1 + 1$

» $1 + 1$ 이라는 표현식은 평가될 때 2라는 값으로 계산되어 축약

– 0과 같이 값 리터럴로 값을 표현해놓은 것

➤ 문자열

– 'hello world'

– "hello" + ", world"

➤ 함수

– lambda (일반적으로 Functional Paradigm에서 지원)

○ 궁극적으로 “평가”되며, 평가된다는 것은 결국 하나의 값으로 수렴한다는 의미

➤ python에서는 기본적으로 left-to-right로 평가

■ 구문(statement) = 문

○ 예약어(reserved word, keyword)와 표현식을 결합한 패턴

○ 컴퓨터가 수행해야 하는 하나의 단일 작업(instruction)을 명시.

➤ 활당(대입, assigning statement)

– python에서는 보통 '바인딩(binding)'이라는 표현을 씀, 어떤 값에 이름을 붙이는 작업.

➤ 선언(정의, declaration)

– 재사용이 가능한 독립적인 단위를 정의. 별도의 선언 문법과 그 내용을 기술하는 블럭 혹은 블럭들로 구성.

» Ex) python에서는 함수나 클래스를 정의

– 블럭

» 여러 구문이 순서대로 나열된 덩어리

» 블럭은 여러 줄의 구문으로 구성되며, 블럭 내에서 구문은 위에서 아래로 쓰여진 순서대로 실행.

» 블럭은 분기문에서 조건에 따라 수행되어야 할 작업이나, 반복문에서 반복적으로 수행해야 하는 일련의 작업을 나타낼 때 사용하며, 클래스나 함수를 정의할 때에도 쓰임.

➤ 조건(분기) : 조건에 따라 수행할 작업을 나눌 때 사용.

– Ex) if 문

➤ 반복문 : 특정한 작업을 반복수행할때 사용.

– Ex) for 문 및 while 문

➤ 예외처리

■ 값에 대한 type이 중요함

■ 값에 따라 서로 다른 기술적인 체계가 필요

- 지원하는 연산 및 기능이 다르기 때문

■ 컴퓨터에서는 이진수를 사용해서 값을 표현하고 관리

- 정확하게 표현하지 못할 수가 있음

- 숫자형 = numeric

- 산술 연산을 적용할수 있는 값
- 정수 : 0, 1, -1 과 같이 소수점 이하 자리가 없는 수. 수학에서의 정수 개념과 동일. (int)
- 부동소수 : 0.1, 0.5 와 같이 소수점 아래로 숫자가 있는 수. (float)
- 복소수 : Python에서 기본적으로 지원

- 문자, 문자열

- 숫자 "1", "a", "A" 와 같이 하나의 낱자를 문자라 하며, 이러한 문자들이 1개 이상있는 단어/문장와 같은 텍스트
- Python에서는 낱자와 문자열 사이에 구분이 없이 기본적으로 str 타입을 적용
 - byte, bytearry

- 불리언 = boolean

- 참/거짓을 뜻하는 대수값. 보통 컴퓨터는 0을 거짓, 0이 아닌 것을 참으로 구분
- True 와 False 의 두 멤버만 존재 (bool)
- Python에서는 숫자형의 일부

○ Compound = Container = Collection

- 기본적인 데이터 타입을 조합하여, 여러 개의 값을 하나의 단위로 묶어서 다루는 데이터 타입
- 논리적으로 이들은 데이터 타입인 동시에 데이터의 구조(흔히 말하는 자료 구조)의 한 종류. 보통 다른 데이터들을 원소로 하는 집합처럼 생각되는 타입들
- Sequence
 - list : 순서가 있는 원소들의 묶음
 - tuple : 순서가 있는 원소들의 묶음. 리스트와 혼동하기 쉬운데 단순히 하나 이상의 값을 묶어서 하나로 취급하는 용도로 사용
 - range
- Lookup
 - mapping
 - » dict : 그룹내의 고유한 이름인 키와 그 키에 대응하는 값으로 이루어지는 키값 쌍(key-value pair)들의 집합.
 - set : 순서가 없는 고유한 원소들의 집합.

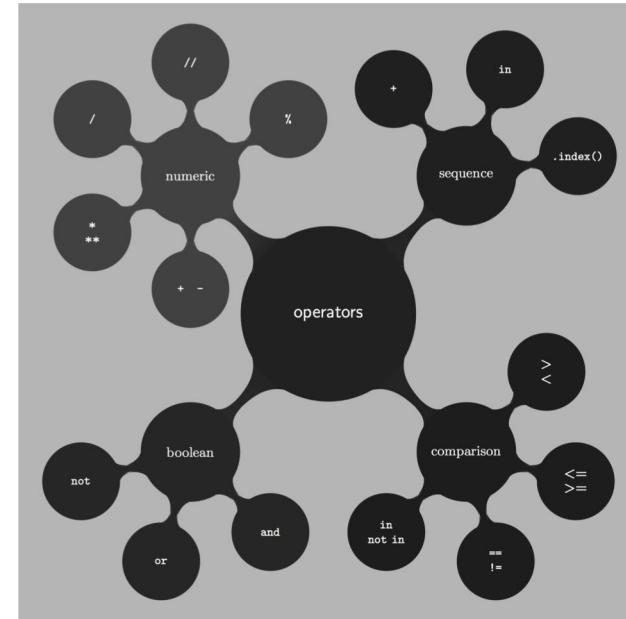
○ None

- 존재하지 않음을 표현하기 위해서 "아무것도 아닌 것"을 나타내는 값
- 어떤 값이 없는 상태를 가리킬만한 표현이 마땅히 없기 때문에 "아무것도 없다"는 것으로 약속해놓은 어떤 값을 하나 만들어 놓은 것
- None 이라고 대문자로 시작하도록 쓰며, 실제 출력해보아도 아무것도 출력되지 않음.
- 값이 없지만 False 나 0 과는 다르기 때문에 어떤 값으로 초기화하기 어려운 경우에 쓰기도 함

Literal vs Type(Object-oriented)

값, 기호로 고정된 값을 대표
간단하게 값 생성

Operation



■ 산술연산

- 계산기

■ 비교연산

- 동등 및 대소를 비교. 참고로 '대소'비교는 '전후'비교가 사실은 정확한 표현
- 비교 연산은 숫자값 뿐만 아니라 문자열 등에 대해서도 적용할 수 있음.

■ 비트연산

■ 멤버십 연산

- 특정한 집합에 어떤 멤버가 속해있는지를 판단하는 것으로 비교연산에 기반을 둠
- is, is not : 값의 크기가 아닌 값 자체의 정체성(identity)이 완전히 동일한지를 검사
- in, not in : 멤버십 연산. 어떠한 집합 내에 원소가 포함되는지를 검사 ('a' in 'apple')

■ 논리연산

- 비교 연산의 결과는 보통 참/거짓. 이러한 불리언값은 다음의 연산을 적용. 참고로 불리언외의 타입의 값도 논리연산을 적용

	Operator	Description
lowest precedence	or	Boolean OR
	and	Boolean AND
	not	Boolean NOT
	in, not in	membership
	==, !=, <, <=, >, >=, is, is not	comparisons, identity
		bitwise OR
	^	bitwise XOR
	&	bitwise AND
	<<, >>	bit shifts
	+, -	addition, subtraction
	*, /, //, %	multiplication, division, floor division, modulo
	+x, -x, ~x	unary positive, unary negation, bitwise negation
highest precedence	**	exponentiation

PEMDAS :

Parentheses - Exponentiation - Multiplication - Division - Addition - Subtraction

Python 문법

■ Python 문법의 특징

- 규칙(키워드 등)의 수가 적고 대부분의 규칙이 일관된 맥락을 가지고 있음

- Python 3 : 35 (True, False)
- Python 2 : 31
- C++ : 62
- Java : 53
- Visual Basic : >120

이해하고 배우기 쉬움?

- **case sensitive**

- **space** sensitive (Indentation / line oriented)

- Indentation used to define **block structure**
 - **Multiple statements can occur at same level of indentation**
- Implicit continuation across unclosed bracketing ((...), [...], {...})
- Forced continuation after `\#` at end of line
- possible to combine multiple statements (Compound statements) on a single line
 - semi-colon(;) is a statement separator
 - **strongly discouraged**

■ <https://docs.python.org/ko/3/index.html>

■ 주석

- 해시 문자(#로 시작하고 줄의 끝까지 이어집니다.

- 주석은 줄의 처음에서 시작할 수도 있고, 공백이나 코드 뒤에 나올 수도 있습니다.
- 하지만 문자열 리터럴 안에는 들어갈 수 없습니다. 문자열 리터럴 안에 등장하는 해시 문자는 주석이 아니라 해시 문자일 뿐입니다.
- 주석은 코드의 의미를 정확히 전달하기 위한 것이고, 파이썬이 해석하지 않는 만큼, 예를 입력할 때는 생략해도 됩니다.

■

- Line-based, i.e., but independent of indentation (but advisable to do so)
- There are no multi-line comments

- " " "

■ #!

- Shell-script friendly, i.e., **#!** as first line
- [https://en.wikipedia.org/wiki/Shebang_\(Unix\)](https://en.wikipedia.org/wiki/Shebang_(Unix))

■ # -*- coding: <encoding-name> -*-

- https://docs.python.org/3/reference/lexical_analysis.html#encoding-declarations

- 대부분 언어는 서로 다른 스타일로 작성될 (또는 더 간략하게, 포맷될) 수 있음. 어떤 것들은 다른 것들보다 더 읽기 쉬움. 다른 사람들이 코드를 읽기 쉽게 만드는 것은 항상 좋은 생각이고, 훌륭한 코딩 스타일을 도입하는 것은 그렇게 하는 데 큰 도움을 줌.

■ style guide

- Offers guidance on accepted convention, and when to follow and when to break with convention
- mostly on layout and naming

■ But there is also programming guidance

■ PEP 8 conformance can be checked automatically

- E.g., <http://pep8online.com/>, pep8, flake8

■ Pythonic style

- 다른 언어와 다른 특색
- 간단 명료, 풍부한 표현력
- Encouraged style is idiomatically more functional than procedural

<https://www.python.org/dev/peps/pep-0008/#naming-conventions>

■ Camel Case

- Second and subsequent words are capitalized, to make word boundaries easier to see. (Presumably, it struck someone at some point that the capital letters strewn throughout the variable name vaguely resemble camel humps.)
 - Example: `numberOfCollegeGraduates`

■ Pascal Case = CapWords

- Identical to Camel Case, except the first word is also capitalized.
 - Example: `NumberOfCollegeGraduates`
 - Camel Case와 혼용해서 사용하기도 함

■ Snake Case

- Words are separated by underscores.
 - Example: `number_of_college_graduates`

■ PEP8

○ 들려 쓰기에 4-스페이스를 사용하고, 탭을 사용하지 마세요.

- 4개의 스페이스는 작은 들여쓰기 (더 많은 중첩 도를 허락한다) 와 큰 들여쓰기 (읽기 쉽다) 사이의 좋은 절충입니다.
탭은 혼란을 일으키고, 없애는 것이 최선입니다.

○ 79자를 넘지 않도록 줄 넘김 하세요.

- 이것은 작은 화면을 가진 사용자를 돋고 큰 화면에서는 여러 코드 파일들을 나란히 볼 수 있게 합니다.

○ 함수, 클래스, 함수 내의 큰 코드 블록 사이에 빈 줄을 넣어 분리하세요.

- 가능하다면, 주석은 별도의 줄로 넣으세요.
- 독스트링을 사용하세요.

○ 연산자들 주변과 콤마 뒤에 스페이스를 넣고, 괄호 바로 안쪽에는 스페이스를 넣지 마세요

- `a = f(1, 2) + g(3,4).`

○ 클래스와 함수들에 일관성 있는 이름을 붙이세요;

- 관례는 클래스의 경우 CamelCase, 함수와 메서드의 경우 lower_case_with_underscores 입니다. 첫 번째 메서드 인자
의 이름으로는 항상 `self` 를 사용하세요

○ 여러분의 코드를 국제적인 환경에서 사용하려고 한다면 특별한 인코딩을 사용하지 마세요. 어떤 경우에 도 파이썬의 기본, UTF-8, 또는 단순 ASCII조차, 이 최선입니다.

○ 마찬가지로, 다른 언어를 사용하는 사람이 코드를 읽거나 유지할 약간의 가능성만 있더라도, 식별자에 ASCII 이외의 문자를 사용하지 마세요.

Assignment

binding

identifier → **a** = **23** ← value
= name



Assignment operator

```
parrot = 'Dead'
```

Simple assignment

```
x = y = z = 0
```

Assignment of single value to multiple targets
= Aliasing

```
lhs, rhs = lhs, rhs
```

Assignment of multiple values to multiple targets

```
counter += 1
```

Augmented assignment

```
world = 'Hello'
```

Global assignment from within a function

```
def farewell():
```

```
    global world
```

```
    world = 'Goodbye'
```

- augmented assignments are **statements** not expressions
- Cannot be **chained** and can only have a single target
- no ++ or --

Arithmetic

`+ =`
`- =`
`* =`
`/ =`
`% =`
`// =`
`** =`

Bitwise

`& =`
`| =`
`^ =`
`>> =`
`<< =`

■ 일반적 의미

- 아직 알려지지 않거나 어느 정도까지만 알려져 있는 양이나 정보에 대한 상징적인 **이름**
 - 대수학 : 수식에 따라서 변하는 값
 - 상수 : 변하지 않는 값

■ 프로그래밍에서의 변수

- 값을 기억해 두고 필요할 때 활용할 수 있음
 - 중간 계산값을 저장하거나 누적 등

■ Python

- 값(객체)을 저장하는 메모리 상의 공간을 가르키는(**object reference**) **이름**
 - A variable is a name for an object **within a scope**
 - **None** is a reference to nothing
- Python은 모든 것이 객체이므로 변수보다는 **식별자**로 언급. 변수로 통용해서 사용하기도 함
 - **변수, 상수, 함수, 사용자 정의 타입** 등에서 다른 것들과 구분하기 위해서 사용되는 변수의 이름, **상수의 이름, 함수의 이름, 사용자 정의 타입의 이름** 등 '이름'을 일반화 해서 지칭하는 용어
 - **Python에는 이름있는 상수 개념 없음**
- **변수의 경우, 선언 및 할당이 동시에 이루어져야 함**
 - A variable is created in the current **scope** on first assignment to its name
 - It's a runtime error if an unbound variable is referenced

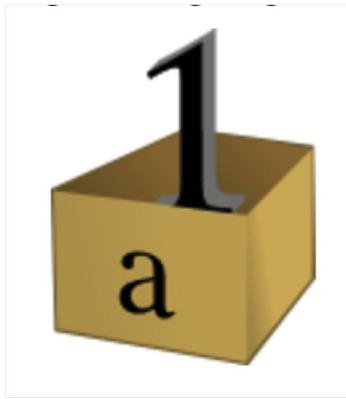
■ case-sensitive names

- E.g., functions, classes and variables

■ Some identifier classes have reserved meanings

Class	Example	Meaning
<code>_*</code>	<code>_load_config</code>	Not imported by wildcard module imports — a convention to indicate privacy in general
<code>__ __</code>	<code>__init__</code>	System-defined names, such as special method names and system variables
<code>__ *</code>	<code>__cached_value</code>	Class-private names, so typically used to name private attributes

Object Reference

C

```
int a = 1;
```



```
a = 2;
```

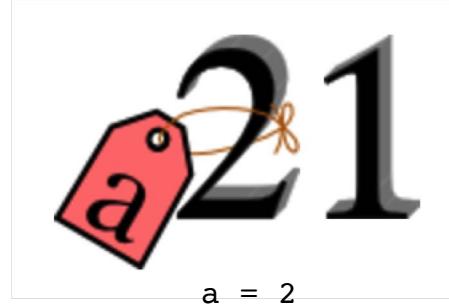


```
int b = a;
```

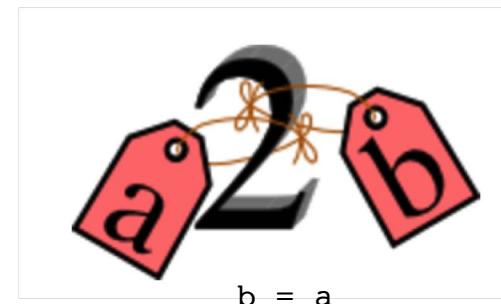
Python



```
a = 1
```

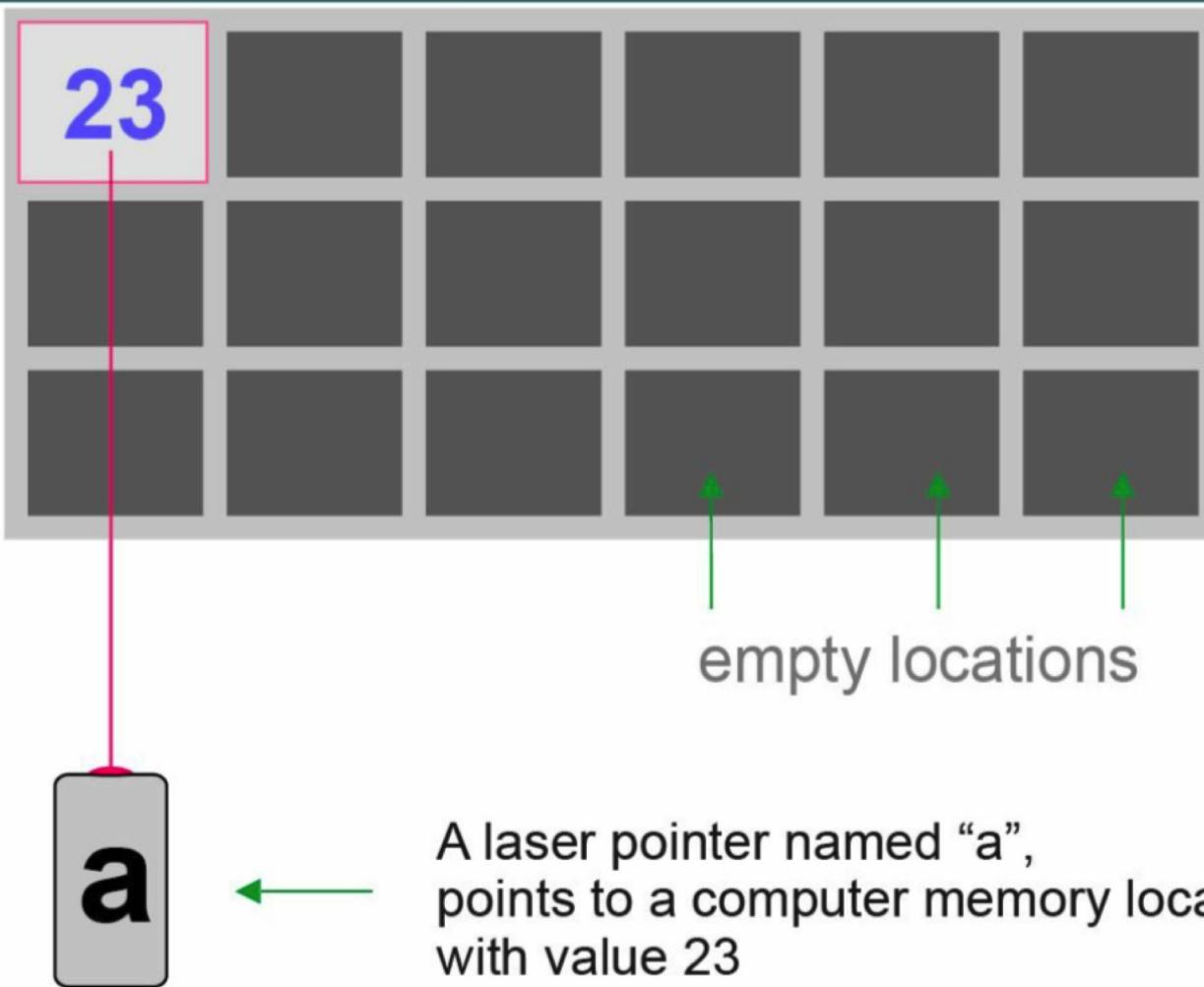


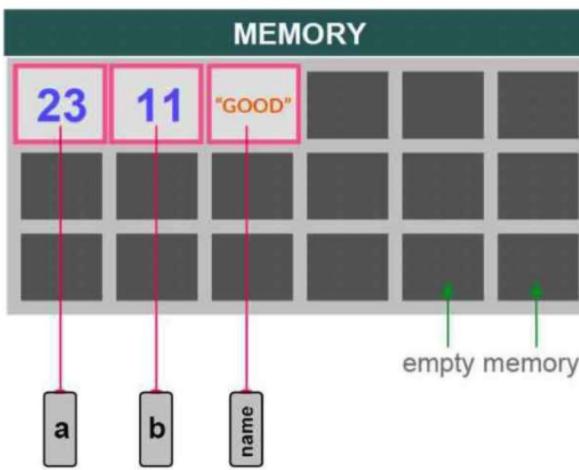
```
a = 2
```



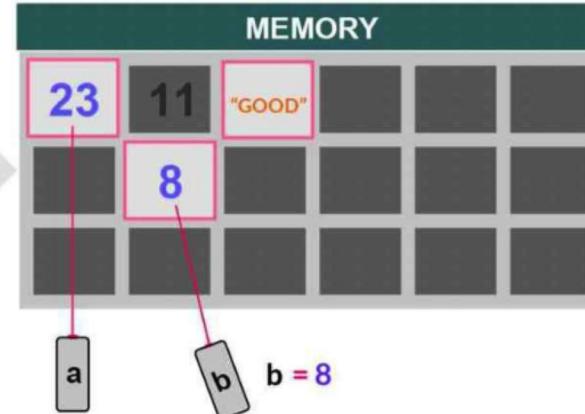
```
b = a
```

COMPUTER MEMORY

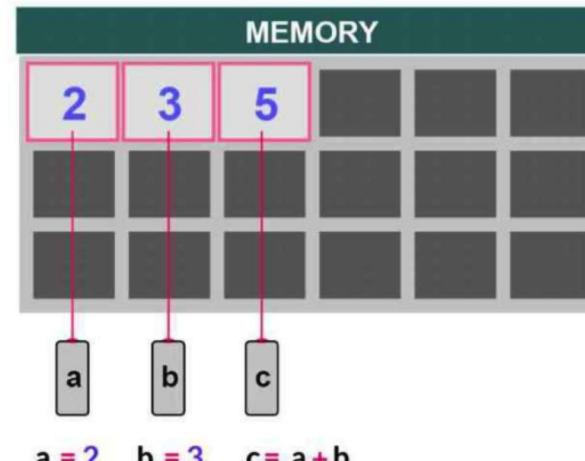
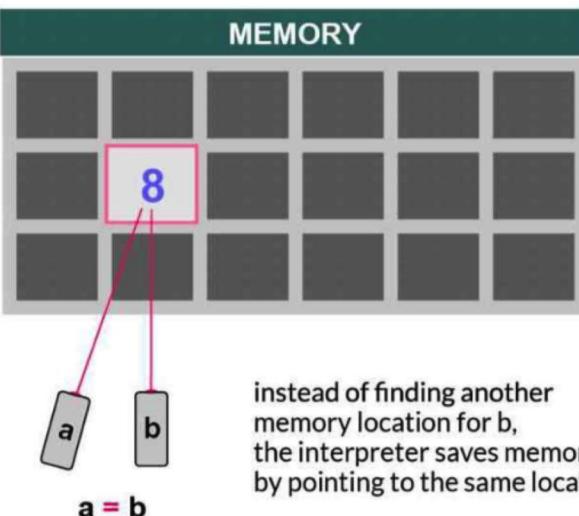




lets change the value of b from 11 to 8



Because numbers are immutable, "b" changes location to the new value.
When there is no reference to a memory location the value fades away and the location is free to use again.
This process is known as garbage collection



Variable Creation

Code

```
status = "off"
```

What Computer Does

Variables

Objects



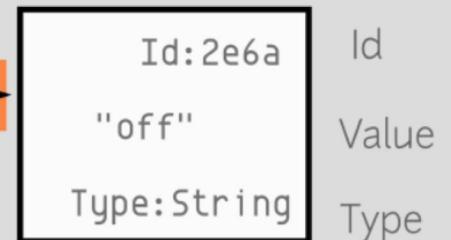
Step 1: Python creates an object

```
status = "off"
```

Variables

Objects

status →



Step 2: A variable is created

Rebinding Variables

Code

a = 400

What Computer Does

Variables

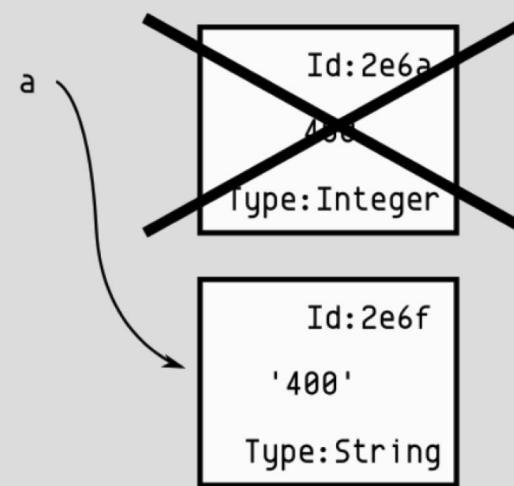
Objects



a = '400'

Variables

Objects



```
import sys  
>>> names = []  
>>> sys.getrefcount(names)
```

Old Object is Garbage Collected

Mutability

■ 객체 (Objects)

- 데이터(data)와 행위를 추상화한 것(abstraction)
- Python 프로그램의 모든 데이터는 객체나 객체 간의 관계로 표현
 - (폰 노이만(Von Neumann)의 "프로그램 내장식 컴퓨터(stored program computer)" 모델을 따르고, 또 그 관점에서 코드 역시 객체로 표현
- 모든 객체는 **아이덴티티(identity)**, **형(type)**, **값(value)**을 가짐.
 - **아이덴티티**
 - 한 번 만들어진 후에는 변경되지 않음.
 - 메모리상에서의 객체의 주소로 생각.
 - 'is' 연산자는 두 객체의 아이덴티티를 비교;
 - id() 함수는 아이덴티티를 정수로 표현한 값을 반환.
 - **형**
 - 객체가 지원하는 연산들을 정의 (예를 들어, "길이를 갖고 있나? ")
 - 그 형의 객체들이 가질 수 있는 가능한 값들을 정의.
 - type() 함수는 객체의 형(이것 역시 객체다)을 돌려줌.
 - 아이덴티티와 마찬가지로, 객체의 형 (*type*) 역시 변경되지 않음
 - » 어떤 제한된 조건으로, 어떤 경우에 객체의 형을 변경하는 것이 가능.

■ 값을 변경할 수 있는 객체들을 **가변(mutable)**

■ 만들어진 후에 값을 변경할 수 없는 객체들을 **불변(immutable)**

○ 가변 객체에 대한 참조를 저장하고 있는 불변 컨테이너의 값은 가변 객체의 값이 변할 때 변경된다고 볼 수도 있음; 하지만 저장하고 있는 객체들의 집합이 바뀔 수 없으므로 컨테이너는 여전히 불변이라고 여겨짐. 따라서 불변성은 엄밀하게는 변경 불가능한 값을 갖는 것과는 다름.

○ 객체의 가변성(mutability)은 그것의 형에 의해 결정

○ 제자리(inplace)에서 멤버를 추가, 삭제 또는 재배치하고 특정 항목을 반환하지 않는 메서드는 컬렉션 인스턴스 자체를 반환하지 않고 None 을 반환

■ 형은 거의 모든 측면에서 객체가 동작하는 방법에 영향을 줌.

○ 불변형의 경우, 새 값을 만드는 연산은 실제로는 이미 존재하는 객체 중에서 같은 형과 값을 갖는 것을 돌려줄 수 있음. 반면에 가변 객체에서는 이런 것이 허용되지 않음.

➤ `a = 1; b = 1` 후에, `a` 와 `b` 는 값 1을 갖는 같은 객체일 수도 있고, 아닐 수도 있음.

➤ `c = []; d = []` 후에, `c` 와 `d` 는 두 개의 서로 다르고, 독립적이고, 새로 만들어진 빈 리스트임이 보장

➤ `c = d = []` 는 같은 객체를 `c` 와 `d` 에 대입

Built-in Types

immutable

1. Numeric

1.1. int

1.2. float

1.3. complex

1.4. bool

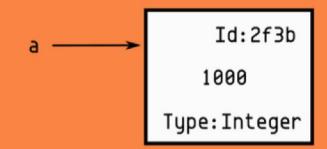
Immutable Integers

Code

```
a = 1000
```

What Computer Does

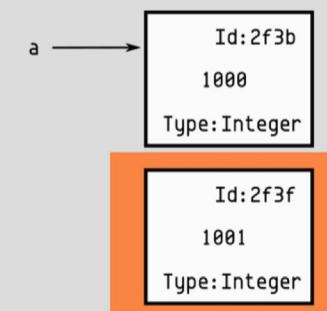
Variables Objects



Step 1: Python creates an integer

```
a = a + 1
```

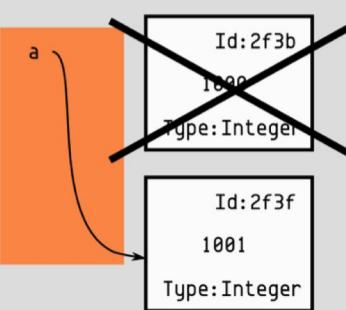
Variables Objects



Step 2: Python adds 1 to a and creates an integer

```
a = a + 1
```

Variables Objects



Step 3: Python rebinds a, garbage collects old object

Type		예시
int	int	14, -14
	int (Hex)	0xe
	int (Octal)	0o16
	int (Binary)	0b1110
float	float	14.0, 0.5, .3, 1.
	float	1.4e1, 1.79e+308 1.8e-308
	infinity	case insensible float("inf"), "Inf", "INFINITY" 및 "iNfINity"는 모두 가능
	Not a Number	case insensible float('nan')
complex	complex	14+0j
bool	bool	True, False
Underscore (readability)		1_000

■ 수를 표현하는 기본 리터럴

○ 정수

- 0, 100, 123 과 같은 표현

○ 실수

- 중간에 소수점 0.1, 4.2, 3.13123 와 같은 식
- 0. 으로 시작하는 실수값에서는 흔히 앞에 시작하는 0 제외 가능. .5 는 0.5를 줄여쓴 표현

○ 부호를 나타내는 - , +를 앞에 붙일 수 있다. (-1, +2.3 등)

■ 기본적으로 그냥 숫자만 사용하는 경우, 이는 10진법 값으로 해석.

○ 10진법외에도 2진법, 8진법, 16진법이 존재.

- 2진법 숫자는 0b로 시작(대소문자를 구분하지 않음)
- 8진법 숫자는 0o로 시작(대소문자를 구분하지 않음)
- 16진법숫자는 0x로 시작(대소문자를 구분하지 않음)

■ 숫자 리터럴 중간에 _ 를 쓰는 것은 무시.

○ 원하는 아무자리에나 가능

■ 참/ 거짓을 의미하는 부울대수값. lssubclass(bool,int)

○ 자체가 키워드로 True/False를 사용하여 표현

■ int (Decimal library)

○ arbitrary precision

- Not limited to machine **word size** (overflow/underflow 없음)
- https://en.wikipedia.org/wiki/Arbitrary-precision_arithmetic

○ import sys; sys.maxsize

○ It supports the usual operators

- / results in a float, so use // if integer division is needed

■ float (<https://docs.python.org/3.6/tutorial/floatingpoint.html>) (Fraction library)

○ 64-bit double-precision (Almost all platforms) according to the IEEE 754 standard

- https://en.wikipedia.org/wiki/IEEE_754_revision
 - maximum value a floating-point number can have is approximately 1.8×10^{308} . (overflow 있음)
 - 이보다 더 큰수는 inf
 - The closest a nonzero number can be to zero is approximately 5.0×10^{-324} .
 - Anything closer to zero than that is effectively zero:
- >>> x = 1.1 + 2.2
 >>> x == 3.3
 False

○ import sys; sys.float_info

○ internally as binary (base-2) fractions. Most decimal fractions cannot be represented exactly as binary fractions, so in most cases the internal representation of a floating-point number is an approximation of the actual value.

- In practice, the difference between the actual value and the represented value is very small and should not usually cause significant problems.

■ Sequential Reduction : Left to Right

○ 1-2-3-4

→ -1-3-4

→ -4-4

→ -8

○ 1-(2-3)-4

→ 1--1-4

→ 0-4

→ -4

■ General Rules

- Left-to-right evaluation, but look-ahead first, checking if a higher priority operation comes next; parentheses force evaluation order; evaluation work can be "queued up" due to operator priority
- `9>1e-4 and {0:"Cavern", 5:'Tunnel'}[0].upper() in 'cave'`
 - True and `{0:"Cavern", 5:'Tunnel'}[0].upper()` in 'cave'
 - True and `"Cavern".upper()` in 'cave'
 - True and `"CAVERN"` in 'cave'
 - True and False
 - False

Operator	Example	Meaning	Result
+ (unary)	+a	Unary Positive	a In other words, it doesn't really do anything. It mostly exists for the sake of completeness, to complement Unary Negation.
+ (binary)	a + b	Addition	Sum of a and b
- (unary)	-a	Unary Negation	Value equal to a but opposite in sign
- (binary)	a - b	Subtraction	b subtracted from a
*	a * b	Multiplication	Product of a and b
/	a / b	Division	Quotient when a is divided by b. The result always has type float.
%	a % b	Modulus	Remainder when a is divided by b
//	a // b	Floor Division (Integer Division)	Quotient when a is divided by b, rounded to the next smallest whole number
**	a ** b	Exponentiation	a raised to the power of b

$10 // -4 == -3$ or $-10 // 4 == -3$

When the result is negative, the result is rounded down to the next smallest (greater negative) integer:

Operator	Example	Meaning	Result
&	a & b	bitwise AND	Each bit position in the result is the logical AND of the bits in the corresponding position of the operands. (1 if both are 1, otherwise 0.)
	a b	bitwise OR	Each bit position in the result is the logical OR of the bits in the corresponding position of the operands. (1 if either is 1, otherwise 0.)
~	~a	bitwise negation bitwise complement	Each bit position in the result is the logical negation of the bit in the corresponding position of the operand. (1 if 0, 0 if 1.)
^	a ^ b	bitwise XOR (exclusive OR)	Each bit position in the result is the logical XOR of the bits in the corresponding position of the operands. (1 if the bits in the operands are different, 0 if they are the same.)
>>	a >> n	Shift right n places	Each bit is shifted right n places.
<<	a << n	Shift left places	Each bit is shifted left n places.

Operation	Provided By	Result
abs(num)	<code>_abs_</code>	Absolute value of num
num + num2	<code>_add_</code>	Addition
bool(num)	<code>_bool_</code>	Boolean conversion
num == num2	<code>_eq_</code>	Equality
float(num)	<code>_float_</code>	Float conversion
num // num2	<code>_floordiv_</code>	Integer division
num >= num2	<code>_ge_</code>	Greater or equal
num > num2	<code>_gt_</code>	Greater than
int(num)	<code>_int_</code>	Integer conversion
num <= num2	<code>_le_</code>	Less or equal
num < num2	<code>_lt_</code>	Less than
num % num2	<code>_mod_</code>	Modulus
num * num2	<code>_mul_</code>	Multiplication
num != num2	<code>_ne_</code>	Not equal
-num	<code>_neg_</code>	Negative
+num	<code>_pos_</code>	Positive
num ** num2	<code>_pow_</code>	Power
round(num)	<code>_round_</code>	Round
num.__sizeof__()	<code>_sizeof_</code>	Bytes for internal representation
str(num)	<code>_str_</code>	String conversion
num - num2	<code>_sub_</code>	Subtraction
num / num2	<code>_truediv_</code>	Float division
math.trunc(num)	<code>_trunc_</code>	Truncation

Operation	Provided By	Result
<code>num & num2</code>	<code>_and_</code>	Bitwise and
<code>math.ceil(num)</code>	<code>_ceil_</code>	Ceiling
<code>math.floor(num)</code>	<code>_floor_</code>	Floor
<code>~num</code>	<code>_invert_</code>	Bitwise inverse
<code>num << num2</code>	<code>_lshift_</code>	Left shift
<code>num num2</code>	<code>_or_</code>	Bitwise or
<code>num >> num2</code>	<code>_rshift_</code>	Right shift
<code>num ^ num2</code>	<code>_xor_</code>	Bitwise xor
<code>num.bit_length()</code>	<code>bit_length</code>	Number of bits necessary

Operation	Result
<code>f.as_integer_ratio()</code>	Returns num, denom tuple
<code>f.is_integer()</code>	Boolean if whole number

[math](#) 모듈의 복소수 버전이 필요하면, [cmath](#)를 사용

abs(value)	Absolute value
bin(integer)	String of number in binary
divmod(dividend, divisor)	Integer division and remainder
float(string)	Floating-point number from string
hex(integer)	String of number in hexadecimal
int(string)	Integer from decimal number
int(string, base)	Integer from number in base
oct(integer)	String of number in octal
pow(value, exponent)	value $\star\star$ exponent
round(value)	Round to integer
round(value, places)	Round to places decimal places
str(value)	String form of number (decimal)
sum(values)	Sum sequence (e.g., list) of values
sum(values, initial)	Sum sequence from initial start

Sequence

hold values in an indexable and sliceable order

■ by holding items

○ Container sequences

- list, tuple, and collections.deque can **hold items of different types**. (Heterogeneous)
- hold references to the objects they contain, which may be of any type

○ Flat sequences

- str, bytes, bytearray, memoryview, and array.array **hold items of one type**. (Homogeneous)
- physically store the value of each item within its own memory space, and not as distinct objects.
- more compact, but they are limited to holding primitive values like characters, bytes, and numbers.

■ by mutability:

○ Mutable sequences

- list, bytearray, array.array, collections.deque, and memoryview

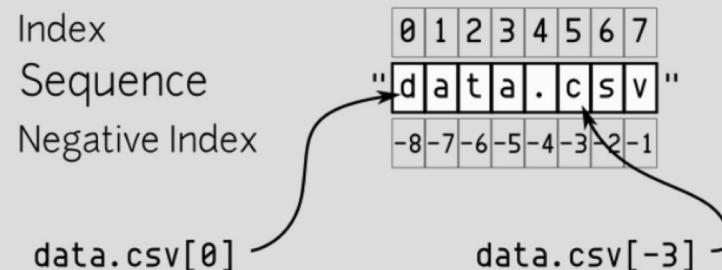
○ Immutable sequences

- tuple, str, and bytes

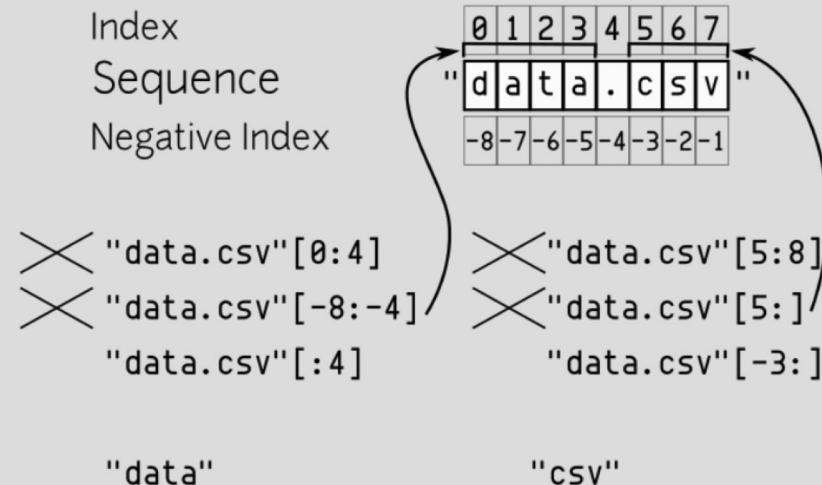
Index Examples

index
= subscript

모든 슬라이스 연산은
요청한 항목들을 포함하
는 새 리스트를 반환. 슬
라이스가 리스트의 새로
운 (얕은) 복사본을 돌려
준다는 뜻



Slicing Examples



■ 너무 큰 값을 인덱스로 사용하는 것은 예러

```
word[42] # the word only has 6 characters
```

```
Traceback (most recent call last): File "<stdin>", line 1, in <module> IndexError: string index out of range
```

■ 범위를 벗어나는 슬라이스 인덱스는 슬라이싱할 때 부드럽게 처리

```
word[4:42]
```

```
word[42:]
```

immutable, flat sequence

2. String

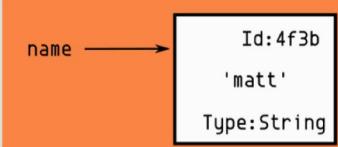
Immutable Strings

Code

```
name = 'matt'
```

What Computer Does

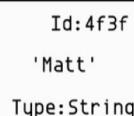
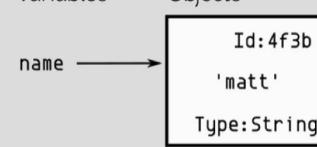
Variables Objects



Step 1: Python creates a string

```
correct = name.capitalize()
```

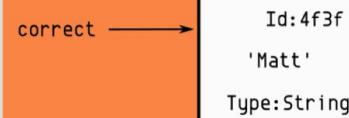
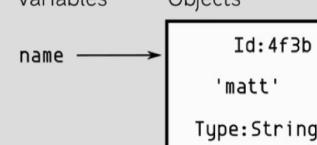
Variables Objects



Step 2: Python creates a new capitalized string

```
correct = name.capitalize()
```

Variables Objects



Step 3: Python creates a new variable

Type	Example
String	"hello\tthere"
String	'hello'
String	""He said, "hello"""
Raw string	r'hello\tthere'
Byte string	b'hello'

Escape Sequence	Output
\newline	Ignore trailing newline in triple quoted string
\\	Backslash
\'	Single quote
\"	Double quote
\a	ASCII Bell
\b	ASCII Backspace
\n	Newline
\r	ASCII carriage return
\t	Tab
\u12af	Unicode 16 bit
\U12af89bc	Unicode 32 bit
\N{BLACK STAR}	Unicode name
\o84	Octal character
\xFF	Hex character

```
print(...)
```

'Single-quoted string'
"Double-quoted string"
'String with\nnewline'

'Unbroken\
 string'
r'\n is an escape code'
'Pasted' ''' 'string'
('Pasted'
 ''
 'string')
"""\String with
newline""""

gives...

Single-quoted string
Double-quoted string
String with
newline
Unbroken string

\n is an escape code
Pasted string
Pasted string

String with
newline

■ str

- 글자, 글자가 모여서 만드는 단어, 문장, 여러 줄의 단락이나 글 전체

■ literal

- 큰 따옴표와 작은 따옴표를 구분하지 않지만 양쪽 따옴표가 맞아야 함. (문자열을 둘러싸는 따옴표와 다른 따옴표는 문자열 내의 일반 글자로 해석)
 - "apple" , 'apple' 은 모두 문자열 리터럴로 apple이라는 단어를 표현
- 두 개의 문자열 리터럴이 공백이나 줄바꿈으로 분리되어 있는 경우에 이것은 하나의 문자열 리터럴로 해석
 - "apple," "banana"는 "apple,banana"라고 쓴 표현과 동일
- 따옴표 세 개를 연이어서 쓰는 경우에는 문자열 내에서 줄바꿈이 허용.
 - 흔히 함수나 모듈의 간단한 문서화 텍스트를 표현할 때 쓰임.
 - """He said "I didn't go to 'SCHOOL' yesterday".""" > He said "I didn't go to 'SCHOOL' yesterday".
 - 여러 줄에 대한 내용을 쓸 때. """HOMEWORK: 1. print "hello, world" 2. print even number between 2 and 12 3. calculate sum of prime numbers up to 100,000 """
- escape를 허용하지 않는 raw string 리터럴 : r"
- 다른 값을 삽입하는 format string 리터럴 f::
- byte를 정의하는 byte string 리터럴 b"

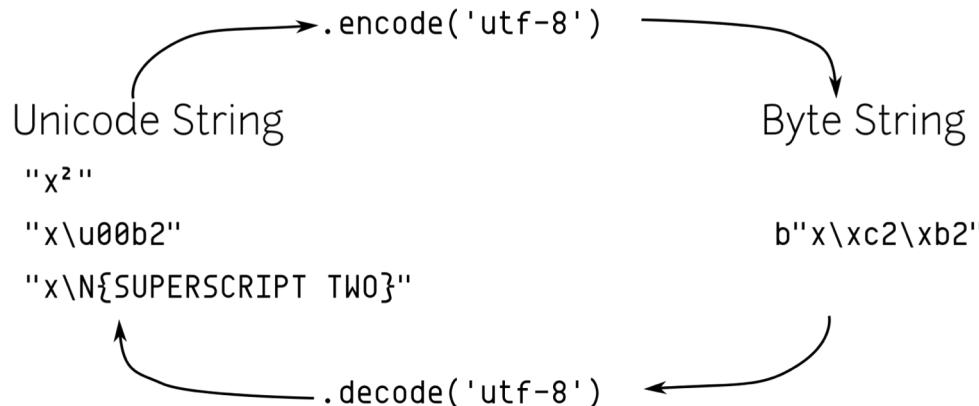
■ str type holds **Unicode characters**

- There is no type for single characters

■ bytes type is used for **strings of bytes, i.e., ASCII or Latin-1**

- The bytearray type is used for modifiable byte sequences

Unicode Encoding & Decoding



- bytes는 (HTTP 응답과 같은 파일)과 (네트워크 리소스)는 바이트 스트림으로 전송되기 때문에 이해하는 것이 중요
- 반면에 사람은 유니 코드 문자열의 편의성을 선호. 그렇기에 상호변환을 하는 경우가 많음

<https://feifeiyum.github.io/2017/07/15/python-fp-textbytes/>

Built-in query functions

chr(codepoint)
len(string)
ord(character)
str(value)

String concatenation

first + second
string * repeat
repeat * string

Substring containment

substring **in** string
substring **not in** string

String comparison (lexicographical ordering)

lhs == rhs
lhs != rhs
lhs < rhs
lhs <= rhs
lhs > rhs
lhs >= rhs

Operation	Provided By	Result
<code>s + s2</code>	<code>_add_</code>	String concatenation
<code>"foo" in s</code>	<code>_contains_</code>	Membership
<code>s == s2</code>	<code>_eq_</code>	Equality
<code>s >= s2</code>	<code>_ge_</code>	Greater or equal
<code>s[0]</code>	<code>_getitem_</code>	Index operation
<code>s > s2</code>	<code>_gt_</code>	Greater
<code>s <= s2</code>	<code>_le_</code>	Less than or equal
<code>len(s)</code>	<code>_len_</code>	Length
<code>s < s2</code>	<code>_lt_</code>	Less than
<code>s % (1, 'foo')</code>	<code>_mod_</code>	Formatting
<code>s * 3, 3* s</code>	<code>_mul_</code>	Repetition
<code>s != s2</code>	<code>_ne_</code>	Not equal
<code>repr(s)</code>	<code>_repr_</code>	Programmer friendly string
<code>s.__sizeof__()</code>	<code>_sizeof_</code>	Bytes for internal representation
<code>str(s)</code>	<code>_str_</code>	User friendly string

문자열에 실수를 곱하거나 문자열에 정수를 더하는 연산 ?
 ValueError 예러

■ Strings can be indexed, which results in a string of length 1

- As strings are immutable, a character can be subscripted but not assigned to
- Index 0 is the initial character
- Indexing past the end raises an exception
- Negative indexing goes through the string in reverse
- I.e., -1 indexes the last character

■ It is possible to take a slice of a string by specifying one or more of...

- A start position, which defaults to 0
- A non-inclusive end position (which may be past the end, in which case the slice is up to the end), which defaults to len
- A step, which defaults to 1
- With a step of 1, a slice is equivalent to a simple substring

string[index]	General form of subscript operation
string[0]	First character
string[len(string) – 1]	Last character
string[-1]	Last character
string[-len(string)]	First character
string[first:last]	Substring from first up to last
string[index:]	Substring from index to end
string[:index]	Substring from start up to index
string[:]	Whole string
string[first:last:step]	Slice from first to last in steps of step
string[::-step]	Slice of whole string in steps of step

■ interpolation (내삽)

○ 문자열에 대한 특별한 연산

- "Tom has 3 bananas and 4 apples." 라는 문자열이 있다고 할 때, 이것을 리터럴로 정의하는 것은 그 내용이 소스코드에 고정되는, 하드 코딩(hard coding)
- 문자열은 한 번 생성된 이후로 변경되지 않는 immutable이므로 Tom이 가지고 있는 사과나 바나나의 개수가 바뀌었을 때, 그 내용을 적절히 변경해 줄 수가 없음

○ 문자열내에 동적으로 변할 수 있는 값을 삽입하여 상황에 따라 다른 문자열을 만드는 방법

- mini-language
- 문자열 내삽의 기본원리는 문자열 내에 다른 값으로 바뀔 치환자를 준비해두고, 필요한 시점에 치환자를 실제 값의 내용으로 바꿔 문자열을 생성
 - 전통적인 포맷 치환자(%)를 사용하는 방법 (Old Style)
 - » 문자열 % (값, ...)의 형식을 이용해서 문자열 내로 변수값을 밀어넣는 방법
 - 문자열의 .format() 메소드를 사용하는 방법 (New Style)
 - » 치환자의 구분없이 사용할 수 있으며, 각 값을 포맷팅할 수 있는 장점
 - Literal String (python 3.6+)
 - » 포맷 메소드를 사용하지 않고 리터럴만으로 2.의 방법을 사용
 - Template String

○ **format string**을 사용자가 입력한 값을 이용하려면, security 이슈를 피하기 위하여 **Template String** 사용. 그렇지 않을 경우, **python3.6+** 사용시 **Literal String** 방식 사용. **python3.6+** 사용하지 않을 경우 **New Style** 사용

■ 포맷 치환자 (printf-like)

- The **mini-language** is based on C's printf, plus some additional features
- Its use is often **discouraged** as it is error prone for large formatting tasks
- 문자열 치환자는 퍼센트 문자 뒤에 포맷형식을 붙여서 치환자를 정의. 치환자를 포함하는 문자열과 각 치환자에 해당하는 값의 tuple을 % 기호로 연결하여 표현
 - %d
 - 정수값
 - d 앞에는 자리수와 채움문자를 넣을 수 있음
 - %04d : 앞의 0은 채움문자이고 뒤는 포맷의 폭 (%04d 는 0으로 시작하는 네자리 정수를 의미)
 - 13이라는 값을 포맷팅할 때, %d 에 치환하면 "13"이 되지만, %04d 에 치환되는 경우에는 "0013"으로 치환
 - %f
 - 실수값
 - f 앞에는 .3 과 같이 소수점 몇 째자리까지 표시할 것인지를 결정하는 확장정보
 - %.3f : 1.5를 "1.500"으로 표시
 - 정수값은 숫자 리터럴과 같이 %b, %o, %x를 이용해서 각각 이진법, 8진법, 16진법으로 표시
 - %s
 - 문자열
 - %r
 - representation으로 타입을 구분하지 않는 값의 표현형
 - 표시되고자 하는 값의 타입이 분명하지 않을 때 사용
 - %d, %f, %s 에 대해서 올바르지 않은 타입의 값을 치환하려 하면 TypeError가 발생할때 사용

```
>>> first_name = "Eric"
>>> last_name = "Idle"
>>> age = 74
>>> profession = "comedian"
>>> affiliation = "Monty Python"
>>> "Hello, %s %. You are %s. You are a %s. You were a me
mber of %s." % (first_name, last_name, age, profession, af
filiation)
'Hello, Eric Idle. You are 74. You are a comedian. You wer
e a member of Monty Python.'
```

readable enough. However, once you start using several parameters and longer strings, your code will quickly become much less easily readable. Things are starting to look a little messy already: Unfortunately, this kind of formatting isn't great because it is verbose and leads to errors, like not displaying tuples or dictionaries correctly.

- 두 개 이상의 문자열 리터럴(즉, 따옴표로 둘러싸인 것들) 가 연속해서 나타나면 자동으로 이어 붙여짐.

```
>>> 'Py' 'thon' 'Python'
```

- 긴 문자열을 쪼개고자 할 때 사용

```
>>> text = ('Put several strings within parentheses ' ... 'to have them joined together.')
>>> text
'Put several strings within parentheses to have them joined together.'
```

- 오직 두 개의 리터럴에만 적용될 뿐 변수나 표현식에는 해당하지 않음

```
>>> prefix = 'Py'
>>> prefix 'thon' # can't concatenate a variable and a string literal ...
SyntaxError: invalid syntax
```

```
>>> ('un' * 3) 'ium' ...
SyntaxError: invalid syntax
```

■ format() : python 2.7+ (PEP 3101)

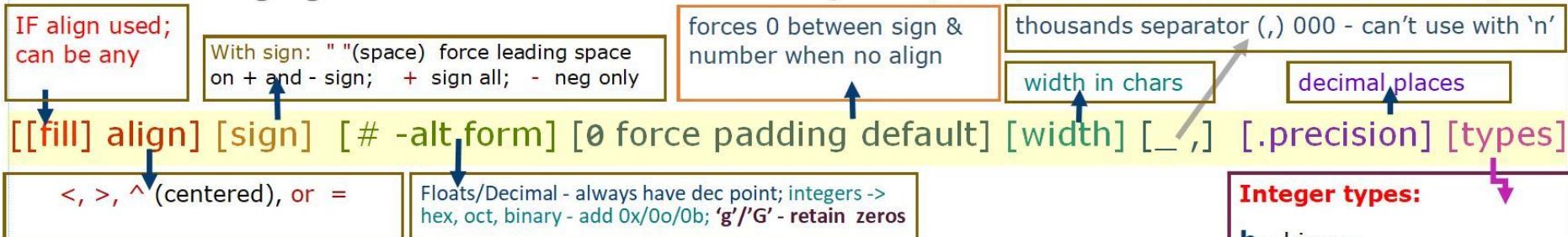
- 전통적인 포맷 치환자가 타입을 가린다는 제약이 있고, 포맷팅의 방법이 제한
 - format()은 Default-ordered, positional and named parameter substitution are supported
 - format()은 숫자값인 경우에는 특별히 d, f, x, b, o 등의 표현타입도 정의 가능
 - format()은 왼쪽 정렬이나 중앙정렬도 설정
- python의 built-in format() 함수와 str의 format 메소드와 동일한 동작
 - format(문자열, 치환값)
 - str.format(치환값)
- {} 를 사용
 - {} 자체가 하나의 값을 의미하며, 번호를 부여해서 한 번 받은 값을 여러번 사용할 수 있음.
 - <https://docs.python.org/3/library/string.html#format-specification-mini-language>

```
'First {}, now {}'.format('that', 'this')  
'First {0}, now {1}'.format('that', 'this')  
'First {1}, now {0}'.format('that', 'this')  
'First {0}, now {0}'.format('that', 'this')  
'First {x}, now {y}'.format(x='that', y='this')
```

Use {{ and }} to embed { and } in a string without format substitution

`format(object, "mini-language-string")`
`"{:mini-language-string}").format(object)`

How the **mini-language** statements are ordered and structured in general: (*Note: symbols must be in the order as shown below!*)



"Hello {name}, your balance is {blc:9.3f}".format(name="Adam", blc=230.2346)

Hello Adam, your balance is ███230.█35

Integer types:	
b	- binary
c	- Unicode char
d	- base 10 integer
o	- Octal
x	- Hex - lower cs
X	- Hex - upper cs
n	- like d but uses local separator definitions
Float/decimal types:	
e	- scientific, e - exponent
E	- E for exponent
f	- fixed point (default 6)

{**0**} – reference first positional argument
{ } – reference implicit positional argument
{result} – reference keyword argument
{bike.tire} – reference attribute of argument
{names[**0**]}) – reference first element of argument

!s – Call `str()` on argument
!r – Call `repr()` on argument
!a – Call `ascii()` on argument

```
class Cat:  
... def __init__(self, name):  
...     self.name = name  
... def __format__(self, data):  
...     return "Format"  
... def __str__(self):  
...     return "Str"  
... def __repr__(self):  
...     return "Repr"  
  
cat = Cat("Fred")  
print("{} {!s} {!a} {!r}".format(cat, cat, cat, ... cat))
```

Format Str Repr Repr

```
>>> first_name = "Eric"
>>> last_name = "Idle"
>>> age = 74
>>> profession = "comedian"
>>> affiliation = "Monty Python"
>>> print(("Hello, {first_name} {last_name}. You are {age}. " +
>>> "You are a {profession}. You were a member of {affiliation}.") \
>>> .format(first_name=first_name, last_name=last_name, age=age, \
>>> profession=profession, affiliation=affiliation)) 'Hello, Eric Idle.
You are 74. You are a comedian. You were a member of Monty Python.'
```

If you had the variables you wanted to pass to **.format()** in a dictionary, then you could just unpack it with **.format(**some_dict)** and reference the values by key in the string, but there has got to be a better way to do this.

■ 복잡하지 않음 (simpler and less powerful mechanism)

- 임의의 변수에 접근할 수 없음
- 복잡한 경우, 다른 mini-language를 사용하는 format 방식 사용

```
from string import Template
t = Template('Hey, $name!')
t.substitute(name=name)

template_string = 'Hey $name, there is a $error error!'
Template(template_string).substitute(name=name, error=hex(errno))

SECRET = 'this-is-a-secret'

class Error:
    def __init__(self):
        pass

err = Error()
user_input = '{error.__init__.globals().__[SECRET]}'
user_input.format(error=err)

user_input = '${error.__init__.globals().__[SECRET]}'
Template(user_input).substitute(error=err)
```

Operation	Result
s.capitalize()	Capitalizes a string
s.casefold()	Lowercase in a unicode compliant manner
s.center(w, [char])	Center a string in w spaces with char (default " ")
s.count(sub, [start, [end]])	Count sub in s between start and end
s.encode(encoding, errors= 'strict')	Encode a string into bytes
s.endswith(sub)	Check for a suffix
s.expandtabs(tabszie=8)	Replaces tabs with spaces
s.find(sub, [start, [end]])	Find substring or return -1
s.format(*args, **kw)	Format string
s.format_map(mapping)	Format strings with a mapping
s.index(sub, [start, [end]])	Find substring or raise ValueError
s.isalnum()	Boolean if alphanumeric
s.isalpha()	Boolean if alphabetic
s.isdecimal()	Boolean if decimal
s.isdigit()	Boolean if digit
s.isidentifier()	Boolean if valid identifier
s.islower()	Boolean if lowercase
s.isnumeric()	Boolean if numeric
s.isprintable()	Boolean if printable
s.isspace()	Boolean if whitespace
s.istitle()	Boolean if titlecased
s.isupper()	Boolean if uppercased

Operation	Result
s.join(iterable)	Return a string inserted between sequence
s.ljust(w, [char])	Left justify in w spaces with char (default ' ')
s.lower()	Lowercase
s.lstrip([chars])	Left strip chars (default spacing).
s.partition(sub)	Split string at first occurrence of substring, return (before, sub, after)
s.replace(old, new, [count])	Replace substring with new string
s.rfind(sub, [start, [end]])	Find rightmost substring or return -1
s.rindex(sub, [start, [end]])	Find rightmost substring or raise ValueError
s.rjust(w, [char])	Right justify in w spaces with char (default " ")
s.rpartition(sub)	Rightmost partition
s.rsplit([sep, [maxsplit=-1]])	Rightmost split by sep (defaults to whitespace)
s.rstrip([chars])	Right strip
s.split([sep, [maxsplit=-1]])	Split a string into sequence around substring
s.splitlines(keepends=False)	Break string at line boundaries
s.startswith(prefix, [start, [end]])	Check for prefix
s.strip([chars])	Remove leading and trailing whitespace (default) or chars
s.swapcase()	Swap casing of string
s.title()	Titlecase string
s.translate(table)	Use a translation table to replace strings
s.upper()	Uppercase
s.zfill(width)	Left fill with 0 so string fills width (no truncation)

Container

3. Another Sequence

- 3.1. list (mutable, container)
- 3.2. tuple (immutable, container)
- 3.3. range (immutable, container)

■ Although everything is, at one level, a dict, dict is not always the best choice

- Nor is list

■ Choose containers based on usage patterns and mutability

- E.g., tuple is immutable whereas list is not

Equality comparison

`lhs == rhs`
`lhs != rhs`

Returns sorted list of container's values
(also takes keyword arguments
for key comparison — `key` — and
reverse sorting — `reverse`)

Built-in queries and predicates

`len(container)`
`min(container)`
`max(container)`
`any(container)`
`all(container)`
`sorted(container)`

Membership (key membership for mapping types)

`value in container`
`value not in container`

■ A tuple is an immutable sequence

- Supports (negative) indexing, slicing, concatenation and other operations

■ A list is a mutable sequence

- It supports similar operations to a tuple, but in addition it can be modified

■ A range is an immutable sequence

- It supports indexing, slicing and other operations

list	list() [] [0, 0x33, 0xCC] ['Albert', 'Einstein', [1879, 3, 14]] [random() for _ in range(42)]
tuple	tuple() () (42,) ('red', 'green', 'blue') ((0, 0), (3, 4))
range	range(42) range(1, 100) range(100, 0, -1)

Search methods

Raises exception if value not found



`sequence.index(value)`
`sequence.count(value)`

Lexicographical ordering
(not for range)

Indexing and slicing

`lhs < rhs`
`lhs <= rhs`
`lhs > rhs`
`lhs >= rhs`

`sequence[index]`
`sequence[first:last]`
`sequence[index:]`
`sequence[:index]`
`sequence[:]`
`sequence[first:last:step]`
`sequence[::-step]`

Concatenation (not for range)

`first + second`
`sequence * repeat`
`repeat * sequence`

■ As lists are mutable, assignment is supported for their elements

- Augmented assignment on subscripted elements
- Assignment to subscripted elements and assignment through slices

■ List slices and elements support del

- Removes them from the list

Index- and slice-based operations

`list[index]`

`list[index] = value`

`del list[index]`

`list[first:last:step]`

`list[first:last:step] = other`

`del list[first:last:step]`

Whole-list operations

`list.clear()`

`list.reverse()`

`list.sort()`

Element modification methods

`list.append(value)`

`list.insert(index, value)`

`list.pop()`

`list.pop(index)`

`list.remove(value)`

Operation	Provided By	Result
<code>I + I2</code>	<code>__add__</code>	List concatenation (see <code>.extend</code>)
<code>"name" in I</code>	<code>__contains__</code>	Membership
<code>del I[idx]</code>	<code>__del__</code>	Remove item at index idx (see <code>.pop</code>)
<code>I == I2</code>	<code>__eq__</code>	Equality
<code>"{}".format(I)</code>	<code>__format__</code>	String format of list
<code>I >= I2</code>	<code>__ge__</code>	Greater or equal. Compares items in lists from left
<code>I[idx]</code>	<code>__getitem__</code>	Index operation
<code>I > I2</code>	<code>__gt__</code>	Greater. Compares items in lists from left
No hash	<code>__hash__</code>	Set to None to ensure you can't insert in dictionary
<code>I += I2</code>	<code>__iadd__</code>	Augmented (mutates I) concatenation
<code>I *= 3</code>	<code>__imul__</code>	Augmented (mutates I) repetition
<code>for thing in I:</code>	<code>__iter__</code>	Iteration
<code>I <= I2</code>	<code>__le__</code>	Less than or equal. Compares items in lists from left
<code>len(I)</code>	<code>__len__</code>	Length
<code>I < I2</code>	<code>__lt__</code>	Less than. Compares items in lists from left
<code>I * 2</code>	<code>__mul__</code>	Repetition
<code>I != I2</code>	<code>__ne__</code>	Not equal
<code>repr(I)</code>	<code>__repr__</code>	Programmer friendly string
<code>reversed(I)</code>	<code>__reversed__</code>	Reverse
<code>foo * I</code>	<code>__rmul__</code>	Called if foo doesn't implement <code>__mul__</code>
<code>I[idx] = 'bar'</code>	<code>__setitem__</code>	Index operation to set value
<code>I.__sizeof__()</code>	<code>__sizeof__</code>	Bytes for internal representation
<code>str(I)</code>	<code>__str__</code>	User friendly string

Operation	Result
<code>l.append(item)</code>	Append item to end
<code>l.clear()</code>	Empty list (mutates l)
<code>l.copy()</code>	Shallow copy
<code>l.count(thing)</code>	Number of occurrences of thing
<code>l.extend(l2)</code>	List concatenation (mutates l)
<code>l.index(thing)</code>	Index of thing else ValueError
<code>l.insert(idx, bar)</code>	Insert bar at index idx
<code>l.pop([idx])</code>	Remove last item or item at idx
<code>l.remove(bar)</code>	Remove first instance of bar else ValueError
<code>l.reverse()</code>	Reverse (mutates l)
<code>l.sort([key=], reverse=False)</code>	In-place sort, by optional key function (mutates l)

■ Tuples are the default structure for unpacking in multiple assignment**■ The display form of tuple relies on parentheses**

- Thus it requires a special syntax case to express a tuple of one item

x = 1,
x = 1, 2
x, y = 1, 2
x = 1, (2, 3)

x = (1,)
x = (1, 2)
(x, y) = (1, 2)
x = (1, (2, 3))

Operation	Provided	Result
t + t2	<code>_add_</code>	Tuple concatenation
"name" in t	<code>_contains_</code>	Membership
t == t2	<code>_eq_</code>	Equality
"{}".format(t)	<code>_format_</code>	String format of tuple
t >= t2	<code>_ge_</code>	Greater or equal. Compares items in tuple from left
t[idx]	<code>_getitem_</code>	Index operation
t > t2	<code>_gt_</code>	Greater. Compares items in tuple from left
hash(t)	<code>_hash_</code>	For set/dict insertion
for thing in t:	<code>_iter_</code>	Iteration
t <= t2	<code>_le_</code>	Less than or equal. Compares items in tuple from left
len(t)	<code>_len_</code>	Length
t < t2	<code>_lt_</code>	Less than. Compares items in tuple from left
t * 2	<code>_mul_</code>	Repetition
t != t2	<code>_ne_</code>	Not equal
repr(t)	<code>_repr_</code>	Programmer friendly string
foo * t	<code>_rmul_</code>	Called if foo doesn't implement <code>_mul_</code>
t. <code>_sizeof_()</code>	<code>_sizeof_</code>	Bytes for internal representation
str()	<code>_str_</code>	User friendly string

Operation	Result
<code>t.count(item)</code>	Count of item
<code>t.index(thing)</code>	Index of thing else ValueError

- 많은 경우에 `range()` 가 돌려준 객체는 리스트인 것처럼 동작하지만, 사실 리스트가 아님
- iterate할 때 원하는 시퀀스 항목들을 순서대로 돌려주는 객체이지만, 실제로 리스트를 만들지 않아서 공간을 절약.
- 공급이 소진될 때까지 일련의 항목들을 얻을 수 있는 무엇인가를 기대하는 함수와 구조물들의 타깃으로 적합

```
range(5, 10)
      5, 6, 7, 8, 9
```

```
range(0, 10, 3)
      0, 3, 6, 9
```

```
range(-10, -100, -30)
      -10, -40, -70
```

4. dict(ionary)

1. mapping
2. mutable

■ dict is a mapping type

- I.e., it maps a key to a correspond value

■ Keys, values and mappings are viewable via keys, values and items methods**■ Keys must be of immutable types****■ This includes int, float, str, tuple, range and frozenset, but excludes list, set and dict**

- Immutable types are hashable (hash can be called on them)

Key-based operations

`key in dict``key not in dict``dict.get(key, default)``dict.get(key)` Equivalent to calling get with a default of None`dict[key]``dict[key] = value` Automagically creates entry if key not already in dict`del dict[key]`

Manipulation methods

Views

`dict.keys()``dict.values()``dict.items()``dict.clear()``dict.pop(key)``dict.pop(key, default)``dict.get(key)``dict.get(key, default)``dict.update(other)`

Operation	Provided By	Result
key in d	<code>__contains__</code>	Membership
<code>del d[key]</code>	<code>__delitem__</code>	Delete key
<code>d == d2</code>	<code>__eq__</code>	Equality. Dicts are equal or not equal
<code>"{}".format(d)</code>	<code>__format__</code>	String format of dict
<code>d[key]</code>	<code>__getitem__</code>	Get value for key (see <code>.get</code>)
<code>for key in d:</code>	<code>__iter__</code>	Iteration over keys
<code>len(d)</code>	<code>__len__</code>	Length
<code>d != d2</code>	<code>__ne__</code>	Not equal
<code>repr(d)</code>	<code>__repr__</code>	Programmer friendly string
<code>d[key] = value</code>	<code>__setitem__</code>	Set value for key
<code>d.__sizeof__()</code>	<code>__sizeof__</code>	Bytes for internal representation

Operation	Result
d.clear()	Remove all items (mutates d)
d.copy()	Shallow copy
d.fromkeys(iter, value=None)	Create dict from iterable with values set to value
d.get(key, [default])	Get value for key or return default (None)
d.items()	View of (key, value) pairs
d.keys()	View of keys
d.pop(key, [default])	Return value for key or default (KeyError if not set)
d.popitem()	Return arbitrary (key, value) tuple. KeyError if empty
d.setdefault(k, [default])	Does d.get(k, default). If k missing, sets to default
d.update(d2)	Mutate d with values of d2 (dictionary or iterable of (key, value) pairs)
d.values()	View of values

5. set & frozenset

5.1. set (mutable)

5.2. frozenset (immutable)

■ **set and frozenset both define containers of unique hashable values**

■ **set is mutable and has a display form**

- set() is the empty set, not {}

■ **frozenset is immutable and can be constructed from a set or other iterable**

■ **set classes are implemented using dictionaries.**

- Accordingly, the requirements for set elements are the same as those for dictionary keys; namely, that the element defines both eq() and hash().
- **sets cannot contain mutable elements** such as lists or dictionaries

```
text = 'the cat sat on the mat'
```

```
words = frozenset(text.split())
```

```
print('different words used:', len(words))
```

Set relations (in addition to equality and set membership)

`lhs < rhs`
`lhs <= rhs lhs.issubset(rhs)`
`lhs > rhs`
`lhs >= rhs lhs.issuperset(rhs)`
`lhs.isdisjoint(rhs)`

Set combinations

`lhs | rhs lhs.union(rhs)`
`lhs & rhs lhs.intersection(rhs)`
`lhs - rhs lhs.difference(rhs)`
`lhs ^ rhs lhs.symmetric_difference(rhs)`

Manipulation methods

set.add(element)
set.remove(element)
set.discard(element)
set.clear()
set.pop()

Removes and returns
arbitrary element



Augmented assignment and updates

lhs |= rhs lhs.update(rhs)
lhs &= rhs lhs.intersection_update(rhs)
lhs -= rhs lhs.difference_update(rhs)
lhs ^= rhs lhs.symmetric_difference_update(rhs)

Operation	Provided By	Result
<code>s & s2</code>	<code>__and__</code>	Set intersection (see <code>.intersection</code>)
<code>"name" in s</code>	<code>__contains__</code>	Membership
<code>s == s2</code>	<code>__eq__</code>	Equality. Sets are equal or not equal
<code>"{}".format(s)</code>	<code>__format__</code>	String format of set
<code>s >= s2</code>	<code>__ge__</code>	<code>s</code> in <code>s2</code> (see <code>.issuperset</code>)
<code>s > s2</code>	<code>__gt__</code>	Strict superset (<code>s >= s2</code> but <code>s != s2</code>).
No hash	<code>__hash__</code>	Set to None to ensure you can't insert in dictionary
<code>s &= s2</code>	<code>__iand__</code>	Augmented (mutates <code>s</code>) intersection (see <code>.intersection_update</code>)
<code>s = s2</code>	<code>__ior__</code>	Augmented (mutates <code>s</code>) union (see <code>.update</code>)
<code>s -= s2</code>	<code>__isub__</code>	Augmented (mutates <code>s</code>) difference (see <code>.difference_update</code>)
<code>for thing in s:</code>	<code>__iter__</code>	Iteration
<code>s ^= s2</code>	<code>__ixor__</code>	Augmented (mutates <code>s</code>) xor (see <code>.symmetric_difference_update</code>)
<code>s <= s2</code>	<code>__le__</code>	<code>s2</code> in <code>s</code> (see <code>.issubset</code>)
<code>len(s)</code>	<code>__len__</code>	Length
<code>s < s2</code>	<code>__lt__</code>	Strict subset (<code>s <= s2</code> but <code>s != s2</code>).
<code>s != s2</code>	<code>__ne__</code>	Not equal
<code>s s2</code>	<code>__or__</code>	Set union (see <code>.union</code>)
<code>foo & s</code>	<code>__rand__</code>	Called if <code>foo</code> doesn't implement <code>__and__</code>
<code>repr(s)</code>	<code>__repr__</code>	Programmer friendly string
<code>foo s</code>	<code>__ror__</code>	Called if <code>foo</code> doesn't implement <code>__or__</code>
<code>foo - s</code>	<code>__rsub__</code>	Called if <code>foo</code> doesn't implement <code>__sub__</code>
<code>foo ^ s</code>	<code>__rxor__</code>	Called if <code>foo</code> doesn't implement <code>__xor__</code>
<code>s.__sizeof__()</code>	<code>__sizeof__</code>	Bytes for internal representation
<code>str(s)</code>	<code>__str__</code>	User friendly string
<code>s - s2</code>	<code>__sub__</code>	Set difference (see <code>.difference</code>)
<code>s ^ s2</code>	<code>__xor__</code>	Set xor (see <code>.symmetric_difference</code>)

Operation	Result
s.add(item)	Add item to s (mutates s)
s.clear()	Remove elements from s (mutates s)
s.copy()	Shallow copy
s.difference(s2)	Return set with elements from s and not s2
s.difference_update(s2)	Remove s2 items from s (mutates s)
s.discard(item)	Remove item from s (mutates s). No error on missing item
s.intersection(s2)	Return set with elements from both sets
s.intersection_update(s2)	Update s with members of s2 (mutates s)
s.isdisjoint(s2)	True if there is no intersection of these two sets
s.issubset(s2)	True if all elements of s are in s2
s.issuperset(s2)	True if all elements of s2 are in s
s.pop()	Remove arbitrary item from s (mutates s). KeyError on missing item
s.remove(item)	Remove item from s (mutates s). KeyError on missing item
s.symmetric_difference(s2)	Return set with elements only in one of the sets
s.symmetric_difference_update(s2)	Update s with elements only in one of the sets (mutates s)
s.union(s2)	Return all elements of both sets
s.update(s2)	Update s with all elements of both sets (mutates s)

형변환 (Type Conversion)

int('314')	314
str(314)	'314'
str([3, 1, 4])	'[3, 1, 4]'
list('314')	['3', '1', '4']
set([3, 1, 4, 1])	{1, 3, 4}

coercion (코어션) : 같은 형의 두 인자를 수반하는 연산이 일어나는 동안, 한 형의 인스턴스를 다른 형으로 묵시적으로 변환하는 것. 예를 들어, int(3.15) 는 실수를 정수 3 으로 변환. 하지만, 3+4.5 에서, 각 인자는 다른 형이고 (하나는 int, 다른 하나는 float), 둘을 더하기 전에 같은 형으로 변환해야 함.. 그렇지 않으면 TypeError 를 일으킴. 코어션 없이는, 호환되는 형들조차도 프로그래머가 같은 형으로 정규화해주어야 함. 예를 들어, 그냥 3+4.5 하는 대신 float(3)+4.5