[**1. 파이썬 기본 이해하기** 6](#_Toc480726977)

[1.1 기본 용어 알아보기 6](#_Toc480726978)

[1.1.1. 리터럴(literal) 과 표현식(expression ) 6](#_Toc480726979)

[1.1.2. 데이터타입, 클래스, 객체 8](#_Toc480726980)

[1.1.3. 문장(statements) 10](#_Toc480726981)

[1.1.4. 식별자(identifier) 12](#_Toc480726982)

[1.1.5. 네임스페이스(Namespace) 관리 규칙 17](#_Toc480726983)

[1.2 파이썬 기본 내장 함수 이해 18](#_Toc480726984)

[1.2.1. help 함수 18](#_Toc480726985)

[1.2.2. print 함수 20](#_Toc480726986)

[1.2.3. type class 21](#_Toc480726987)

[1.2.4. dir 함수 22](#_Toc480726988)

[1.2.5. globals 함수 23](#_Toc480726989)

[1.2.6. locals 함수 25](#_Toc480726990)

[1.2.7. 객체 비교 방식(is, ==) 26](#_Toc480726991)

[**2. 숫자 원자형 데이터 타입 이해하기** 28](#_Toc480726992)

[2.1 내장 숫자 데이터 타입 이해하기 28](#_Toc480726993)

[2.1.1. 숫자 데이터 타입의 특징 28](#_Toc480726994)

[2.1.2 정수 int class 30](#_Toc480726995)

[2.1.3. 불리언 타입 : bool 31](#_Toc480726996)

[2.1.4 실수 float 34](#_Toc480726997)

[2.1.5 복소수 complex class 37](#_Toc480726998)

[2.1.6. 유리수 Fraction class 39](#_Toc480726999)

[2.1.7. 큰 수 Decimal class 40](#_Toc480727000)

[2.1.8. 2, 8, 16진법 변환 하기 42](#_Toc480727001)

[2.1.9 숫자 타입의 형변환 43](#_Toc480727002)

[2.2. 연산자(Operator) 46](#_Toc480727003)

[2.2.1. 연산자 우선순위 46](#_Toc480727004)

[2.2.2. 산술연산자 47](#_Toc480727005)

[2.2.3. 비교 연산자 48](#_Toc480727006)

[2.2.4. 비트 연산자 49](#_Toc480727007)

[2.2.5. 논리 연산 및 단축연산 52](#_Toc480727008)

[**3. 파이썬 문장 이해하기** 55](#_Toc480727009)

[3.1. 추가 정보에 대한 문장 구성 55](#_Toc480727010)

[3.1.1. 줄 들여쓰기 (Lines and Indentation) 55](#_Toc480727011)

[3.1.2. 멀티 라인 문 (Multi-Line Statements) 56](#_Toc480727012)

[3.1.3. 인용 (Quotation): 57](#_Toc480727013)

[3.1.4. 주석 (Comments): 58](#_Toc480727014)

[3.2. 프로그램 문장 59](#_Toc480727015)

[3.2.1. 할당문장 이해하기 59](#_Toc480727016)

[3.2.2. Import 구문 60](#_Toc480727017)

[3.2.3. 단순 제어문 62](#_Toc480727018)

[3.2.4. 복합 제어문 63](#_Toc480727019)

[3.2.5 순환문 처리 : for 65](#_Toc480727020)

[3.2.6. 순환문 처리 : while 65](#_Toc480727021)

[3.2.7 continue/break 66](#_Toc480727022)

[3.2.8. 순환문에 else 문 추가하기 67](#_Toc480727023)

[3.2.9. context 제한문 : with 69](#_Toc480727024)

[3.2.10. global 문 과 nonlocal 문 처리 71](#_Toc480727025)

[3.2.11 try,except 문 처리 74](#_Toc480727026)

[3.3 모듈/패키지 처리 76](#_Toc480727027)

[3.3.1 모듈 생성 및 실행하기 76](#_Toc480727028)

[3.3.2. 모듈 네임스페이스 확인하기 78](#_Toc480727029)

[3.3.3 패키지 처리 79](#_Toc480727030)

[**4. 파이썬 Sequence데이터 타입** 84](#_Toc480727031)

[4.1. 파이썬 Sequence데이터 타입의 특징 84](#_Toc480727032)

[4.1.1. 런타임에 속성 추가 여부 84](#_Toc480727033)

[4.1.1. 변경가능 여부 : Mutable & Immutable 85](#_Toc480727034)

[4.1.3. 컬렉션 여부 86](#_Toc480727035)

[4.1.4. sequence 타입 내의 메소드 처리 기준 90](#_Toc480727036)

[4.2. 문자열 데이터 타입 91](#_Toc480727037)

[4.2.1. 문자열 생성 방법 91](#_Toc480727038)

[4.2.2. 문자열 주요 메소드 92](#_Toc480727039)

[4.3 바이트 데이터 타입 97](#_Toc480727040)

[4.3.1. 바이트 생성 97](#_Toc480727041)

[4.3.2 바이트 주요 메소드 98](#_Toc480727042)

[4.4. bytearray 데이터 타입 101](#_Toc480727043)

[4.4.1. bytearray 생성 101](#_Toc480727044)

[4.4..2. bytearray 내의 메소드 101](#_Toc480727045)

[4.5. memoryview 데이터 타입 103](#_Toc480727046)

[4.51. memoryview 생성 103](#_Toc480727047)

[4.5.2. memoryview 메소드 105](#_Toc480727048)

[4.6. 튜플 데이터 타입 107](#_Toc480727049)

[4.6.1. 튜플 생성 107](#_Toc480727050)

[4.6.2. 튜플 메소드 108](#_Toc480727051)

[4.6..3. 튜플 원소로 Mutable 원소 처리 방법 109](#_Toc480727052)

[4.7. 리스트 데이터 타입 110](#_Toc480727053)

[4.7.1. 리스트 생성 110](#_Toc480727054)

[4.7.2 리스트 내의 메소드 110](#_Toc480727055)

[4.7.3 리스트 내에 리스트를 넣을 때 초기화 하는 법 116](#_Toc480727056)

[4.7.4. +, \* 연산자 처리 118](#_Toc480727057)

[4.8. sequence 타입 slicing 이해하기 120](#_Toc480727058)

[4.8.1. index 순서 이해하기 120](#_Toc480727059)

[4.8.2. slice 클래스 이해하기 123](#_Toc480727060)

[4.9 배열 array.array 모듈 126](#_Toc480727061)

[4.10. Sequence 타입 형변환 128](#_Toc480727062)

[4.11. 지능형 리스트 처리 130](#_Toc480727063)

[4.12. packing과 unpacking 처리 132](#_Toc480727064)

[4.13. 포매팅 처리 136](#_Toc480727065)

[4.13.1 포맷함수와 메소드 이해하기 136](#_Toc480727066)

[4.13.2. 포맷함수 및 메소드 인자 매핑 140](#_Toc480727067)

[4.1.3.3 포매팅 꾸미기 143](#_Toc480727068)

[**5. 파이썬 Mapping/Set데이터 타입** 145](#_Toc480727069)

[5.1. dict 타입 이해하기 145](#_Toc480727070)

[5.1.1 dict 의 키 생성 기준 145](#_Toc480727071)

[5.1.2. dict 생성하기 147](#_Toc480727072)

[5.1.3. dict 검색 149](#_Toc480727073)

[5.1.4. dict 데이터 타입 추가 /삭제 152](#_Toc480727074)

[5.1.5. dict 타입에 keys, values, items 객체 이해하기 154](#_Toc480727075)

[5.1.5. dict 타입에 packing/unpacking 158](#_Toc480727076)

[5.2. set 데이터 처리 하기 160](#_Toc480727077)

[5.2.1 set 160](#_Toc480727078)

[5.2.2. frozenset 이해하기 163](#_Toc480727079)

[5.3. 지능형 dict/set 165](#_Toc480727080)

**1. 파이썬 기본 이해하기**

파이썬은 다양한 개념들이 추가되면서 진화한 프로그램 언어이다. 객체지향, 함수형언어의 특징도 추가되면서 데이터 과학 등으로 확장된 기능들을 제공하면서 발전하고 있다.

프로그램언어를 배울 때 필요한 가장 기본 개념들을 이번 장에서 배우면서 하나하나씩 확장하는 것이 공부에 가장 좋다.

이번 장에서는 모든 프로그램언어에서 설명하는 기본 요소인 용어, 연산자, 데이터 타입 등에 대해 알아보겠다.

## 1.1 기본 용어 알아보기

### 1.1.1. 리터럴(literal) 과 표현식(expression )

* 리터럴

리터럴이란, 프로그램 언어로 작성된 코드에서 값을 대표하는 용어이다. 기본으로 사용되는 정수, 부동 소수점 숫자, 문자열, 불린 등의 실제 값들과 표현식의 결과 등을 나타낸다.

아래의 예제는 주피터 노트북을 실행한 후에 셀에 1을 입력한 후에 실행(shift+enter)하면 결과가 1이라는 정수인 리터럴 값을 바로 출력한다.



* 표현식

표현식(expression)은 프로그래밍 언어가 해석하는 하나 이상의 명시적 값, 상수, 변수, 연산자조합이고 함수도 실행되면 하나의 값이 되어 이를 조합해도 표현식으로 인식한다.

표현식을 평가해서 실행될 때에는 우선 순위 및 연관 규칙에 따라 해석해 실행되며 평가된 결과는 하나의 값인 리터럴로 표현된다. 표현식 처리 순서는 좌측부터 우측으로 가며 평가하고 ( ) 연산자를 만나면 최우선으로 처리된다.

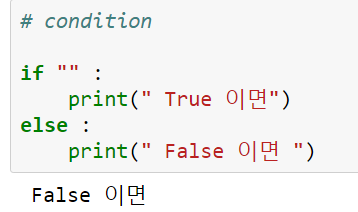
아래의 예는 3+4라는 표현식을 주피터 노트북에 입력하고 실행하면 표현식을 평가해서 최종 값을 결과로 처리한다.



* 조건식(condition)

조건식은 표현식이지만 특정 조건을 판단할 수 있는 구문인 If, while 문 등에 사용되며 조건이 평가되면 True나 False로 결과가 처리된다.

예제는 주피터 노트북에 if 문을 넣고 문자열에 아무 값도 없다는 조건을 입력하면 “”는 False로 인지해서 else 문장을 처리한다.



### 1.1.2. 데이터타입, 클래스, 객체

.

* 데이터 타입, 클래스, 객체

객체란 실제 파이썬 언어에서 사용되는 모든 값들은 객체로 보고 처리되며 리터럴고 표현식의 결과가 객체로 나타낸다.

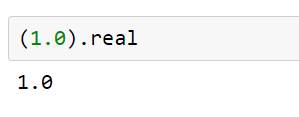
파이썬에는 객체가 클래스(class) 객체와 인스턴스(instance) 객체로 나누어 설명하며 클래스 객체는 인스턴스 객체를 생성하고 메소드를 관리하는 역할을 담당하며, 인스턴스 객체는 자신의 속성들을 가지고 있고 클래스 객체의 메소드를 이용해서 실제 실행을 담당한다

리터럴 등의 값은 특정한 데이터 타입을 갖고 있고 이를 객체지향에서는 클래스로 표현한다.

* 클래스의 구성
* 속성 (attribute)

객체 내부의 변하되는 상태를 관리하는 변수이다.

예제는 1.0은 float 객체의 인스턴스이고 이 내부의 속성 real로 값을 조회한다.



* 메소드(method)

객체의 행위(behavior) 즉 상태의 변화를 발생시키는 함수이다

예제는 1.0 인스턴스 객체의 is\_integer 메소드를 호출하면 소수점이하 숫자가 없으므로 정수로 전환이 가능하므로 True 결과값을 출력한다.



* 프로퍼티

객체의 행위에 연관되지만 속성처럼 접근하도록 처리하는 패턴

* 강한 타입 과 약한 타입

파이썬 언어는 인터프리터이지만 데이터 타입은 강한 타입으로 구성되면 리터럴 값으로 선언된 데이터 타입 자체를 변경할 수 없다.

파이썬에 데이터 타입 모델이 미리 정의되어 있고 이 타입을 변경할 규칙은 제공하지 않으며 이를 처리하는 장점은 프로그래머에게 엄격한 규칙을 부과하고 결과의 일정한 일관성을 보장한다는 것이다.

* 객체가 생성시 데이터 타입 즉 클래스가 확정되면 변하지 않는다.



* 동적 타입

파이썬은 특정 변수나 함수의 리턴 결과 등에 명기적으로 타입을 지정할 수없다. 처리되어진 객체가 기본으로 변수에 레퍼런스만 할당하므로 처리결과에 따라 그 변수의 타입이 변경되는 동적타입 구조를 가지고 있다.

### 1.1.3. 문장(statements)

문장은 리터럴과 표현식 등이 조합으로 완전한 처리결과를 할 수 있는 구조이면서 처리결과에 대한 상태를 관리할 수 있다.

함수정의인 키워드 def, 클래스 정의인 키워드 class 정의로 정의된 것도 완전한 문장으로 인식한다.

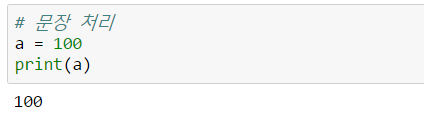
다양한 문장 블록을 처리를 위해 순환, 제어 등을 하는 경우도 완전한 문장으로 인식한다.

* 할당문(assignment)

리터럴이나 표현식의 결과에 대한 상태를 저장하기 위해서는 할당문이 필요하다. 변수에 값, 표현식, 함수 등의 처리된 결과를 할당연산자( =)를 이용해서 저장하는 문장이다.

실제 값이 저장되는 것이아니라 파이썬 객체들의 주소인 레퍼런스가 저장된다.

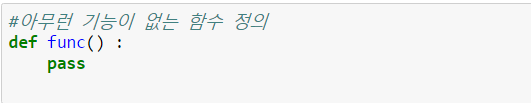
예제는 100이라는 리터럴 즉 정수 클래스이 인스턴스 객체인 100의 주소 즉 레퍼런스를 변수 a에 저장하는 것이다. 변수 a를 출력하면 변수 a에 있는 주소를 검색해서 100을 찾아와 출력하는 것이다.



* 함수(function) 정의문

함수를 정의하는 이유는 함수를 정의하면 파이썬 내부적으로 이 정의를 보고 함수를 객체화해서 함수이름에 저장해서 다시 사용할 수 있도록 만든다.

에제는 함수 정의시 아무런 행위도 없을 경우 pass 키워드를 사용해서 정의만 하면 실제 호출시 아무런 행위는 없지만 하나의 함수 인스턴스 객체가 생겼다는 것을 알 수 있다.



* 클래스(class) 정의문

함수 정의문 처럼 인스턴스 객체를 만들기 위해서는 클래스를 정의해야 한다.

주로 사용자 정의 데이터 타입인 클래스를 만들어서 다양한 기능을 처리할 수 있도록 정의하면 파이썬 내부에서는 클래스 객체를 만들어준다.

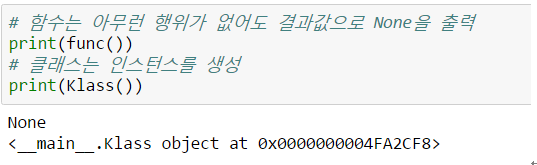
예제는 클래스가 아무런 일을 하지 않는 것을 정의했다. 하지만 클래스는 인스턴스 생성는 기능만은 자동으로 생긴다.



* pass 문

실제 문장의 완성을 위해 아무런 일을 하지 않는 표현이 필요하다. 파이썬에서는 pass를 하나의 문장처럼 사용하도록 정의했다.

위의 정의된 함수와 클래스를 실행해보면 아무런 일은 하지 않지만 클래스인 경우는 인스턴스를 하나 생성한다.

. 

### 1.1.4. 식별자(identifier)

식별자는 프로그램언어로 프로그램을 작성할 때 변수, 함수, 클래스, 모듈, 패키지등 명확한 구별이 필요한 부분에 사용되는 개념이다.

파이썬에서 식별자가 중요한 이유는 모든 것을 이름 즉 식별자로만 관리한다.

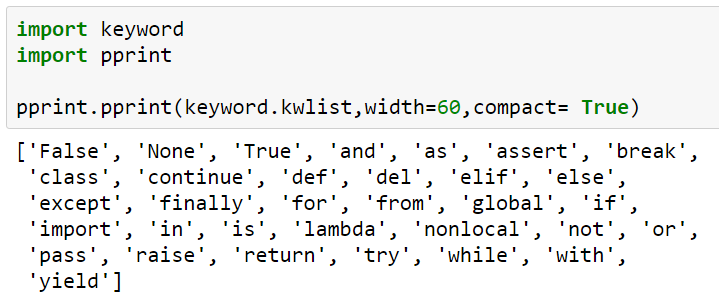
이름이 중복되었을 때 원하는 것을 처리하지 않는 경우가 발생한다.

파이썬에서 식별자가 충돌되는 이유는 모듈마다 관리되는 Namespace가 하나만존재하고 그 안에 변수, 함수, 클래스 등이 식별자들을 키로 관리하게 되어 중복이 발생하면 값만 변경하는 체계로 구성되어 있다.

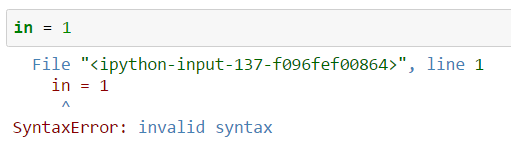
* 예약어(keyword)

파이썬 내부에서 사용하는 용어를 예약어(keyword)로 등록되어 식별자로 사용하지 못하게 정했다.

파이썬에서는 keyword 모듈에 자신들이 사용하는 예약어 목록을 관리한다. 사용자가 프로그램을 작성시 식별자로 명명하는 변수, 함수, 클래스, 모듈명으로 키워드를 사용하면 안된다 .



* 키워드로 변수를 정의할 때 SyntaxError를 처리



* 변수(Variable)

파이썬에서 변수는 아직 알려지지 않거나 어느 정도까지만 알려져 있는 양이나 정보 즉 인스턴스 객체에 대한 상징적인 이름이다.

프로그램을 작성할 때 변수가 정의되어야 재사용이 가능하므로 항상 모든 것은 변수에 저장되어야 하며, 실제 변수에는 값이 저장되는 것이 아니라 그 값이 위치한 주소인 레퍼런스가 저장된다.

실행 중에 변수는 항상 변경이 가능하고 변경될 때마다 주소인 레퍼런스가 변경되어 실제 값인 인스턴스 객체를 가르치는 것을 보관하는 용도로 사용될 뿐이다.

* 변수를 처리하는 예시



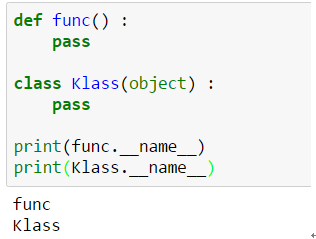
* 함수(function)명/클래스(class) 명/모듈(module)명

파이썬에서는 함수명, 클래스명, 모듈명을 정의하면 관리하는 기준도 두개로 분리된다. 하나는 변수명으로 Namespace에 등록하고 실제 함수, 클래스, 모듈 이름은 객체 내의 \_\_name\_\_속성에 문자열로 저장한다.

예외는 익명함수인 lambda 인 경우만 \_\_name\_\_속성에 이름이 없다는 표시로 lambda를 등록해서 처리한다.

* + 함수와 클래스의 이름 확인

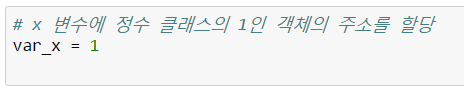
함수와 클래스를 정의해서 함수와 클래스 명을 확인할 수 있다..



* 식별자 명명규칙
  + 문자, 숫자, underscore

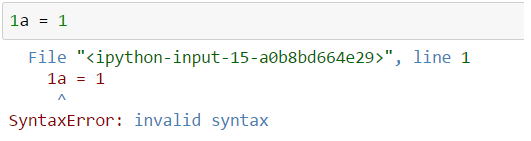
식별자 명명 규칙은 문자, 숫자, underscore(\_)의 조합으로 구성되지만 첫번째 위치에는 절대 숫자가 올 수 없다.

예제는 문자와 언더스코어로 구성된 변수를 지정해서 1을 할당한 것이다.



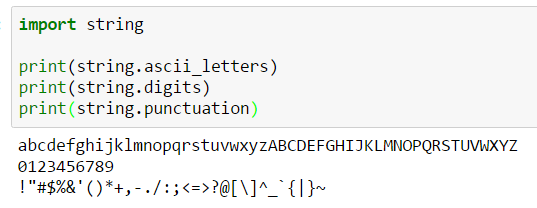
* + 숫자를 첫번째 위치로 정했을 경우

명명규칙에 첫자리는 반드시 문자나 \_가 와야 하므로 문장구성의 오류인 Syntax 에러를 처리한다.



* 식별자 가능 문자

파이썬이 문자열을 아스키코드에서 유니코드로 변경했으므로 다양한 유니코드 문자도 수용하지만 관행상 아스키코드인 영어로 표현한다.

예제는 아스키 문자를 관리하는 문자열(string) 모듈에 있는 정보를 확인해 본것이다. 관행 상 ascii\_letters, digits, punctuation 중에 \_만으로 식별자를 표현하는게 좋다.   


### 1.1.5. 네임스페이스(Namespace) 관리 규칙

파이썬은 모듈, 클래스, 함수 등에 식별자인 변수를 지정하면 Namespace 영역이 생기고 그 영역별로 접근할 수 있는 scope 규칙이 정해져 있다.

Namespace 은 키와 값으로 관리되므로 각 변수는 키에 해당하고 인스턴스 객체의 레퍼런스는 값으로 할당되어 관리된다.

Namespace에 대한 세부 사항은 모듈, 함수, 클래스 영역에서 상세한 설명을 하기로 하겠다.

## 1.2 파이썬 기본 내장 함수 이해

파이썬 언어의 특징은 내장함수를 이용해서 프로그램 내부의 다양한 정보를 확인할 수 있다.

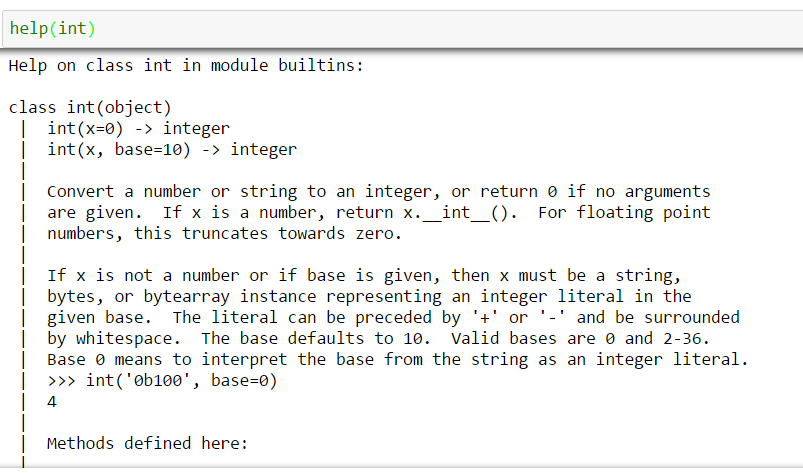
제일 먼저 이런 정보를 어떻게 검색하는지에 대해 알아보기로 하겠다.

### 1.2.1. help 함수

파이썬 내부의 함수나 클래스 등의 기능을 검색하기 위한 함수를 제공한다.

* 내장 데이터 타입 int에 대한 help 조회

정수 값들을 관리하는 int는 함수가 아닌 클래스이며 내부의 속성과 메소드들에 대한 정보가 출력된다.

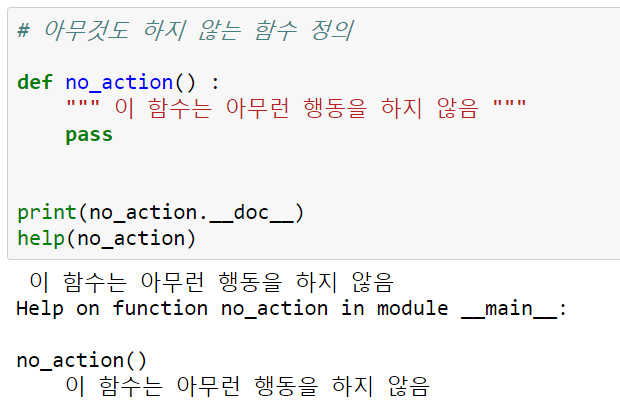


* 사용자 정의 함수에 help 함수가 실행을 위한 작성 기준

사용자 함수를 def 키워드를 이용해서 함수를 정의할 때 블록이 첫번째 라인의 주석을 함수의 설명으로 인식한다.

함수의 설명은 “ “, “”” “”” 등 문자열로 처리하면 자동으로 이를 docstring으로인지해서 함수 내부속성인 \_\_doc\_\_에 저장해둔다.

향후에 help 함수로 함수를 조회하면 이 정보를 조회해서 화면에 보여준다..

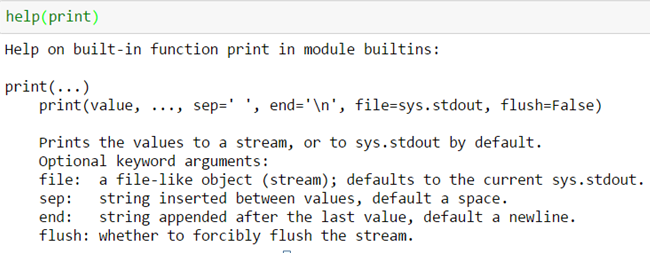


### 1.2.2. print 함수

파이썬으로 프로그램을 작성한 경우에 실행된 결과나 중간과정을 출력하고 싶은 때가 많다. 함수 print를 이용해서 표준 출력창에 출력이 되도록 처리하는 함수이다.

* help함수를 이용해서 print함수를 조회

파라미터 명에 값이 할당된 이유는 기본값이며 일단 print 함수를 사용해도 기본값을 사용할 경우 이 파라미터에 인자를 주지 않아도 된다는 뜻이다.



* 함수 print 처리 예시

문자열은 두개 넣어서 출력하면 분리자(sep)가 기본으로 빈공칸을 가지므로 겹쳐나오지는 않는다.

분리자에 쉼표를 “,” 넣고 출력하면 문자열 중간에 쉼표가 표시된다.

출력된 마지막에 개행문자 대신 \*\*\*를 넣으면 마지막에 출력된다.

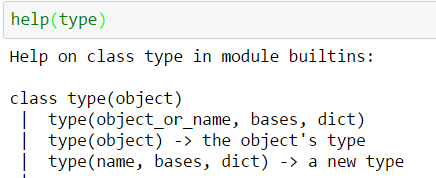


### 1.2.3. type class

파이썬이 모든 것은 객체로 구성되어 있고 이 객체에는 데이터 타입인 존재하므로 모든 리터럴이 어떤 클래스로 만들어졌는지를 확인해 보고 싶어질 것이다.

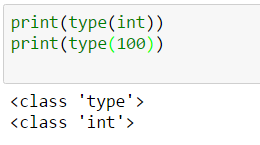
* help함수를 이용해서 type class 조회

type은 함수가 아닌 class 객체이며 인자로 객체를 넣으면 어떤 클래스에 위해 만들어졌는지를 보여준다



* 타입 체크하기

파이썬에서 타입체크는 type 클래스를 이용해서 정보를 확인하는 것이다. 100 인스턴스를 넣고 조회하면 명확히 int가 출력되지만 int를 넣고 조회하면 type이 나온다. 이 부분은 뒤에 메타클래스에서 상세한 설명을 하겠다.



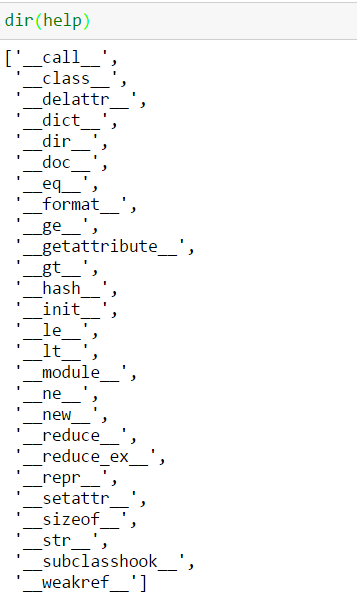
### 1.2.4. dir 함수

파이썬은 모든 것이 객체로 관리하므로 객체의 구성요소들이 자동으로 만들어진다. 이 내부를 조회하기 위해 dir 함수를 이용해서 내부 구성을 확인할 수 있다.

.

* 함수 help 내부 조회

함수도 인스턴스 객체이므로 내부를 조회하면 메소드와 속성들을 가지고 있다. 상세한 설명은 이 책을 진행하면서 자세히 설명하기로 하겠다.

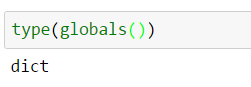


### 1.2.5. globals 함수

파이썬 모듈(.py)로 작성되면 내부에는 Namespace가 존재하며 이 내용을 조회할 수 있는 내장함수인 globals를 제공한다.

* 글로벌 함수의 결과는 dict 타입

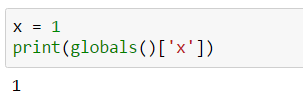
모듈 내에서 globals()함수를 실행하고 함수의 결과값에 대한 타입을 조사해보면dict 클래스가 나온다. 클래스 dict은 키와 값으로 원소를 구성하는 데이터 타입이다.



* Namespace에 변수 할당하는 방법

모듈에서 변수 x를 정의하면 파이썬 인터프리터는 자동으로 Namespace에 등록한다.

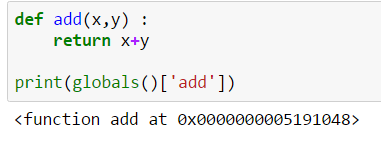
함수 globals()를 싱행한고 결과에서 문자열로 변수명을 주고 조회하면 변수 x의값을 출력한다.



* Namespace에 함수이름을 등록

모듈에서 함수 add를 정의하면 파이썬 인터프리터는 namespace 에 함수명을 변수와 동일한방법으로 등록한다.

변수와 동일한 방법으로 조회하면 함수 객체에 대한 정보를 출력한다.

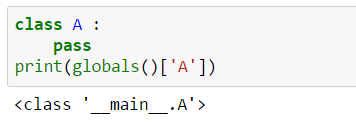


* Namespace에 클래스이름을 등록

모듈 내에서 클래스 A를 정의하면 파이썬 인터프리터는 namespace 에 클래스 명을 변수와 동일한방법으로 등록한다.

변수와 동일한 방법으로 조회하면 클래스 객체에 대한 정보를 출력한다.

.



### 1.2.6. locals 함수

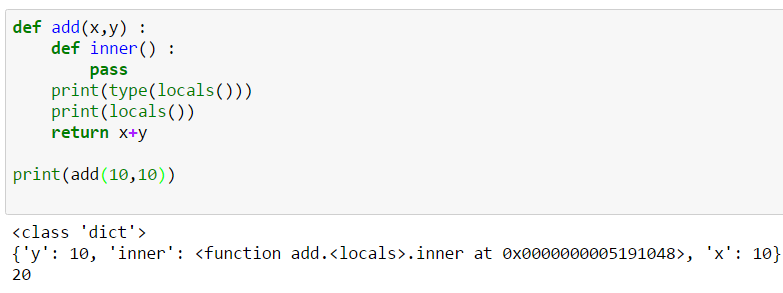
파이썬은 함수별로 각각 자신의 Namespace를 관리하게 만들었다. 함수가 정의되면 내부 로직을 처리하는 영역에 대한 local namespace가 추가적으로 만들어진다.

Local namespace 영역을 조회하려면 locals 함수를 사용해야 하고 locals 함수는 항상 함수 내부에서만 조회가 된다.

* 함수 내부의 로컬 영역에 대한 조회

함수가 실행시 즉시 local namespace가 만들어진다. 함수가 종료되며 사라진다.

함수 add를 정의하고 내부에서 함수 locals 를 이용해서 출력해보면 함수의 파라미터와 내부함수가 local namespace에 할당되어 있는 것을 확인할 수 있다..



### 1.2.7. 객체 비교 방식(is, ==)

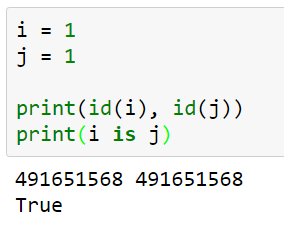
파이썬은 모든 것을 객체로 인식하기에 객체들을 비교할 때 is 연산을 사용한다.

실제 파이썬 권고사항은 객체를 비교할 때도 객체 내의 값을 가지고 비교하는 것을 권고하므로 객체의 비교도 주로 == 연산자를 이용해서 처리한다.

* 키워드 is

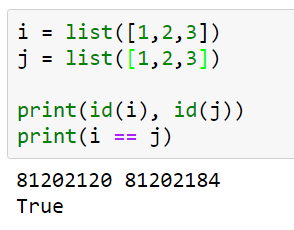
두 객체 동일한 객체인지를 비교할 때 is를 사용한다.

특히 None 처럼 인스턴스를 비교할 경우에만 관행상으로 is 연산을 사용한다.



* == 연산자

두 객체 동일한 객체를 비교하는 것이 아닌 내부에 있는 원소들이 동일한지를 비교한다.



**2. 숫자 원자형 데이터 타입 이해하기**

숫자형 데이터 타입이 특징은 원자형 값만 가지고 있어 내장 값에 대해서는 절대로 변경할 수 없는 Immutable 객체이고 인스턴스 객체를 생성시 리터럴과 생성자 2가지 방법을 사용한다.

또한 런타임에 속성이나 메소드를 추가할 수 없어 제공되어진 속성과 메소드만을 사용해야 한다. 추가적인 기능이 필요하면 사용자 정의 클래스를 만들어서 사용해야 한다.

이번 장에서는 숫자 타입과 이를 지원하는 연산자를 이해해본다.

## 2.1 내장 숫자 데이터 타입 이해하기

### 2.1.1. 숫자 데이터 타입의 특징

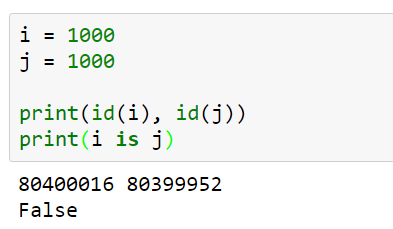
파이썬 내부 인터프리터 엔진에서 원자적 타입이 일부는 항상 동일하게 처리한다.하지만 특정 수보다 큰 경우에는 동일한 수도 별도의 인스턴스로 만들어서 관리한다.

특정 숫자가 동일하지만 별도의 레퍼런스를 가지므로 동일여부 체크시 == 연산자 사용을 권고한다.

.

* 동일한 숫자지만 레퍼런스가 상이

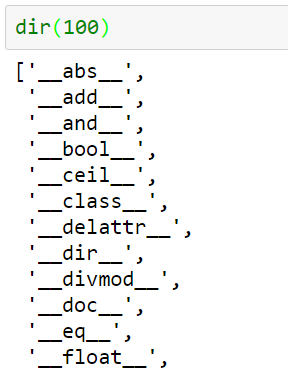
특정 숫자도 동일한 객체여야 하지만 파이썬에서는 별도의 객체로 만든다. 이런 이유로 동일한 처리를 하려면 == 연산을 이용해서 처리하는게 좋다.



* 런타임 속성 추가 불가

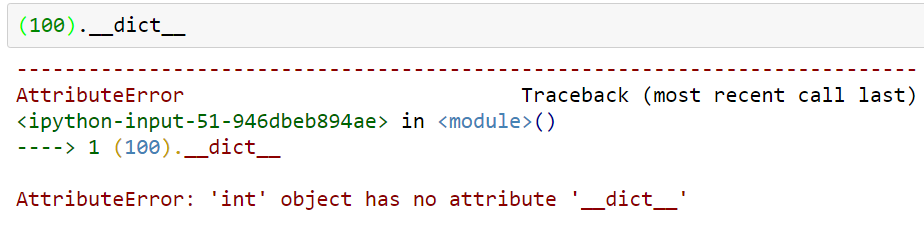
내부에 \_\_dict\_\_ 속성이 존재하지 않아서 런타임에 속성이나 메소드를 추가할 수 없다.

* + 함수 dir 조회시 \_\_dict\_\_ 미존재



* + 인스턴스에서 \_\_dict\_\_ 조회

속성이 존재하지 않다고 에러 발생시킨다.

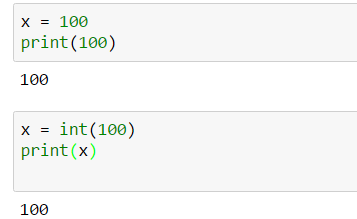


### 2.1.2 정수 int class

파이썬에서 수학의 정수와 동일하며 파이썬 내의 부동소수점 숫자인 float를 제외한 모든 정수(integer)를 관리하는 데이터 타입이다.

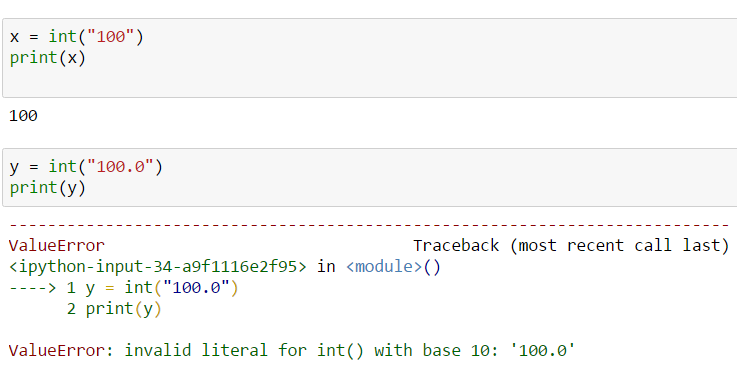
* 생성방법

리터럴 값을 직접 작성하거나 클래스 int(리터럴)를 이용해서 생성도 가능하다.



* 정수 생성시 오류 조건

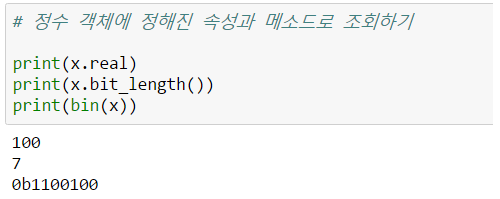
문자열로 받은 소수점을 가진 숫자는 정수에서는 문자로 인식해서 오류로 처리된다. 소수점 숫자가 문자열로 들어오면 float로 전환한 후에 int로 전환처리가 필요하다.



* 주요 속성 및 메소드

속성 real로 정수 값을 조회할 수 있다. 정수에서만 이진 연산도 가능하므로 이진수로 변환한 경우 bit\_length 메소드로 이진수 크기를 알수 있다.

정수를 bin 함수를 이용해서 변환하면 0b인 이진수 표현을 제외하면 총 7개 bit로 숫자가 만들이진 것을 볼 수 있다.



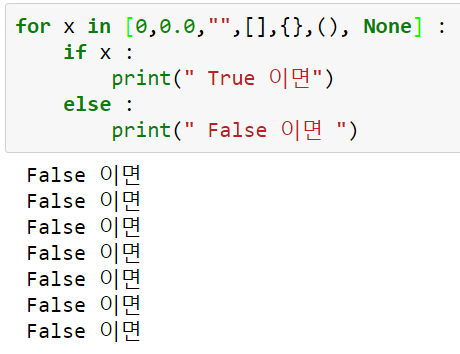
### 2.1.3. 불리언 타입 : bool

파이썬 bool 클래스는 int 클래스를 상속받아 처리되는 클래스이므로 int 클래스의 메소드가 다 실행되므로 사칙연산 등이 다 가능하다.클래스 bool 의 인스턴스는 True와 False 2개만 만들어져 있다.

* False로 추론되는 경우

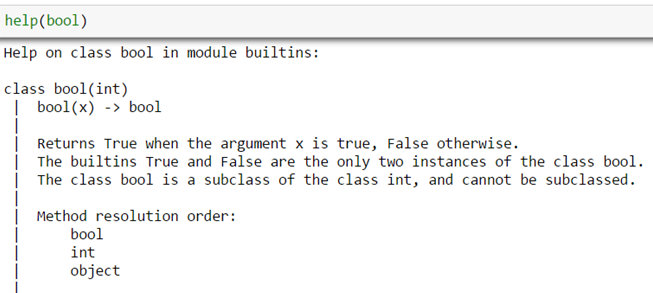
파이썬에서는 bool 타입 체크는 다양한 조건을 수용해서 일단 데이터가 없거나내부에 원소를 가지지 않았다는 것을 전부 False로 추론해서 처리한다.

None, 숫자0 ,문자열에 없음(“”), 빈 리스트([ ]), 빈 튜플(()), 빈 dict({}) 등은 False이고 나머지 대부분은 True 본다.



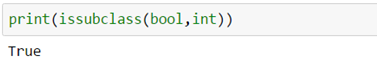
* help로 내부 정보 조회

클래스 bool를 조회해 보면 int를 상속했고 특정 값을 bool 타입으로 변환한다는설명이 나온다.

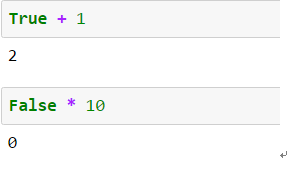


* bool 타입은 int를 상속을 받아 처리

실제 상속을 받았는지를 확인해보면 결과도 True로 표시한다.



* + 숫자 타입과 숫치계산도 가능



* 생성 방법

클래스 Bool로 생성해보면 True와 False로 결과값을 리턴한다. .

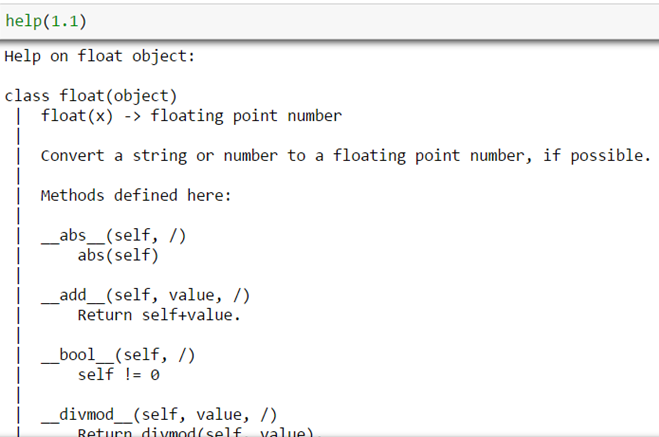


### 2.1.4 실수 float

파이썬에서 부동소수점 숫자인 float를 관리하는 데이터 타입이다.

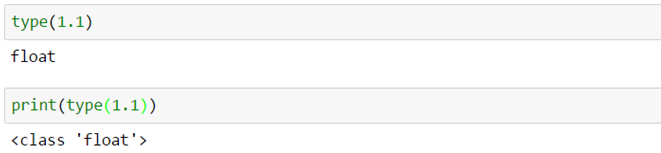
* help로 내부 정보 조회

help함수, print 함수, type 클래스로 인스턴스 객체(1.1)에 대한 내부 정보를 조회해보면 아래와 같은 결과가 나온다.



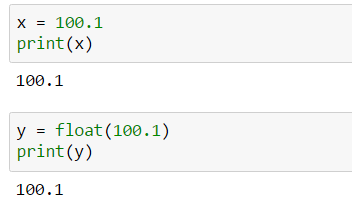
* 실수에 대한 타입 확인

실수 리터럴 즉 실수 인스턴스를 넣어서 조회하면 float 클래스를 표시한다. 두 표현이 다른 것은 str(), repr() 호출되어 문자열 처리하는 방법이 다르기 때문이다.



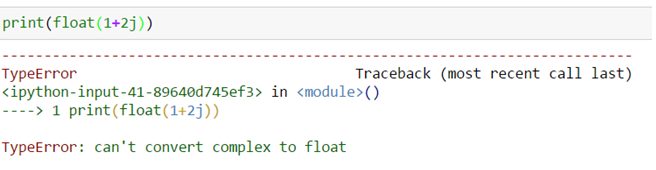
* 생성방법

직접 리터럴로 생성하거나 클래스 float를 이용해서 인스턴스를 생성하면 된다.



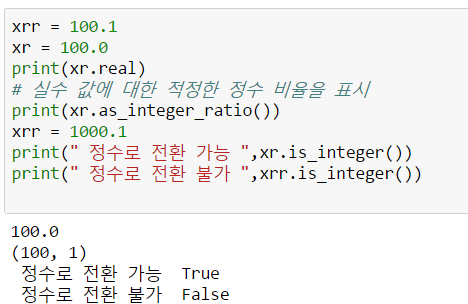
* 실수 생성시 오류

수학도 복소수가 실수보다 더 큰 범위이므로 파이썬에서도 float에 은 복소수complex가 들어오면 처리를 할 수 없도록 만들었다.



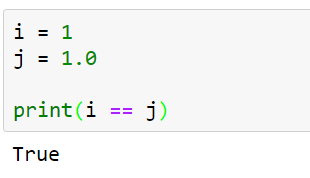
* 속성과 메소드

속성으로는 real, 메소드는 정수 값으로 실수에 대한 비율을 표시하는 as\_integer\_ratio와 정부 전환 가능여부를 체크하는 is\_integer 가 있다.



* 정수와 실수 동일한 값 확인

동일한 값인 1과 1.0에 대해 어떻게 판단하는지를 확인해 본다. 정수 int를 단순히 float화 한 경우를 비교하면 True로 인식한다.

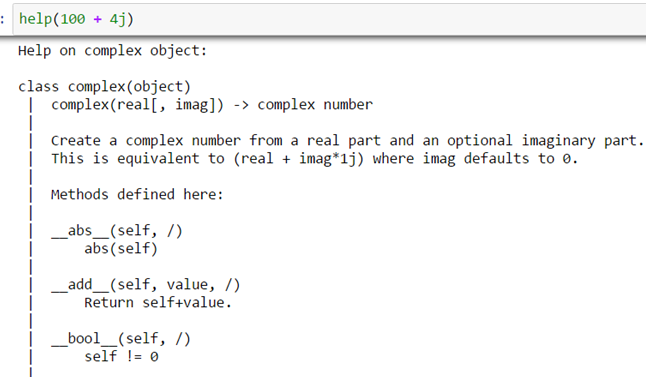


### 2.1.5 복소수 complex class

파이썬에서 복소수 숫자인 complex를 관리하는 데이터 타입이다. 복소수의 허수 표기법이 i에서 j로 표시된다.

* help로 내부 정보 조회

help함수, print 함수, type 클래스로 인스턴스 객체(100+4j)에 대한 내부 정보를 조회해보면 아래와 같은 결과가 나온다.



* 복소수에 대한 타입 체크



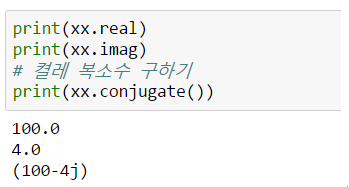
* 생성방법

리터럴과 클래스로 인스턴스를 생성하는 두가지 방법이 다 가능하다.



* 속성과 메소드

속성으로는 real,imag가 있고 메소드는 켤레복소수를 표시하는 conjugate 가 있다.

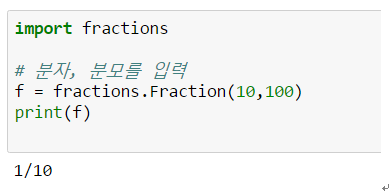


### 2.1.6. 유리수 Fraction class

유리수는 별도의 fraction 모듈에서 보관하므로 사용시 import로 로딩을 해서 처리해야 하며 정수와 연산 시만 유리수가 유지되면 float/complex와 계산시는 더 상위 개념인 float/complex 타입으로 변경된다.

* 생성방법

분모 (denominator )와 분자(numerator**)**에 대한 값을 정수로 넣어 유리수를 생성한다.



* 생성이 안 되는 경우

유리수는 분자와 분모가 정수일 경우만 처리가 되므로 실수값이 오면 생성이 되지 않는다.

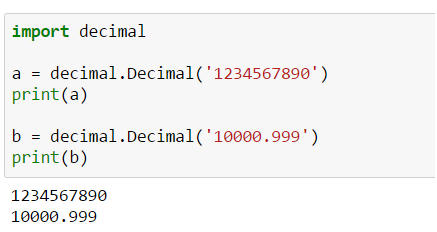


### 2.1.7. 큰 수 Decimal class

숫자를 계산시 정밀한 수치를 계산이 필요한 경우 decimal 모듈을 이용해서 처리해야 한다.

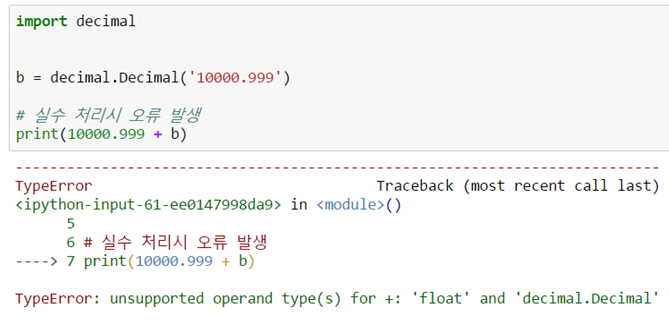
* 생성방법

정수와 실수에 대해 정의해서 decimal을 생성한다.



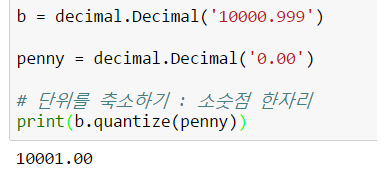
* 실수와 계산시 오류 발생

파이썬 float 연산에서 decimal 모듈과 처리시 데이터 타입에 대한 지원상 에러가 발생한다.



* 주요 메소드

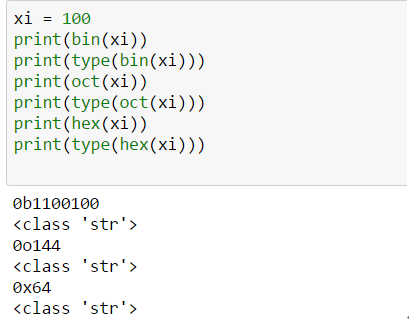
특정 단위로 계산이 필요할 경우 명확한 단위를 별도의 인스턴스로 만들고 quantize 메소드로 단위를 확정하도록 계산하면 된다.



### 2.1.8. 2, 8, 16진법 변환 하기

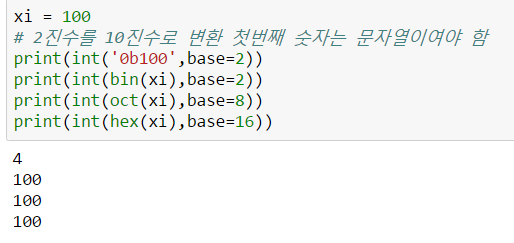
정수일 경우만 2진수 bin 함수 : 0b1111, 8진수 oct 함수 : 0o1111, 16진수 hex 함수 : 0x111를 이용해서 변환이 가능하다. 변환된 결과는 문자열로 표시한다.

* 정수를 2/8/16진법으로 변환



* 2/8/16진법을 정수로 변환

정수로 변환 시에는 반드시 진법을 base에 지정해야 정수로 전환된다.

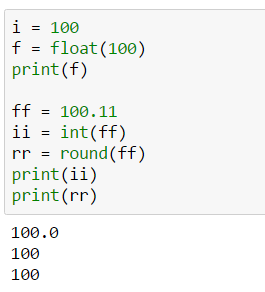


### 2.1.9 숫자 타입의 형변환

파이썬에서 형변환은 실제 클래스의 인스턴스 객체로 다시 만드는 것이다. 기존 객체가 아닌 별도의 객체가 만들어지는 것이다.

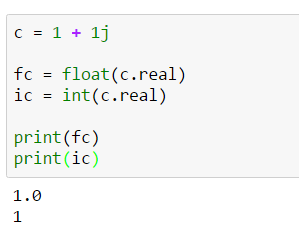
* int와 float 변환

정수와 실수는 바로 변환이 가능하면 소수점 이하는 정수 변환이 절사처리된다.



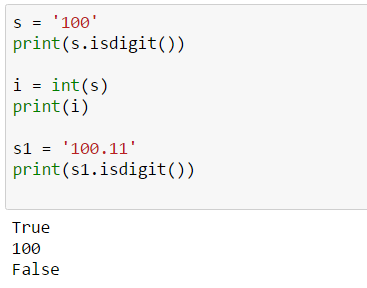
* complex, int, float 변환

복소수 값은 변화은 정수나 실수로 변환이 되지 않으므로 허수부를 제외한 부분만 변환처리하면 정수나 실수로 변환이 된다.



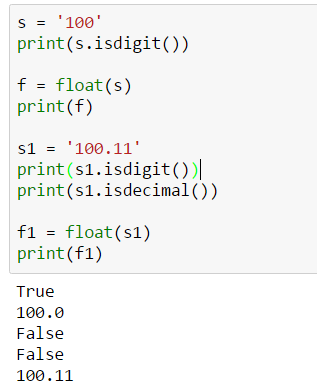
* 문자열(str)과 정수( int) 변환

문자열에서 isdigit 메소드에서 False 값이 나오면 int로 변환이 불가능하다



* 문자열(str)과 실수(float) 변환

소수점을 가진 숫자 값일 경우 변환이 가능하다.



## 2.2. 연산자(Operator)

수학을 기반으로 연산자를 만들었으므로 수학에서 처리되는 연산자의 원칙에 따라 파이썬 내에서도 연산방식이 동일하게 처리된다.

연산시 연산자 우선순위에 따라 처리되므로 이 기준을 명확히 이해하고 처리해야 정확한 값이 계산된다.

연산자 우선순위를 정확히 맞추기 위해서는 제인 우선순위가 높은 괄호 연산을 사용해서 묶어주면 된다.

### 2.2.1. 연산자 우선순위

표현식 내의 연산자의 우선순위는 아래의 표처럼 처리 된다. 연산자 우선순위 처리시 오류가 발생할 수 있으므로 그룹 연산자인 괄호 ( )로 표시해서 처리해야 한다.

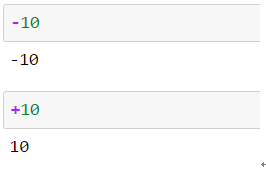
|  |  |  |  |
| --- | --- | --- | --- |
| 순위 | 구분 | **Operator** | **Description** |
| 0 | 그룹 | **( )** | Parentheses (grouping) |
| 1 | 함수 | ***f*(args...)** | 함수 호출 |
| 2 | 참조 | ***x*[index:index]** | 부분구조 읽기(슬라이싱) |
| 3 | 참조 | ***x*[index]** | 원소 참조 |
| 4 | 참조 | ***x.attribute*** | 속성 참조 |
| 5 | 산술 | **\*\*** | 제곱 구하기 |
| 6 | 비트 | **~*x*** | Bitwise not |
| 7 | 부호 | **+*x*, -*x*** | 양수, 음수 |
| 8 | 산술 | **\*, /, %** | 곱하기, 나누기, 나머지구하기 |
| 9 | 산술 | **+, -** | Addition, subtraction |
| 10 | 비트 | **<<, >>** | 쉬프트 연산 |
| 11 | 비트 | **&** | 비트 and |
| 12 | 비트 | **^** | 비트XOR |
| 13 | 비트 | **|** | 비트 OR |
| 14 | 비트 | **in, not in, is, is not, <, <=,  >,  >=,<>, !=, ==** | Comparisons, membership, identity |
| 15 | 논리 | **not *x*** | NOT 논리연산 |
| 16 | 논리 | **and** | AND 논리연산 |
| 17 | 논리 | **or** | OR 논리연산 |
| 18 | 함수 | **lambda** | 람다 표현식 |

### 2.2.2. 산술연산자

산술 연산자는 수학의 사칙연산 등 수치에 대한 계산을 처리하도록 한다.

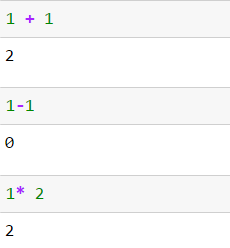
* 부호 연산 : +, -

부호 표시를 하는 +,-는 단항연산자(unary)로 숫자에 대한 부호를 표시한다.



* 사칙연산 : +, -, \*

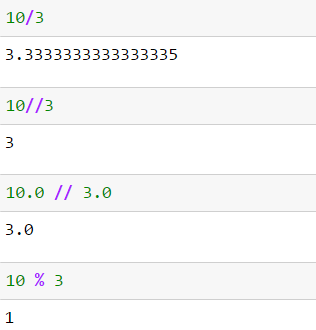
이항 연사인 덧셈, 뺄셈, 곱셈은 숫자 타입의 값을 계산해서 연산자 특징의 맞춰 계산된 결과를 리턴한다.



* 사칙연산 : /, //, %

나누기도 float을 계산하는 /, int로 계산하는 //, 나머지 값을 계산하는 % 연산자들로 구성하면 계산된 값을 리턴한다.

// 나누기는 float일 경우에도 몫만 구하고 소수점 이하는 생략한다.

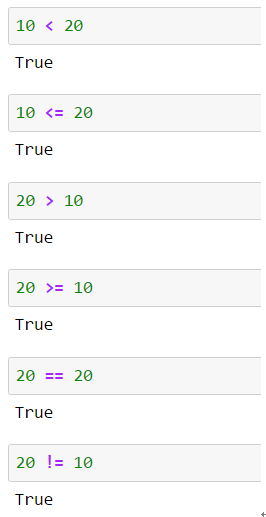


### 2.2.3. 비교 연산자

두 수식을 비교하는 연산자로서 비교 연산한 결과를 True, False로 표현하는 연산자들이다.

>(큼), <(작음), >=(크거나 같음), <= (작거나 같음), !=(같지 않음), ==(같음)

비교 연산은 조건식을 평가하는 if 문이나 while문에 사용되면 비교결과에 대한 값으로 제어문과 순환문에 대한 처리를 제한한다.



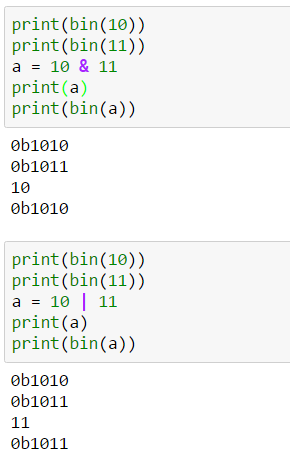
### 2.2.4. 비트 연산자

파이썬의 비트 연산자는 숫자를 2 진법으로 계산하는 연산자이다. 비트 연산자를 사용할 때는, 숫자를 0과 1로 구성된 문자열로 취급한다는 말이다.

숫자 타입에서 비트연산자은 내장타입 int 타입일 경우에만 처리된다

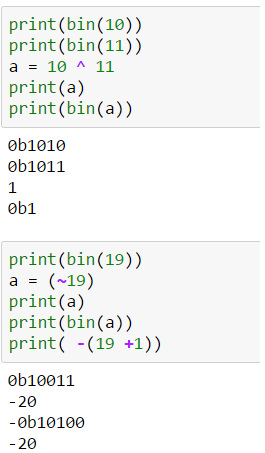
* 비트 처리 : &, |

|  |  |
| --- | --- |
| 연산자 | 설명 |
| &/and | 논리곱 연산자로서 비트가 두 항에 모두 나타나는 경우 비트 결과를 표시한다. |
| |/or | 논리합 연산자로서 비트가 두 항 중 어느 곳에 나타나는 경우 비트 결과를 표시한다. |



* 비트 처리 : ^, ~

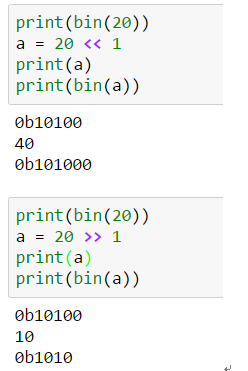
|  |  |
| --- | --- |
| 연산자 | 설명 |
| ^ | 배타적 논리합 연산자로서 어느 한 쪽의 항에만 비트가 존재할 경우 비트 결과를 표시한다. |
| ~ | 부정 연산자로서 비트를 뒤집어서 각 비트에 대하여 정확히 반대를 반환하며 실제 처리결과는 숫자 1을 추가하고 부호를 바꾸면 된다. |



* 시프트 연산

곱하기와 나누기를 처리하는 것과 동일한 결과가 나오는 연산자이다.

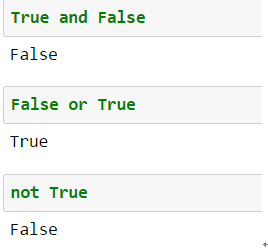
|  |  |
| --- | --- |
| 연산자 | 설명 |
| x<<n | 왼쪽으로 이동 (숫자 x에 2를 n번 곱한 것과 동등함) |
| x>>n | 오른쪽으로 이동 (숫자 x를 2로 n번 나눈 것과 동등함) |



### 2.2.5. 논리 연산 및 단축연산

* 논리 연산처리

|  |  |
| --- | --- |
| 조건부 | 논리 |
| and | x and y 평가에 있어서, x가 거짓으로 판명되면 그것을 반환하고, 그렇지 않은 경우에는 y를 평가하여 결과값을 반환한다 |
| or | x or y 평가에 있어서, x가 참으로 판명되면 그것을 반환하고, 그렇지 않은 경우에는 y를 평가하여 결과값을 반환 |
| not | not x 평가에 있어서, x가 거짓이라는 것은 x의 반대를 의미함 |

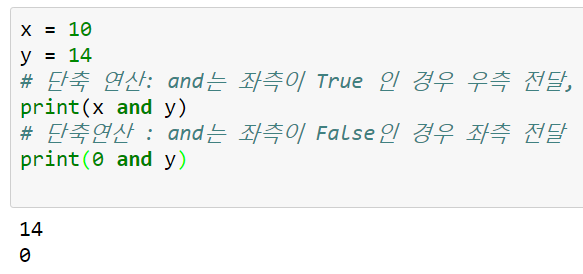


* 축약연산처리

표현식에 대한 축약은 and와 or 연산으로 처리가 가능한다.

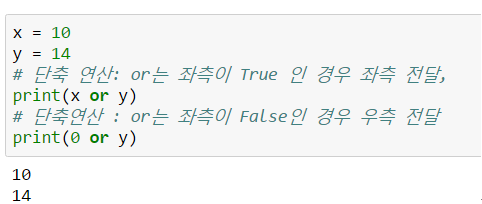
* and 축약연산

and 연산은 첫번째가 True 일 경우는 and 우측 편을 리턴하고 첫번째가 False이면 and의 좌측 편을 리턴한다.



* Or 축약 연산

첫번째가 True 일 경우는 Or 좌측 편을 리턴처리하고 첫번째가 False이면 or의 우측 편을 리턴한다.



**3. 파이썬 문장 이해하기**

파이썬을 실제 실행하고 제어하도록 하는 것은 문장을 구성해야 하는 것이다.순환과 제어 등 다양한 문장 작성법을 이해하고 이를 모듈화하는 법까지 이번장에서 알아본다.

## 3.1. 추가 정보에 대한 문장 구성

### 3.1.1. 줄 들여쓰기 (Lines and Indentation)

파이썬 프로그램 시 첫번째 주의 사항은 코드 블록에 대한 세부적인 기호가 시작할 때만 있다.

클래스와 함수, 제어문과 순환문 등의 블록처리는 콜론( : )이고 이 다음의 블록을 인지하려면 줄 들여쓰기를 사용해서 구분한다.

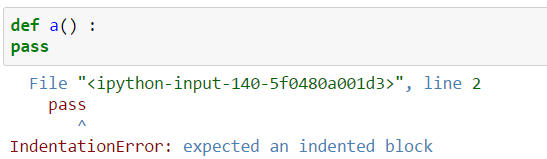
{ }, ( ) ,[ ] 기호로 시작할 경우 여러 줄로 작성되어도 하나의 문장으로 인식하기에 들여쓰기가 인식되지 않으며 줄 연결 기호( 역슬래쉬 \)를 사용해도 아직 한 문장이 끝나지 않았으므로 들여쓰기가 인지되지 않는다.

들여쓰기는 공백(Space), 탭(Tab) 모두 가능하지만 단, 들여쓰기 공백 수는 블럭내에 동일 수로 해야 한다.

* 문장의 블록 만들기 : 4칸 들여쓰기

: 는 블럭구문을 하는 표시이므로 반드시 이후에 작성된 문장은 4칸을 들여쓰기 해야 함

* + 4칸 들여쓰기 안하면 IndentationError 발생

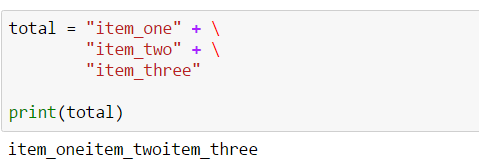


### 3.1.2. 멀티 라인 문 (Multi-Line Statements)

파이썬은 일반적으로 새로운 줄이 되면 끝난다.그러나 줄 연속 문자(\)의 사용으로 줄을 계속 사용할 수 있다.

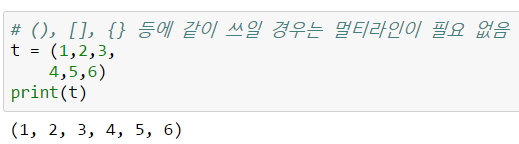
* 역슬래쉬 사용해서 문장 연결하기

표션식을 사용할 경우 여러 문장을 하나의 문장으로 표시할 때 역슬래쉬를 이용해서 하나의 문장으로 만든다



* ( ), [ ], { } 일 경우 역슬래쉬 없이 문장이 연결

이 기호가 사용되면 닫힌 표시가 있기전까지는 하나의 문장이 끝난 것으로 보지 않기에 닫혀야 문장으로 인식한다.

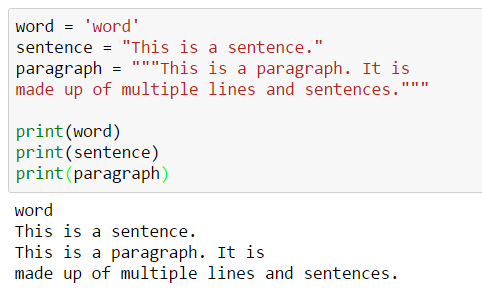


### 3.1.3. 인용 (Quotation):

ingle('), double("), triple(''' or """) 를 사용하여 문자열을 감싸서 사용한다.

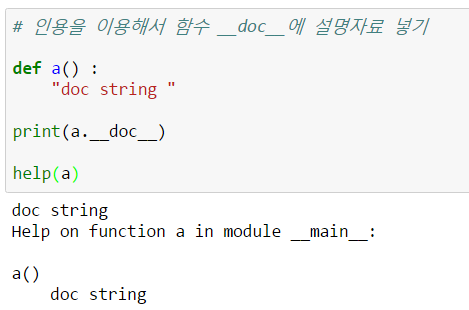
triple 따음표는 여러 줄에 걸쳐 문자열을 사용할 수 있다.

* 인용 예시



* 함수 설명 달기

함수나 클래스 정의 후 첫문장에 문자열로 설명하면 \_\_doc\_\_ 자동으로 처리된다.



### 3.1.4. 주석 (Comments):

기호 (#)를 사용해 주석을 시작하며 기호 (#) 후 물리적 라인 끝까지 모든 문자를 주석으로 간주한다



## 3.2. 프로그램 문장

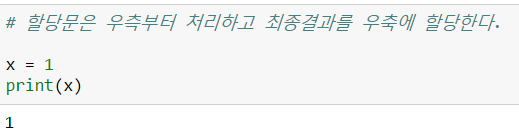
### 3.2.1. 할당문장 이해하기

파이썬에서 할당문이 중요한 이유는 리터럴과 표현식이 결과를 변수에 할당할 경우 그 변수가 이전에 정의가 없으면 처음으로 변수가 초기화 된 것으로 본다.

변수가 할당없이 할당문 우측에 사용되면 상위 namespace를 확인하고 없으면 오류 처리하므로 변수 사용이 필요한 경우 반드시 초기값이라도 할당이 먼저 있어야 한다.

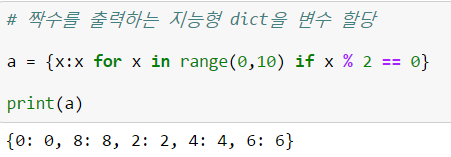
* 변수에 리터럴 할당

직접 값을 변수에 할당하게 표현이 되었지만 실제는 변수에 인스턴스 레퍼런스만 할당해서 이 인스턴스 객체를 찾을 수 있는 참조만 할당한다.



* 변수에 지능형으로 리터럴 할당

지능형 타입이 실행되는 순간 dict 타입이 요소가 생겨서 이를 처리할 수 있는 참조만 할당된다.



### 3.2.2. Import 구문

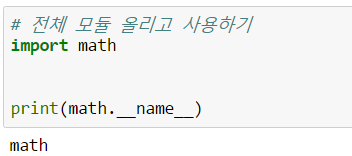
현재 작생되는 모듈에 타 모듈을 이용하기 위해 그 모듈을 가져다 쓰는 방식이 import 구문이다. import문 모듈과 내부 모듈에 작성되는 변수 등이 이름이 동일하면 사용할 수 없을 수도 있다.

* 전체 모듈을 올리기 : import 모듈명
* 모듈 내에 일부 요소만 올리기 : from 모듈명 import 요소명
* 이름이 충돌을 막기 위해 별칭 사용 as 별칭



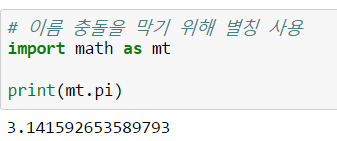
* 모듈 전체 import

모듈 전체를 import 할 경우에는 import 구문만 사용한다. 내부 모듈이 속성은 닷연산자이 다음에 내부에 포함된 요소를 붙여서 사용한다.



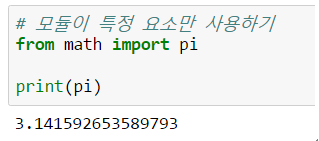
* 모듈 전체 import하고 별칭 사용하기

모듈 이름이 길어지면 이를 줄여서 별칭으로 사용할 수 있다.



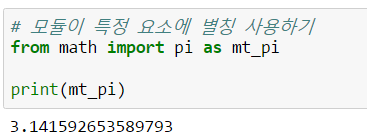
* From import로 모듈 내에 특정 속성만 사용하기

특정 모듈이나 패키지에 전부를 import 하는 것이 아니라 특정 요소만 사용이 필요한 경우에 필요한 요소만 지정해서 import 처리한다.



* From import로 모듈 내에 특정 속성에 별칭 부여하기

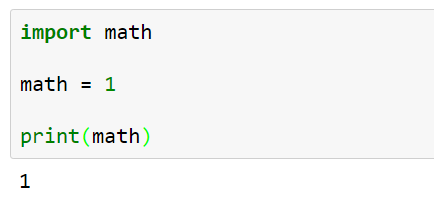
Import 되는 요소에 대해 as 키워드를 사용해서 별칭을 부여할 수 있다. 이름이 충돌이 나지않는 이름을 부여한다.



* 중첩된 명명이면 나중에 정의된 것으로 변경

파이썬은 모든 것을 변수로 생각해서 namespace에 보관하므로 모듈명, 함수, 클래스, 변수 명등이 동일하지 않도록 해야 한다.

모듈을 import 했는데 변수를 모듈과 동일한 이름으로 정의 시 내부 모듈에서 최종으로 정의된 변수로 인지해서 처리된다.

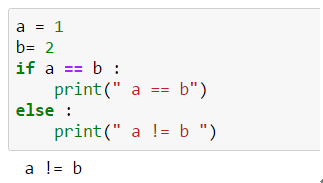


### 3.2.3. 단순 제어문

제어문은 표현식이 조건에 따라 if 구문에는 True와 else 구문에는 False 값에 따라 처리하는 법

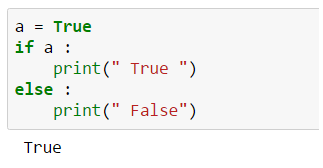
* 비교연산으로 처리

If/else 문을 이용해서 조건식에 참과 거짓에 따라 if문이나 else 문에 정해진 로직을 처리한다.



* True/False로 처리

if문은 True만 처리하므로 True 값이 전달되면 if 블록만 처리된다.

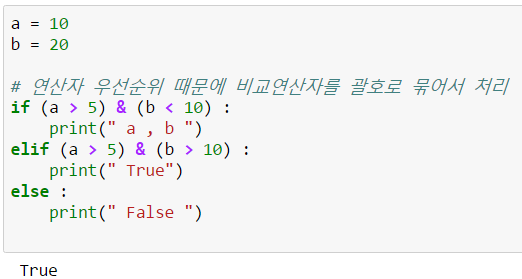


### 3.2.4. 복합 제어문

제어문은 표현식이 조건에 따라 if 구문 True와 elif 구문 True, Else 구문에는 False 값에 따라 처리한다.

* 복합 제어문 elif

다양한 조건식을 사용하고 싶을 경우 elif문에 조건을 추가해서 로직을 넣는다. 해당 조건이 만족할 경우 elif 블록 내의 로직을 처리한다.



* 스위치 케이스 문 처리 방법

파이선에는 스위치문이 없어서 대안으로 dict 타입에 함수를 정의해서 switch문의 특징 하나의 case만 실행하도록 처리할 수 있다.



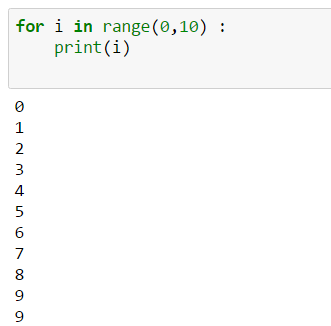
### 3.2.5 순환문 처리 : for

파이썬 for문은 다른 언어와 차이점은 반복형(iterable)/반복자(iterator)를 처리하는 것이다.

for문에 반복형이 들어오면 반복자를 생성시켜서 원소 하나씩 순환해서 처리하도록 해준다. 반복형이나 반복자가 원소가 다 처리되면 StopIteration 예외 처리를 하지만 for문 내에서는 이를 마지막으로 인식하기 때문에 예외 없이 처리가 가능하다.

문자열, 리스트, 튜플, dict, 파일 등 반복자나 반복형 등의 타입이 오면 전부 처리가 가능하다

* 반복형 range로 처리

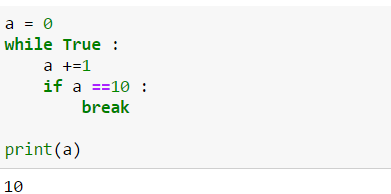


### 3.2.6. 순환문 처리 : while

while 문은 조건식이 만족할 때까지 처리가 된다.

* True로 무한순환

while문은 조건이 참일 경우에만 내부 블록이 실행되므로 while의 조건을 True 지정하면 무한 순환이 되어 처리하므로 내부에 종료할 수 있는 로직을 처리한다.



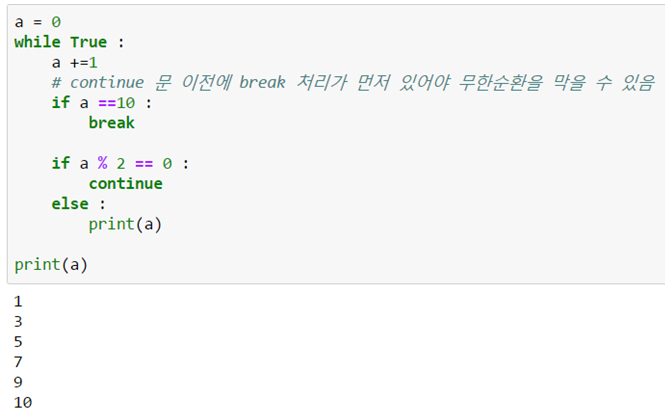
### 3.2.7 continue/break

continue : 순환 처리는 해야하지만 특정 로직을 처리하지 않을 경우 순환문으로 돌아감

break : 순환처리시 순환을 종료하는 경우 사용

* 순환문에서 continue와 break 처리 예시

While 문으로 무한 순환을 시키고 내부에서 특정 조건을 부여해서 로직을 스킵할지 계속할지를 처리하도록한다.



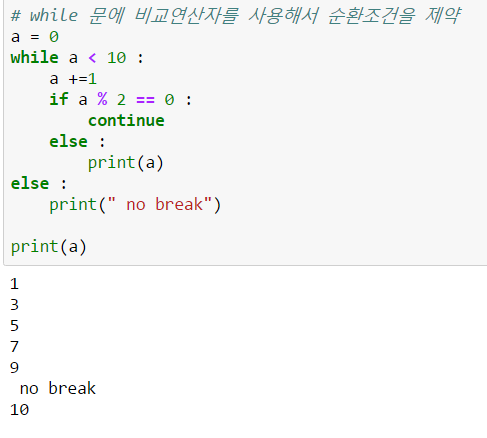
### 3.2.8. 순환문에 else 문 추가하기

순환문이 break 처리 없이 수행이 되었는지를 확인하기 위해 마지막에 else문을 추가 할 수 있다.

* 순환문에서 break 없을 경우 예시

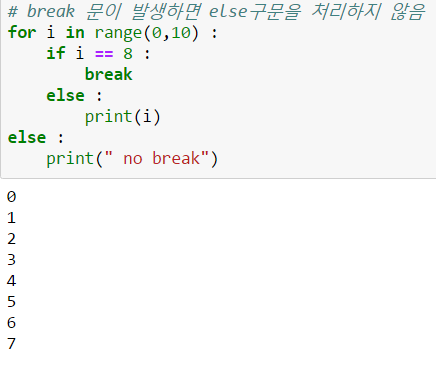
순환문에 else 문을 첨가하면 순환 작업이 break 없이 처리되면 그 다음 로직을 처리할 수 있는 부문을 이 블록 내부에 정의해서 실행하도록 처리한다.

* 정상 처리하는 경우 else구문이 실행



* 순환문에서 break 발생할 경우

순환문에서 break가 발생하면 else문은 처리되지 않고 종료한다.

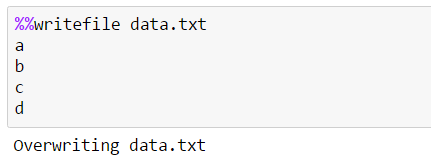


### 3.2.9. context 제한문 : with

with는 블록단위의 프로세스의 시작과 끝에 대한 처리를 해준다. context manager에 의해서 실행되는 \_\_enter\_\_()과 \_\_exit\_\_()을 정의하여, with 구문 body 의 앞부분과 뒷부분에 실행되는 코드를 대신할 수 있다.

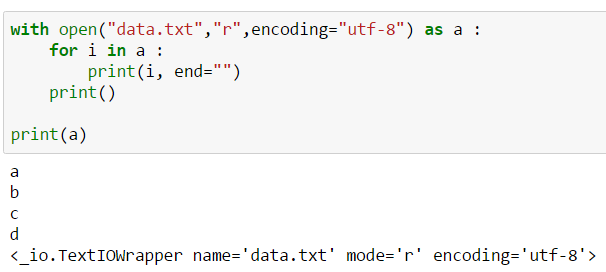
* 파일 생성

주피터 노트북에 cell에서 %% 기호와 writefile 명령어를 쓰고 파일이름을 적고 다음칸부터 파일에 들어갈 값을 넣고 실행(shift+enter)하면 파일이 저장된다.



* 파일 읽기

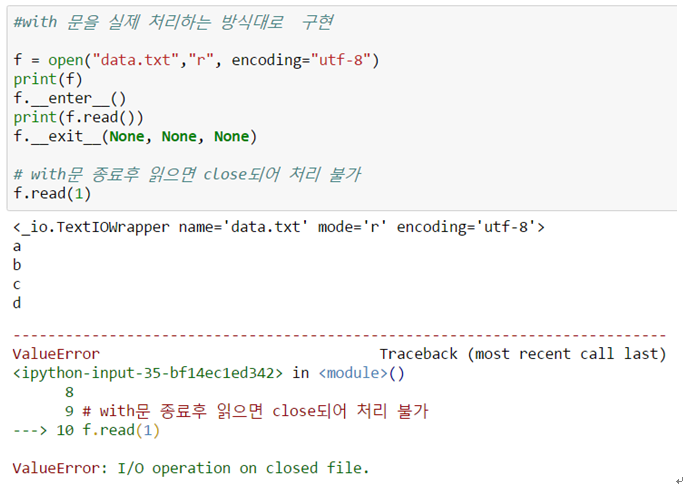
컨텍스트 구문에 open 함수를 이용해서 파일을 읽으면 as 구문 다음의 변수에 파일에 대한 인스턴스 객체를 넘겨 내부에서 파일을 읽고 오게 만든다.



* context를 내부 메소드로 처리

with 구문 내에 \_\_enter\_\_와 \_\_exit\_\_ 스페셜 메소드가 있고, File은 context를처리할 수 있어 메소드가 정의 되어있어 파일처리를 \_\_enter\_\_, \_\_exit\_\_ 으로 처리하므로 with 구문과 동일한 처리를 할 수 있다.

파일에서 \_\_exit\_\_ 메소드를 호출하면 파일이 종료되므로 다시 파일을 읽으면 오류가 발생한다.



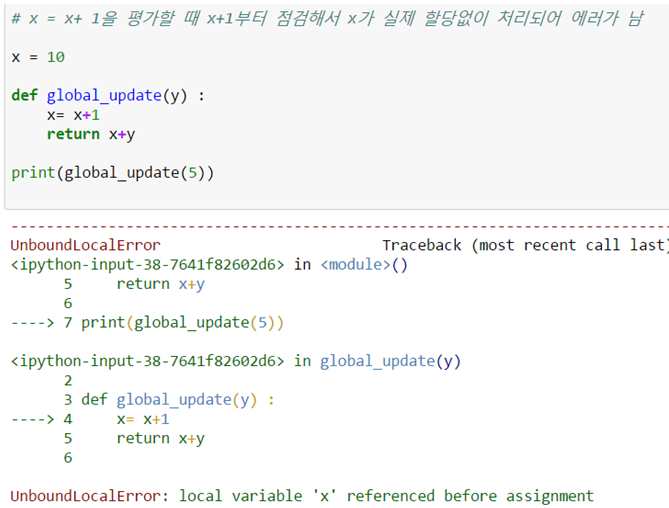
### 3.2.10. global 문 과 nonlocal 문 처리

Namespace 별로 변수가 관리되므로 특정 영역을 참조할 경우에 선언하는 문장이다. global 문은 주로 함수 영역에서 모듈의 글로벌 영역의 변수를 참조해서 갱신이 필요할 경우 정의해서 사용한다.

nonlocal문은 주로 함수 내에 함수를 정의할 경우 내부 함수에서 외부 함수에 있는 변수를 갱신할 경우 정의해서 사용한다.

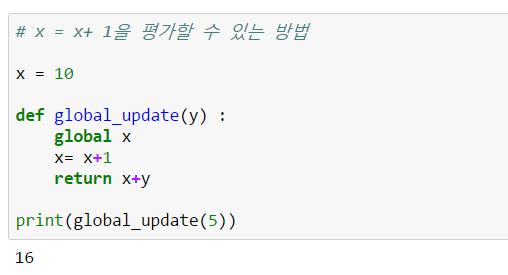
* 함수내에서 지정되지 않는 변수변경

파이썬에서는 변수 정의는 변수에 값을 할당하는 것이다. 표현식에 사용되는변수가 처음으로 사용되어지면서 미리 할당을 하지 않으면 그 변수를 글로벌 영역에서 검색하고 없으면 에러가 나온다.



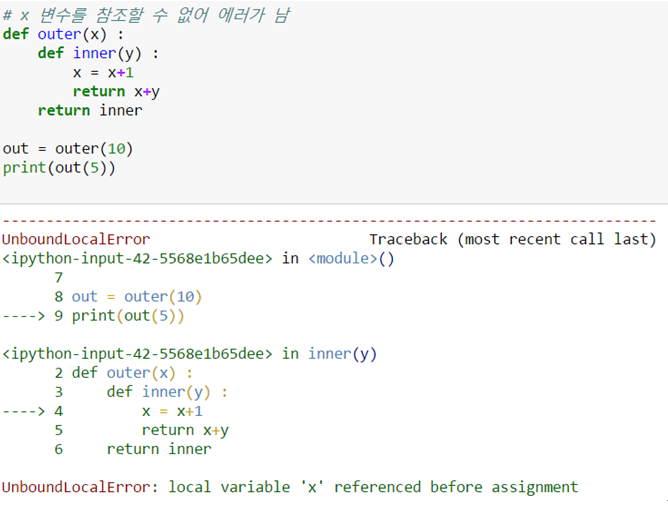
* 키워드 global를 사용

함수 내에 변수를 할당하면 로컬 변수로 인식되므로 글로벌 변수를 처리하지않는다. 갱신 불가한 글로벌 변수를 함수 내부에서 갱신하고 싶을 경우는 명시적으로 global 키워드로 변수를 정의하고 갱신해야 한다.



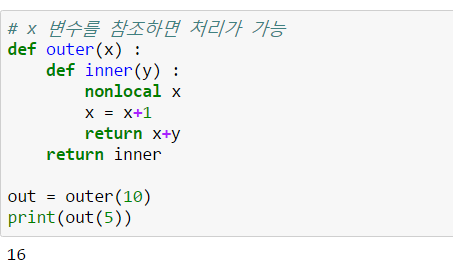
* 내부함수에서도 변수 할당 없이 갱신하는 경우

함수 내부에 함수를 정의할 경우에도 로컬 변수가 별도로 생성되므로 변수 할당 없이 사용하면 변수가 정의가 되지않고 사용하는 것이므로 예외가 발생한다. 예외 발생을 해결하려면 변수를 할당해서 정의한 후에 사용해야 한다.



* 외부 함수의 변수를 내부함수에서 갱신

외부 함수의 변수를 nonlocal로 정의하면 외부함수의 변경불가능한 변수를 갱신할 수 있다. X라는 변수가 외부 함수에 있는 변수를 명기하는 것이므로 이를 참조해서 갱신을 한다.

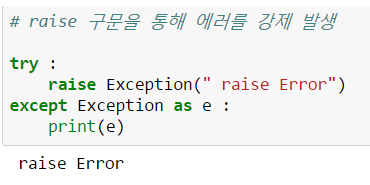


### 3.2.11 try,except 문 처리

에러가 발생할 경우 처리하는 구문으로 에러가 발생할 수 있는 문장을 try 구문 다음에 넣고 에러가 발생안 예외처리를 except 항목으로 추가한다.

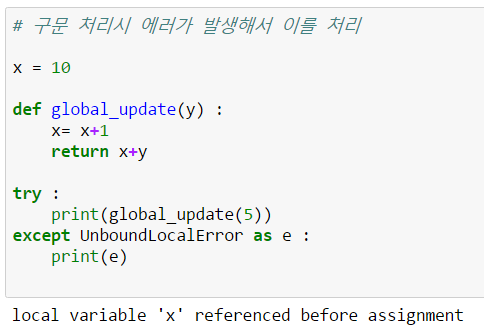
* raise예외 발생

예외를 강제로 발생시키고 except 구문에서 예외를 잡고 에러처리한다.



* 예외 발생

함수에서 예외가 발생한 것을 except 구문에서 처리한다.



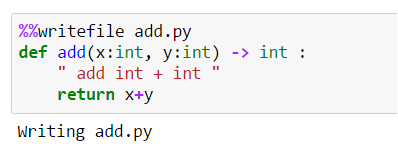
## 3.3 모듈/패키지 처리

### 3.3.1 모듈 생성 및 실행하기

* 모듈을 생성하고 import 후에 메인 모듈에서 처리하기

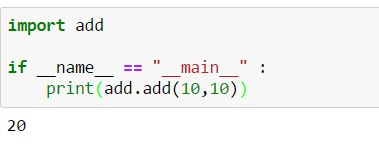
파이썬에는 main 함수가 제공하지 않는다 대신 module이 \_\_name\_\_ 관리 규칙이 현재 실행되는 모듈일 경우는 “\_\_main\_\_” 으로 관리하므로 이 기준으로 실행을 확인할 수 있다.

* 함수를 정의해서 파일로 저장한다

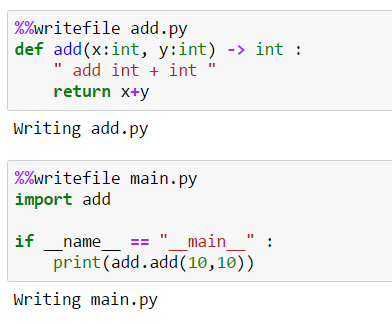


* 파일을 import 해서 처리

현재 실행하는 모듈을 python shell로 실행시키면 그 내부 모듈이 이름이 \_\_main\_\_으로 변경되므로 이 구문을 만날 경우에 함수 호출이 실행된다.



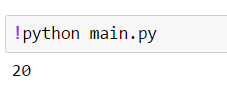
* 모듈을 생성하고 import 후에 메인 모듈에서 처리하기
  + 두개의 모듈을 생성



* 쉘에서 실행

쉘에서 main.py 을 실행하면 이 모듈이 \_\_name\_\_이 값이 “\_\_main\_\_”으로 처리되므로 if문 내부의 로직을 처리한다

Jupyter notebook에서 !(느낌표)쉘명령을 사용하면 쉘명령이 실행된다.



### 3.3.2. 모듈 네임스페이스 확인하기

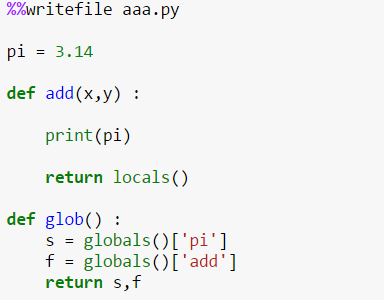
모듈과 함수에 대한 namespace를 dict 타입으로 관리한다. 내장함수 dir를 사용하지 않고 globals함수와 locals 함수를 사용해서 조회하면 내부의 dict를 확인할 수 있다.

* 모듈을 생성

함수를 2개 만들어서 글로벌과 로컬에 대한 namespace 영역을 출력하도록 만든다.

* 함수 정의 모듈 작성

함수 globals를 이용해서 모듈의 네임스페이스를 내의 변수들을 확인하는 함수를 작성하고 그 값을 리턴한다.



* 모듈을 import하고 모듈별 global namespace 확인

다른 모듈 aaa를 import해보면 pi라는 변수 처리 결과는 aaa 모듈 내의 값을 처리한다.

현재의 모듈의 pi는 지정되었지만 함수가 만들어진 aaa 모듈이 아니라서 참조가 되지 않는다. 이 예제를 보면 모듈 별로 namespace가 별도로 만들어지는 것을 확인할 수 있다.

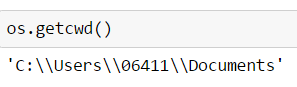




### 3.3.3 패키지 처리

* test 패키지 생성하기
  + 현재 디렉토리를 확인

Import os를 한 후에 현재 디렉토리를 확인한다



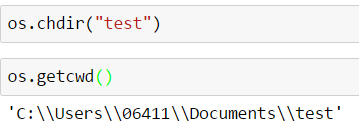
* + 패키지 만들기 : test

Makedirs를 이용해서 현재 디렉토리에 test를 만든다.



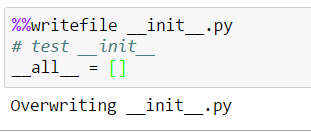
* + 디렉토리 이동하기

아래의 명령을 가지고 디렉토리를 이동하고 이동된 디렉토리를 불러와서 확인해 본다.



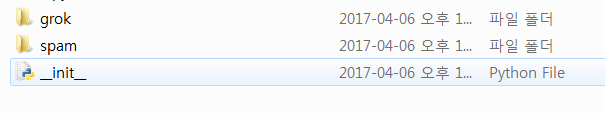
* + \_\_init\_\_ 파일 및 내부 패키지 만들기

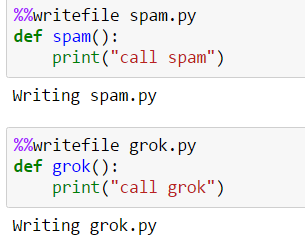
파이썬은 디렉토리도 객체로 관리하므로 반드시 \_\_init\_\_.py를 만들어야 한다.



* + 내부 패키지 내의 함수정의 후 모듈만들기

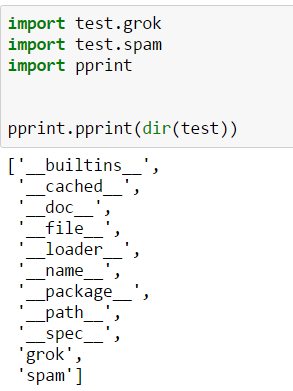
내부 패키지 내에 각각 파이썬 모듈과 \_\_init\_\_.py 를 만들어야 한다. 파이썬에서는 패키지에 \_\_inin\_\_.py가 없으면 파이썬 패키지로 인식하지 않는다.





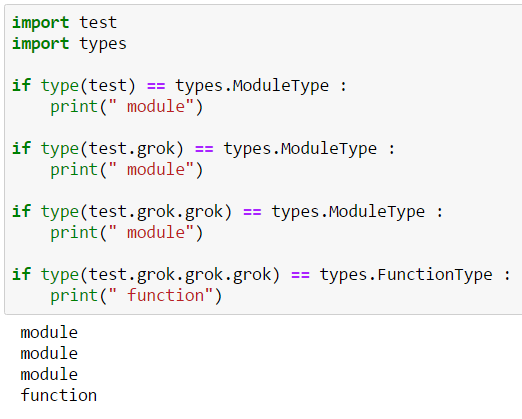
* test 패키지 호출
  + package를 import 해서 내부 속성을 확인

파이썬은 패키지도 하나의 객체로 인지하므로 기본으로 갖는 속성들이 정보를 확인할 수 있다.



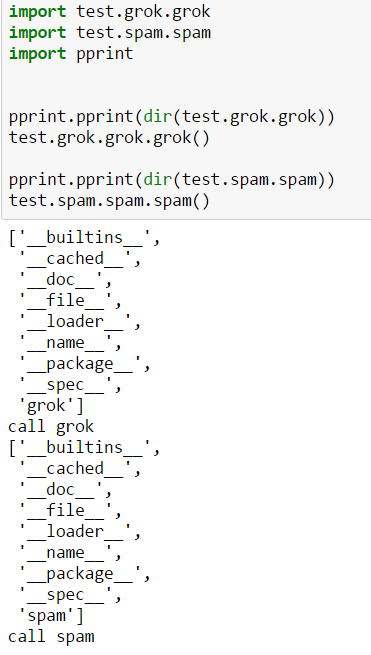
* + 내부 타입 체크

모듈 types는 파이썬 인터프리터에서 사용하는 데이터 타입을 가지고 있다이 모듈을 이용해서 실제 파이선 패키지, 모듈, 함수들 만들어진 타입을 비교해 본다.



* + import로 호출하기

패키지 test 내부에 패키지 spam 밑에 spam 모듈 내에 spam 함수를 호출해서 처리한다.



**4. 파이썬 Sequence데이터 타입**

파이썬 숫자 타입은 주로 원자타입이지만 Sequence 타입인 문자열, 리스트, 튜플 등은 원소가 여러 개를 가질 수 있는 분자타입이다.

다른 표현으로는 컨테이너 데이터 타입이며 왜 여러 개의 원소가 들어가서 생기고 이 각 원소들을 어떻게 검색하고 갱신하는지에 대해 알아보기로 하겠다.

## 4.1. 파이썬 Sequence데이터 타입의 특징

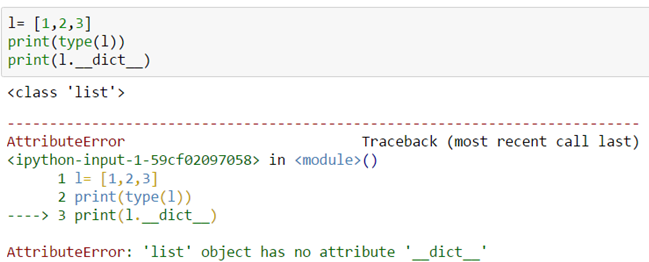
### 4.1.1. 런타임에 속성 추가 여부

내장 데이터 타입은 실제 런타임에 내부 속성은 조회가 가능하나 추가는 할 수 없다. \_\_dict\_\_ 속성을 제공하지 않는다.

* 파이썬 객체 namespace(\_\_dict\_\_) 조회

내부 데이터 타입들은 런타임에 속성 추가가 없어서 내부에 \_\_dict\_\_ 속성이 없다. 내부속성을 조회하려고 \_\_dict\_\_으로 조회하면 에러가 나므로 dir 함수를 이용해서 접근을 해야 한다.

예제는 리스트를 만들어서 내부 속성인 \_\_dict\_\_를 조회하면 실제 속성이 없어서 AttributeError가 나온다.



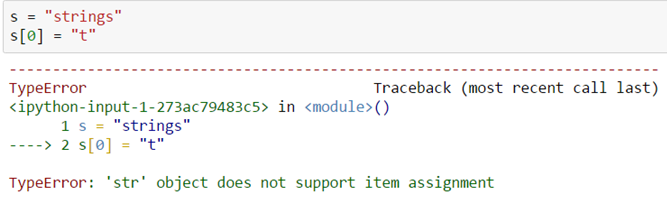
### 4.1.1. 변경가능 여부 : Mutable & Immutable

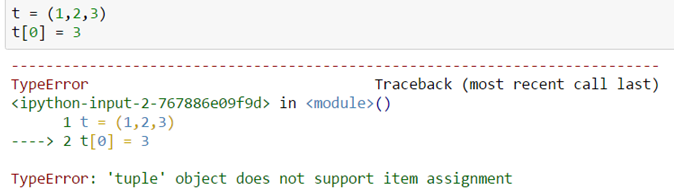
변경가능(Mutable) 하다는 말은 객체 내부의 원소들을 추가, 삭제, 변경 등을 처리할 수 있다는 것이니 객체 자신을 바꾸는 것은 아니다.

* 변경불가 데이터 타입( Immutable )
  + 문자열, 튜플, 바이트는 변경불가

변경불가하다는 뜻은 내부 원소의 값을 변경할 수 없다는 것이다. 실제 이 클래스의 메소드들을 보면 변경해서 리턴결과를 보내는데 실제 기존 인스턴스를 바꾸는 것이 아니라 별도의 인스턴스를 만들어서 새로운 것을 보내는 것이다.

예제는 문자열이나 튜플을 생성해서 문자열이나 튜플의 첫번째 위치를 변경하려고 하면 문자열이나 튜플은 변경이 불가해서 TypeError가 나온다.

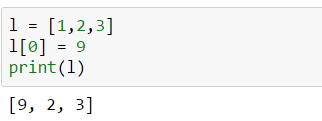




* 변경가능 데이터 타입(Mutable )

리스트 바이트어레이 등이 데이터 타입은 기존 원소들을 삭제 및 추가 등이 가능해서 변경이 가능하다. 배열을 처리하는 모듈 array.array도 동일하게 원소들 추가 및 삭제 등이 가능하다.

예제는 리스트를 생성해서 리스트 첫번째 위치를 변경해서 출력하면 첫번째 위치 원소가 변경된 것을 확인할 수 있다.



### 4.1.3. 컬렉션 여부

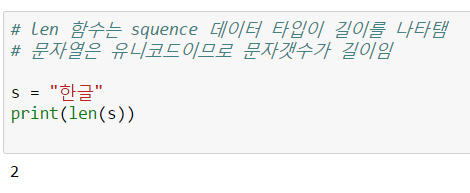
컬렉션이란 다양한 원소를 가진 데이터 구조를 말한다. Sequence 데이터 타입은 여러 원소들로 구성되므로 기본 컬렉션 이다.

컬렉션 여부는 원소개수,포함관계, 반복가능형 여부를 확인할 수 있는지를 확인할 수 있으면 된다.

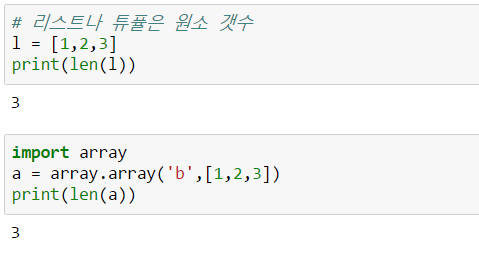
* 데이터타입 내의 원소 개수 확인 : len 함수

파이썬 Sequence 데이터 타입은 내부에 여러 원소를 가지므로 그 데이터에 대한 원소의 개수를 len함수로 확인할 수 있다.

예제는 한글로 작성된 문자열의 길이를 확인하는 것이다. 유니코드 문자를 기준으로 표시해서 두 글자의 길이를 출력한다.



예제는 리스트나 array.array 즉 배열들의 길이는 가지고 있는 원소의 수와 동일한 것을 알 수 있다.



* 데이터타입의 원소를 반복해서 읽을 수 있음 : iter 함수

Sequence 데이터 타입들은 내부에 원소들이 없거나 연속적으로 들어있어 반복해서 원소를 읽을 수 있어 반복형(Iterable)로 처리가 가능하다.

예제는 함수 iter로 Sequence 타입을 변환하면 반복자(Iterator)로 변환된 것을 알수 있다.

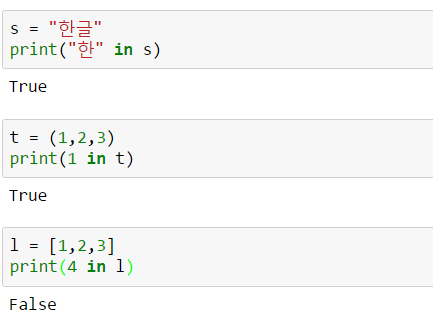
함수 iter의 결과를 가지고 for 문이나 list 등 생성자, 지능형 등에 사용하면 자동으로 반복처리가 되어진다.



* Sequence 데이터 타입 내에 원소 포함 확인 : in 연산자

Sequence 데이터 타입 내의 원소와 일치 여부를 확인이 가능한지를 in 연산자를 통해 처리한다

예제는 Sequence 타입으로 생성해서 그 내부의 원소가 포함되어있는지를 확인하고 처리결과는 True, False로 표시된다.

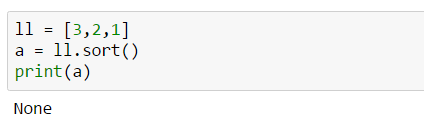


### 4.1.4. sequence 타입 내의 메소드 처리 기준

Mutable 타입 과 Immutable 타입 내의 메소드 처리 결과가 약간씩 상이한 이유는 그 기능으로 자기 자신이 객체 원소를 고칠지 아니면 새로운 객체를 만들지에 따른 처리가 다르기 때문이다.

* 변경 가능할 경우 메소드 처리

메소드 대부분은 자기 객체 내를 변경하므로 메소드 처리 결과값을 None으로 처리한다.



* 변경불가능한 경우는 별도의 객체를 만들어서 리턴 처리

메소드는 자기 객체를 변경할 수 없으므로 메소드 처리 결과값으로 새로운 Immutable 객체를 만들고 리턴한다.



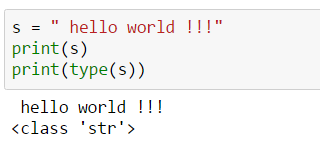
## 4.2. 문자열 데이터 타입

파이썬 3 버전은 문자열이 기본 unicode로 처리 되며 문자열은 한번 생기면 그 안의 내용을 변경할 수 없다. 단, 새로운 문자열이 만들어져 다시 변수에 재할당되도록 처리할 수는 있다.

### 4.2.1. 문자열 생성 방법

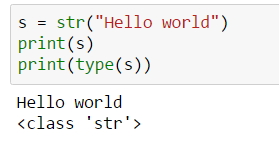
* 직접 리터럴 문자열로 지정

문자열 리터럴를 만들면 파이썬은 자동으로 str 클래스의 인스턴스로 만든 후에 변수에 레퍼런스를 할당한다.



* str('문자열) 생성

str 클래스로 생성할 때 인자로 문자열을 지정해서 처리해도 인스턴스를 생성하고 그 인스턴스의 레퍼런스를 변수에 할당한다.

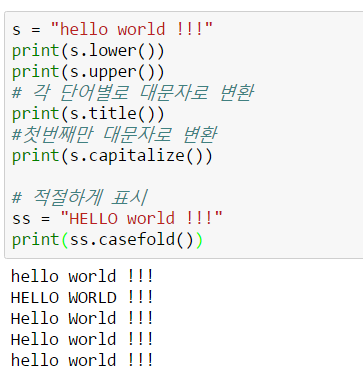


### 4.2.2. 문자열 주요 메소드

문자열은 생성되면 원소의 값을 변경할 수 없으므로 대부분 메소드는 새로운 문자열을 생성해서 결과를 리턴한다.

* 문자열 대소문자 처리 메소드

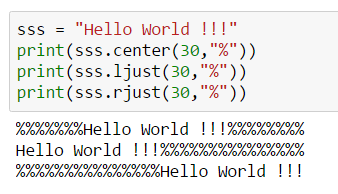
문자열 내의 모든 것을 대문자(upper) 소문자(lower)로 변경이 가능하고 제목형태(title), 첫 단어만 대문자(capitalize), 대소문자 변환처리(casefold) 등이 메소드로 문자열을 변경해서 새로운 문자열 객체를 만들어낸다.



* 문자열 위치 조정해서 꾸미기

문자열에 대한 크기를 정하고 빈공칸에 들어갈 값을 정하면 문자열이 위치를 조정하고 빈 공간에 새로운 값들이 들어가도록 처리한다.

기본 값은 공백문자이므로 크기만 지정하면 위치만 변경된다.

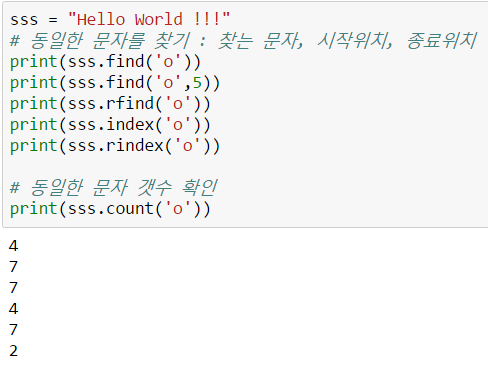


* 문자열 찾기

메소드 find는 문자나 부분 문자열을 찾는 메소드로 처음 매칭되는 것을 찾으면 그 인덱스를 알려준다.

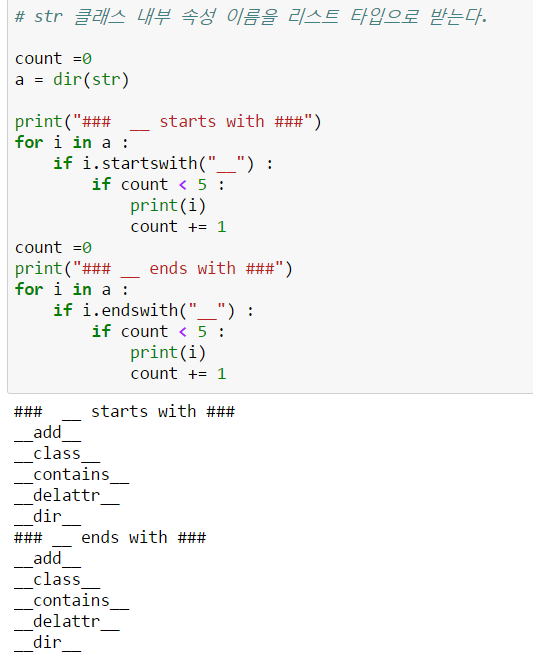
메소드 index는 동일한 문자가 어느 index에 위치한지를 확인한다.

메소드 count는 주어진 문자나 부분문자열이 이 문자열에 몇 개가 있는지를 확인해 준다.



* 문자열 시작점과 끝점 찾아서 문자열 패턴 매칭하기

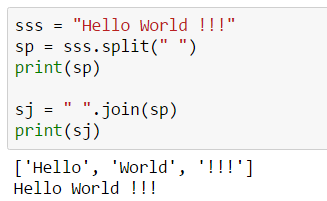
처음과 마지막에 구성된 부분문자열을 체크해서 있으면 True, 없으면 False를 제공하는 메소드들이다.



* 문자열을 분리 및 문자열 결합

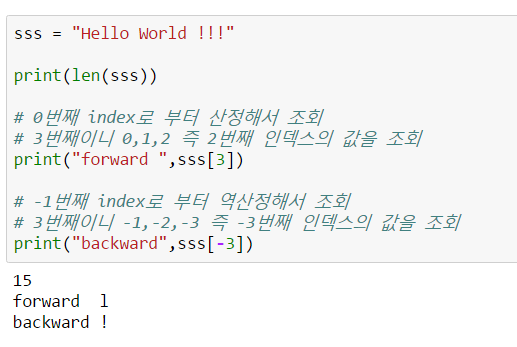
메소드 split은 문자열을 분리하려면 문자열 분리자를 인자로 넣어서 분리자별로 리스트의 원소로 구성되도록 한다.

메소드 join은 분리자를 문자열로 정하고 리스트를 인자로 받아 리스트의 원소를 분리자별로 나눠 문자열로 변경한다.



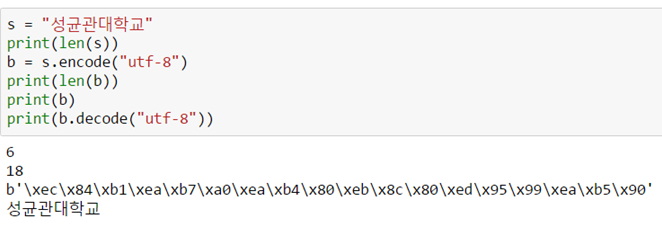
* 문자열 길이 및 검색

Len 함수로 문자열의 길이를 확인한다.



* unicode 에서 bytes로 전환하기

유니코드는 다양한 문자를 한 바이트에서 여러 바이트로 관리한다. 바이트로 전환하면 한글인 경우 대부분 문자당 2바이트로 전환된다.



## 4.3 바이트 데이터 타입

문자열은 유니코드를 처리하고 바이트 데이터 타입은 bytes 단위의 문자열을 처리하도록 되어있어 거의 동일한 메소드를 제공한다.

### 4.3.1. 바이트 생성

* 리터럴과 bytes 생성자로 생성

ASCII 타입일 경우는 영문으로 표시되지만 그외 타입은 encoding 처리 후에 hex값으로 표시된다.



* 문자열을 바이트로 전환 후에 확인하기

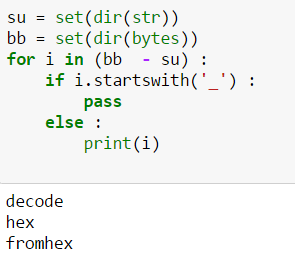
문자열에 encode 메소드로 바이트 타입으로 전환할 수 있다. 유니코드는 문자 길이를 처리시 문자 단위로 표시하지만 바이트로 전환하면 길이가 바이트 단위로 바뀐다.



### 4.3.2 바이트 주요 메소드

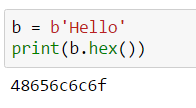
* 문자열과 다른 메소드들

문자열 데이터 타입과 다른 메소드를 확인하기 위해 내부 속성의 이름으로 집합을 만들어서 2개의 집합의 차집합을 이용해서 바이트 타입에만 있는 3개의 메소드를 확인이 가능하다.



* 바이트 타입에서 hex 메소드 처리

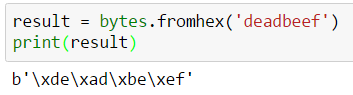
바이트 단위의 값에서 hex값으로만 표시하기 위해서는 메소드 hex로 변경을 해야한다.



* 바이트 타입 에서 fromhex 메소드

hex값으로 들어온 것을 바이트 타입의 hex로 전환한다.

일반 hex와 다른 것은 역슬래쉬와 x가 붙여있는 값으로 표시한다.



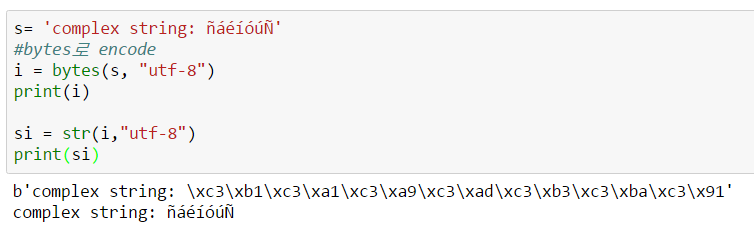
* 바이트 타입 에서 문자열로 전환

한글 등을 바이트 타입에서 사용하려면 문자열에서는 encode가 되어 바이트 타입으로 전환되어야 하고 바이트 타입을 문자열로 전환하려면 decode 메소드를 사용해서 전환해야 한다.



* bytes/str 생성자에서 직접 encode, decode 하기

클래스에서 직접 인스턴스를 만들면서 encoding하면 encode/decode 메소드를 사용하지 않는다.



## 4.4. bytearray 데이터 타입

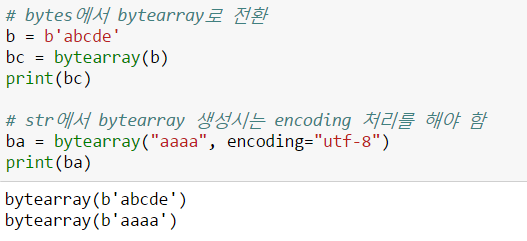
문자열과 거의 동일한 메소드들을 제공하지만 내부 원소에 대한 변경이 가능하므로 리스트에 필요한 메소드가 추가된다.

### 4.4.1. bytearray 생성

* Bytearray 생성

클래스 byte는 변경이 불가능하므로 변경가능한 bytearray를 제공한다. 바이트 단위로 처리해야 하므로 문자열을 가지고 변경등을 사용하려면 encoding 처리가 되어야 한다.

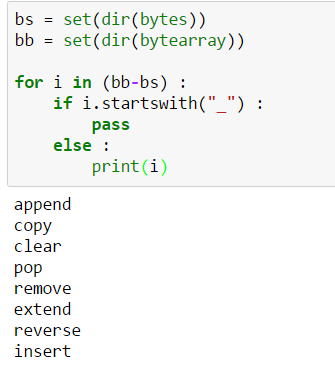
이 예시는 바이트로 지정된 것을 바이트 어레이로 변환하거나 문자열은 직접 enconding 처리한다.



### 4.4..2. bytearray 내의 메소드

* Bytes 타입과 메소드 비교

변경가능한 bytearray 타입은 list에 가지고 있는 메소드들이 있어 삭제, 추가, 변경 등이 가능하다



* 문자열을 bytearray 로 전환

문자열을 바이트 어레이에 대한 encode 처리하면 한글은 hex 값으로 전환되어 표시되고 다시 decode 처리하면 유니코드 문자열로 변환된다.



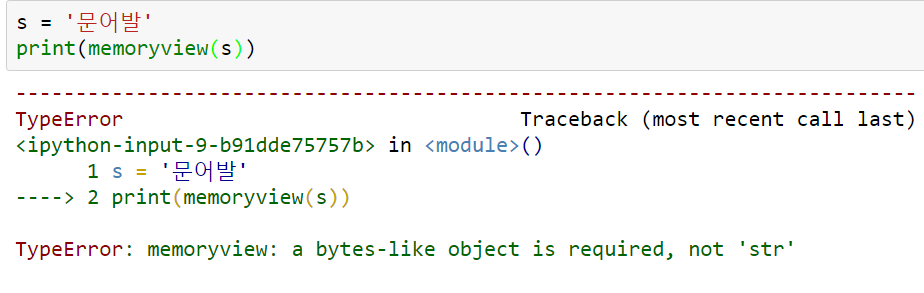
## 4.5. memoryview 데이터 타입

메모리에 있는 것을 직접 참조할 수 있는 데이터 타입을 만들어 준다. 동일한 메모리를 참조할 수 있는 데이터 타입이다.

### 4.51. memoryview 생성

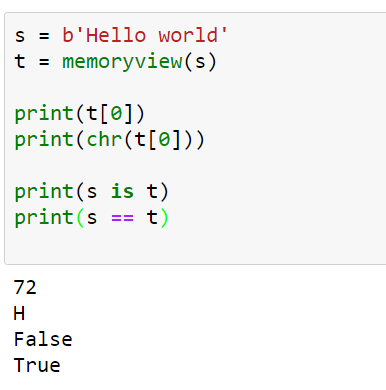
* 바이트 타입일 경우만 처리

유니코드를 바이트로 데이터 타입을 전환해야 memoryview를 사용할 수 있다.



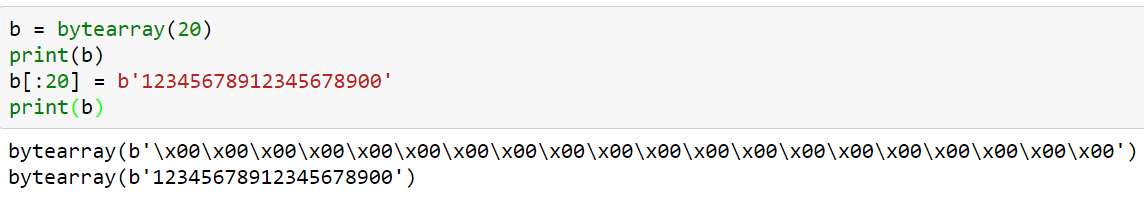
* 동일한 것을 참조해서 조회하기

문자열 등 변하지 않는 값을 동일한 참조를 유지하기 위해 memoryview로 처리한다. 두 개의 객체는 다르지만 실제 참조하는 곳은 동일하다.



* 버퍼처리하기 : bytearray

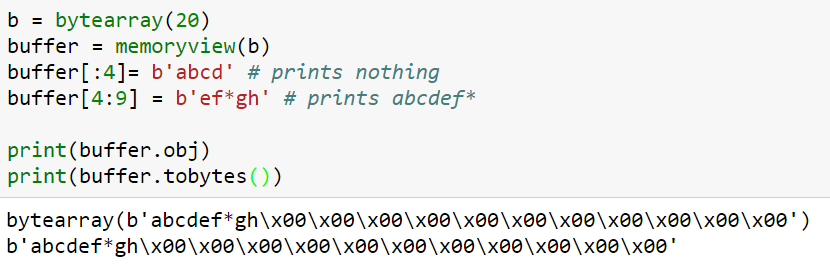
Bytearray 타입에 숫자를 주고 생성하면 버퍼가 생긴다. 버퍼에 값을 넣어서 처리할 수 있다.



### 4.5.2. memoryview 메소드

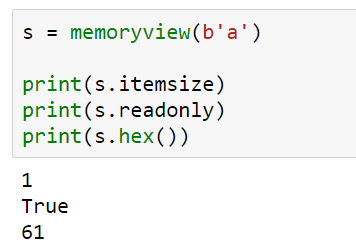
* 버퍼처리하기 : memoryview로 wrapping 하기

Bytearray를 가지고 버퍼를 만들고 이것을 Memoryview로 래핑해서 처리



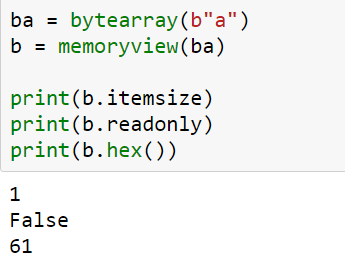
* 메모리에 대한 처리 : bytes 타입

Bytes 타입은 변경 불가하므로 readonly 속성이 True 표시되어 변경할 수 없다는것을 알 수 있다.



* 메모리에 대한 처리 : bytearray타입

Bytearray는 변경가능하므로 readonly 속성이 False로 표시되어 변경이 가능하다는 것을 알 수 있다.



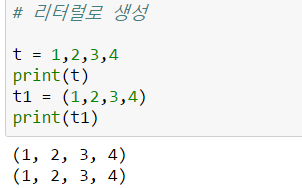
## 4.6. 튜플 데이터 타입

튜플 타입은 리스트 타입과 유사하지만 변경이 불가한 데이터 타입이다. 파이썬내부에서 변경불가한 기능을 처리하기 위해 사용된다.

### 4.6.1. 튜플 생성

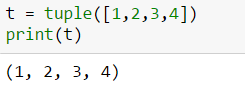
* 튜플 리터럴로 생성

튜플 리터럴 생성은 괄호가 없이 쉼표를 기준으로 인식된다.



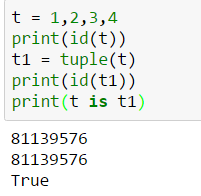
* 생성자로 생성

tuple클래스로 생성은 sequence 데이터 타입을 인자로 받아 생성이 가능하다.



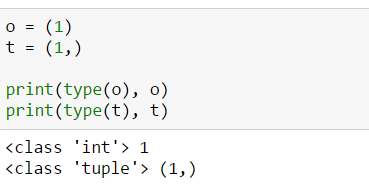
* Interning 발생

튜플은 변경이 불가한 데이터 타입이므로 기존 생성된 튜플을 가지고 생성자로 튜플을 생성하면 새로운 객체를 만드는 것이 아니라 기존에 만들어진 객체를 리턴해서 동일한 객체를 유지한다.



* 튜플 내의 원소가 하나 일 경우 주의 사항

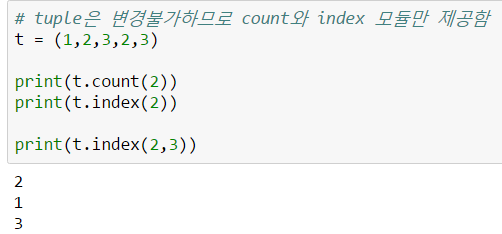
튜플 인스턴스의 원소가 하나인 경우에는 반드시 마지막에 꼭 쉼표(,)를 부여해야 한다.



### 4.6.2. 튜플 메소드

* count, index 메소드

원소에 대한 개수와 원소의 인덱스를 찾아주는 메소드가 존재하며 두번째 인자에 위치를 주면 그 위치 이후에 인덱스 값을 찾아준다.

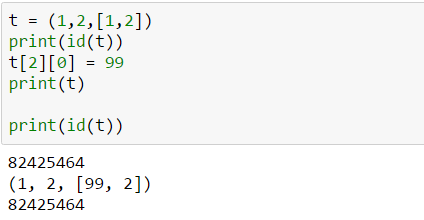


### 4.6..3. 튜플 원소로 Mutable 원소 처리 방법

튜플 내의 리스트는 변경가능하므로 변경이 가능하지만 튜플 객체 자신에 대한 정보는 바뀌는 것이 아니다.

* 튜플 내부 원소로 리스트 타입이 있을 경우

리스트에 대해서는 변경이 가능하므로 원소도 변경이 불가능하도록 처리하려면 튜플로 처리해야 한다.



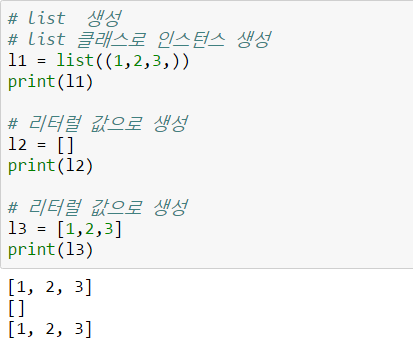
## 4.7. 리스트 데이터 타입

변경가능한 Sequece 타입으로 다양한 데이터 타입을 원소로 구성될 수 있다.

### 4.7.1. 리스트 생성

* 생성자와 리터럴 값으로 생성하기

리터럴과 생성자를 이용해서 생성하면 동일한 결과가 나온다



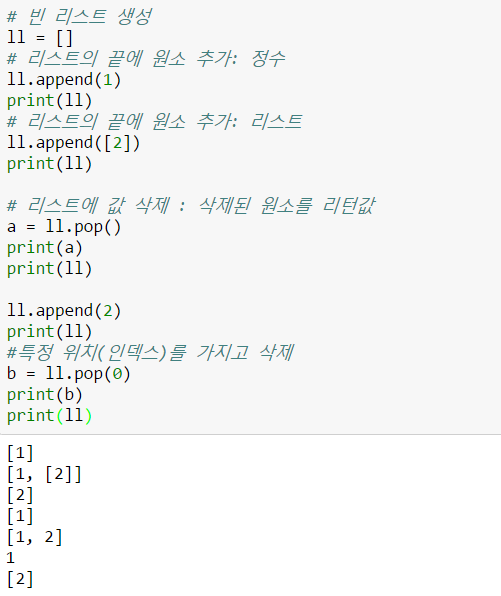
### 4.7.2 리스트 내의 메소드

리스트 처리를 위한 주요 메소드에 대한 리스트이다.

|  |  |
| --- | --- |
| Method | Description |
| list.append(obj) | 리스트에 원소 추가 |
| list.count(obj) | 리스트 내의 동일한 원소 갯수 |
| list.extend(seq) | 리스트의 원소를 확장 |
| list.index(obj) | 리스트의 원소가 위치 정보 |
| list.insert(index,obj) | 리스트 내의 위치에 원소 삽입 |
| list.pop(obj=list[-1]) | 리스트 내의 마직막 원소를 삭제, 위치를 지정하면 지정된 위치의 원소 삭제 |
| list.remove(obj) | 리스트 내의 값으로 삭제 |
| list.reverse() | 리스트를 역으로 소팅 |
| list.sort([func]) | 리스트를 내림차순으로 소팅 |

* 리스트에 마지막 위치에 추가 및 삭제

리스트 내의 원소는 기본으로 마지막 부분에 추가와 삭제를 처리한다.

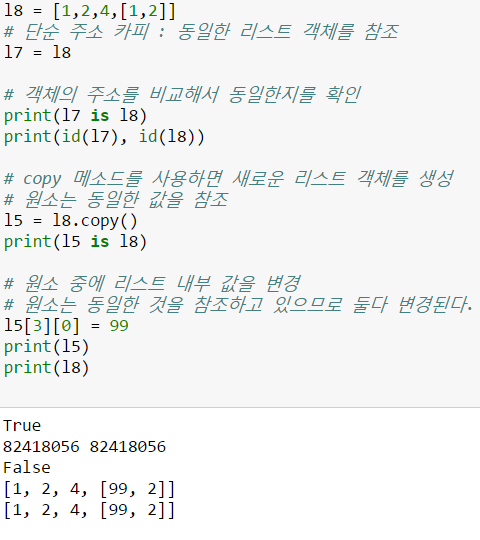


* 리스트 복사하기

리스트를 복사하면 기본적으로 리스트 타입만 새로운 것으로 복사되고 리스트 내부의 원소들까지 새로운 것으로 만들지 않는다.

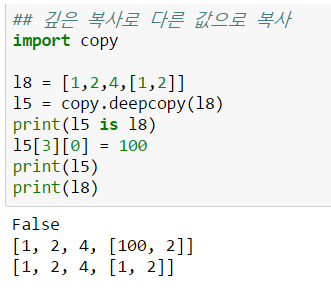
* + 복사에는 2가지가 존재
* 얕은 복사(shallow copy) : 새로운 객체를 만들지만 내부 원소는 기존 원소를 참조
* 깊은 복사(deep copy) : 새로운 객체를 만들고 내부 원소들도 다른 원소로 만듦
* 얕은 복사하기

자기 자신을 새로운 사본으로 만들어서 새로운 객체를 만든다. 내부 원소까지 새로운 사본을 만들지 않기 때문에 원소로 리스트 등이 있을 시에 원소 내부의 값이 변경되면 원본과 사본이 동시에 갱신된다.



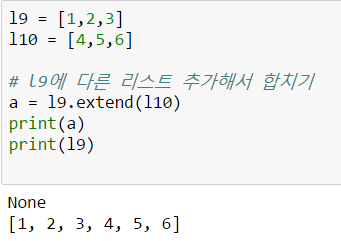
* deepcopy를 하는 이유

자기 자신과 자기 내부의 원소 전부를 새로운 객체로 만들어서 사용이 필요하다. 내부 원소까지 전부 새로운 객체로 만들기 위해서는 copy 모듈을 import해서 deepcopy 함수를 이용해서 완전히 새로운 객체를 만들어야 한다.



* 리스트를 합치기

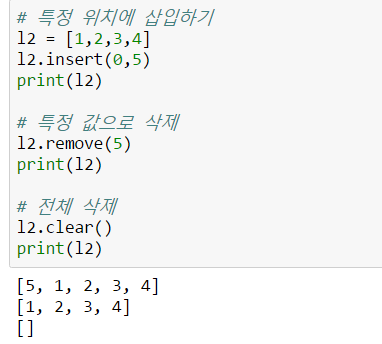
두 개의 리스트를 원본 리스테 내의 하나로 합친 리스트를 만든다.



* 리스트 특정 위치에 삽입 및 전체 삭제

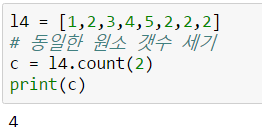
리스트 내의 특정 위치에 삽입은 insert 메소들을 사용해서 추가한다. Insert의 첫번째 인자는 위치이고 두번 째 인자는 원소이다.

특정 값을 가지고 삭제는 remove 메소드를 사용하면 전체 원소를 전부 삭제하고싶으면 clear 메소드를 사용한다.



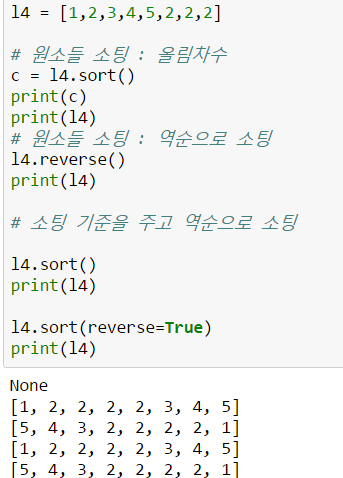
* 동일한 원소 갯수를 세기

리스트 내의 원소가 동일한 경우가 있을 경우를 확인하기 위해서 사용하고 동일한 값의 개수를 확인해서 숫자로 알려준다.



* 리스트 원소들 소팅하기

리스트 내부 원소들을 내림차순 별로 소팅도 가능하고 올림차순으로도 소팅이 가능하다.

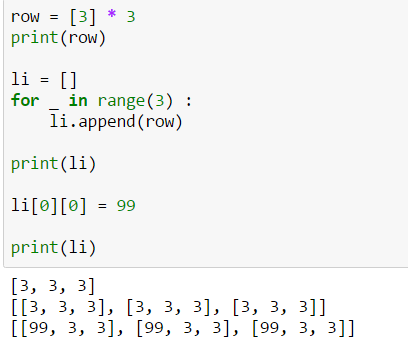


### 4.7.3 리스트 내에 리스트를 넣을 때 초기화 하는 법

리스트 내에 리스트를 원소로 구성할 때 내부 원소의 리스트가 명확히 다른 것으로 인식되려면 내부 리스트를 넣을 때 매번 초기화 처리를 해줘야 한다.

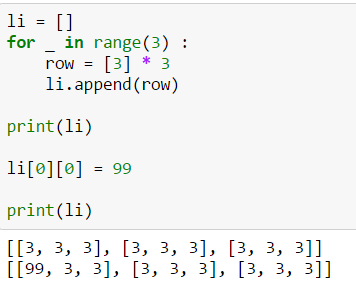
* 리스트를 먼저 초기화

리스트를 초기화한 후에 for 문을 통해 빈 리스트에 원소로 추가하면 내부에 들어간 모든 원소가 동일한 레퍼런스를 가지고 있어 리스트를 원소를 변경할 때 모든 값들이 같이 변경된다.



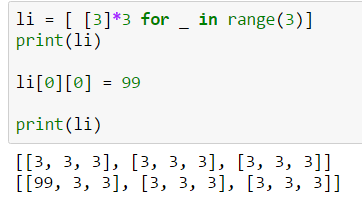
* 리스트를 for 문 안에서 초기화

for 문 내에서 내부 리스트 원소로 들어가는 것을 초기화하면 항상 새로운 리스트를 만들어서 리스트 내부의 원소로 들어간다.



* 지능형 리스트로 처리하기

지능형 리스트도 항상 for문으로 처리하는 이유는 for문이 작동할 때 새롭게 내부리스트 원소를 초기화 처리를 한다.

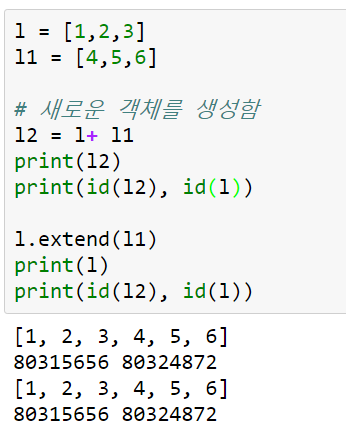


### 4.7.4. +, \* 연산자 처리

+ 연산자는 extend 연산자와 같이 두 개의 sequence를 합치고 \* 연산자는 반복해서 생성한다.

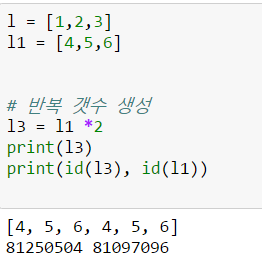
* 리스트로 +와 extend 비교하기

연산자(+)를 이용해서 생성하면 새로운 객체를 만들고 extend 메소드는 자기 객체 내의 원소들을 추가 한다.



* 리스트로 \* 연산자 비교하기

같은 원소를 반복해서 만들어 준다.



## 4.8. sequence 타입 slicing 이해하기

Sequence 타입을 검색, 갱신, 삭제를 할 때는 원소하나만 처리하는 경우는 indexing 처리를 수행한다. 부분 집합을 처리할 때는 slicing으로 처리한다.

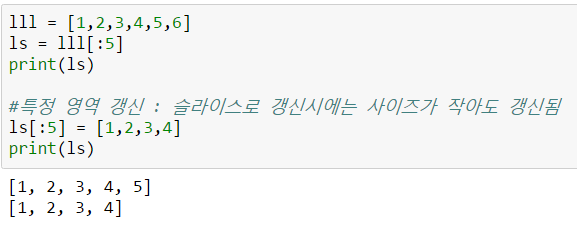
검색 및 갱신 등을 직접 [ ] 기호를 이용한 연산자를 사용하도록 하지만 파이선 내부적으로는 스페셜 메소드에 위해 처리가 된다.

### 4.8.1. index 순서 이해하기

* 순방향 인덱스

파이썬은 순방향 인덱스는 0 부터 시작하기에 Sequence 데이터 타입의 객체의 원소 개수 -1개 만큼만 순환할 수 있도록 구성된다. Index 범위가 넘을 경우는 예외가 발생한다.

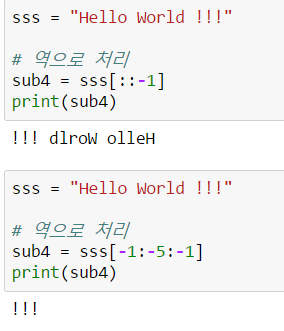
예제는 순방향으로 읽을 때 관행상 첫번째부터 읽을 경우는 0을 표시하지 않는다.



* 역방향 인덱스

파이썬은 역방향으로 바로 접근할 수 있도록 -1 부터 역방향 index를 지원한다.

역방향으로 전체를 읽을 때는 시작과 끝은 표시를 하지 않고 step에 -1로 처리하고, 역으로 특정영역을 읽을 경우는 역으로 시작점과 종료점 그리고 step도 명기해야 한다



* 문자열 슬라이싱 처리해서 검색하기

문자열을 슬라이싱하면 별도의 문자열 객체가 반환된다.



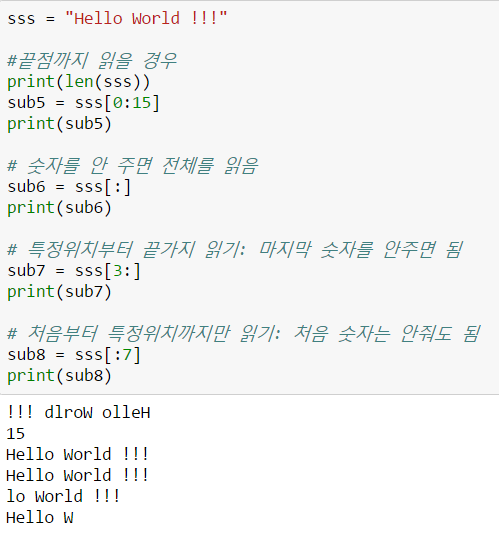
* 슬라이싱에서 step 사용하기

2칸씩 이동하면서 검색해서 부분 문자열을 만든다.



* 시작점과 끝점을 없이 처리 가능

시작부터 시작하면 index 값을 아무것도 주지 않아도 되고 마지막까지

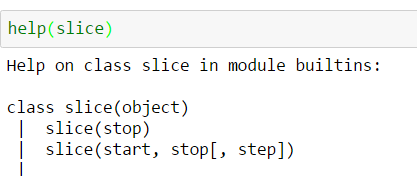


### 4.8.2. slice 클래스 이해하기

Sequence 데이터 타입에서 indexing 처리( [ ]연산자)는 하나의 값만 들어간다. 슬라이싱으로 표현 [start:end:step] 으로 표현하지만 실제 들어가는 값은 slice 클래스의 인스턴스가 들어가므로 하나의 값만 표시되는 것이다.

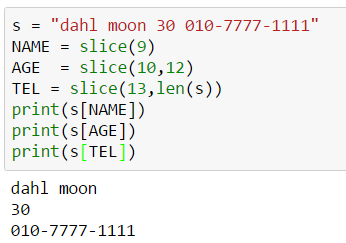
* Slice도 하나의 클래스

Slice를 이용해서 하나의 인스턴스 객체를 만든다.



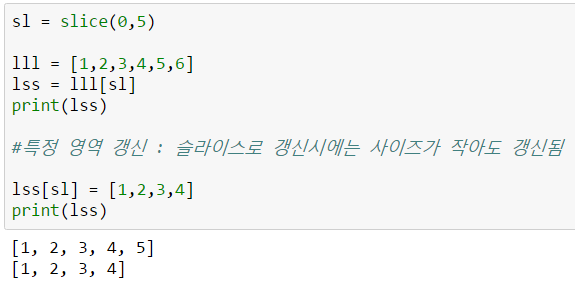
* 변수에 할당하기

Sequence 타입을 처리할 때 먼저 slice로 객체를 만들어서 위치를 고정해서 변수에 할당하고 이를 Sequence 검색 및 슬라이싱 연산자([ ]) 에 넣고 처리가 가능하다.



* Slice로 갱신하기

slice(시작점, 종료점,스템)을 객체로 만들면 [시작점, 종료점,스텝]가 동일한 결과를 나타내며 슬라이스 크기에 상관없이 주어진 원소를 처리하기에 슬라이스 크기보다 작은 경우에도 처리하므로 전체 원소의 개수가 줄여들 수도 있다.

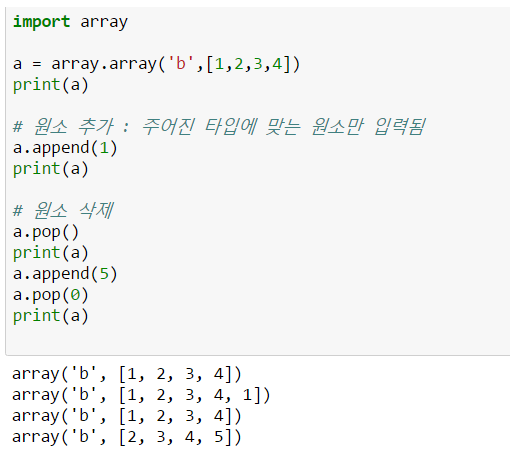


## 4.9 배열 array.array 모듈

동일한 타입만을 처리하면서 list보다 빠르게 처리할 수 있도록 이 모듈을 제공한다.

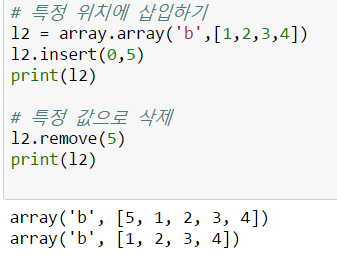
* 생성, 추가, 삭제 처리

타입코드 'b'는 : signed integer(1 bytes)를 표시하고 이를 기준으로 생성한다. 대부분 리스트처리와 유사하다.



* 특정위치 삽입 및 제거

파이썬은 유사한 처리가 필요한 경우 메소드 이름을 동일하게 작성해서 처리하도록 구성하면서 처리도 거의 유사하다.



* 슬라이싱 처리

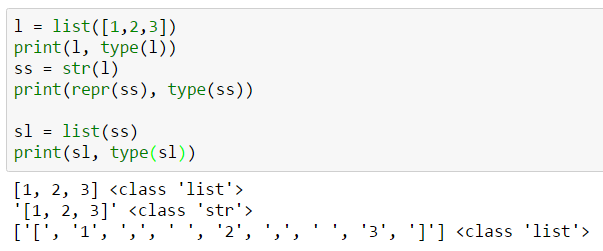
슬라이스 즉 부분 집합을 만드는 것도 Sequence 데이터 타입과 같다.



## 4.10. Sequence 타입 형변환

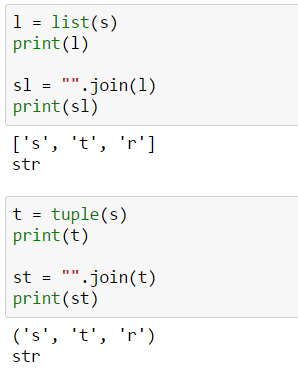
* 리스트와 문자열과의 형변환 : 클래스만 사용

리스트를 문자열로 형변환을 하면 리스트 전체를 문자열로 만들어 버린다. 이를 다시 리스트로 전환하면 원하지 않는 결과가 나온다.



* 리스트와 문자열 형변환 : 문자열 메소드로 이용

리스트를 문자열로 형변환을 하고 싶을 경우에는 문자열의 join을 사용하면 문자열 값이 원하는 것으로 만들 수 있다.



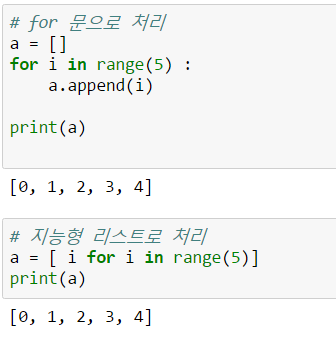
## 4.11. 지능형 리스트 처리

* 지능형 리스트 : 리스트를 특정 시점에 생성시키는 방식

[ 표현식 for 표현식 in iterable/iterator if 표현식]

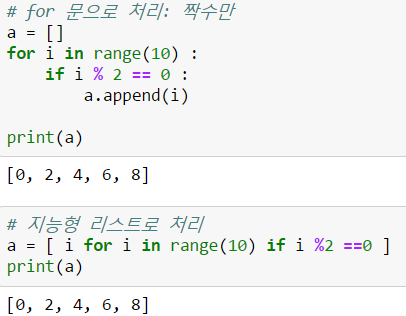
* 지능형 리스트

for문을 이용해서 원소를 추가하는 것보다 지능형 리스트를 이용해서 한번에 처리하는 것이 사용상 편리하다.



* 지능형 리스터의 필터링 처리

for문 다음에 if문을 이용하면 if 조건에 만족하는 경우만 처리하도록 필터가 된다.



## 4.12. packing과 unpacking 처리

파이썬에서 변수 할당이나 파라미터 할당 등은 내부적으로는 변수와 객체들을 매핑하는 방법이다. Sequence 타입은 내부에 원소들이 많아 이 원소들과 변수간의 매핑을 처리하는 방법이 packing과 unpacking 규칙이다.

Sequence 데이터 타입과 변수가 일대일로 매핑되는 경우는 변수에 Sequence 데이터 타입의 객체가 바로 매핑되므로 packing과 unpacking이 발생하지 않는다.

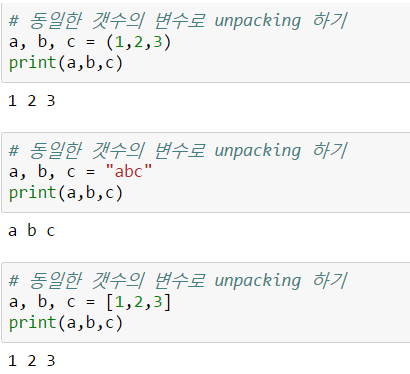
.

Sequence 나 Mapping 타입을 처리하기 위해 분리할 것인지 합칠 것인지를 처리하며 Sequence 타입에는 \*를 사용하고 Mapping 타입에는 \*\*를 사용한다. .

* Sequence 타입을 동일 개수의 변수에 unpacking 처리 하기

변수에 Sequence 데이터 타입 객체의 원소를 동일한 개수로 매핑하는 것을 unpacking 이라고 한다.

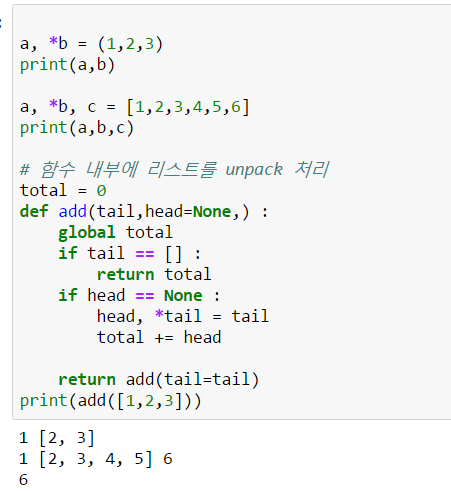
예제는 일대일로 매핑되어 각 원소들이 변수에 할당되어지는 것을 볼 수 있다. .



* 다른 개수의 변수로 unpacking 하기

변수의 개수가 부족한 일대 다 매핑일 경우 에러가 발생하므로 특정 변수에 \*를표시하면 \*가 붙은 변수는 Sequence 타입의 데이터를 받을 수 있도록 처리가 된다.

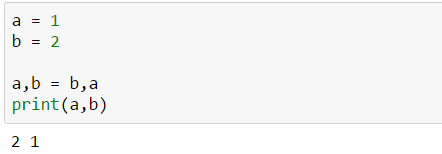
예제는 Sequence 타입을 다양하게 unpacking 처리한다.



* swapping 처리하기

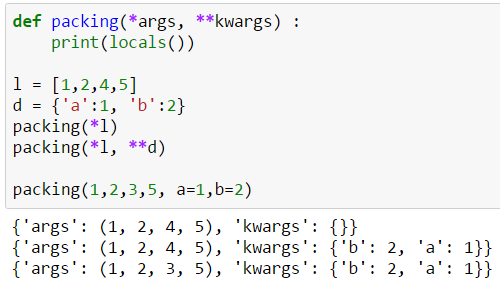
변수들 끼리 내부의 값을 바꾸는 것을 swap이라 한다. 파이썬에서는 별도의임시저장변수를 사용하지 않아도 이를 간단히 처리한다.

이 예제가 처리되는 이유는 좌측 편은 실제 튜플형이 데이터이고 그 내부 원소에 변수로 지정한 값이 들어가 있고 좌측 편에는 변수가 정해져 있어 단순히 매칭만 하면 변수의 값을 가르치는 레퍼런스를 바꿔주는 역할만 하는 것이다.



* 함수 인자에서 packing 과 unpacking 하기

함수 인자에 리스트, dict 등의 데이터 타입을 넣을 경우는 하나의 원소로만 인 되므로 이를 unpacking해서 원소의 개수를 가변화하는 것이고, 함수 정의된 파라미터는 이 인자들을 받아 packing 처리해서 하나의 변수에 넣는 것이다.



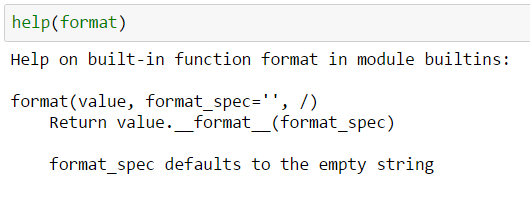
## 4.13. 포매팅 처리

파이썬은 기본 문자열로 출력을 하므로 문자열 기준으로 포매팅 체계를 제공한다. Format 함수와 문자열의 format 메소드를 이용해서 처리한다.

### 4.13.1 포맷함수와 메소드 이해하기

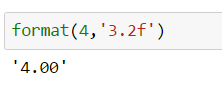
* 포맷 내장함수

Format 함수의 파라미터는 값과 포매팅 폼이 필요하다.



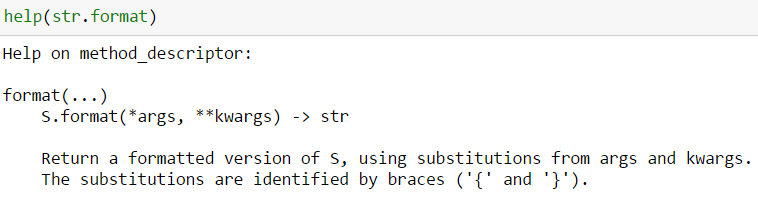
* + 내장함수 포매팅 처리

4를 입력하고 실수의 정밀도를 주는 포매팅 폼을 주면 해당 값을 정수로 표시한문자열을 결과로 제공한다.



* 문자열 포매팅 메소드

문자열 내에 포매팅 폼을 지정하고 메소드 인자에 실제 값들이 들어와서 포매팅 폼과 매칭해서 처리하는 메소드이다.



* 포맷코드

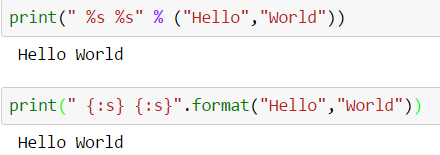
문자열 포매팅 폼을 정의할 때 인자로 받는 값들이 타입을 확인하고 적절하게 포매팅하도록 지원하는 코드들이다.

* + 주요한 포매팅 코드

|  |  |
| --- | --- |
| **코드** | **설명** |
| s | 문자열 (String) |
| d, i | 정수 (Integer) |
| f | 부동소수 (floating-point) |
| o | 8진수 |
| x | 16진수 |
| X | 16진수 |
| %, %% | % 표시 |
| !r, %r | \_\_repr\_\_ 처리 |
| !s. %r | \_\_str\_\_ |

* 포매팅 코드 추가하기

인자를 매핑하는 기호(% 다음이나 { })들 내부에 세미콜론( : )를 붙이고 포매팅 방법을 추가하면 포맷코드에 맞도록 출력이 된다.



* 숫자로 위치인자를 표시하기

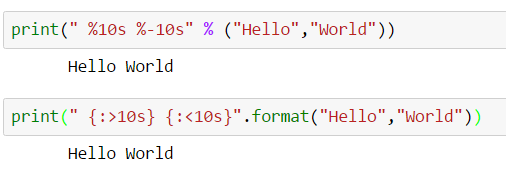
메소드 format의 인자의 위치로 포매팅 코드 내에서 매핑하도록 강제화할 수 있다.

예제는 위치 인자를 바꿔 출력 값을 변경했다.



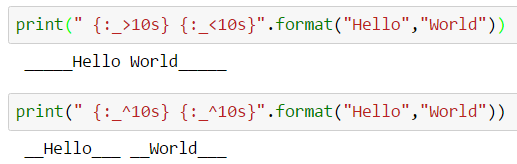
* 출력 값들을 정렬하기

포매팅코드 앞에 길이를 부여하고 정렬의 방향을 좌측(new 방식:>)이나 우측(old 방식: -, new 방식 : <)으로 부여하면 길이에 맞춰 포매팅 처리가 된다.



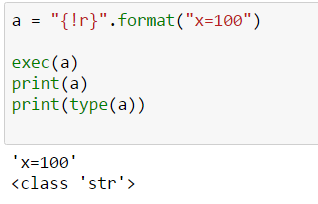
* 빈 공간에 의미의 값 넣기

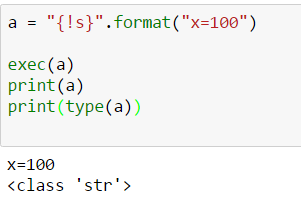
정렬을 좌측, 우측, 중앙으로 처리할 수 있도록 하고 나머지 빈 공간에 임의의 문자( \_ )를 출력하도록 처리한다.



* repr/str 호출 처리

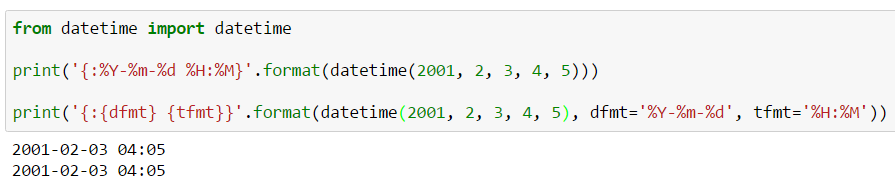
포매팅 폼을 호출하는 것이 아닌 실제 \_\_repr\_\_, \_\_str\_\_ 내부 메소드를 호출을 하도록 포매팅을 만들어서 처리한다.





* 날짜 포매팅 코드 사용하기

Datatime 내의 처리 규칙에 따라 포매팅 지정해서 사용하기



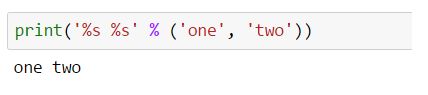
### 4.13.2. 포맷함수 및 메소드 인자 매핑

* 포매팅 폼에서 인자를 받는 방법

문자열에서 포매팅 폼과 인자 사이의 연결을 하는 방법은 % , { } 두 가지 방법이 있다.

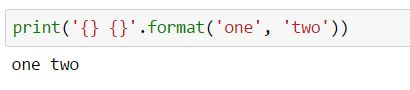
* + Old 방식 처리 : %

문자열 내에 %로 지정된 인자 개수를 지정하고 문자열 밖에 %로 표시된 다음에 인자의 개수를 넣으면 매칭되어 처리한다.



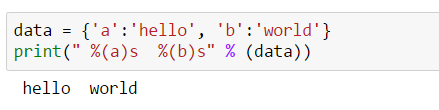
* + New 방식 처리 : { }

문자열 내에 중괄호{ } 를 지정해서 format 메소드 내의 인자와 매핑해 처리되도록 처리한다.



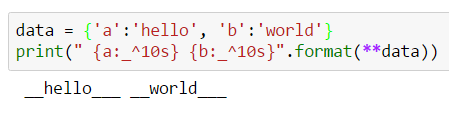
* 인자명을 가지고 처리하기 : old 방식

Dict 타입으로 전달 시 가져올 키값의 이름을 ( ) 내에 정의하면 실행 시 내부 키값을 기준으로 데이터가 처리된다.



* 인자명을 가지고 처리하기 : new방식
  + 키워드 인자로 처리하기

Format 메소드에 dict 타입을 unpacking해서 키워드 인자로 넣어야 내부 키의 이름으로 매칭되어 처리된다.



* + item으로 직접 접근하기

Format 메소드에 dict 타입을 직접 할당하면 그 dict을 [ ] 연산자를 이용해서 처리한다.



* + 객체 속성 접근연산자로 접근

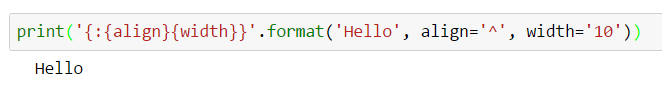
Format 메소드 내에서 인스턴스를 생성하면 객체 접근을 위한 닷연산으로 직접 접근이 가능하다.



### 4.1.3.3 포매팅 꾸미기

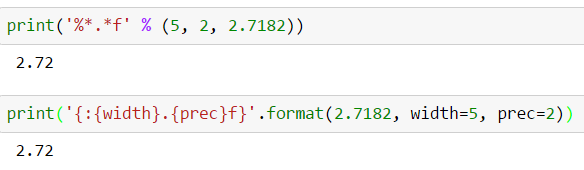
* 포매팅을 파라미터로 받아서 처리하기

포매팅 기호를 변수명으로 할당해서 처리도 가능하다.



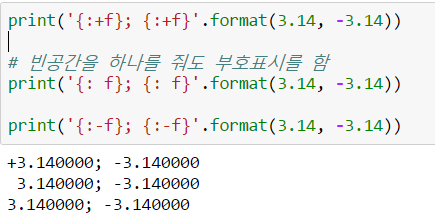
* 포매팅을 파라미터로 받아서 정밀도 처리하기

숫자의 정밀도도 변수로 받아서 처리가 가능하다.



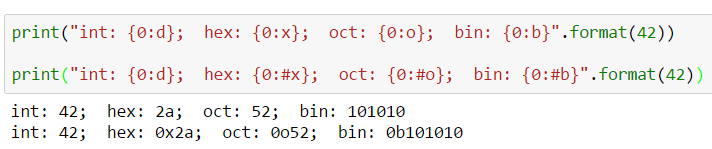
* 부호 처리하기

음수나 양수도 표시를 할 수 있다.

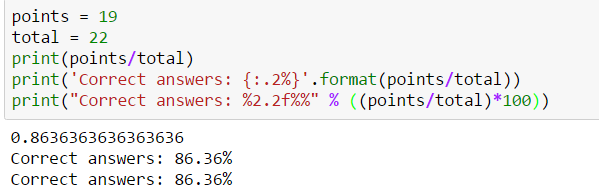


* 숫자를 표시하기

#을 넣으면 각 숫자의 추가구성항목인 진법표시를 한다.



* 퍼센트 표시하기



* 천 단위 쉼표 표시하기



**5. 파이썬 Mapping/Set데이터 타입**

원소를 key와 값으로 구성하는 데이터 타입이고 각 Key는 해쉬를 생성해서저장하므로 순서와 상관없이 저장되는 컨테이너 타입을 이번 장에서 알아보기로 하겠다.

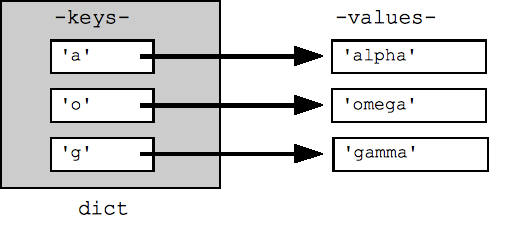
## 5.1. dict 타입 이해하기

### 5.1.1 dict 의 키 생성 기준

변경불가능한 데이터 타입(int, float, tuple, str, bytes, frozenset 등)이 가능

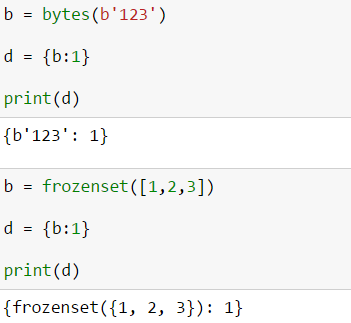
* Dict 타입의 구조

키와 값으로 구성된 mapping 타입이 구조



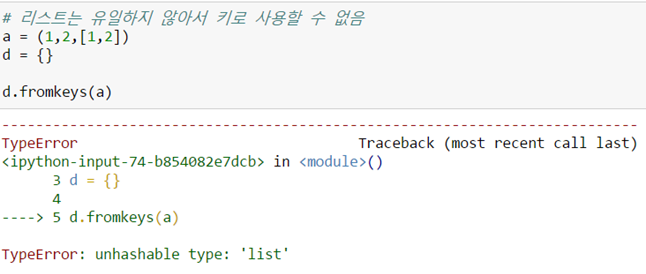
* bytes와 frozenset도 키로 사용가능

변경불가한 데이터 타입으로 키를 생성이 가능하므로 bytes와 frozenset도 키가 가능하다.



* tuple일 경우 원소가 list일 경우

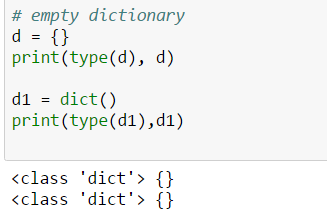
튜플의 원소가 리스트이면 변경이 가능해져 dict 타입의 키로 사용이 불가하다



### 5.1.2. dict 생성하기

* dict 생성하기 : 빈 dict 생성

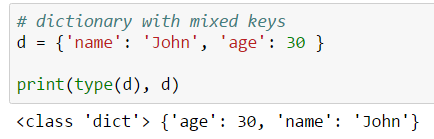
{ } 기호는 빈 dict 데이터 타입의 인스턴스를 표시한다.



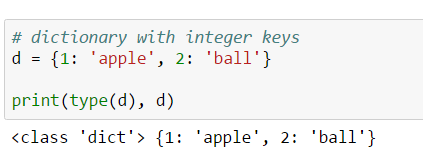
* dict 생성 : 리터럴

문자열이나 숫자로 키를 만든 리터럴로 dict 인스턴스를 생성한다.

* 문자열로 키를 만들고 생성하기



* 숫자를 키로 생성



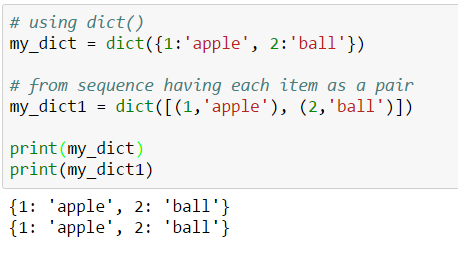
* 키를 문자와 숫자로 혼용 생성



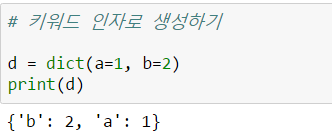
* dict 클래스 이용

dict 생성자는 리터럴 값을 넣고 생성하면 다양한 리터럴 값을 인자(리스트,키워드 인자 등)로 받아dict 타입 인스턴스를 만들 수 있다.

* + 2 튜플 원소를 가진 리스트 인자



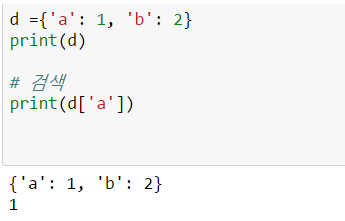
* 키워드 인자



### 5.1.3. dict 검색

* dict 검색은 key로 값 찾기

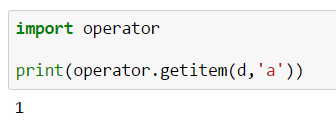
검색 연산자( [ ] )는 sequence 데이터 타입과 동일하지만 내부에 들어가는 값은 index가 아닌 Key를 넣어 조회한다.



* operator 모듈의 함수 이용하기

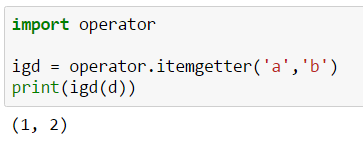
operator 모듈 내의 getitem, itemgetter를 이용해서 원소를 검색할 수 있다.

* + getitem 으로 검색



* + itemgetter로 검색

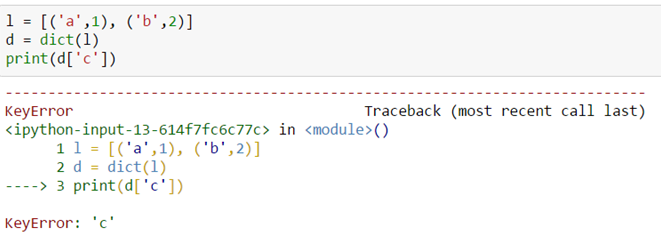
여러 키값을 동시에 읽고 싶을 경우에는 operator 모듈의ㅣ itemgetter를 이용해서 여러 개의 키를 동시에 검색한다.



* 없는 키로 조회 또는 세팅할 때 에러가 안 나게 하는 방법

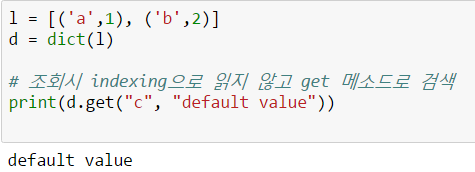
[ ]로 처리해서 에러가 날 수도 있으므로 get 메소드를 이용

* + 키가 없을 경우 에러



* + Get 메소드로 검색해서 예외처리 미발생

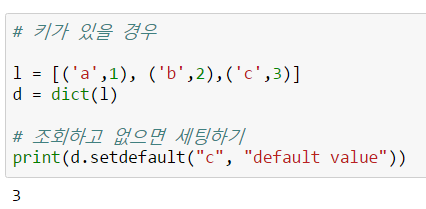
내부에 default 값을 지정할 수 있어 없으면 조회시 없으면 default값을 전달한다.



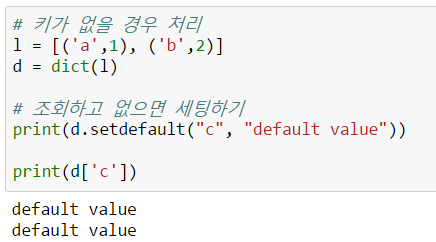
* 없는 것을 조회할 수 있어 dict 내부에 세팅하고 조회하기

Setdefault 메소드를 사용하면 키값이 없는 경우에도 속성을 추가한 후에 처리하고 있는 경우에는 기존 값을 조회해서 처리한다.

* 기존에 키가 있을 경우는 조회해서 처리

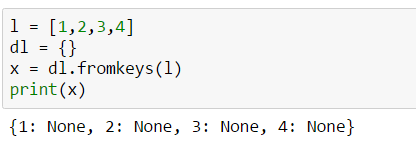


* 기존에 키가 없을 경우는 default 값을 세팅하고 조회 처리



* 특정 key를 가져와서 dict 데이터 타입 생성

리스트 내의 원소가 dict 타입이 키가 가능할 경우 fromkeys 메소드를 이용해서 키값을 추가하면서 default. 값으로 None으로 처리

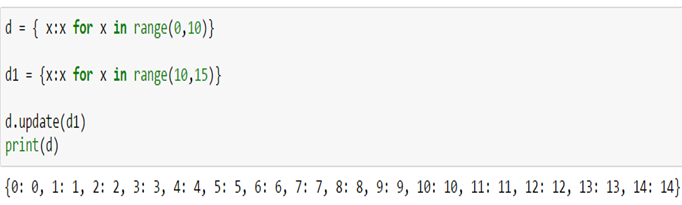


### 5.1.4. dict 데이터 타입 추가 /삭제

* dict에 dict 타입 추가 : update

dict 내에 다른 dict 타입의 원소를 전부 삽입이 필요할 경우 이용한다.

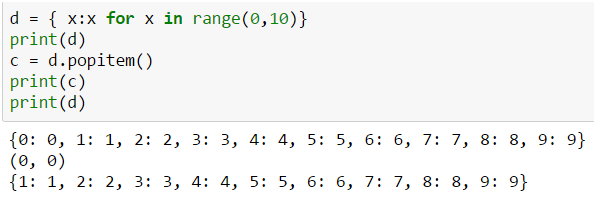
지능형 dict 생성 표현식을 이용해서 두개의 dict 타입을 만든 것이다.



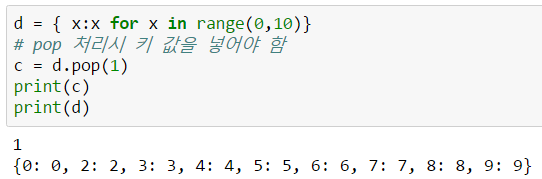
* dict 타입 항목 삭제

dict 타입은 순서가 없으므로 임의의 항목(key/value)를 삭제하고 결과로 삭제된 항목을 리턴한다.

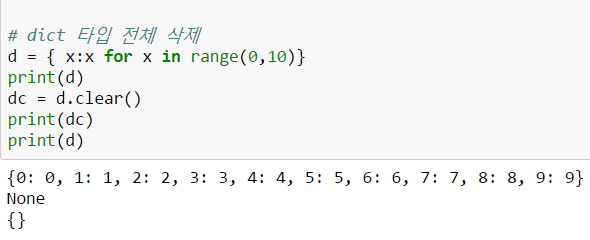
* popitem 메소드를 이용해서 삭제



* pop메소드를 이용해서 삭제



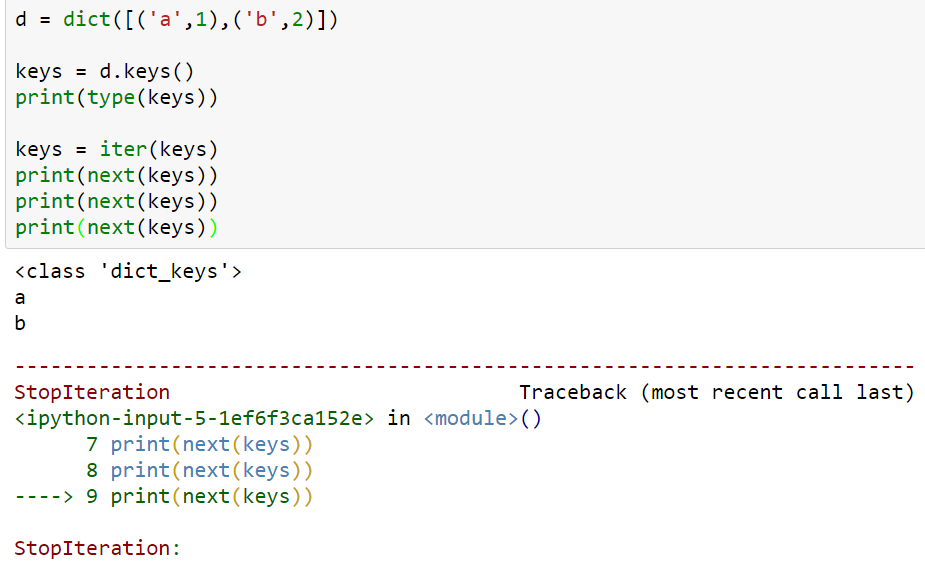
* dict 내부 전체 원소 삭제



### 5.1.5. dict 타입에 keys, values, items 객체 이해하기

* 반복형 처리가 가능하도록 처리

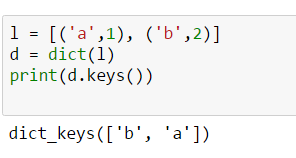
반복형이 가능하도록 클래스 정의 내부에 \_\_iter\_\_가 생성되어 있어 iter 함수를 이용해서 반복형을 만들고 next 함수로 호출이 가능하도록 구성된다.



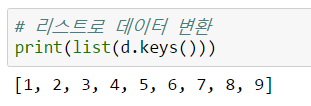
* dict 타입 내부 조회하기 : keys

dict 타입 내에 key 값을 가져오는 메소드로 결과값을 views 타입의 인스턴스로 리턴한다.

* Key 메소드 결과에 대한 데이터 타입



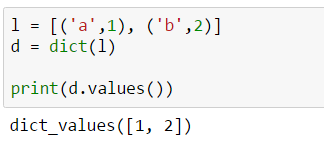
* Key 메소드 결과를 리스트로 변환



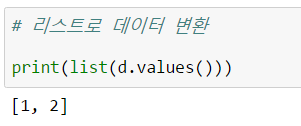
* dict 타입 내부 조회하기 : values

dict 타입 내에 value값을 가져오는 메소드로 결과값을 views 타입의 인스턴스로 리턴한다.

* value메소드 결과에 대한 데이터 타입



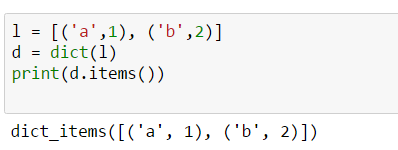
* values 메소드 결과를 리스트로 변환



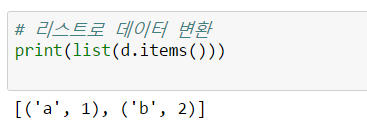
* dict 타입 내부 조회하기 : items

key와 value를 전부 조회할 경우 사용하는 메소드이다.

* items메소드 결과에 대한 데이터 타입

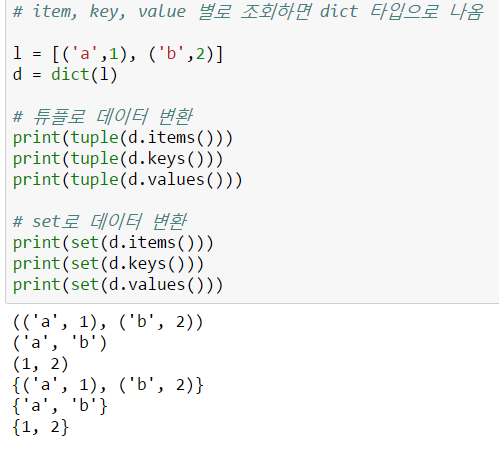


* items메소드 결과를 리스트로 변환



* dict 내부 원소를 조회하고 tuple/set 변환

items, keys, values 메소드의 결과 tuple, set으로 변환도 가능도 가능한 이유는 반복자 타입으로 만들어져 있기에 생성자와 만나면 그 생성자에 맞춰 새로운 데이터 타입의 인스턴스가 만들어지는 것이다.



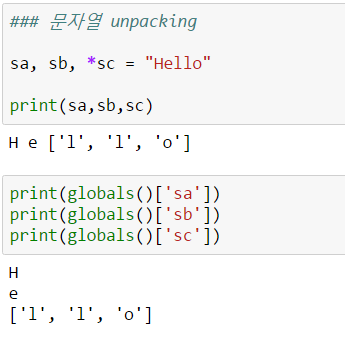
### 5.1.5. dict 타입에 packing/unpacking

* 파이썬 변수 할당 방식

파이썬 변수를 관리하는 namespace가 dict 타입으로 만들어지므로 실제 변수할당시 key와 value 형태로 정의가 되어야 한다.

* + 모듈이 변수 할당

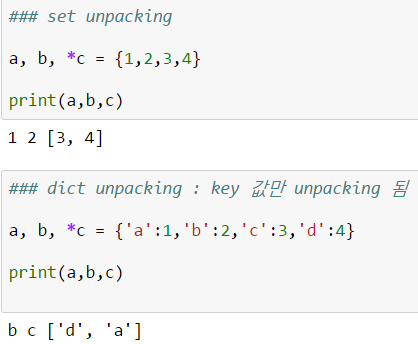
모듈에 변수를 할당하면 global 영역에 들어가는 그 들어가 있는 변수들에 대한 값을 조사해 보면 \*가 붙은 변수는 리스트 타입으로 들어가 있다.



* Set과 dict 타입 변수 할당에서 처리

Mapping 타입으로 unpacking 처리하면 일단 key값만 변수에 들어간다.

특히 dict 타입은 키와 값이 존재하므로 두 가지를 전부 변수에 할당하고 싶으면 dict.items 메소드를 이용해서 키와 값을 가져와서 unpacking 처리가 되어야 한다.



## 5.2. set 데이터 처리 하기

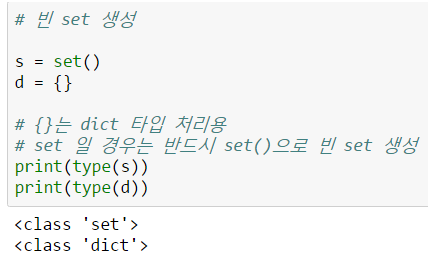
수학의 집합을 표현하는 set, frozenset 데이터 타입을 제공한다. 기본 집합연산을 처리할 수 있다.

### 5.2.1 set

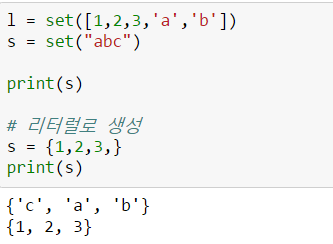
* set 생성하기

빈 set 타입은 set() 으로 표현한다. { }은 빈 dict 타입 표현으로 사용한다.

* 빈 set 생성은 반드시 set()으로 호출해서 만들어야 함

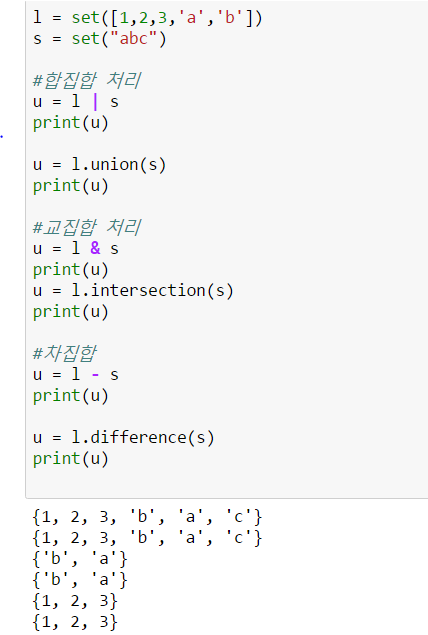


* set()으로 호출해서 만들어야 함



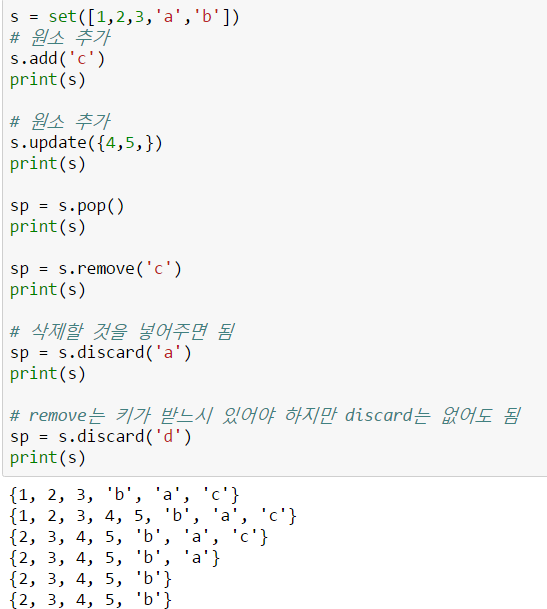
* set 기본 연산 처리하기

합집합, 교집합, 차집합 등의 집합연산을 처리하는 메소드들을 제공한다.



* set 내부 원소값 조정하기

set 데이터 타입은 변경가능하므로 내부 원소에 대한 원소 추가, 삭제 등을 처리하는 메소드를 제공한다.



### 5.2.2. frozenset 이해하기

변경 불가능한 집합이므로 메소드를 통해 만들어지는 것은 새로운 객체들이다.

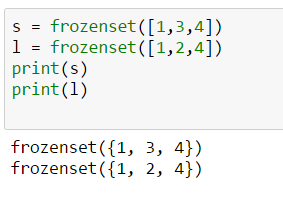
* 생성하기

변경 불가한 frozenset 생성하기

* 빈객체 생성하기

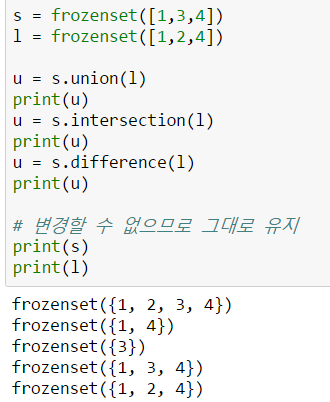


* 원소를 넣고 생성하기



* 연산 처리

변경불가한 데이터 타입이므로 집합 연산 결과는 새로운 인스턴스로 만들어진다.



## 5.3. 지능형 dict/set

* 지능형 dict : dict를 특정 시점에 생성시키는 방식
  + 지능형 dict

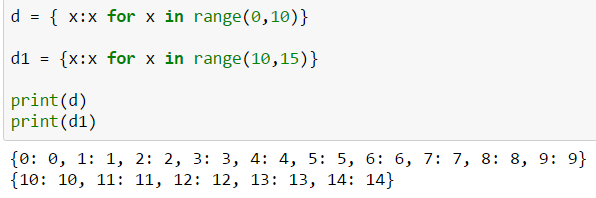
{key: value 표현식 for 표현식 in iterable/iterator if 표현식}

* + 지능형 set

{key 표현식 for 표현식 in iterable/iterator if 표현식}

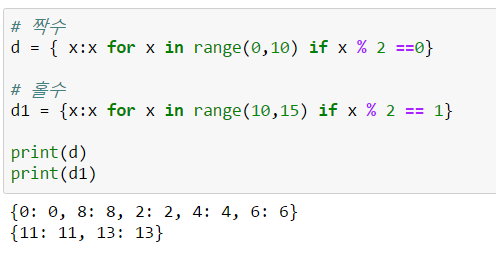
* 지능형 dict 생성

지능형 set과 동일하지만 키와 캆을 처리하도록 구성한다



* 지능형 dict 생성시 필터링 처리

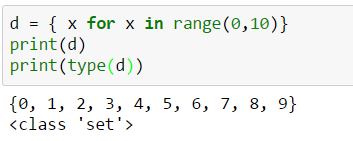
특정 조건만 만족한 것을 원소로 생성되도록 필터링 처리가 가능하다.



* 지능형 set

표기법은 지능형 dict가 유하하지만 key만 존재하고 key를 기준으로 생성한다.

* 지능형 set



* 지능형 set 필터링 하기

