

# 처음부터 시작하는 파이썬



파이썬 3 기준!  
버전 3.7 지원!

비매품

# 목차

## 1

### 파이썬 시작하기

#### 1-1 파이썬이란?

## 2

### 개발 환경 준비

#### 2-1 파이썬 설치하기

#### 2-2 파이썬 둘러보기

#### 2-3 IDLE 사용해보기

## 3

### 자료형

#### 3-1 파이썬 자료형의 특징

#### 3-2 기본 자료형

#### 3-3 리스트

#### 3-4 기타 자료형

#### 3-5 퀴즈

## 4

### 함수와 입출력

#### 4-1 함수란?

#### 4-2 print()

#### 4-3 input()

#### 4-4 pow()

#### 4-5 퀴즈

# 5

## 연산

- 5-1 사칙연산
- 5-2 몫과 나머지
- 5-3 거듭제곱

# 6

## 문자열

- 6-1 문자열이란?
- 6-2 문자열 활용하기
- 6-3 문자열 포매팅

# 7

## 조건문

- 7-1 조건에 따라 코딩하기
- 7-2 if,else
- 7-3 elif
- 7-4 연습문제

# 8

## 반복문

- 8-1 While문
- 8-2 For문
- 8-3 While문과 For문의 공통점 / 차이점
- 8-4 중첩 반복문

# 1

## 파이썬 시작하기

“들어올 때는 마음대로지만 나갈 때는 아니란다.”

# 1-1 파이썬이란?

## 파이썬(Python)

파이썬 (Python)은 범용 프로그래밍 언어로서 코드 가독성과 간결한 코딩을 강조한 언어입니다.

파이썬은 인터프리터(interpreter) 언어로서, 리눅스, Mac OS X, 윈도우 등 다양한 시스템에 널리 사용됩니다.

Python은 원래 그리스 신화에서 그리스 중부 델포이를 지배하였던 큰 뱀인데, 제우스의 아들 아폴로에 의해 죽게 됩니다. 파이썬의 로고가 뱀 모양인 이유가 여기에 있습니다.

Python은 1989년 12월 네덜란드 개발자 귀도 반 로섬 (Guido van Rossum)에 의해 개발되기 시작하여, 약 1년간 개발하여 1991년 처음 Python 0.9 버전을 세상에 내놓았습니다. 그 후 정식 Python 1.0 버전은 1994년에 출시되었으며, Python 2.0은 2000년에, Python 3.0은 2008년에 각각 출시되었습니다. 귀도는 파이썬이라는 이름을 자신이 좋아하는 코미디 쇼인 "몬티 파이썬의 날아다니는 서커스(Monty Python's Flying Circus)"에서 따왔다고 합니다.

## 파이썬 특징

1. 파이썬은 인터프리터(interpreter) 언어입니다.
2. 객체지향, 절차지향 등 다양한 프로그래밍 기법을 지원합니다.
3. 라이브러리가 다양합니다.
4. 동적 타이핑 언어(dynamic typing)로 실행 중에 자료형을 검사하고 자동으로 메모리 관리 됩니다.
5. 플랫폼 독립적인 언어로 어떤 운영체제든 상관없이 사용 할 수 있습니다.

# 컴파일러와 인터프리터란

구분	컴파일러	인터프리터
작동 방식	소스코드를 기계어로 먼저 번역하고, 해당 플랫폼에 최적화되어 프로그램을 실행함	별도의 번역 과정 없이 소스코드를 실행 시점에 해석하여 컴퓨터가 처리할 수 있도록 함
장점	실행 속도가 빠름	간단히 작성, 메모리가 적게 필요
단점	한 번에 많은 기억 장소가 필요함	실행속도가 느림
주요 언어	C, Java, C++, C#	파이썬, 스칼라

## 파이썬 철학

읽어만 보자

import this

The Zen of Python, by Tim Peters Beautiful is better than ugly. Explicit is better than implicit. Simple is better than complex. Complex is better than complicated. Flat is better than nested. Sparse is better than dense. Readability counts. Special cases aren't special enough to break the rules. Although practicality beats purity. Errors should never pass silently. Unless explicitly silenced. In the face of ambiguity, refuse the temptation to guess. There should be one-- and preferably only one --obvious way to do it. Although that way may not be obvious at first unless you're Dutch. Now is better than never. Although never is often better than \*right\* now. If the implementation is hard to explain, it's a bad idea. If the implementation is easy to explain, it may be a good idea. Namespaces are one honking great idea -- let's do more of those!

**"Life is too short, You need Python."**

## 파이썬의 활용

구글, 인스타그램, 요기요 등이 파이썬으로 만들어졌으며, 웹 프로그래밍, 데이터베이스 프로그래밍, 데이터 분석, 인공지능 분야에서 파이썬이 활용 되고 있습니다.

# 2

## 개발 환경 준비

파이썬 공식 홈페이지에서 파이썬을 설치 해봅시다.  
설치한 파이썬을 사용하며 감을 익혀봅시다.  
파이썬 IDLE를 사용하여 본격적인 프로그램 작성을 시작해봅시다.

## 2-1 파이썬 설치하기

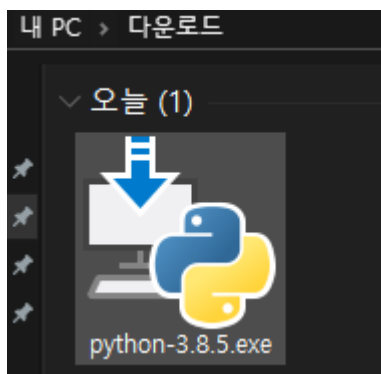
파이썬 공식 다운로드 페이지

(<https://www.python.org/downloads/>) 에서 윈도우용 파이썬 최신버전을 다운 받습니다.

(2020.08.03 기준 최신 버전은 3.8.5)

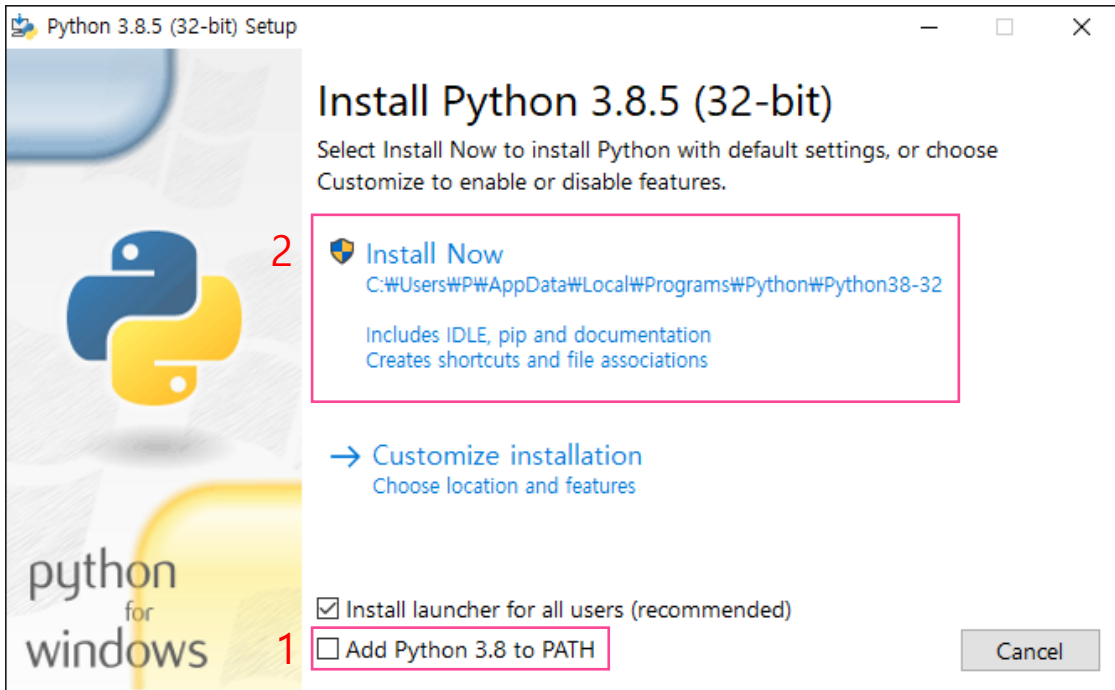


다운로드가 완료되면, 다운로드 받은 프로그램을 실행 시킵니다.





프로그램을 실행 시키면 이러한 창이 뜨게 됩니다.



첫번째로,

Add **Python 3.8 to PATH**를 **꼭** 눌러서 체크해줍니다.

만약 체크하지 않을 경우 나중에 오류로 고생할 수 있습니다.

두번째로 Install Now를 눌러 본격적인 설치를 시작합니다.

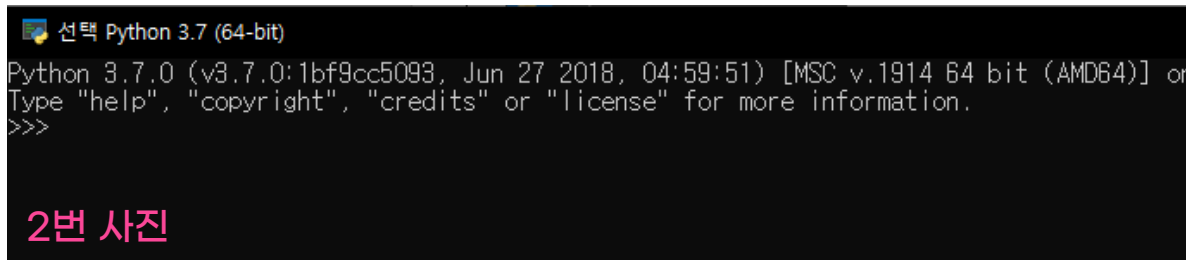
## 2-1 파이썬 둘러보기

설치가 완료되었으면 이제 파이썬을 실행해볼 차례입니다.

**1번 사각형**을 참고하여  
시작메뉴에서 파이썬을 찾아 실행합니다.



**1번 사각형**



실행이 완료되면 **2번 사진**과 같은 화면이 나타납니다.  
위와 같은 창을 **대화형 인터프리터**라고 합니다.  
인터프리터란 사용자가 입력한 소스 코드를 실행하는 환경을 뜻합니다.  
따라서 대화형 인터프리터는 프로그램과 대화하듯이 사용자가  
소스코드를 입력하고, 컴퓨터가 그 결과를 돌려주는 환경입니다.

여기에서 소개하는 내용은 나중에 다시 자세하게 다룰 예정이니 이해가 되지 않는다고 절망하거나 속상해하지 않아도 됩니다.

## 사칙연산

```
>>> 1+3
4
>>> 1-4
-3
>>> 3*8
24
>>> 9/3
3.0
>>>
```

더하기는 (+) 기호를 이용하여 사용할 수 있고,  
빼기는 (-) 기호를 이용하여 사용할 수 있고,  
나눗셈과 곱셈은 (/) 기호와 (\*) 기호를 이용하여 사용할 수 있습니다.

## 변수에 숫자 대입하고 계산하기

```
>>> a = 1
>>> b = 6
>>> a+b
7
>>> _
```

a = 1을, b = 6을 대입한 다음 (+) 기호를 이용해 a와 b를 더해주면  
결과값으로 7이 나오게 됩니다.

## 변수에 문자 대입하고 출력하기

```
>>> a = "Hello world!"
>>> print(a)
Hello world!
>>>
```

변수 a에 Hello world! 라는 값을 대입한 다음 print(a)라고 작성하면 a에 저장되어있는 값을 출력하게 됩니다.

## 조건문 if

```
>>> a = 7
>>> if a>6:
...     print("a는 6보다 큼니다")
...
a는 6보다 큼니다
```

위의 예제는 a 가 6보다 크면 “a는 6보다 큼니다” 라는 문장을 출력(print)하라는 뜻 입니다. 위 예제는 a는 7로 6보다 크므로 두번째 “...” 이후에 Enter 키를 누르면 if문이 종료되고 a는 6보다 큼니다 라는 문장이 출력됩니다.

위 사진의 “if a>6:” 다음 문장은 **Tab 키** 또는 **Spacebar 4번**을 통해 **반드시 들여쓰기** 한 후에 작성해야 합니다.

## 2-3 IDLE 사용해 보기

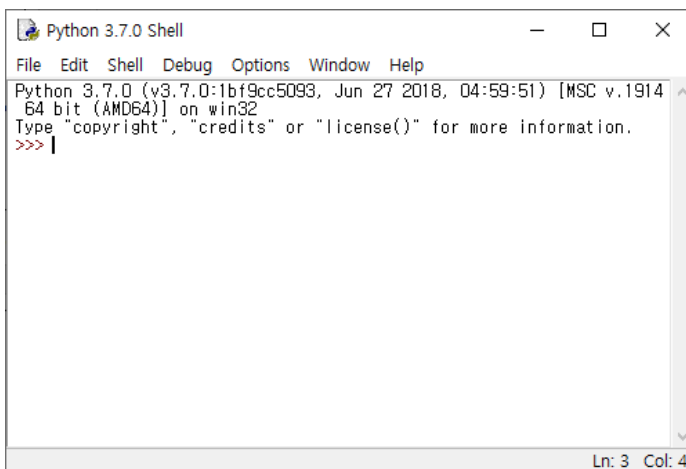
여태까지 파이썬 인터프리터를 이용해 예제를 입력해봤습니다.  
하지만 인터프리터는 여러 줄의 프로그램을 작성하기에 불편합니다. 또  
종료하자마자 프로그램이 사라지기 때문에 다시 사용하지 못합니다.  
이러한 단점들을 보완하기 위해서 IDLE를 사용해 봅시다.

1번 사각형을 참고하여  
시작메뉴에서 IDLE를 찾아 실행합니다.

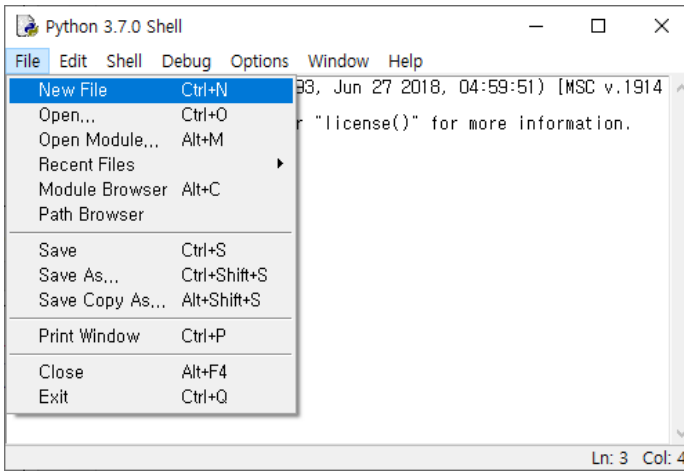


1번 사각형

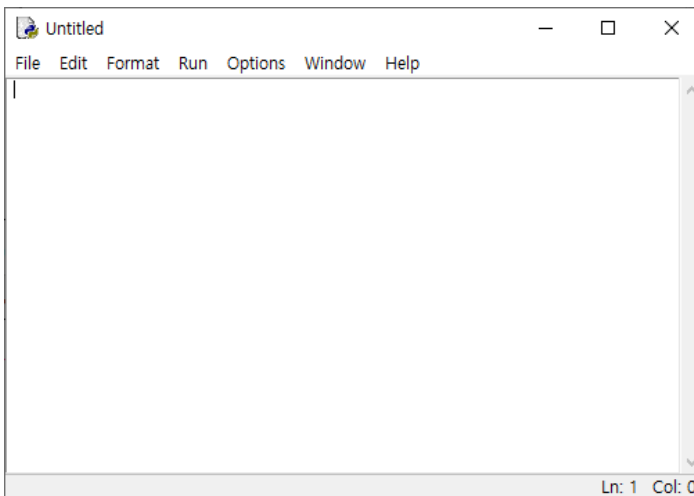
실행하게 되면 아래 사진과 같은 창이 나타나게 됩니다.



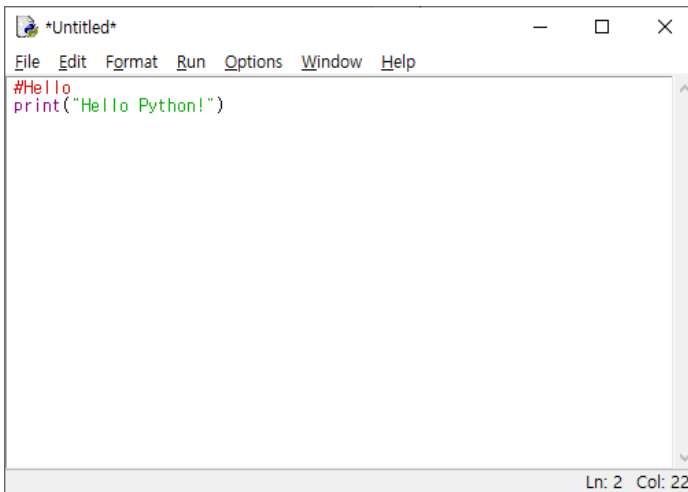
여기서 File -> New File 을 선택합니다.



누르면 다음과 같은 빈 창이 생성되게 되는데 이 창이 IDLE 에디터 입니다.

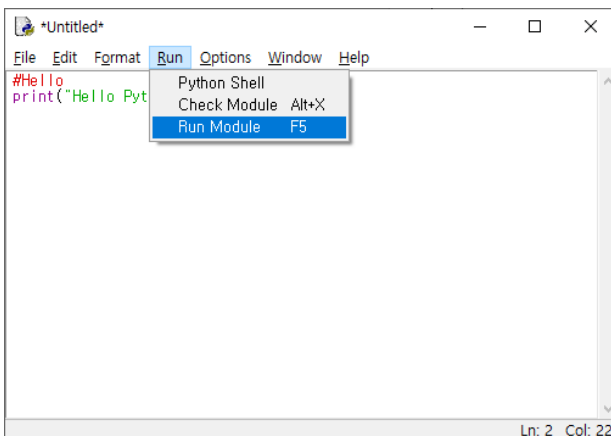


이제 아래 사진과 같은 프로그램을 작성해봅시다.



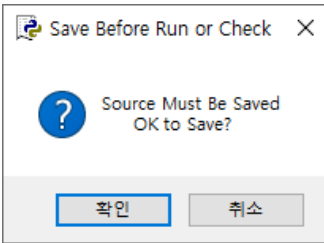
위의 #Hello 문장은 주석입니다. #으로 시작하는 문장은 프로그램 수행에 영향을 주지 않습니다.

주로 코드에 코멘트를 달 때에 사용합니다.

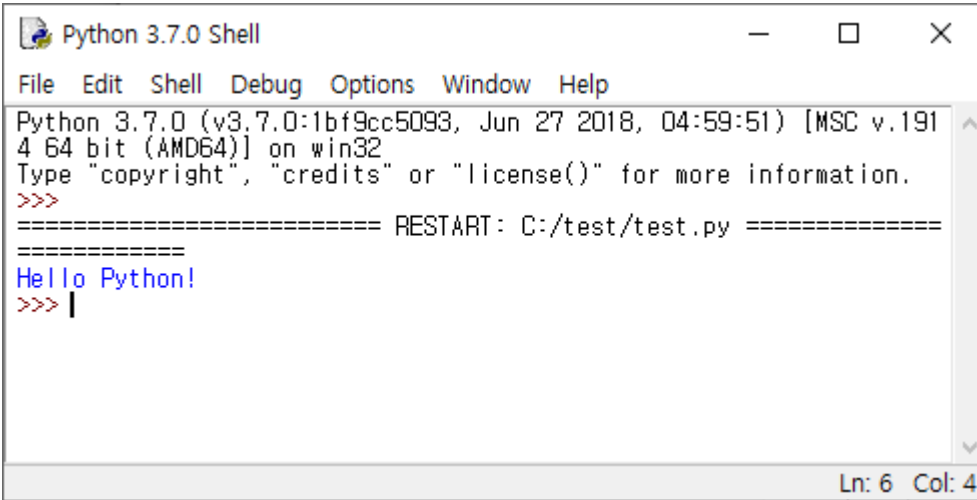


이제 Run -> Run Module 을 눌러 프로그램을 실행시켜봅시다.

실행 버튼을 누르게 되면, 이러한 창이 뜨게 됩니다.



확인을 눌러 원하는 경로와 이름으로 저장을 해줍니다.



저장을 완료하게 된다면 이렇게 실행 결과가 출력됩니다.

앞으로 간단한 예제는 인터프리터에 입력해 결과를 바로 확인하고, 여러 줄을 한꺼번에 작성하거나 여러 번 사용해야 하는 프로그램은 IDLE 에디터에서 작성하게 될 것 입니다.



# 3

## 자료형

자료형은 자료의 속성 중 하나로, 컴퓨터와 프로그래머에게 어떤 종류의 자료를 다루고 있는지 알려줍니다. 이는 해당 자료형에 할당할 수 있는 값, 가능한 연산과 명령, 그리고 저장하는 방식을 제한합니다.

# 3-1 파이썬 자료형의 특성

## 자료형의 특성

기본 자료형을 알아보기 전, 자료형의 기본적인 특성들을 알아보시다. 이 특성들을 이해하고 숙지하면 파이썬의 여러 자료형을 적재적소에 잘 활용할 수 있습니다.

### Mutable / Immutable 자료형

다른 언어와 달리, 파이썬의 자료형에는 요소의 값을 변경할 수 있는 것과 없는 것으로 나뉘는데, 각각 mutable 자료형, immutable 자료형으로 불립니다. Mutable 자료형의 값은 사용자가 얼마든지 바꿀 수 있으나, immutable 자료형의 값은 바꿀 수 없습니다. 다만 원래 가리키던 값의 주소를 다른 값의 주소로 갱신할 수 있습니다.  
숫자형, 문자열, 튜플 레인지 자료형이 immutable에 해당합니다.



## 자료형의 갈래

자료형들은 크게 시퀀스(Sequence), 매핑(Mapping), 집합(Set) 자료 구조로 나뉩니다.

시퀀스는 데이터를 순서 있게 나열한 자료 구조로, 연속한 메모리 공간에 데이터를 저장합니다. 이런 특성 덕에 특정 데이터의 위치를 참조할 수 있습니다.

매핑은 데이터 저장 시 키(Key)값과 Value(변수)끼리 짝 지어 저장하는 자료 구조입니다.

집합은 순서나 키(Key)가 존재하지 않는 자료형입니다. 이름에서도 알 수 있듯이 수학의 집합 개념이므로 수학적 집합 연산이 가능합니다. 또한 동일한 값의 데이터를 중복적으로 저장할 수 없습니다.

시퀀스 :                    [0] [1] [2] [3] ...  
                              값  값  값  값  ...

매핑 :    {키<sub>1</sub> : 값<sub>1</sub>, 키<sub>2</sub> : 값<sub>2</sub>, 키<sub>3</sub> : 값<sub>3</sub>, ... }

집합 :    {Immutable 값<sub>1</sub>, Immutable 값<sub>2</sub>, ... }

## 3-2 기본 자료형

### 파이썬의 기본 자료형

파이썬의 기본 자료형에는 정수형, 실수형, 그리고 문자열이 존재합니다. 이들 모두 요소의 값을 변경할 수 없는 immutable 자료형입니다.

정수형은 -2, 0, 5 같은 음의 정수, 0, 양의 정수형 데이터를 저장하는데 사용됩니다.

실수형은 3.14 같은 정수가 아닌 유리수로 이루어진 데이터를 저장하는데 사용됩니다.

그리고 문자열 자료형은 Immutable인 시퀀스(Sequence) 자료형, 즉 값을 바꿀 수 없는 연속적인 문자들의 나열 구조의 자료형입니다.

참고로 파이썬에서는 문자 한 개만을 저장하는 자료형이 존재하지 않습니다. 그래서 문자 하나인 경우도 문자열로 처리하게 됩니다.

정수: -3, -2, -1, 0, 1, 2

실수: 3.14, -1.0, 0.0

문자열: 

H	e	l	l	o
---	---	---	---	---

## 3-3 리스트

---

### 리스트

리스트는 Mutable인 시퀀스(Sequence) 자료형입니다. 이름 그대로 여러 요소들을 연속적으로 저장하는 목록이라는 뜻이며, C/C++에서의 배열 개념과 유사합니다. 시퀀스 자료형이기 때문에 각각의 요소마다 인덱스(위치)가 존재합니다.

리스트에는 다음과 같은 큰 특징이 있습니다.

- 대괄호 [ ]로 값을 묶습니다.
- 각각의 요소들은 심표로 구분합니다.
- 어떠한 자료형도 리스트에 저장할 수 있습니다.
- 서로 다른 자료형을 저장할 수 있습니다.

특히 서로 다른 자료형을 저장할 수 있다는 점이 기존의 배열 개념과의 큰 차이점입니다.

인덱스 : [0] [1] [2] ...

리스트 변수 = [값<sub>1</sub>, 값<sub>2</sub>, 값<sub>3</sub>, ...]

# 리스트 연산

리스트는 여러가지 시퀀스 연산이 가능합니다. 한 번 다양한 연산들을 알아봅시다.

## 리스트끼리의 연산

- (리스트) + (리스트) : 각각의 리스트를 하나의 리스트로 합칩니다.
- (리스트) \* N : 해당 리스트 요소들을 자연수 N번만큼 복제합니다.

## 멤버 연산자

- (변수) in (리스트) : 변수가 리스트의 요소인지 확인합니다. 있으면 True, 없으면 False를 반환합니다.
- (변수) not in (리스트) : 변수가 리스트의 요소가 아닌지 확인합니다. In과 반대로, 없으면 True, 있으면 False를 반환합니다.

## 시퀀스 연산

- 인덱싱 : (리스트)[N] / N번 인덱스에 위치한 값을 반환합니다.
- 슬라이싱 : (리스트)[N:M] / N번 인덱스부터 (M - 1)번 인덱스까지의 값들을 반환합니다.
- 리스트 간격 : (리스트)[A:B:C] / A번부터 B번 인덱스에 위치한 값을 (C - 1)만큼 생략하면서 반환합니다.  
※(리스트)[::-1] : 역순으로 반환합니다.

## 중첩 리스트

중첩 리스트는 리스트 속에 또 리스트가 있는 형태입니다. 다른 언어에서의 이차원 배열의 개념과 유사합니다.

중첩 리스트도 인덱싱, 슬라이싱 등의 연산이 가능합니다. 다만, 일차원 리스트가 아니기 때문에 인덱스 번호에 유의하여 코드를 작성하도록 합니다.

인덱스 : [0][0] [0][1] [0][2] [1][0] [1][1] [1][2] . . .

중첩 리스트 변수 = [ [값<sub>1</sub>, 값<sub>2</sub>, 값<sub>3</sub>], [값<sub>1</sub>, 값<sub>2</sub>, 값<sub>3</sub>], . . . ]

---

## Quiz!

- 리스트는 서로 다른 어떠한 자료형도 모두 저장이 가능합니다. — ○ / X
- 리스트 [2, 3, 4, 5, 6, 7, 8, 9]가 있습니다. 리스트를 [2:6]로 슬라이싱하여 나오는 결과값들의 합을 구하세요.
- 중첩 리스트 [ ['a', 'b', 'c'], [1, 2, 3, 4, 5] ]가 있습니다. 중첩 리스트 [1][2]에 위치한 요소는 무엇인가요?

답: ○, 22, 3

# 리스트 메소드

리스트 메소드란, 리스트에서만 사용 가능한 함수를 말합니다.

## 요소 추가 메소드

- (리스트).append(값) : 리스트 맨 끝에 요소를 추가합니다.
- (리스트).insert(인덱스, 값) : 해당 인덱스에 값을 추가합니다. 원래 지점에 있던 값들은 뒤로 밀려납니다.
- (리스트).extend(또 다른 리스트) : 기존 리스트에 또 다른 리스트를 추가하여 확장시킵니다. (리스트)+(리스트) 연산과 비슷합니다.

## 요소 삭제 메소드

- (리스트).pop() : 리스트 맨 끝의 요소를 삭제합니다.
- (리스트).remove(값) : 해당 값을 리스트에서 삭제합니다.
- (리스트).clear() : 리스트를 비웁니다.

## 요소 검색 메소드

- (리스트).count(값) : 해당 값의 개수를 반환합니다.
- (리스트).index(값) : 해당 값의 인덱스 위치를 반환합니다.

## 요소 정렬 메소드

- (리스트).sort() : 문자 및 숫자 요소를 오름차순으로 정렬합니다. 단, 같은 자료형만 가능합니다.



## 기타 메소드

- (리스트).reverse(): 요소들을 역순으로 출력합니다. 단, 내용은 변경되지 않습니다.
- 값.join(리스트): 기존 리스트 속의 요소들을 문자열 '요소+값+요소...'로 출력 및 반환합니다.
- 리스트.split(값): '요소+값+요소...'에서 해당 값을 삭제하고, 리스트 형태인 [요소,요소...]로 출력 및 반환합니다.

## 통계 함수

- Sum(리스트): 요소들의 합을 반환합니다.
- Min(리스트): 요소들 중 최솟값을 반환합니다.
- Max(리스트): 요소들 중 최댓값을 반환합니다.

## 복사 메소드

※ == : 비교 연산자이며, 두 변수의 내용(값)을 비교합니다.

※ is : identity 연산자이며, 두 변수가 같은 주소를 가리키는지의 여부를 판단합니다.

- 슬라이싱 : 리스트<sub>1</sub> = 리스트<sub>2</sub>[N:M] / 리스트를 슬라이싱 하여 다른 리스트 변수에 요소들을 복사하여 저장합니다.
- 얇은 복사 : 리스트<sub>1</sub> = copy.copy(리스트<sub>2</sub>) / 중첩 리스트인 경우 가리키는 주소는 다르나, 값을 공유합니다.
- 깊은 복사 : 리스트<sub>1</sub> = copy.deepcopy(리스트<sub>2</sub>) / 위의 경우에서 값을 공유하지 않습니다. copy 함수와의 큰 차이점입니다.

중첩 리스트 A → [ [값<sub>1</sub>, 값<sub>2</sub>, 값<sub>3</sub>], [값<sub>1</sub>, 값<sub>2</sub>, 값<sub>3</sub>] ]  
중첩 리스트 B → [ [값<sub>1</sub>, 값<sub>2</sub>, 값<sub>3</sub>], [값<sub>1</sub>, 값<sub>2</sub>, 값<sub>3</sub>] ]

↑ 중첩 리스트인 경우, 얇은 복사를  
할 시 서로 참조하는 주소, 즉 id는  
다르지만 내부의 리스트를 서로  
공유하는 문제점이 생깁니다.

## 3-4 기타 자료형

---

### 튜플(Tuple)

튜플은 Immutable한 시퀀스 자료형입니다. 구조는 리스트랑 다르지 않지만 내용 변경 가능 여부가 다릅니다. 그래서 읽기 전용 리스트라고도 합니다. 튜플은 immutable이기 때문에 튜플에 mutable 요소는 저장할 수 없습니다.

### 딕셔너리(Dictionary)

딕셔너리는 Mutable한 매핑 자료형입니다. 앞서 말했듯이 매핑은 키와 값이 서로 유의미하게 대응되는 자료 구조입니다. 주의할 점은, 키값으로 immutable한 값을 저장해야 합니다.

### 집합(Set)

집합은 Mutable한 집합 자료형입니다. 중복되는 데이터는 저장 불가능하며, 순서와 키 개념이 없습니다. 데이터의 연속성과 순서가 없기 때문에 값 참조를 할 수 없습니다. 또한 삭제 메소드를 호출할 시 어떤 데이터가 삭제될지 모릅니다.

## 3-5 퀴즈

---

1) 다음 중 자료형에 관해 옳은 설명을 고르세요.

1. 파이썬의 자료형의 자료구조는 크게 시퀀스(Sequence), 딕셔너리(Dictionary), 그리고 집합(Set)으로 나뉜다.
2. 튜플 자료형은 인덱싱, 슬라이싱 등의 시퀀스 연산을 통해 값을 참조하는 동시에 요소도 변경이 가능하다.
3. 집합 자료형은 특정 요소의 인덱스 번호를 참조할 수 있다.
4. 리스트 자료형에는 서로 다른 자료형의 요소들을 저장할 수 있다.
5. 딕셔너리 자료형의 키값으로는 mutable한 값을 저장할 수 있다.

2) 다음 코드의 결과를 작성하세요.

```
>> list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>> result = list[1, 7]
>> result.insert(3, 10)
>> print(max(result))
```

3) 다음 중 리스트의 내용을 변경하지 않는 메소드를 고르세요.

1. .pop()
2. .remove()
3. .index()
4. .extend()
5. copy.deepcopy()

답: 4, 10, 3

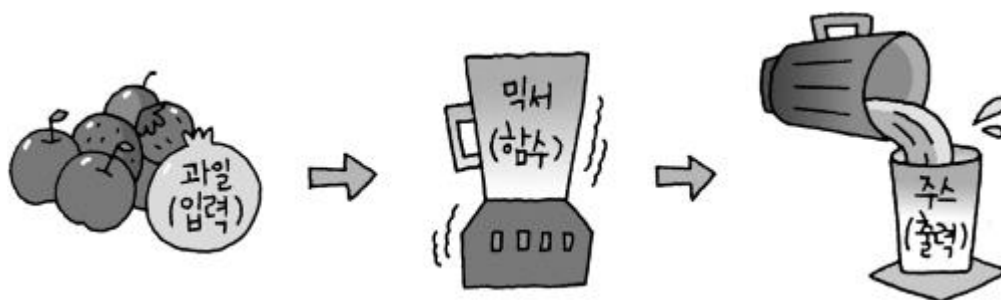
# 4

## 함수와 입출력

함수와 입출력에 대해 알아보자.

## 4-1 함수란?

함수를 설명하기 전에 믹서를 생각해 보자. 우리는 믹서에 [과일]을 넣는다. 그리고 [믹서]를 사용해서 과일을 갈아 과일 [주스]를 만든다. 우리가 믹서에 넣는 과일은 [입력]이 되고 과일 주스는 [출력 or 결과값]이 된다.



그렇다면.. [믹서]는 무슨 역할을 할까?  
이 믹서의 역할이 바로 [함수]이다!

### 함수를 왜 쓸까?

- ➔ 프로그램의 흐름을 일목요연하게 보여준다.
- ➔ 반복적으로 수행해야 할 내용을 쉽게 바꿔준다.

## 4-2 print()

---

### print() 함수란?

➔ 자기가 원하는 결과물을 출력시키는 장치이다.

### print() 함수의 사용법?

➔ 예를 들어,  
Hello World! 라고 출력하고 싶다면,

```
print('Hello World!')
```

라고 쓰면 된다.

즉, print() 의 소괄호 안에 큰따옴표(“)나 작은따옴표(‘)로 넣을 글자를 감싸면 된다.

### 사용 예시

```
print('Hello', end=' ')\nprint('Python')\nprint('Hello', end='&&&')\nprint('Python')\nprint('Hello', 'Python', sep='#')\nf = open('dump.txt', 'w')\nprint('Hello Python', file=f)\nf.close()\nimport sys\nprint('Hello Python', file=sys.stderr)
```

## 4-3 input()

---

### input() 함수란?

➔ 입력하고 싶은 내용을 저장시키는 장치이다.

### input() 함수의 사용법?

➔ 예를 들어,  
어떤 값을 입력받고 싶다면,

```
a = input()
```

라고 쓰면 된다.

이러한 경우 입력된 값과 관계없이 모두 문자열로 입력된다.

### 사용 예시

#### 정수 입력

```
a = int(input())
```

#### 실수 입력

```
a = float(input())
```

#### 입력 및 출력

```
a = input('값을 입력해주세요.')
```

이 코드는 아래 코드와 동일하다.

```
print('값을 입력해주세요.')
```

```
a = input()
```

## 4-4 pow()

---

### pow() 함수란?

➔ 제공하는 연산을 보다 쉽게 만들어주는 함수

### 사용 예시

pow(2, 3) -> 2 \* 2 \* 2

### 기타(자주 쓰이는 함수)

--숫자, 문자열--

print(a)

a.replace('찾을문자열','바꿀문자열')

a.split('기준 문자')

--리스트--

a.append('추가할 데이터')

a.extend(['추가할 데이터들'])

del a[index]

--딕셔너리--

a.keys()

a.values()

a.items()

--셋--

a.add('추가할 데이터')

a.update(['추가할 데이터들'])

a.remove('삭제할 데이터')



## 4-5 퀴즈

---

### 문제

1

Hello World!를 출력하시오

2

자신이 입력한 값을 그대로 출력하시오.

3

‘ “를 출력하시오.

### 정답

1

```
print('Hello World!')
```

2

```
A = input()
print(A)
```

3

```
print('"' + ' ' + '"')
```

# 5

## 연산

더하기, 빼기, 나누기, 곱하기, 몫과 나머지, 같은 수를 여러 번 곱하는 경우 등을 Python에서 어떻게 구현하는지 알아보시다.

# 5-1 사칙연산

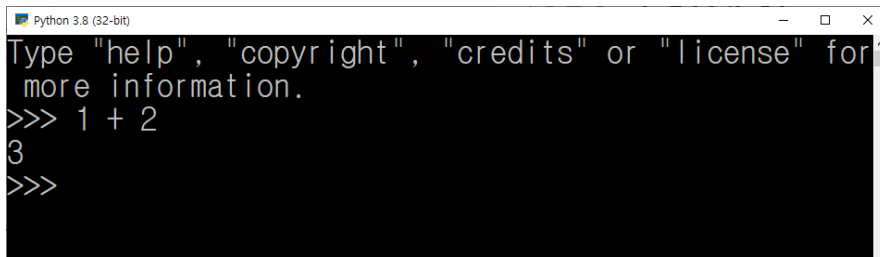
## + 연산자

셀에 다음과 같이 작성해보자!

1 + 2

3이 출력 될 것이다.

+ 연산자는 앞에 있는 숫자와 뒤에 있는 숫자를 더해준다.



```
Python 3.8 (32-bit)
Type "help", "copyright", "credits" or "license" for
more information.
>>> 1 + 2
3
>>>
```

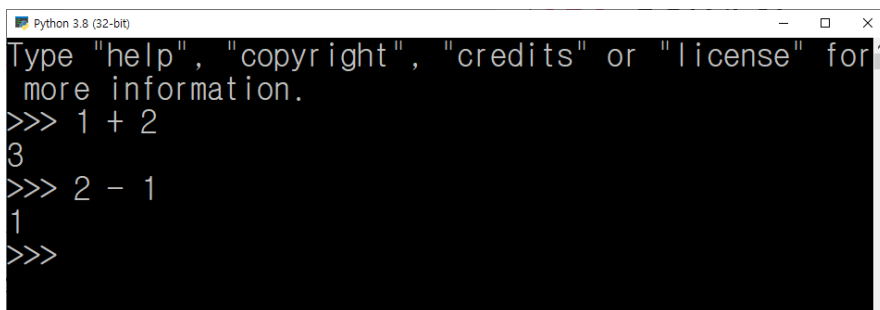
## - 연산자

셀에 다음과 같이 작성해보자!

2 - 1

1이 출력 될 것이다.

- 연산자는 앞에 있는 숫자와 뒤에 있는 숫자를 빼준다.



```
Python 3.8 (32-bit)
Type "help", "copyright", "credits" or "license" for
more information.
>>> 1 + 2
3
>>> 2 - 1
1
>>>
```

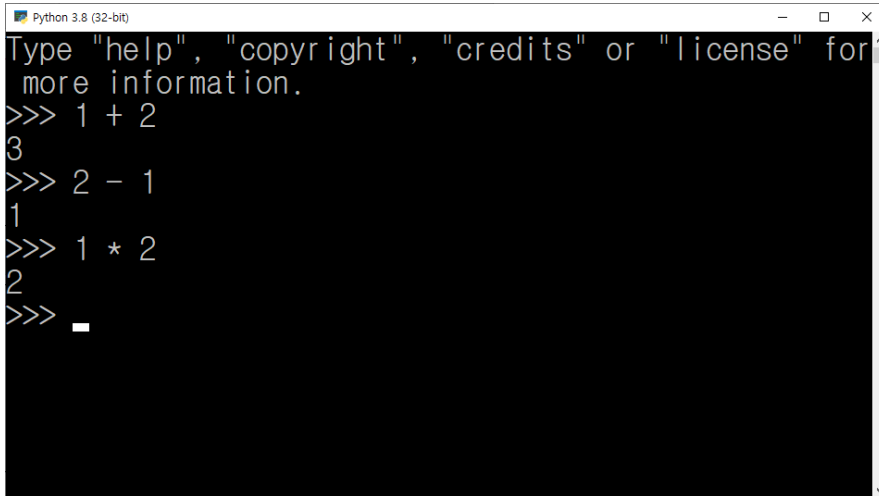
## \* 연산자

셀에 다음과 같이 작성해보자!

1 \* 2

2가 출력 될 것이다.

\* 연산자는 앞에 있는 숫자와 뒤에 있는 숫자를 곱해준다.

A screenshot of a Python 3.8 (32-bit) terminal window. The window title is "Python 3.8 (32-bit)". The prompt is "Type 'help', 'copyright', 'credits' or 'license' for more information." The user has entered three commands: ">>> 1 + 2", ">>> 2 - 1", and ">>> 1 \* 2". The corresponding outputs are "3", "1", and "2". The prompt ">>> \_" is visible at the bottom.

```
Python 3.8 (32-bit)
Type "help", "copyright", "credits" or "license" for
more information.
>>> 1 + 2
3
>>> 2 - 1
1
>>> 1 * 2
2
>>> _
```

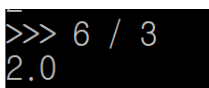
## / 연산자

셀에 다음과 같이 작성해보자!

6 / 3

2.0이 출력 될 것이다.

\* 연산자는 앞에 있는 숫자와 뒤에 있는 숫자를 나눴준다.

A screenshot of a Python 3.8 (32-bit) terminal window showing the division operation. The prompt is "Type 'help', 'copyright', 'credits' or 'license' for more information." The user has entered the command ">>> 6 / 3". The output is "2.0".

```
Python 3.8 (32-bit)
Type "help", "copyright", "credits" or "license" for
more information.
>>> 6 / 3
2.0
```

## 5-2 몫과 나머지

### 몫을 구하는 방법

나누기는 / 연산자를 통해 할 수 있지만 몫만 구하는 방법은 무엇일까?

셀에 다음과 같이 작성해보자!

```
4 // 3
```

1이 출력 될 것이다.

// 연산자는 앞에 있는 숫자와 뒤에 있는 숫자를 나눠서 나온 몫을 알려준다.

```
>>> 4 // 3
1
```

### 나머지를 구하는 방법

// 연산자를 통해 몫을 구하는 방법을 배웠다. 그렇다면 나머지는 어떻게 구할까?

셀에 다음과 같이 작성해보자!

```
4 % 3
```

1이 출력 될 것이다.

% 연산자는 앞에 있는 숫자와 뒤에 있는 숫자를 나눠서 나온 나머지를 알려준다.

```
>>> 4 % 3
1
>>>
```

## 5-3 거듭제곱

### 거듭제곱을 해보자

Python에선 거듭제곱을 어떻게 할까? 가장 먼저 생각나는 방법을 시도해보자!

셀에 다음과 같이 작성해보자!

```
2 * 2 * 2
```

8이 출력 될 것이다.

이미 배웠던 \* 연산자를 이용해서 2를 3번 곱해서 거듭제곱을 구현하였다.

### \*\* 연산자

\* 연산자와 비슷해 보이는 \*\* 연산자는 무엇일까?

셀에 다음과 같이 작성해보자!

```
2 ** 3
```

8이 출력 될 것이다.

\*\* 연산자는 앞에 있는 숫자를 뒤에 있는 숫자만큼 제공해 준다.

```
>>> 2 * 2 * 2
8
>>> 2 ** 3
8
>>> 
```

# Pow함수

Pow함수란 무엇일까? 그리고 어떻게 쓰는 걸까?

셀에 다음과 같이 작성해보자!

```
pow(2, 3);
```

8이 출력 될 것이다.

Pow함수는 첫 번째에 넣어준 숫자를 두 번째에 넣은 숫자만큼 제공해 준다.

```
>>> pow(1,2);  
1  
>>> pow(2,3);  
8
```



# 문자열

문자열의 의미를 알고 활용할 수 있다.



# 6-1 문자열이란?

---

## 문자열(String)

보통 프로그래밍에서는 'a', 'b'와 같이 알파벳 글자 하나를 문자라고 부르며, 'python'같이 두 개 이상의 문자로 구성된 것을 문자열(String)이라고 합니다.

즉, 문자열(String)이란 문자, 단어, 숫자 등으로 구성된 문자들의 집합을 의미합니다. 다시 말해 문자들을 나열한 것으로 생각할 수 있습니다.

## 6-2 문자열 활용하기

---

문자열을 만드는 방법은 기본적으로 4가지가 있습니다.

1. 큰따옴표(“)로 둘러싸기

“Hello world”

2. 작은따옴표(‘)로 둘러싸기

‘I am happy.’

3. 큰따옴표 3개 연속(“”)하여 사용하기

“””Life is too short, you need python”””

4. 작은따옴표 3개 연속(‘’)하여 사용하기

’’’Life is too short, you need python’’’

파이썬의 큰 특징은 단순함인데 왜 문자열 만드는 방법을 4가지나 만들었을까요?

먼저 1, 2번 같은 경우에는 문자열 내에 큰따옴표나 작은따옴표를 사용하는 경우를 위하여 존재합니다.

만약 작은따옴표(‘)로 둘러쌌는데 문자열 내에 ‘가 존재하면 컴퓨터는 문자열 닫았다고 생각할 것입니다. 예를 들어

‘I’m happy.’

라고 작성하였을 경우 컴퓨터는 문자열을 I 밖에 모르는 것입니다. 큰따옴표도 마찬가지라고 생각하면 됩니다.

큰따옴표를 포함하기 위해서는 작은따옴표로 둘러싸고 작은따옴표를 사용하기 위해서는 큰따옴표로 둘러싸면 되는 것입니다. 하지만 따옴표를 포함하고 싶을 때 역 슬래시(**\**)를 사용한다면 신경 쓰지 않아도 괜찮습니다.

**'I\****'m happy'**  
**"\****"python is very easy\****" he says."**

그렇다면 3, 4번처럼 3개를 연속해서 사용하는 것은 무엇을 위함일까요?

먼저 문자열을 작성할 때 문자열이 항상 한 줄은 아닐 것입니다. 그렇기에 우리는 여러 줄의 문자열을 변수에 대입해야 하는 경우가 생길 수 있습니다. 그럴 때에 우리는 줄 바꿈 문자인 **\n**을 사용할 수 있습니다.  
예를 들어

**Life is too short**  
**You need python**

이라는 2줄의 문자열을 대입한다고 할 때 우리는

**needpython = 'Life is too short\nYou need python'**

이라고 작성하면 우리는 2줄의 문자열을 얻을 수 있습니다. 하지만 줄이 더 많거나 내용이 더 길 경우 **\n**의 사용이 늘어날 것이고 이로 인해 읽기가 불편해지고 줄이 더 길어질 것입니다. 이러한 점을 보완하고자 3, 4번의 방식을 사용하는 것입니다.

**needpython = "**  
**Life is too short**  
**You need python**  
**"**

이런 식으로 작성했을 때 **\n**을 사용하는 것보다 훨씬 읽기 편해졌다는 것을 느낄 수 있을 것입니다.

## 이스케이프 코드

여러 줄의 문장을 처리할 때 역 슬래시 + 소문자 n의 조합인 `\n`을 사용하였는데 이처럼 역 슬래시 + 문자의 조합을 이스케이프 코드라고 합니다. 이스케이프 코드는 프로그래밍을 할 때 사용할 수 있도록 미리 정의해둔 것으로 주로 출력물을 보기 좋게 정렬하는 용도로 사용됩니다.

<code>\n</code>	문자열 안에서 줄을 바꿀 때 사용
<code>\t</code>	문자열 사이에 탭 간격을 줄 때 사용
<code>\\</code>	문자 <code>\</code> 를 그대로 표현할 때 사용
<code>\'</code>	작은따옴표(')를 그대로 표현할 때 사용
<code>\"</code>	큰따옴표(")를 그대로 표현할 때 사용
<code>\r</code>	캐리지 리턴(줄 바꿈 문자, 현재 커서를 가장 앞으로 이동)
<code>\f</code>	폼 피드(줄 바꿈 문자, 현재 커서를 다음 줄로 이동)
<code>\a</code>	벨 소리(출력할 때 PC 스피커에서 '뽕' 소리가 난다)
<code>\b</code>	백 스페이스
<code>\000</code>	널 문자

이중에서 `\n`, `\t`, `\\`, `\'`, `\"`이 주로 사용 되고 나머지는 잘 사용되지 않습니다.

# 문자열 연산하기

파이썬에서는 문자열을 더하거나 곱할 수 있습니다. 다른 언어에서는 쉽게 볼 수 없는 기능으로, 파이썬의 특징이 드러나는 파이썬의 장점이라고 볼 수 있습니다.

## 문자열 더하기

```
print('python'+'is fun') 와  
head = 'python'  
tail = 'is fun'  
print(head + tail)
```

의 출력 결과는 모두 **python is fun**으로 동일합니다. 이처럼 +표시로 문자열을 연결 할 수 있습니다.

## 문자열 반복

```
print('python'*2) 와  
a = 'python'  
print(a*2)
```

의 출력결과는 모두 **pythonpython** 입니다. 문자열에서의 \*는 숫자 곱하기와의 의미와 다르게 문자열의 반복을 의미합니다.

## 문자열 길이 구하기

문자열의 길이는 len함수를 이용하여 쉽게 구할 수 있습니다. len함수는 내장함수로 별다른 설정없이 이용할 수 있습니다.

```
a = 'python is fun'  
len(a)  
>>>13
```

## 멤버 연산자

멤버 연산자는 문자열에 해당 문자열이 있는지 검사합니다  
in은 포함하는지 검사하는 연산자이고, not in은 포함하지 않는지 검사합니다.

```
print('p' in 'python')  
print('j' not in 'python')  
>>>True  
>>>True
```

# 문자열 인덱싱과 슬라이싱

인덱싱(Indexing)이란 무엇인가를 "가리킨다"는 의미이고, 슬라이싱(Slicing)은 무엇인가를 "잘라낸다"는 의미입니다. 이런 의미를 생각하면서 다음 내용을 살펴봅시다.

## 문자열 인덱싱

문자열 인덱싱은 문자열에서 특정한 위치의 문자를 읽을 때 사용합니다.

인덱스는 0부터 시작하고 연산자 []를 사용합니다.

만약 음수 인덱스를 사용할 경우 뒤에서부터 접근합니다. 즉, -1은 뒤에서 첫번째라는 뜻으로 만약 17자리 문자열일 경우 인덱스 16과 -1은 같은 문자를 의미합니다.

```
a = 'Python uses zero-based index'
print(a[0])
print(a[1])
print(a[-1])
print(a[-2])
>>>P
>>>y
>>>x
>>>e
```

## 문자열 슬라이싱

문자열 인덱싱에서 문자를 추출하였다면 슬라이싱에서는 문자열을 추출합니다.

- 문자열[start:stop]은 start~stop-1에 해당하는 문자열을 반환합니다.
- 문자열[:]은 처음부터 끝까지 자릅니다.
- 문자열[start:]은 start부터 끝까지 자릅니다.
- 문자열[:stop]은 처음부터 stop-1까지 자릅니다.

```
print(a[:7])
print(a[7:])
print(a[:])
print(a[:-1])
>>>Python
>>>uses zero-based index
>>>Python uses zero-based index
>>>Python uses zero-based inde
```

## 6-3 문자열 포매팅

문자열에서 또 중요한 것은 문자열 포매팅(formatting)입니다. 만약 여러분이 현재의 기온을 알려주는 프로그램을 만들었다고 합시다. 그렇다면 “현재 기온은 20도입니다.”라고 출력할 것이고 시간이 지나면 기온이 바뀌어 “현재 기온은 18도입니다.”라고 출력할 것입니다. 위 2문자열은 모두 같은데 기온을 나타내는 숫자만 다릅니다. 이렇게 문자열에서 특정한 값을 바꿔야하는 경우가 있을 때 이를 가능하게 해주는 것이 문자열 포매팅입니다. 다시 말해 문자열 포매팅이란 문자열 안에 어떤 값을 삽입하는 것입니다.

### %-formatting

%연산자를 사용하는 경우 입니다. %d : 정수, %f : 실수, %c : 문자 1개, %s : 문자열 등이 사용 됩니다.

```
print('I ate %d apples' %3)
print('I ate %d %s' %(3, 'apples'))
>>>I ate 3 apples
>>>I ate 3 apples
```

### {}.format() 메소드

필요한 위치에 {}를 삽입하여 {}에 번호를 기입하거나, 변수를 지정하여 사용할 수 있습니다.

```
print('{} and {}'.format('You', 'I'))
print('{0} and {1} and {0}'.format('You', 'I'))
print('I ate {number} {food}'.format(number=3,food=apples'))
>>>You and I
>>>You and I and You
>>>I ate 3 apples
```

# 7

## 조건문

‘돈이 있으면 버스를 타고, 없으면 걸어 간다.’와 같은 조건이 있을 때,  
돈이 있는지 없는지 판단하고 버스를 탈지 걸어갈지 판단할 수 있다.

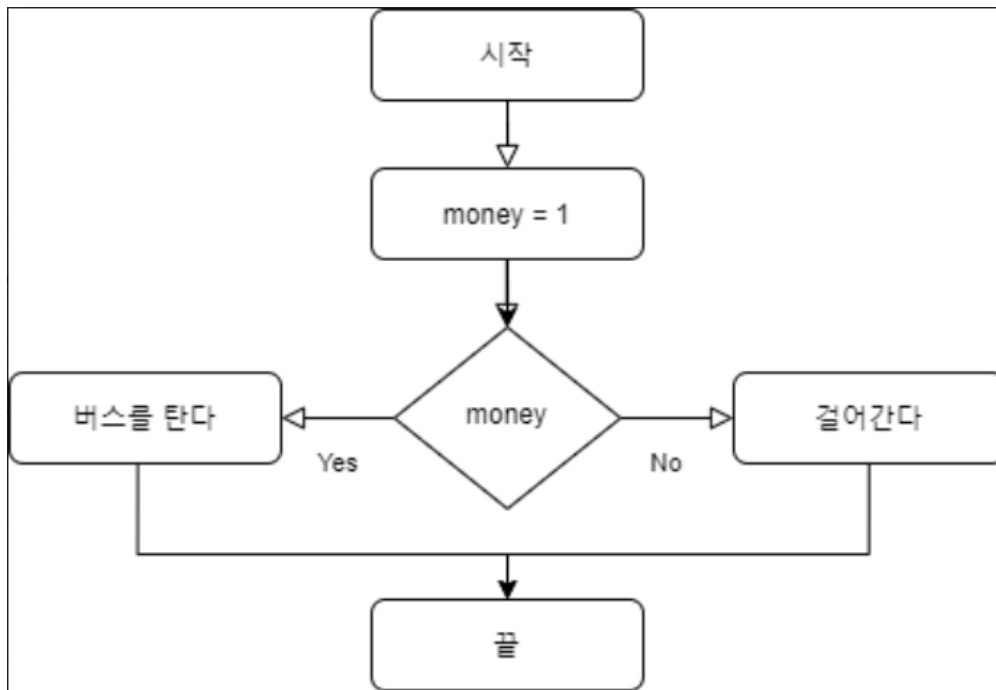


## 7-1 조건에 따라 코딩하기

‘돈이 있으면 버스를 타고, 없으면 걸어 간다.’ 라는 조건을 다음과 같이 코딩할 수 있다.

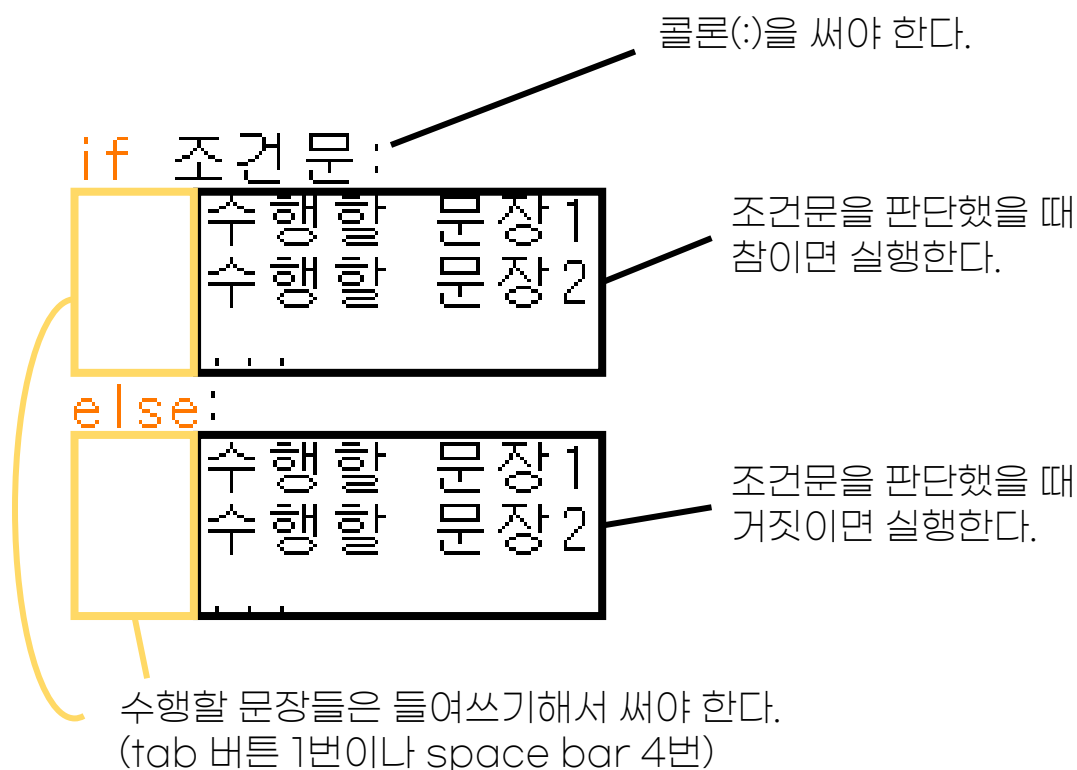
```
money = 1
if money:
    print("버스를 타고 간다")
else:
    print("걸어간다")
```

money = 1 은 돈이 있다는 것을 의미한다.



# 7-2 if, else

조건문에는 다양한 것들이 있는데 그 중에서도 if,else문은 기본이다.  
기본 구조는 다음과 같다.



if 다음에 사용하는 조건문에는 어떤 것들이 있을까?

```
money = 1
if money:
    print("버스를 타고 간다")
else:
    print("걸어간다")
```

자료형	참	거짓
숫자	0이 아닌 숫자	0
리스트	[]이 아닌 리스트	[]
문자열	""이 아닌 문자열	""
튜플	()이 아닌 튜플	()
딕셔너리	{ }이 아닌 딕셔너리	{ }

앞쪽에서 봤던 예제를 보면 조건문에 money라고 쓰여져 있다. 코드의 첫번째 줄에서 money = 1이라고 정의를 했기 때문에 조건문은 참이 되고 결과는 '버스를 타고 간다'가 출력되는 것이다.

조건문에는 비교 연산자를 많이 사용한다. 다음은 비교 연산자이고, 이 비교 연산자들은 True 나 False를 반환한다.

비교연산자	설명
X>y	X가 y보다 크다
X>=y	X가 y보다 크거나 같다
X<y	X가 y보다 작다
X<=y	X가 y보다 작거나 같다
X!=y	X가 y와 다르다
X==y	X가 y와 같다

앞에서 살펴봤던 예제를 비교연산자를 사용해서  
'돈이 1000원 이상 있으면 버스를 타고, 없으면 걸어 간다.' 를 코딩해 보자

```
>>> x=3
>>> y=5
>>> x!=y
True
>>> x==y
False
>>> x<y
True
>>> x>=y
False
>>> |
```

x를 3으로 초기화했다.

y를 5으로 초기화했다.

x는 3, y는 5로 서로  
다르기 때문에 참 반환

x, y는 서로 같지 않기  
때문에 거짓 반환

x가 y보다 작기  
때문에 참 반환

x가 y보다 크거나  
같지 않기 때문에  
거짓 반환

돈이 2000원 있을 때, '돈이 1000원 이상 있으면 버스를 타고, 없으면 걸어 간다.' 를 다음과 같이 코딩할 수 있다.

```
money = 2000
if money>=1000:
    print("버스를 타고 간다")
else:
    print("걸어간다")
```

if, else문만 사용했을 때, 조건문을 단 하나만 판별해서 참, 거짓일 때 수행할 문장을 지정할 수 있다. 하지만 여러 조건을 판별하고 싶다면 어떻게 해야 할까?

## 7-3 elif

---

조건을 여러 개 판별하고 싶으면 elif를 사용하면 된다.

돈이 900원 있고 교통카드에 돈이 1900원 있고 카드에 돈이 9900원 있을 때  
'돈이 1000원 이상 있으면 버스를 타고,  
교통카드에 2000원 이상 있으면 지하철을 타고,  
카드에 10000원 이상 있으면 택시를 타고,  
없으면 걸어 간다.'를 다음과 같이 코딩할 수 있다.

```
money = 900
traffic_card = 1900
card = 9900
if money >= 1000:
    print("버스를 타고 간다")
elif traffic_card >= 2000:
    print("지하철을 타고 간다")
elif card >= 10000:
    print("택시를 타고 간다")
else:
    print("걸어간다")
```

## 7-4 연습문제

---

1

돈이 9000원 있고 카드에 30000원이 있을 때  
'돈이 10000원 이상 있으면 돈을 내고, 카드에 25000원 이상 있으면 카드를  
내고, 없으면 옷가게를 나간다' 를 코딩하시오.

2

수학 점수가 78점일 때  
'점수가 90~100 이면 A,  
점수가 80~89 이면 B,  
점수가 70~79 이면 C,  
점수가 60~69 이면 D를 출력하는 프로그램'  
을 코딩하시오.

```
money = 9000
card = 30000
if money >= 10000:
    print("돈을 낸다")
elif card >= 25000:
    print("카드를 낸다")
else:
    print("옷가게를 나간다")
```

```
math_score = 78
if math_score >= 90:
    print("A")
elif math_score >= 80:
    print("B")
elif math_score >= 70:
    print("C")
else:
    print("D")
```

# 8

## 반복문

반복문에 대해서 알고 활용할 수 있다.

# 8-1 While 문

---

## While 문

While안의 조건이 충족되는 동안 계속해서 수행되는 반복문이다.

흔히 얼마나 반복해야 하는지 명확하지 않을 때 주로 사용한다.

사용방법 :

→ 형식

**while** 조건식 :

수행할문장1

수행할문장2

수행할문장3

<조건문>

>> while문이 실행될 조건을 담아두는 곳이다.

이 부분이 거짓이 되기 전까지 반복을 멈추지 않는다.

While문은 조건이 충족되지 않으면 계속 반복하기 때문에 몇 가지 경우에서 특정 조건일 때 반복을 멈추거나 건너뛰어야 할 수 있다.

반복 멈추기 : break

While안의 조건이 만족되지 않았음에도 반복문을 종료시켜야 할 때가 있는데 이런 경우에는 break문을 사용하면 된다.



break

```
cnt = 0
```

```
while cnt < 10:           1
    cnt += 1              2
    if cnt == 5:          3
        break # 반복문 탈출 **** 4
    print(cnt)
```

반복 건너뛰기: continue

특정 조건일 때 실행을 무시하고 다음 반복으로 넘어가게 하려면  
continue문을 사용하면 된다.

continue

```
cnt = 0
```

```
while cnt < 10:
    cnt += 1
    if cnt == 3 or cnt == 5 or cnt == 7:
        continue
    print(cnt) # 1 2 4 6 8 9 10
```

# 무한 루프

: 프로그램이 일련의 명령을 무한히 반복하는 것

주로 프로그래밍 과정 실수가 있었거나 의도적으로 탈출 조건을 설정한 뒤 사용하기도 한다.

While문은 조건이 참이면 계속 반복하므로 무한 루프가 발생할 수 있다.

```
while True:
    print("Hello world")
```

무한 루프가 발생하면 거의 대부분 에러가 발생하거나 메모리 초과가 발생하기 때문에 무한 루프가 발생하지 않도록 조건을 잘 확인하는 것이 중요하다.

따라서, break문을 사용하여 의도적으로 무한 루프를 실행한 뒤 탈출 조건을 통해 사용할 수도 있다.

# 언제 사용할까?

## 별 출력하기

다음 예시는 뒷부분의 ‘중첩 반복문’에서 좀 더 자세하게 다루어 볼 예정이다.

```
*****      n=10
*****      i=0
*****      j=0
*****      while i<n:
*****          while j<n-i:
*****              print('*',end="")
*****              j+=1
*****              print("")
*****              i+=1
*****              j=0
*****
*****
*****
```

## 더하기 사이클

0보다 크거나 같고, 99보다 작거나 같은 정수가 주어질 때 다음과 같은 연산을 할 수 있다. 먼저 주어진 수가 10보다 작다면 앞에 0을 붙여 두 자리 수로 만들고, 각 자리의 숫자를 더한다. 그 다음, 주어진 수의 가장 오른쪽 자리 수와 앞에서 구한 합의 가장 오른쪽 자리 수를 이어 붙이면 새로운 수를 만들 수 있다. 다음 예를 보자.

26부터 시작한다.  $2+6 = 8$ 이다. 새로운 수는 68이다.  $6+8 = 14$ 이다. 새로운 수는 84이다.  $8+4 = 12$ 이다. 새로운 수는 42이다.  $4+2 = 6$ 이다. 새로운 수는 26이다.

위의 예는 4번만에 원래 수로 돌아올 수 있다. 따라서 26의 사이클의 길이는 4이다.

N이 주어졌을 때, N의 사이클의 길이를 구하는 프로그램을 작성하시오.

```
N = int(input())
n = -1
t = 0
while n != N:
    if n == -1: n = N
    n = (n//10 + n%10)%10 + (n%10)*10
    t += 1
print(t)
```

## 8-2 For문

### For문이란?

- 반복횟수가 명확하거나 컬렉션을 순회할 때 사용한다.
- 컬렉션의 첫 번째 요소부터 마지막 요소까지 차례대로 변수에 대입하면서 문장을 반복 실행한다.

```
for 변수 in 컬렉션:  
    문장
```

#### 변수

>> 컬렉션의 요소들을 0번째부터 끝까지 차례대로 저장하는 변수이다.

#### 문장

>> for문이 실행되면서 반복 실행될 명령문이다.

#### 컬렉션

>> 파이썬의 대표적인 자료형으로, 시퀀스, 매핑, 셋 등이 있다.

for문에서 흔하게 사용되는 컬렉션으로는 레인지(range) 라는 자료형이 있다.

> ‘파이썬의 기본 자료형’ 단원의 ‘컬렉션’ 부분 참고!

# 따라해보기

## 자료형 : 리스트

```
1  arraylist = [1, 2, 3, 4, 5]
2
3  for num in arraylist:
4      print(num, end=' ')
```

변수 num에 arraylist의 값  
1부터 5가 들어가고 num을  
print한다.

```
[Running] python -u "d:\sunrin\VolT\python\for_loop.py"
1 2 3 4 5
```

-> 실행 결과 (파일명 : for\_loop.py)

## 자료형 : 문자열

```
1  hello = "hello world"
2
3  for i in hello:
4      print(i, end=' ')
```

```
[Running] python -u "d:\sunrin\VolT\python\for_loop.py"
h e l l o   w o r l d
```

-> 실행 결과 : 문자열도 처음부터 끝까지 한 글자씩!

## 자료형 : 딕셔너리

```
1  dic = {
2      "name": "홍길동",
3      "phone": "010-1234-5678",
4      "job": "student",
5      "age": 18,
6  }
7
8  for k in dic:
9      print(k, end=' ')
```

```
[Running] python -u "d:\sunrin\VolT\python\for_loop.py"
name phone job age
```

※ 딕셔너리 사용 주의 : for문 안의 k변수에는 딕셔너리의 key부분만 저장됨

### ● 생각해보기1 :

- 딕셔너리의 value 부분을 가져오려면 위의 코드를 어떻게 수정해야 할까?

### ○ 생각해보기1 예시 :

```
for k in dic:
    print(dic[k], end=' ')
```

k 대신 dic[k]으로 수정해서 value값을 가져올 수 있다.

## 자료형 : 레인지(range)

```
1  for i in range(1, 5):
2      print(i, end=' ')
```

```
[Running] python -u "d:\sunrin\VolT\python\for_loop.py"
1 2 3 4
```

레인지 사용 방법 (시작 값 ~ 끝 값 - 1)

```
1. range(<시작 값>, <끝 값>)
2. range(<끝 값>)
3. range(<시작 값>, <끝 값>, <간격>)
```

예 :

```
1. range(1, 5)      -> 1, 2, 3, 4
2. range(7)         -> 0, 1, 2, 3, 4, 5, 6
3. range(0, 10, 2)  -> 0, 2, 4, 6, 8
   range(0, 11, 2)  -> 0, 2, 4, 6, 8, 10
```

## 응용해보기

### 이중 For문 (별찍기)

```
1  for i in range(5):
2      for j in range(5):
3          print("*", end='')
4      print()
```

결과는 어떻게 나올까요?

```
*****
*****
*****
*****
*****
```

-> 가로 세로 별이 5개씩 있는 직사각형이 출력됩니다

이런 별은 어떻게 찍을 수 있을까요?

### 문제 1번

```
*
**
***
****
*****
```

### 문제 2번

```
  *
 **
***
****
*****
```

### 문제 3번

```
  *
 ***
*****
*****
*****
*****
```

### 문제 1번 답 코드

```
for i in range(5):
    for k in range(i + 1):
        print("*", end='')
    print()
```

### 문제 2번 답 코드

```
1  for i in range(5):
2      for j in range(5 - i):
3          print(" ", end='')
4      for k in range(i + 1):
5          print("*", end='')
6      print()
```

### 문제 3번 답 코드

```
1  for i in range(5):
2      for j in range(5 - i):
3          print(" ", end='')
4      for k in range(2 * i + 1):
5          print("*", end='')
6      print()
```

- ➔ 무엇이 달라졌는지 관찰해보기
- ➔ 자세한 내용은 8-4(중첩 반복문) 참고



## 8-3 For문과 while문의 공통점과 차이점

---

공통점 : 둘 다 반복문이기 때문에 둘은 같은 결과를 항상 만들어 낼 수 있다.

**while 조건식 :**      **for 변수 in 시퀀스자료형(문자열,리스트,튜플) :**

수행할문장1                      수행할 문장(명령어)

수행할문장2                      수행할 문장(명령어)

수행할문장3

차이점 : for문은 구간을 명확히 정하기 때문에 처음부터 얼마나 순회해야 하는지 아는 작업을 알 때 유용하게 사용한다.

반면 while문은 얼마나 수행해야 하는지 모르는 작업에서 더 많이 사용한다.

## 8-4 중첩 반복문

### 중첩 반복문이란?

중첩 반복문은 반복문 여러 개를 중첩으로 사용한 것이다.  
말이 어렵다면 아래 예시를 통해 확인해 보자.

```
for i in range(1,5):  
    print(i)  
    for j in range(1,5):  
        print(j)
```

이처럼 중첩 반복문은 반복문 안에 또 다른 반복문을 넣은 것을 말한다.  
위 사진처럼 반복문 안에 또 다른 반복문을 넣은 경우(2중첩) 뿐만 아니라  
3, 4, 5...개의 반복문을 반복문 안에 넣어 사용할 수도 있다.

```
for i in range(1,5):  
    print(i)  
    for j in range(1,5):  
        print(j)  
        for k in range(1, 5):  
            print(k)  
            for l in range(1, 5):  
                print(l)
```

그러나, 이런 식으로 코드를 짜게 된다면 보기 어려울 뿐만 아니라,  
코드를 이해하는 데에도 어려움이 있고, 컴퓨터의 자원을 지나치게 많이  
사용하는 문제점이 있기 때문에 아주 특수한 사례를 제외하면 반복문 안에  
반복문을 넣은 2중첩 반복문까지를 주로 사용한다.

# 왜 사용할까?

## 중첩된 자료형

중첩 반복문을 사용하는 이유는 여러 가지가 있지만, 주로 중첩 리스트나 중첩 튜플 등, 중첩된 자료형을 편리하게 관리하기 위해 사용한다.

아래 예시를 통해 조금 더 자세하게 알아보자.

```
a = [
    [1, 2, 3, 4, 5],
    [6, 7, 8, 9, 10],
    [11, 12, 13, 14, 15],
    [16, 17, 18, 19, 20],
    [21, 22, 23, 24, 25]]
```

먼저, a라는 중첩 리스트를 선언해 준다.

그 다음, 기존에 사용했던 방식(중첩 for문 사용X)으로 for문을 이용해 1부터 25까지 각각 출력해 보겠다.

```
for i in range(0, 5):
    print(a[0][i], end = " ")

print()
for i in range(0, 5):
    print(a[1][i], end = " ")

print()
for i in range(0, 5):
    print(a[2][i], end = " ")

print()
for i in range(0, 5):
    print(a[3][i], end = " ")

print()
for i in range(0, 5):
    print(a[4][i], end = " ")
```

보다시피, 매우 복잡하고, 중복되는 코드도 많다.  
특히, 리스트의 길이가 늘어나게 되면 이런 식으로 칠 수 없을 정도로 코드가  
매우 길어지게 될 것이다.  
그렇다면, 이를 중첩 for문으로 나타내면 어떻게 될까?

```
for i in range(0, 5):  
    for j in range(0, 5):  
        print(a[i][j], end = " ")  
    print("")
```

이런 식으로 중복된 코드가 사라지고, 훨씬 간편하게 사용할 수 있다.

## 언제 사용할까?

앞에서 설명한 중첩된 자료형 이외에도 다양한 곳에서 중첩 반복문을 사용할 수 있다.

### 경우의 수 출력하기

경우의 수를 출력할 때에도 중첩 for문을 사용하면 편하게 출력할 수 있다.  
다음은 주사위 2개에 숫자가 나올 수 있는 경우의 수를 모두 출력하는 코드이다.  
주사위 2개에는 각각 6개의 면이 있으므로, 나올 수 있는 경우의 수는 36가지라는 것을 먼저 예측해 볼 수 있겠다.

```
for dice01 in range(1, 7):  
    for dice02 in range(1, 7):  
        print(dice01, end = " ")  
        print(dice02)
```

1	1	6	3
1	2	6	4
1	3	6	5
1	4	6	6

이렇게 1 1 부터 6 6 까지 경우의 수의 결과값이 총 36개가 나오는 것을 확인할 수 있다.

### 개수를 늘려가면서 출력하기

위 제목을 말로만 들어서는 잘 이해가 가지 않을 수 있다.  
따라서, 이 부분도 간단한 예시를 준비해 보았다.

```

*
**
***
****
*****
*****
*****
*****
*****
*****
*****

```

위 코드는 첫째 줄에 별 하나, 둘째 줄에 별 두 개… 열째 줄에 별 열 개를 출력한 코드이다.

그렇다면, 위와 같은 결과를 출력하기 위해서는 어떻게 해야 할까?

```

for i in range(1, 11):
    for j in range(0, i):
        print('*', end = ' ')
    print('')

```

이렇게 코드를 짜게 되면, 처음 바깥쪽의 for문이 돌아갈 때는  $i=1$ 이 되므로, 안쪽 for문은 1번 돌아가게 되기 때문에 별이 첫 줄에는 하나가 출력된다.

그 다음 안쪽 for문을 빠져나와 두 번째로 바깥쪽 for문으로 들어가게 되면  $i=2$ 가 되고 안쪽 for문은 두 번 실행된다.

이렇게 바깥쪽 for문을 총 10번 반복하고 프로그램이 끝나게 되고, 결과적으로 컴퓨터는 위와 같은 결과를 출력하게 된다.

## 개수를 줄여가면서 출력하기

방금 전 예시에서 첫째 줄에 별 하나, 둘째 줄에 별 두 개… 열째 줄에 별 열 개를 출력했다.

이번에는 반대로 첫째 줄에 별 열 개, 둘째 줄에 별 아홉 개… 열째 줄에 별 한 개를 출력하는 코드를 작성해 보겠다.

```
for i in range(1, 11):
    for j in range(0, 11-i):
        print('*', end = ' ')

    print('')
```

아까 전에 작성했던 코드와 별 차이가 없어 보인다.  
그러나, 아까와는 다른 곳이 한 군데가 있다.

```
for j in range(0, 11-i):
```

바로 이 부분이다.

기존 코드에서 `i`를 `11-i`로 고친 것이다.

이렇게 코드를 작성하게 되면, 처음 바깥쪽 for문을 돌릴 때는 안쪽 for문이 `range(0, 10)`만큼, 즉 10번 실행되고, 다음은 `range(0, 9)`, `range(0, 8)`... 이런 식으로 출력하다 보면 바깥쪽 for문이 10번째(즉 마지막으로) 돌아갈 때에는 별이 한 번 출력된다.

## 중첩 while문

지금까지 중첩 for문에 대해서만 살펴 보았는데, 위 내용을 모두 이해했다면 중첩 while문도 쉽게 사용할 수 있을 것이다.

### 중첩 리스트에서의 사용

다시 중첩 리스트를 가지고 왔다.

이번에는 이 리스트를 while문을 이용해 1부터 25까지 순서대로 출력해 보겠다.

```
a = [
    [1, 2, 3, 4, 5],
    [6, 7, 8, 9, 10],
    [11, 12, 13, 14, 15],
    [16, 17, 18, 19, 20],
    [21, 22, 23, 24, 25]]
```

```
i = 0
j = 0
while i < 5:
    while j < 5:
        print(a[i][j])
        j = j + 1

    j = 0
    i = i + 1
```

이런 식으로 코드를 작성할 수 있다.  
아래 코드는 모두 앞에서 살펴 봤던 것과 동일하게 작동하는 코드이다.

## 경우의 수 출력하기

```
dice01 = 1
dice02 = 1

while dice01<7:
    while dice02<7:
        print(dice01, dice02, sep=" ")
        dice02 = dice02 + 1

    dice01 = dice01 + 1
    dice02 = 1
```

## 개수를 늘려가면서 출력하기

```
i = 1
j = 0

while i<11:
    while j<i:
        print('*', end = '')
        j = j + 1

    i = i + 1
    j = 0
    print('')
```

## 개수를 줄여가면서 출력하기

```
i = 1
j = 0

while i<11:
    while j<11-i:
        print('*', end = '')
        j = j + 1

    i = i + 1
    j = 0
    print('')
```

