

파이썬 입문

✚ 파이썬이란

파이썬을 지금 막 배우고 있는 당신에게 누군가 파이썬이 무엇이냐고 묻는다면 무엇이냐 답변하시겠습니까? 한 문장을 정리하여 이야기 한다면 1990년대에 개발된 컴퓨터언어로 인터프리터 언어 입니다 라고 할수있습니다.

이렇게는 누구나 이야기 할 수 있지만 이것을 제대로 이해한 상태로 이야기를 하기 위해선 몇 가지 사전지식들이 필요합니다. 그것에 대해 미리 공부를 해야 파이썬의 구동방식을 이해할 수 있으며 다른 언어들과의 차별점을 알고 응용분야에 대한 깊은 고찰이 가능해 집니다.

기본적인 사전 지식들을 익히며 Python 과 IDE를 설치하고 Smoke Test(기본 동작 테스트) 까지 진행해보는 시간을 갖도록 하겠습니다.



■ 컴퓨터 언어란

컴퓨터 언어는 프로그래밍 언어라고도 불리우며 **기계**에게 **명령**을 내리는 목적으로 만들어 졌습니다.쉽게하면 “컴퓨터를 이용하기 위한 언어” 입니다.

왼쪽의 그림과 같이 실제로 프로그래밍 언어는 다양합니다. 이 프로그래밍 언어는 컴퓨터가 이해할 수 있는 기계어로 바꾸는 과정에서 변환되는 방식이나 독자적인 문법에 따라 나누어 집니다.

➤ 기계어(1세대)

기계(CPU)가 읽고 실행할수있는 이진코드

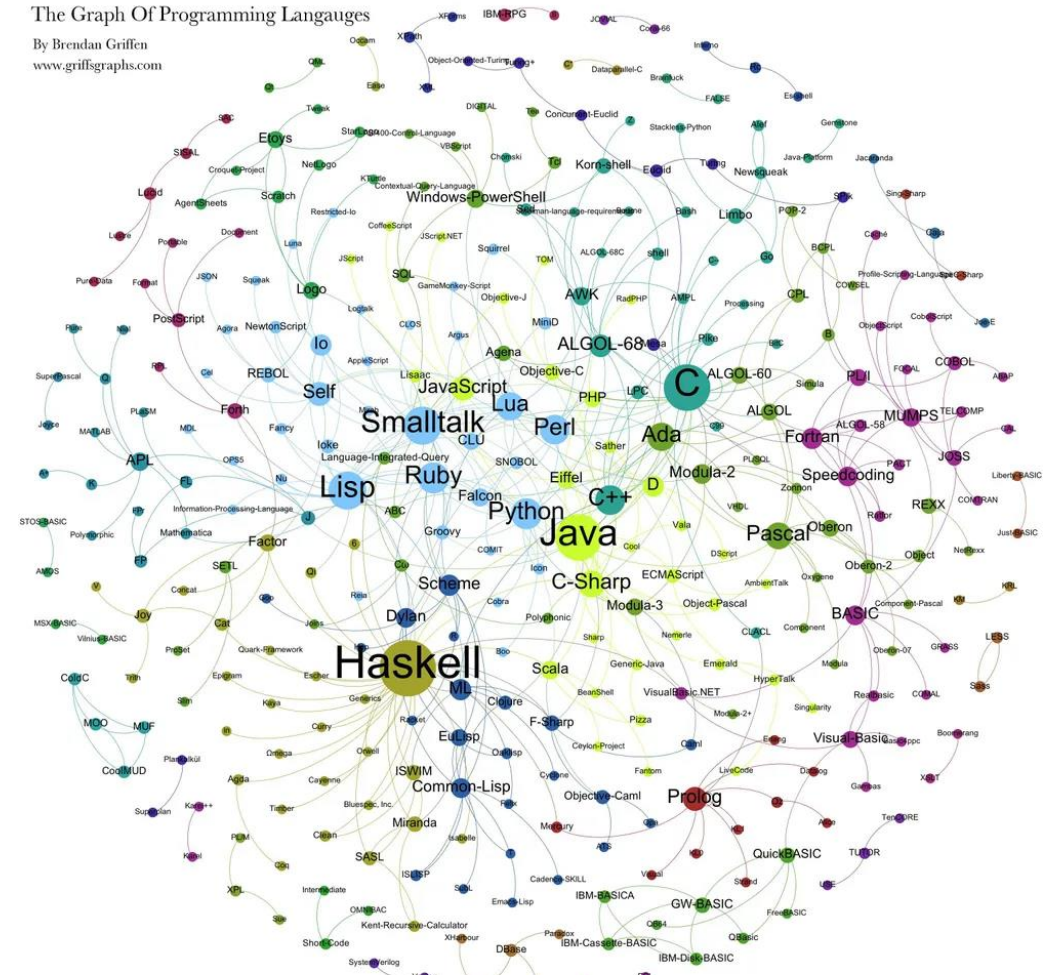
➤ 어셈블리어(2세대)

기계어보다는 한단계 높은 기계어를 개발자가 알아보기 쉽게 가공한 언어입니다.

➤ 고급 프로그래밍 언어(3세대)

개발자들이 읽고 이해하기 쉬운 문법이 적용이 되면서 고급언어로 불립니다. 기계어로 번역해주는 번역기가 필요하며 절차 지향 프로그래밍언어입니다.

The Graph Of Programming Languages
By Brendan Griffen
www.griffgraphs.com



➤ 객체지향 프로그래밍 언어(4세대)

객체라는 개념을 도입하여 코드의 재사용성을 향상 시키고 대규모 프로그램개발에 적합한 언어입니다.

➤ 인공지능 프로그래밍 언어(5세대)

프로그램에 주어지는 제약을 사용하여 문제를 해결하며 빅데이터 분석에 특화되어있고 머신러닝과 딥러닝등 다양한 분야에 활용이 됩니다.

■ 컴파일링 vs 인터프리팅

고급언어로 개발자가 프로그래밍을 했다면 컴퓨터가 이해할수있는 기계어로 변환이 되어야 합니다.

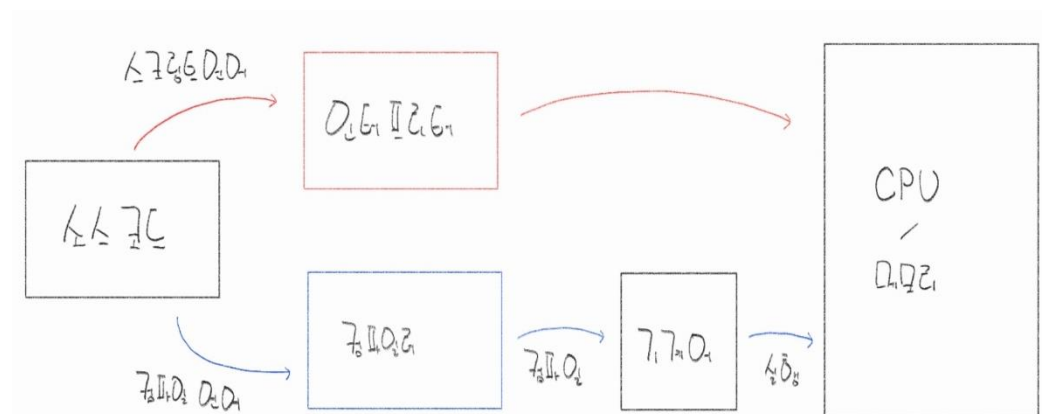
컴파일링과 인터프리팅은 프로그램을 해석하고 실행하는 대표적인 방법입니다.

➤ 컴파일러

고급언어로 만들어진 프로그램을 컴퓨터가 이해할 수 있는 언어로 변환하는 프로그램입니다.

➤ 인터프리터

고급언어로 작성된 코드를 문장 단위로 한 단계씩 해석하며 실행시키는 프로그램 입니다.



■ 인터프리터 언어란

소스코드를 컴파일 하지 않고 인터프리터로 소스코드를 한 줄씩 읽어서 바로 실행하는 방식으로 동작을 하는 언어를 이야기 합니다 줄(Row)단위로 실시간으로 해석하며 실행시키기 때문에 컴파일 언어보다 속도가 느리지만 번역과 실행이 동시에 이루어져 OS 환경에 맞는 라이브러리와 빌드를 하기위한 구축 환경없이 바로 실행이가능 하므로 별도의 실행파일이 존재하지 않습니다.











또한 실시간 Debugging 및 코드수정이 가능 하며 메모리를 별도로 할당 받아 수행되지 않으며, 필요할 때 할당하여 사용합니다.

대표적인 언어로는 Python, JavaScript 등이 있습니다.

■ 파이썬의 입지

꾸준히 상위권을 유지하는 Python의 인기는 최근 인공지능 개발과 관련하여 수요가 크게 증가하였습니다.

구글에서 만든 소프트웨어의 50% 이상이 파이썬으로 작성되었다는 이야기도 있을 정도이며 인스타그램, 넷플릭스, 아마존 등 우리가 알고 있는 많은 IT 기업에서 파이썬을 사용합니다.

May 2024	May 2023	Change	Programming Language		Ratings
1	1			Python	16.33%
2	2			C	9.98%
3	4	▲		C++	9.53%
4	3	▼		Java	8.69%
5	5			C#	6.49%
6	7	▲		JavaScript	3.01%
7	6	▼		Visual Basic	2.01%
8	12	▲		Go	1.60%
9	9			SQL	1.44%
10	19	▲		Fortran	1.24%

■ 파이썬 특징

모든 프로그래밍 언어는 저마다 장점이 있지만 파이썬은 다른 언어에서 쉽게 찾아볼 수 없는 독특한 매력을 가지고 있습니다.. 파이썬의 특징을 알면 왜 파이썬을 공부해야 하는지, 과연 시간을 투자할 만한 가치가 있는지 판단할 수 있을 것 입니다.

➤ 직관적인 표현

파이썬은 사람이 생각하는 방식을 그대로 표현할 수 있는 언어입니다. 따라서 파이썬을 사용하는 프로그래머는 굳이 컴퓨터의 사고 체계에 맞추어 프로그래밍하려고 애쓸 필요가 없습니다. 이제 곧 어떤 프로그램을 구상하자마자 머릿속에서 생각한 대로 코드를 술술 써 내려가는 자신의 모습을 보고 놀라게 될 것입니다.

➤ 쉬운 문법

파이썬은 문법이 매우 쉽고 간결하며 사람의 사고 체계와 매우 닮아 있습니다. 배우기 쉬운 언어, 활용하기 쉬운 언어가 가장 좋은 언어라고 생각합니다. 프로그래밍 경험이 조금이라도 있다면 파이썬의 자료형, 함수, 클래스 만드는 법, 라이브러리 및 내장 함수 사용 방법 등을 익히는 데 일주일이면 충분하다고 생각합니다.

➤ 무료

파이썬은 오픈 소스(open source)이며 무료 소프트웨어입니다. 사용료 걱정 없이 언제 어디서든 파이썬을

내려받아 사용할 수 있습니다.

➤ 즐거운 프로그래밍

파이썬은 프로그래머가 다른 부수적인 개념이나 제한 사항 등에 신경 쓸 필요 없이 만들고자 하는 기능에만 집중할 수 있게 해 줍니다.. 파이썬을 배우고 나면 다른 언어로 프로그래밍하는 것이 지루하다고 느낄지도 모릅니다.

마지막으로 이 모든 특징을 유명한 한마디로 정리하겠습니다.

“Life is too short, you need python” (인생은 너무 짧으니 파이썬이 필요해.)

■ 파이썬 응용분야

- 웹 프로그래밍
- 인공지능과 머신러닝
- 수치연산 프로그래밍
- 데이터분석
- DATA BASE 프로그래밍
- 사물인터넷

파이썬 설치

■ 공식 설치 파일 vs 배포판(아나콘다)

➤ 공식 설치파일

사이트: <https://www.python.org/>

용량: 25 MB

➤ 배포판(아나콘다)

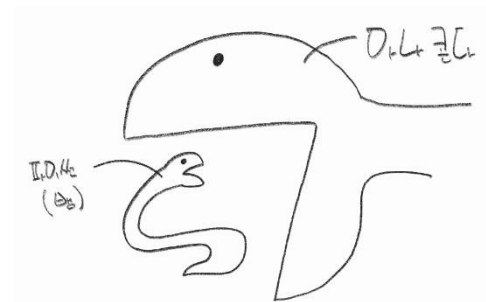
사이트: <https://www.anaconda.com/>

용량: 2GB

다양한 라이브러리가 설치가 되어있습니다. (공식이 노옵션차라고 한다면 배포판은 풀옵션차.)

■ 아나콘다

많은 회사가 배포판 버전을 만들었지만 현재 가장 많이 이용하는 배포판은 아나콘다라는 회사에서 제공을 하여 제품명이 아나콘다 입니다. 다양한 라이브러리가 미리 설치가 되어있어 용량이 큰것이 단점이지만 파이썬을 처음 이용하는 프로그래머에게 추천이 되어집니다.



■ 설치



아나콘다 다운로드



전체 이미지 쇼핑 동영상 뉴스 : 더보기

검색결과 약 205,000개 (0.20초)



Anaconda

<https://www.anaconda.com> > download

Download Anaconda Distribution

Download Anaconda's open-source Distribution today. Discover the easiest way to perform Python/R data science and machine learning on a single machine.



Products Solutions Resources Partners Company

Free Download Sign In

Distribution

Register to get everything you need to get started on your workstation

- ✓ Distribution installation on Windows, MacOS, or Linux
- ✓ Easily search and install thousands of data science, machine learning, and AI packages
- ✓ Manage packages and environments from a desktop application or work from the command line
- ✓ Deploy across hardware and software platforms

Commercial use at a company of more than 200 employees requires a Business or Enterprise license. [See Pricing](#)

Provide email to download Distribution

Don't miss out! Get access to: Cloud Notebooks, Anaconda Assistant, easy application deployment, learning resources, and updates from Anaconda.

Email Address:

☐ I agree to receive communication from Anaconda regarding relevant content, products, and services. I understand that I can revoke this consent [here](#) at any time.

By continuing, I agree to Anaconda's [Privacy Policy](#) and [Terms of Service](#).

Submit >

[Skip registration](#)



Products Solutions

Download N

For installation assistance, refer to [Troubleshooting](#).

Download Distribution by choosing the proper ins

Download

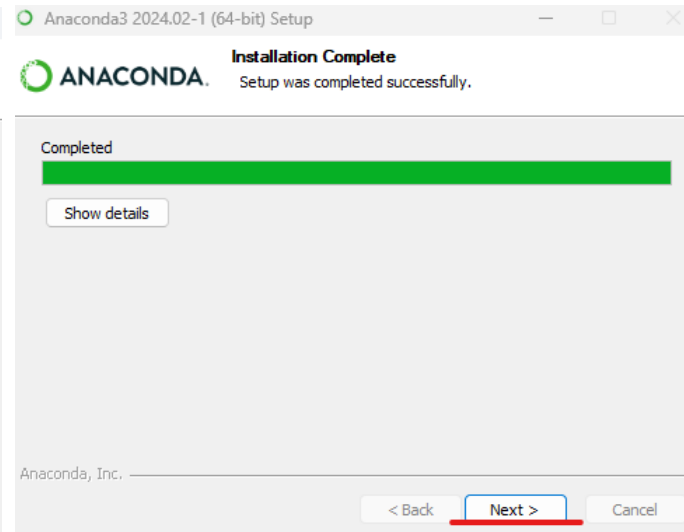
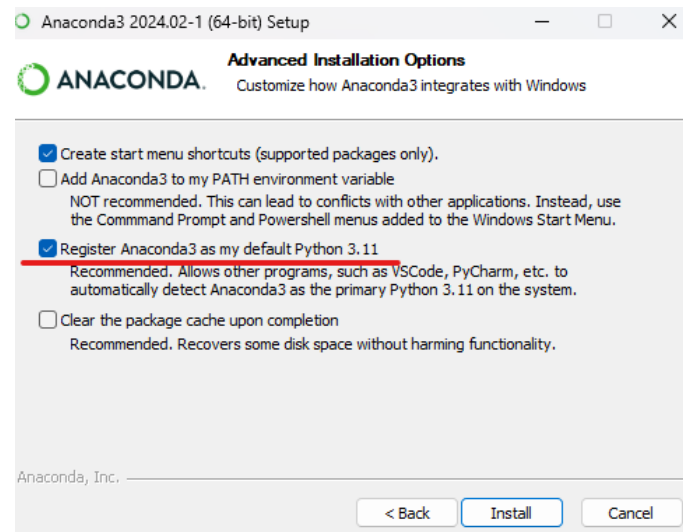
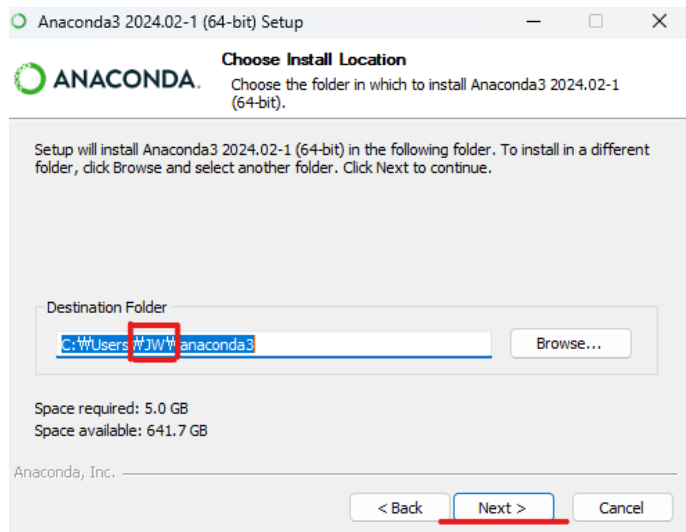
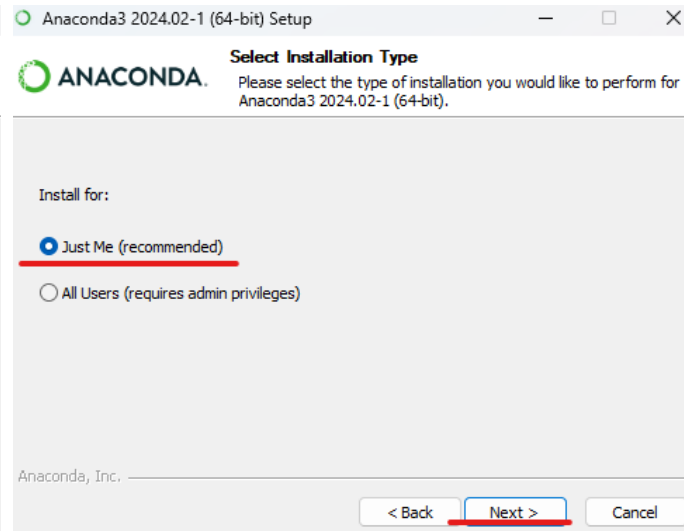
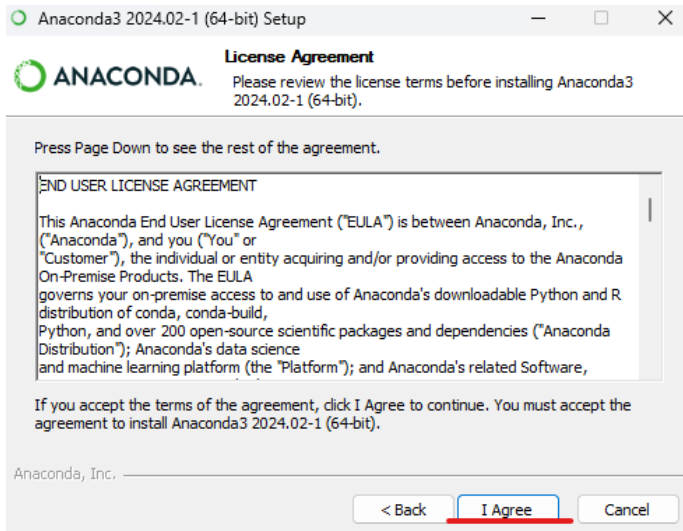
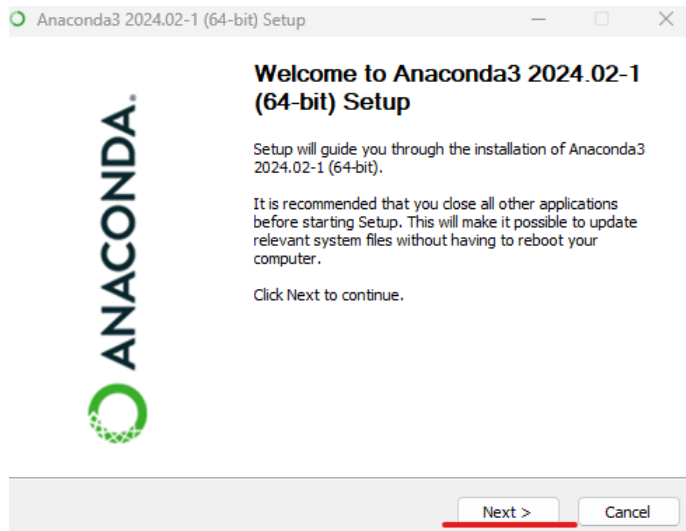
Anaconda Installer

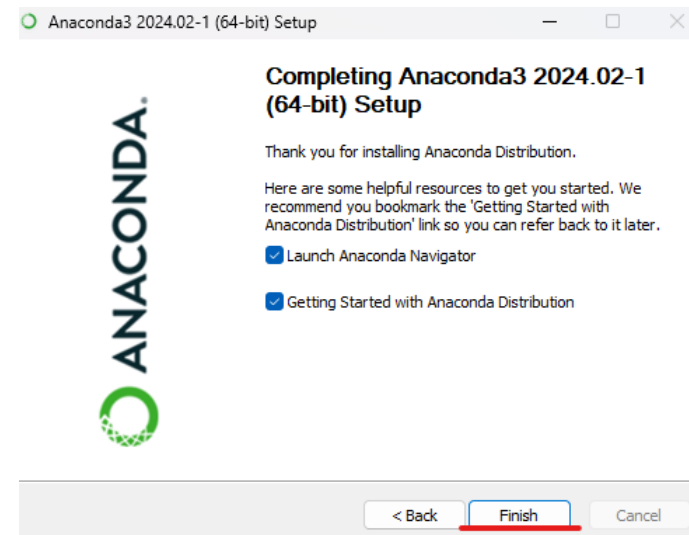
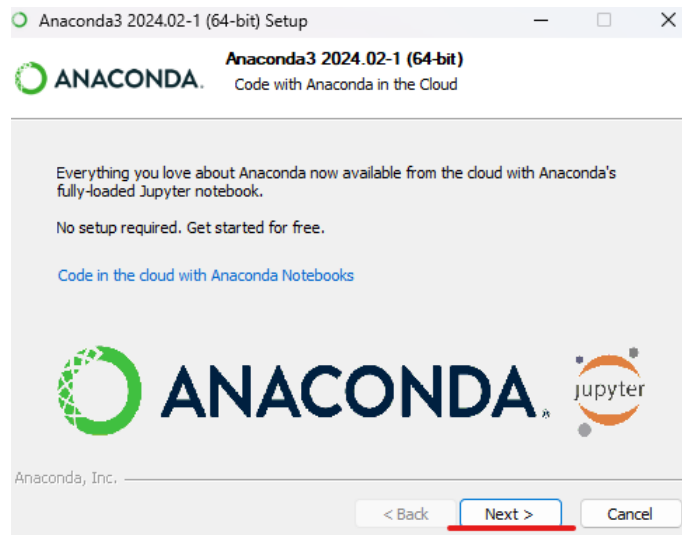


Windows

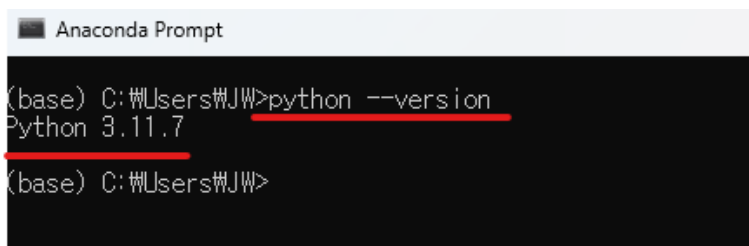
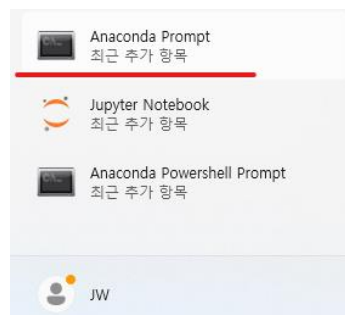
Python 3.11

64-Bit Graphical Installer (904.4M)





■ 설치 확인





IDE

■ IDE란

Integrated Development Environment

효율적으로 소프트웨어를 개발하기 위한 통합개발환경 소프트웨어 어플리케이션 인터페이스입니다. 코드 편집기, 디버거, 컴파일러, 인터프리터 등을 포함하고 개발자에게 제공합니다.


통합개발환경은 개발자가 소프트웨어를 개발하는 과정에 필요한 모든 작업을 하나의 소프트웨어에서 처리할 수 있는 환경을 제공합니다. 초기 소프트웨어 개발 시스템은 코드 편집기, 컴파일링, 디버깅 등과 같은 개발 과정을 각각의 프로그램에서 사용했으며 콘솔을 통한 개발이 불가능했습니다.

■ 파이참

JetBrains에서 제작한 Python용 통합 개발 환경 프로그램 입니다.


현용 파이썬 개발 툴 중 가장 기능이 강력하고 완성도가 높다고 여겨지고 수준 높은 코드 자동완성 기능을 제공합니다. 특히 타입 힌트를 적극 사용하면 웬만한 정적 타입 언어 수준의 코드 자동완성을 제공합니다.

➤ 설치




[전체](#) [이미지](#) [동영상](#) [뉴스](#) [쇼핑](#) [더보기](#) [도구](#)

검색결과 약 6,210,000개 (0.21초)


 **JetBrains**
<https://www.jetbrains.com> > [ko-kr](#) > [pycharm](#) > [download](#) [⋮](#)

PyCharm 다운로드: 데이터 과학 및 웹 개발을 위해 ...

Windows, macOS 또는 Linux용 최신 버전의 PyCharm을 다운로드하세요.

 **PyCharm**
데이터 과학 웹 개발 EAP 새로운 기능
JetBrains IDEs

[Windows](#) [macOS](#) [Linux](#)

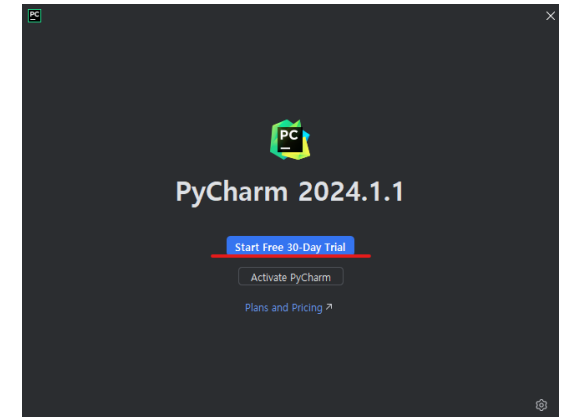
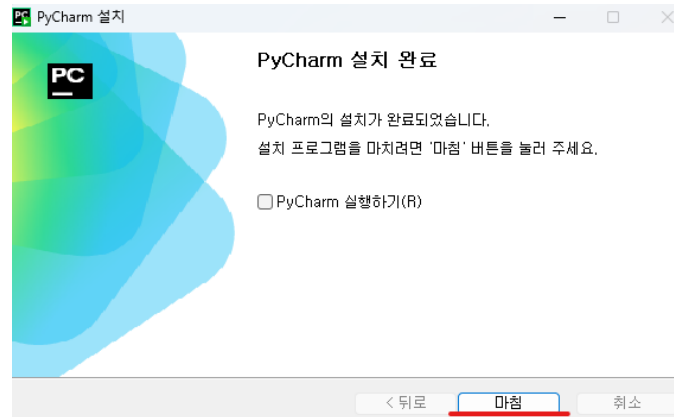
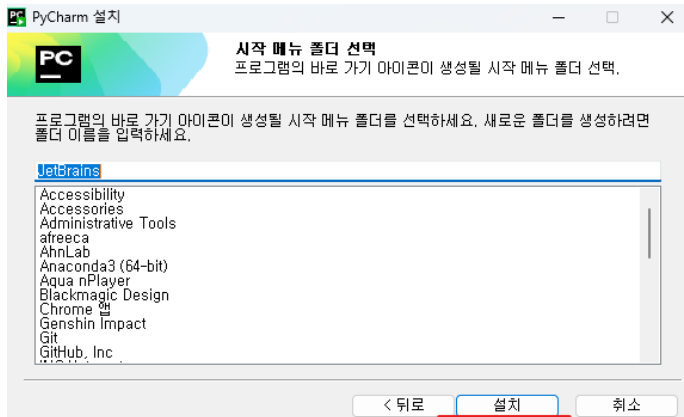
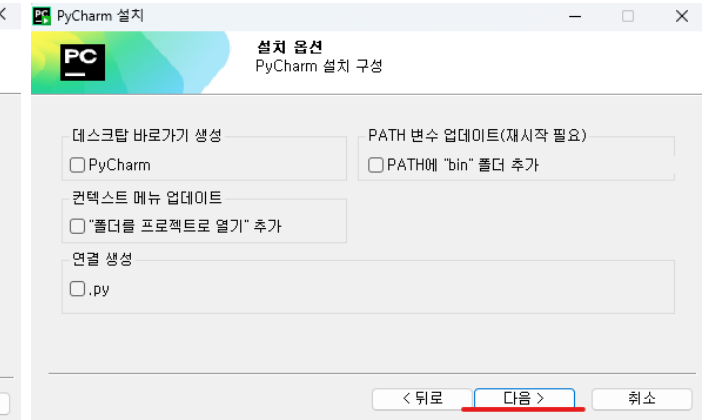
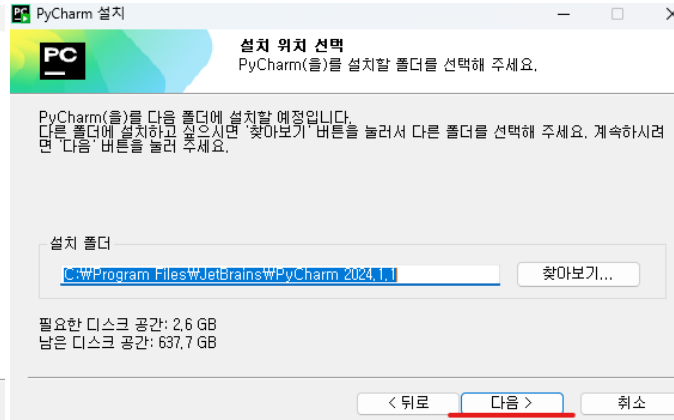
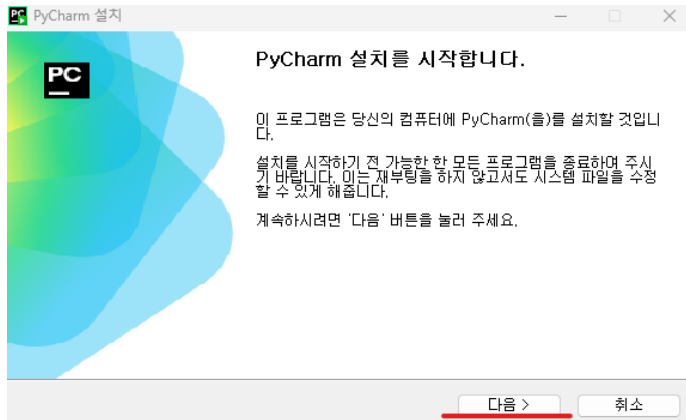


PyCharm Professional

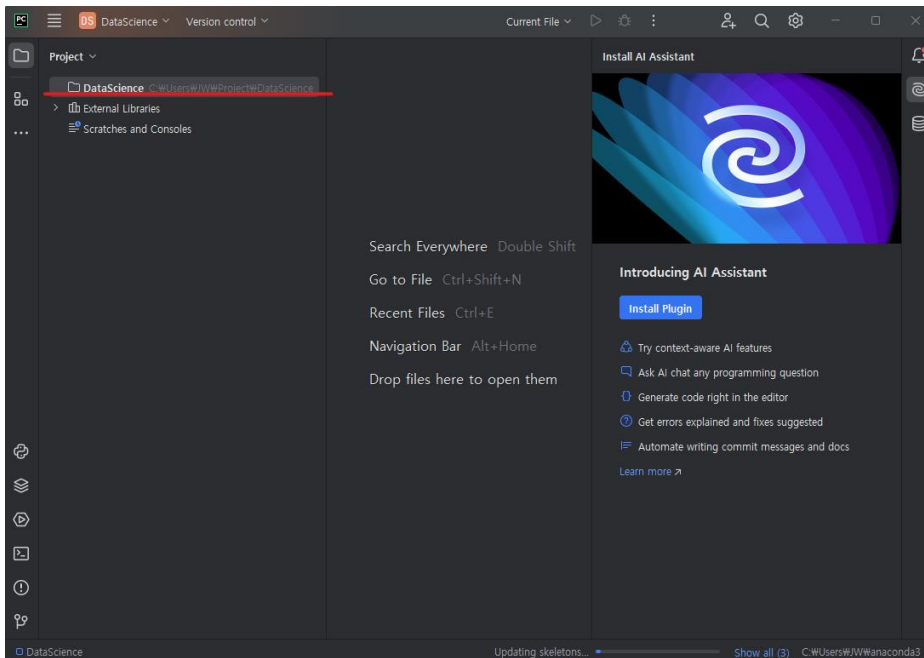
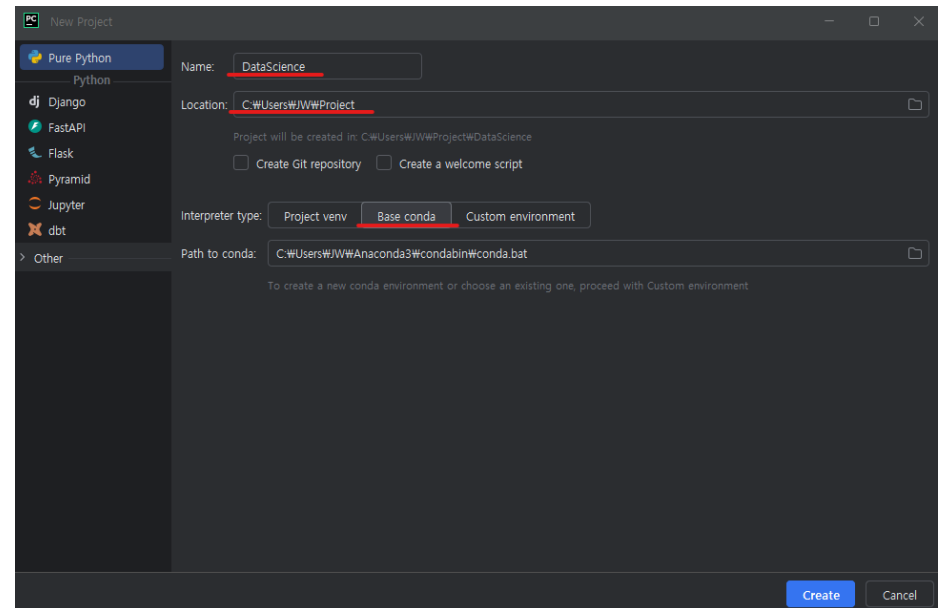
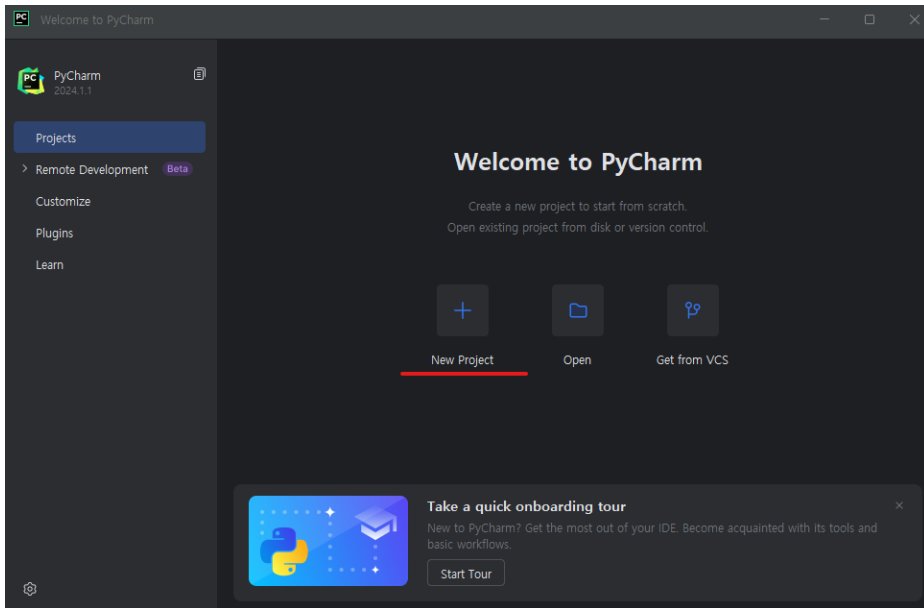
데이터 과학 및 웹 개발을 위한
Python IDE

[다운로드](#) [.exe \(Windows\) ▼](#)

30일 무료 평가판



■ 프로젝트 생성



■ 환경설정

➤ 대표적인 단축키

Setting창 열기 : Ctrl + Alt + S

주석 처리/해제 : Ctrl + /

함수 정의된 곳 이동 : Ctrl + B

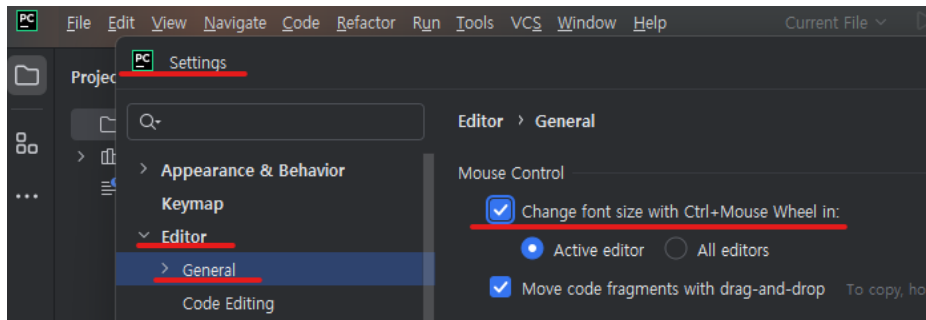
새로운 파일 생성 : Alt + insert

이름변경 : Shift + F9

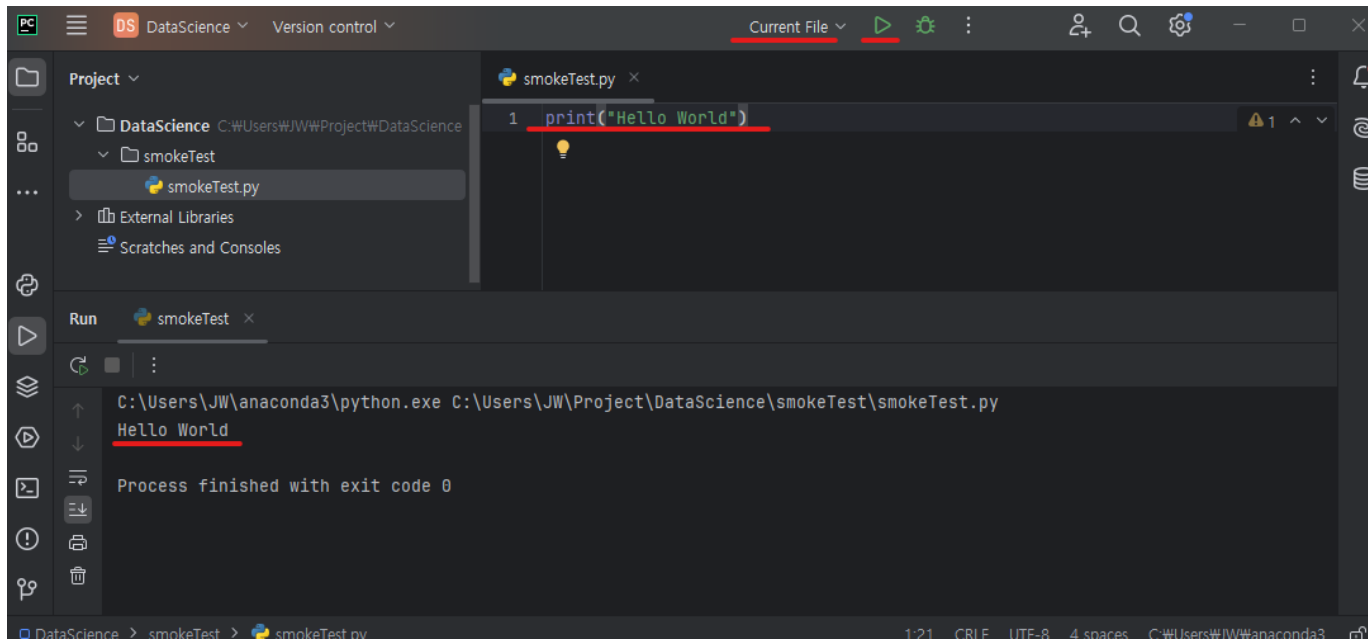
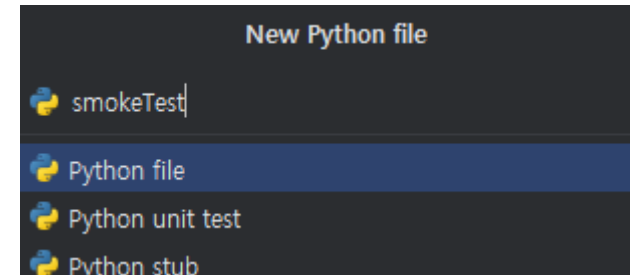
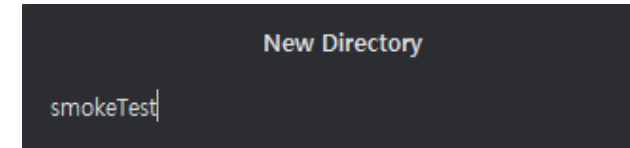
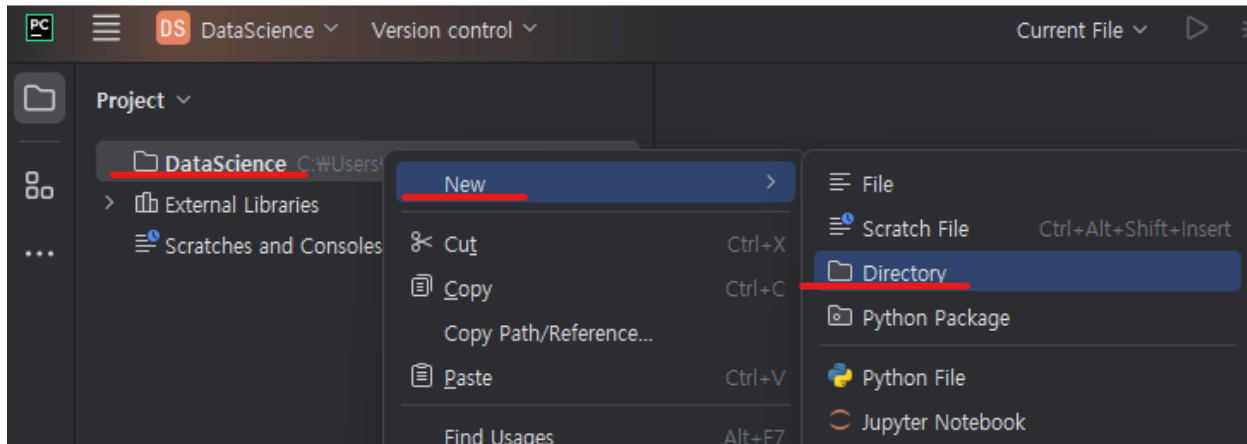
자동정렬 : Alt + Ctrl + L

파이썬 파일 실행 : Shift + F10

지정코드 바로console에서 실행 : Alt + Shift + E



SMOKE_TEST



파이썬 자료형

■ 숫자형

➤ 정수형

정수형(integer)이란 말 그대로 정수를 뜻하는 자료형을 말합니다. 다음은 양의 정수와 음의 정수, 숫자 0을 변수 a, b, c에 대입하는 예입니다.

```
a = 1004
b = -486
c = 0
```

➤ 실수형

파이썬에서 실수형(floating-point)은 소수점이 포함된 숫자를 말합니다. 다음은 실수를 변수 a에 대입하는 예이다. 일반적으로 볼 수 있는 실수형의 소수점 표현 방식입니다.

```
a = 1.004
b = -4.86
```

➤ 사칙연산

프로그래밍을 한 번도 해 본 적이 없는 독자라도 사칙 연산(+, -, *, /)은 알고 있을 것입니다. 파이썬 역시

계산기와 마찬가지로 다음처럼 연산자를 사용해 사칙 연산을 수행합니다.

```
...: a = 3
...: b = 4
```

```
...: print(a+b)
```

```
...:
```

7

```
...: print(a-b)
```

```
...:
```

-1

```
...: print(a*b)
```

```
...:
```

12

```
...: print(a/b)
```

```
...:
```

0.75

➤ x의 y제곱을 나타내는 ** 연산자

```
...: print(a**b)
...:
81
```

➤ 나눗셈 후 나머지를 리턴하는 % 연산자

3을 4으로 나누면 나머지는 3

```
...: print(a%b)
...:
3
```

➤ 나눗셈 후 몫을 리턴하는 // 연산자

3을 4으로 나누면 몫은 0

```
...: print(a//b)
...:
0
```

➤ 복합 연산자

사칙 연산자(+, -, *, /)와 대입 연산자(=)를 합쳐 놓은 것입니다.

a += 1이라는 표현식은 a = a + 1을 줄여서 표현한 것으로 완전히 동일한 기능을 합니다. 여기에서 사용

한 += 와 같은 연산자를 복합 연산자라 부릅니다.

```
...: a = 1
...: a += 1
...: print(a)
...:
```

2

a += 1이라는 표현식은 a = a + 1

a -= 1이라는 표현식은 a = a - 1

a *= 1이라는 표현식은 a = a * 1

a /= 1이라는 표현식은 a = a / 1

a //= 1이라는 표현식은 a = a // 1

a %= 1이라는 표현식은 a = a % 1

a **= 1이라는 표현식은 a = a ** 1

■ 문자열 자료형

➤ 표현방법

```
"Hello World"  
'Hello World'  
"""Hello World"""  
'''Hello World'''
```

파이썬에서 문자열을 만드는 방법은 총 4가지 입니다.

➤ 문자열 안 따옴표처리1

```
...: food = 'Python's favorite food is perl'  
...: print(food)  
...:  
Cell In[5], line 1  
    food = 'Python's favorite food is perl'  
                                   ^  
SyntaxError: unterminated string literal (detected at line 1)
```

작은따옴표(')가 포함된 문자열은 작은따옴표(')가 아닌 큰따옴표(") 이용하면 됩니다

```
...: food = "Python's favorite food is perl"  
...: print(food)  
...:  
Python's favorite food is perl
```

```
...: say = '"Python is very easy." he says.'  
...: print(say)  
...:  
"Python is very easy." he says.
```

작은따옴표(') 안에 사용된 큰따옴표(") 는 문자열을 만드는 기호로 인식되지 않습니다.

➤ 문자열 안 따옴표처리2

```
...: food = 'Python\'s favorite food is perl'
...: print(food)
...: say = "\"Python is very easy.\" he says."
...: print(say)
...:
Python's favorite food is perl
"Python is very easy." he says.
```

역슬래시(\)를 사용하는 것이다. 즉, 역슬래시를 작은 따옴표나 큰따옴표 앞에 삽입하면 역슬래시 뒤의 작은 따옴표나 큰따옴표는 문자열을 둘러싸는 기호의 의미가 아니라 '나 ' 자체를 뜻하게 됩니다.

➤ 다중행 문자열

```
...: multiline='''
...: Life is too short
...: You need python
...: '''
...: print(multiline)
...:
...: multiline2 = "Life is too short\nYou need python"
...: print(multiline2)
...:

Life is too short
You need python

Life is too short
You need python
```

작은따옴표 3개(''), 큰따옴표 3개(""")를 사용합니다.

줄 바꿈 문자인 \n을 삽입하는 방법이 있지만, 읽기가 불편하고 줄이 길어지는 단점이 있습니다.

➤ 이스케이프 코드

코드	설명
\	문자열 안에서 줄을 바꿀 때 사용

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

➤ 문자연산

```
...: head = "Python"
...: tail = " is fun!"
...: print(head + tail)
...:
Python is fun!
```

더하기는 눈에 보이는 데로 문자열을 합치는 것입니다.


```
...: a = "python"
...: print(a * 2)
...:
pythonpython
```

곱하기는 문자열의 반복을 뜻하는 의미로 사용되었다.

```
...: print("=" * 50)
...: print("My Program")
...: print("=" * 50)
...:
=====
My Program
=====
```

문자열 곱하기를 좀 더 응용해 볼 수 있습니다.

➤ 문자열 길이

```
...: a = "Life is too short"
...: print(len(a))
...:
17
```

len 함수는 print 함수처럼 파이썬의 기본 내장 함수로, 별다른 설정 없이 바로 사용할 수 있습니다. 문자열의 길이에는 공백 문자도 포함됩니다.

➤ 인덱싱

데이터를 빠르고 효율적으로 접근하고 조작할 수 있도록 하는 방법입니다.

```
...: a = "Life is too short, You need Python"
...: print(a[0],a[1],a[2],a[3],a[4],a[5],a[6],a[7])
...:
L i f e   i s
```

각 요소는 순서대로 번호(인덱스)가 매겨져 있고 첫 요소는 0부터 시작합니다.

```

...: a = "Life is too short, You need Python"
...: print(a[-6],a[-5],a[-4],a[-3],a[-2],a[-1])
...:
P y t h o n

```

a[-1]은 뒤에서부터 세어 첫 번째가 되는 문자를 말합니다.

➤ 슬라이싱

```

...: a = "Life is too short, You need Python"
...: print(a[0:4])
...: print(a[19:])
...: print(a[:17])
...: print(a[:])
...: print(a[19:-7])
...:
Life
You need Python
Life is too short
Life is too short, You need Python
You need

```

단순히 한 문자만을 뽑아 내는 것이 아니라 단어를 뽑아 내는 방법은 슬라이싱을 하는 것입니다.

a[시작_번호:끝_번호]를 지정할 때 끝 번호에 해당하는 문자는 포함하지 않습니다.

a[시작_번호:끝_번호]에서 끝 번호 부분을 생략하면 시작 번호부터 그 문자열의 끝까지 뽑아 냅니다.

a[시작_번호:끝_번호]에서 시작 번호를 생략하면 문자열

의 처음부터 끝 번호까지 뽑아 냅니다.

a[시작_번호:끝_번호]에서 시작 번호와 끝 번호를 생략하면 문자열의 처음부터 끝까지 뽑아 냅니다.

슬라이싱에서도 인덱싱과 마찬가지로 -(빼기) 기호를 사용할 수 있습니다.

```

...: a = "20230331Rainy"
...: year = a[:4]
...: day = a[4:8]
...: weather = a[8:]
...:
...: print("year: " + year)
...: print("day: " + day)
...: print("weather: " + weather)
...:
year: 2023
day: 0331
weather: Rainy

```

```

...: a = "Python"
...: a = a[:1] + 'y' + a[2:]
...: print(a)
...:
Python

```

➤ f 문자열 포매팅

```

...: name = 'jane'
...: age = 22
...: a = f'나의 이름은 {name}입니다. 나이는 {age}입니다.'
...: print(a)
...:
나의 이름은 jane입니다. 나이는 22입니다.

```

응용1

예시는 자주 사용하는 슬라이싱 기법 중 하나입니다.

응용2

변경 불가능한 문자열을 바꿀 때 슬라이싱을 이용하면 가능합니다.

문자열 포매팅이란 문자열 안에 어떤 값을 삽입하는 방법입니다. 문자열 안의 특정한 값을 바꿔야 할 경우가 있을 때 이것을 가능하게 해 주는 것이 바로 문자열 포매팅입니다.

파이썬 3.6 버전부터는 f 문자열 포매팅 기능을 사용할 수 있습니다. 문자열 앞에 f 접두사를 붙이면 f 문자열 포매팅 기능을 사용할 수 있고 name, age와 같은 변수값을 생성한 후에 그 값을 참조할 수 있습니다.

```

...: a = f'{"hi":<10}'
...: print(a)
...: b = f'{"hi":>10}'
...: print(b)
...: c = f'{"hi":^10}'
...: print(c)
...:

```

```

hi
      hi
    hi

```

f 문자열 포매팅은 표현식을 지원하기 때문에 다음과 같은 것도 가능합니다.

정렬은 다음과 같이 할 수 있습니다. 10칸의 가상의 공간을 만든 후 위에서부터 왼쪽, 오른쪽, 가운데 정렬입니다.

```

...: a = f'{"hi":=^10}'
...: print(a)
...: b = f'{"hi":!<10}'
...: print(b)
...:
====hi====
hi!!!!!!!

```

공백 채우기는 다음과 같이 할 수 있습니다. 10칸의 가상의 공간을 만든 후 빈공백을 '=' 와 '!'로 채웁니다.

```

...: age = 22
...: a = f'나이는 {age + 3}입니다.'
...: print(a)
...:
나이는 25입니다.

```

f 문자열 포매팅은 표현식을 지원합니다.

```

...: d = {'name':'김철수', 'age':22}
...: a = f'나의 이름은 {d["name"]}입니다. 나이는 {d["age"]}입니다.'
...: print(a)
...:
나의 이름은 김철수입니다. 나이는 22입니다.

```

딕셔너리는 f 문자열 포매팅에서 다음과 같이 사용할 수 있습니다

```

...: y = 3.42134234
...: a = f'{y:0.4f}'
...: print(a)
...: b = f'{y:10.4f}'
...: print(b)
...:
3.4213
3.4213

```

소수점은 다음과 같이 표현할 수 있습니다. 소수점 4 자리까지만 표현하는 방법과 총자리수를 10자리로 맞추고 소수점 4자리까지만 표현하는 방법입니다.

➤ 문자열 관련 함수들

문자열 자료형은 자체적으로 함수를 가지고 있습니다. 이들 함수를 다른 말로 '문자열 내장 함수'라고 합니다. 이 내장 함수를 사용하려면 문자열 변수 이름 뒤에 '.'를 붙인 후 함수 이름을 써 주면 됩니다. 이제 문자열의 내장 함수에 대해서 알아보겠습니다.

```

...: a = "hobby"
...: print(a.count('b'))
...:
2

```

count 함수로 문자열 중 문자 b의 개수를 리턴합니다..

```

...: a = "Python is the best choice"
...: print(a.find('b'))
...: print(a.find('k'))
...:
14
-1

```

find 함수로 문자열 중 문자 b가 처음으로 나온 위치를 반환합니다. 만약 찾는 문자나 문자열이 존재하지 않는다면 -1을 반환합니다.

```
...: a = "Life is too short"
...: print(a.index('t'))
...:
```

8

index 함수로 문자열 중 문자 t가 맨 처음으로 나온 위치를 반환합니다.

```
...: a = ",".join('abcd')
...: print(a)
...: b = "!".join(['a', 'b', 'c', 'd'])
...: print(b)
...:
```

```
a,b,c,d
a!b!c!d
```

join 함수로 abcd 문자열의 각각의 문자 사이에 ','를 삽입합니다. 문자열뿐만 아니라 앞으로 배열 리스트나 튜플도 입력으로 사용할 수 있습니다.

```
...: a = "hi"
...: print(a.upper())
...: b = "HI"
...: print(b.lower())
...:
```

```
HI
hi
```

upper 함수는 소문자를 대문자로 바꾸어 줍니다.

lower 함수는 대문자를 소문자로 바꾸어 줍니다.

lstrip 함수는 문자열 중 가장 왼쪽에 있는 한 칸 이상의 연속된 공백들을 모두 지웁니다

```
...: a= "  hi  "
...: print(a.rstrip())
...: print(a.lstrip())
...: print(a.strip())
...:
```

```
hi
hi
hi
```

rstrip 함수는 문자열 중 가장 오른쪽에 있는 한 칸 이상의 연속된 공백을 모두 지웁니다

strip 함수는 문자열 양쪽에 있는 한 칸 이상의 연속된 공백을 모두 지웁니다.

```
...: a = "Life is too short"
...: b = a.replace("Life", "Your leg")
...: print(b)
...:
Your leg is too short
```

replace 함수는 replace(바뀔_문자열, 바꿀_문자열)처럼 사용해서 문자열 안의 특정한 값을 다른 값으로 치환해 줍니다.

```
...: a = "Life is too short"
...: b = a.split()
...: print(b)
...:
['Life', 'is', 'too', 'short']
```

split 함수는 a.split()처럼 괄호 안에 아무 값도 넣어주지 않으면 공백을 기준으로 문자열을 나누어 줍니다.

```
...: a = "a:b:c:d"
...: b = a.split(':')
...: print(b)
...:
['a', 'b', 'c', 'd']
```

괄호 안에 특정 값이 있을 경우에는 괄호 안의 값을 구분자로 해서 문자열을 나누어 줍니다. 이렇게 나눈 값은 리스트에 하나씩 들어갑니다.

■ 리스트 자료형

➤ 표현방법

리스트명 = [요소1, 요소2, 요소3, ...]

리스트를 만들 때는 위에서 보는 것과 같이 대괄호([])로 감싸 주고 각 요소값은 쉼표(,)로 구분해 줍니다.

```
a = []  
b = [1, 2, 3]  
c = ['Life', 'is', 'too', 'short']  
d = [1, 2, 'Life', 'is']  
e = [1, 2, ['Life', 'is']]
```

```
...: a = list()  
...: print(a)  
...:  
[ ]
```

a처럼 비어 있는 리스트([])일 수도 있고, b처럼 숫자로만, c처럼 문자열로만 요소값을 가질 수도 있습니다. 또한 d처럼 숫자와 문자열을 함께 가질 수도 있고, e처럼 리스트 자체를 요소값으로 가질 수도 있습니다.

비어 있는 리스트는 a = list()로 생성할 수 있습니다.

➤ 인덱싱

```
...: a = [1, 2, 3]  
...: print(a)  
...: print(a[0])  
...: print(a[-1])  
...: print(a[0] + a[2])  
...:  
[1, 2, 3]  
1  
3  
4
```

리스트도 문자열처럼 인덱싱과 슬라이싱이 가능합니다

문자열을 공부할 때 이미 살펴보았지만, 파이썬은 숫자를 0부터 세기 때문에 a[1]이 리스트 a의 첫 번째 요소가 아니라 a[0]이 리스트 a의 첫 번째 요소입니다. a[-1]은 문자열에서와 마찬가지로 리스트 a의 마지막 요소값을 말합니다.


```

...: a = [1, 2, 3, ['a', 'b', 'c']]
...: print(a[0])
...: print(a[3])
...: print(a[-1])
...: print(a[-1][0])
...: print(a[-1][2])
...:

```

```

1
['a', 'b', 'c']
['a', 'b', 'c']
a
c

```

리스트를 요소값으로 갖는 리스트의 인덱싱도 알아보겠습니다.

a[-1]은 마지막 요소값 ['a', 'b', 'c']를 나타내고 리스트에서 'a' 값을 인덱싱을 사용해 끄집어 내기위해서는 리스트에서 첫 번째 요소를 불러오기 위해 [0]을 붙여주면 됩니다.

```

...: a = [1, 2, ['a', 'b', ['Life', 'is']]]
...: print(a[-1][2][0])
...:
Life

```

삼중 리스트도 같은 방법으로 인덱싱을 하면 됩니다.

➤ 슬라이싱

```

...: a = [1, 2, 3, 4, 5]
...: print(a[0:2])
...: b = "12345"
...: print(b[0:2])
...:
[1, 2]
12

```

문자열과 마찬가지로 리스트에서도 슬라이싱 기법을 적용할 수 있고 예시를 통해 문자열에서 슬라이싱 했던 예와 비교해보았습니다.

```
...: a = [1, 2, 3, 4, 5]
...: print(a[:2])
...: print(a[2:])
...:
[1, 2]
[3, 4, 5]
```

문자열 슬라이싱과 마찬가지로 두 부분으로 나누는 전형적인 방법입니다.

```
...: a = [1, 2, 3, ['a', 'b', 'c'], 4, 5]
...: print(a[2:5])
...: print(a[3][:2])
...:
[3, ['a', 'b', 'c'], 4]
['a', 'b']
```

중첩 리스트도 슬라이싱 방법은 똑같이 적용됩니다.

a[3]은 ['a', 'b', 'c']를 나타내고 따라서 a[3][:2]는 ['a', 'b', 'c']의 첫 번째 요소부터 세 번째 요소 직전까지의 값, 즉 ['a', 'b']를 나타냅니다.

➤ 리스트 연산

```
...: a = [1, 2, 3]
...: b = [4, 5, 6]
...: print(a+b)
...: print(a*2)
...:
[1, 2, 3, 4, 5, 6]
[1, 2, 3, 1, 2, 3]
```

리스트 역시 +를 사용해서 더할 수 있고 *를 사용해서 반복할 수 있습니다. 문자열과 마찬가지로 리스트에서도 되는지 직접 확인해보겠습니다.

```
a = [1, 2, 3]
print(a[2] + "hi")
```

```
print(a[2] + "hi")
~~~~~^~~~~~
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

a[2]에 저장된 값은 3이라는 정수인데 "hi"는 문자열이다. 정수와 문자열은 당연히 서로 더할 수 없기 때문에 오류가 발생한 것입니다.

```
...: a = [1, 2, 3]
...: print(str(a[2]) + "hi")
...:
3hi
```

a[2]의 값인 3과 문자열 hi가 더해져서 3hi가 출력될 것이라고 생각할 수 있습니다. 하지만 다음 결과를 살펴보니 오류가 발생했습니다. 오류의 원인은 무엇일까요?

숫자와 문자열을 더해서 '3hi'를 만들고 싶다면 다음처럼 숫자 3을 문자 '3'으로 바꾸어야 합니다.

➤ 리스트 길이 구하기

```
...: a = [1, 2, 3]
...: print(len(a))
...:
3
```

리스트 길이를 구하기 위해서는 다음처럼 len 함수를 사용해야 합니다.

len은 문자열, 리스트 외에 앞으로 배울 튜플과 딕셔너리에도 사용할 수 있는 함수입니다. 자주 사용하므로 잘 기억해 두어야 합니다.

➤ 리스트 요소 수정과 삭제

리스트는 값을 수정하거나 삭제할 수 있습니다.

```
...: a = [1, 2, 3]
...: a[1] = 4
...: print(a)
...: del a[1]
...: print(a)
...:
[1, 4, 3]
[1, 3]
```

```
...: a = [1, 2, 3, 4, 5]
...: del a[2:]
...: print(a)
...:
[1, 2]
```

a[2]의 요소값 3이 4로 수정되었습니다.

del a[x]는 x번째 요소값을 삭제합니다.

del 함수는 파이썬이 자체적으로 가지고 있는 삭제 함수입니다.

슬라이싱 기법을 사용하여 리스트의 요소 여러 개를 한꺼번에 삭제할 수도 있습니다.

➤ 리스트 관련 함수

리스트 변수 이름 뒤에 '.'를 붙여 여러 가지 리스트 관련 함수를 사용할 수 있습니다.

```
...: a = [1, 2, 3]
...: a.append(4)
...: print(a)
...: a.append([5, 6])
...: print(a)
...:
[1, 2, 3, 4]
[1, 2, 3, 4, [5, 6]]
```

append의 사전적 의미는 '첨부하다' 입니다. append(x)는 리스트의 맨 마지막에 x를 추가합니다.

리스트 안에는 어떤 자료형도 추가할 수 있습니다. 예제는 리스트를 추가한 결과입니다.

```

...: a = [1, 4, 3, 2]
...: a.sort()
...: print(a)
...: b = ['a', 'c', 'b']
...: b.sort()
...: print(b)
...:
[1, 2, 3, 4]
['a', 'b', 'c']

```

sort 함수는 리스트의 요소를 순서대로 정렬해 줍니다.

원본 리스트의 요소를 직접 정렬 해준다는 사실을 기억해 두셔야 합니다.

```

...: a = ['a', 'c', 'b']
...: a.reverse()
...: print(a)
...:
['b', 'c', 'a']

```

reverse 함수는 리스트를 역순으로 뒤집어 줍니다. 현재의 리스트를 정렬하는 것이 아니고 그대로 거꾸로 뒤집어 줍니다.

```

...: a = [1, 2, 3, 4, 5]
...: print(a.index(4))
...:
3

```

index(x) 함수는 리스트에 x 값이 있으면 x의 인덱스 값(위치값)을 리턴해줍니다.

```

...: a = [1, 2, 3]
...: a.insert(0, 4)
...: print(a)
...: a.insert(3, 5)
...: print(a)
...:
[4, 1, 2, 3]
[4, 1, 2, 5, 3]

```

insert(a, b)는 리스트의 a번째 위치에 b를 삽입하는 함수이다. 파이썬은 숫자를 0부터 센다는 것을 반드시 기억해 놔야 합니다.

```

...: a = [1, 2, 3, 1, 2, 3]
...: a.remove(3)
...: print(a)
...: a.remove(3)
...: print(a)
...:
[1, 2, 1, 2, 3]
[1, 2, 1, 2]

```

remove(x)는 리스트에서 첫 번째로 나오는 x를 삭제하는 함수입니다.

같은 요소가 두개 있다면 두 번 삭제를 해야 모두 삭제가 가능합니다.

```

...: a = [1, 2, 3]
...: print(a.pop())
...: print(a)
...:
3
[1, 2]

```

pop()은 리스트의 맨 마지막 요소를 리턴하고 그 요소는 삭제합니다.

```

...: a = [1, 2, 3, 1]
...: print(a.count(1))
...:
2

```

count(x)는 리스트 안에 x가 몇 개 있는지 조사하여 그 개수를 리턴해주는 함수입니다.

```

...: a = [1, 2, 3]
...: a.extend([4, 5])
...: print(a)
...: b = [6, 7]
...: a.extend(b)
...: print(a)
...:
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5, 6, 7]

```

extend(x)에서 x에는 리스트만 올 수 있으며 원래의 a 리스트에 x 리스트를 더하게 됩니다.

a += [4, 5]는 a = a + [4, 5]와 동일한 표현식입니다.

■ 튜플 자료형

튜플(tuple)은 몇 가지 점을 제외하곤 리스트와 거의 비슷하며 리스트와 다른 점은 두가지가 있습니다.

1. 리스트는 [], 튜플은 ()으로 둘러싼다.
2. 리스트는 요소값의 생성, 삭제, 수정이 가능하지만, 튜플은 요소값을 바꿀 수 없다.

➤ 표현방법

튜플을 표현할 때 리스트와 다른 2가지 차이점이 있습니다.

```
t1 = ()  
t2 = (1,)   
t3 = (1, 2, 3)  
t4 = 1, 2, 3  
t5 = ('a', 'b', ('ab', 'cd'))
```

t2 = (1,)처럼 단지 1개의 요소만을 가질 때는 요소 뒤에 쉼표(,)를 반드시 붙여야 한다는 것과 t4 = 1, 2, 3처럼 소괄호()를 생략해도 된다는 점입니다.

➤ 튜플 요소 삭제와 수정

```
t1 = (1, 2, 'a', 'b')  
del t1[0]
```

```
TypeError: 'tuple' object doesn't support item deletion
```

튜플은 요소값을 지울 수 없다는 오류 메시지를 확인할 수 있습니다.

```
t1 = (1, 2, 'a', 'b')  
t1[0] = 'c'
```

```
TypeError: 'tuple' object does not support item assignment
```

튜플의 요소값을 변경하려고 해도 오류가 발생하는 것을 확인할 수 있다.

➤ 튜플 인덱싱

```
...: t1 = (1, 2, 'a', 'b')
...: print(t1[0])
...: print(t1[3])
...:
1
b
```

문자열, 리스트와 마찬가지로 `t1[0]`, `t1[3]`처럼 인덱싱이 가능합니다.

➤ 튜플 슬라이싱

```
...: t1 = (1, 2, 'a', 'b')
...: print(t1[:1])
...: print(t1[1:])
...:
(1,)
(2, 'a', 'b')
```

`t1[1]`을 기준으로 튜플의 요소를 나누는 슬라이싱 예이다.

➤ 튜플 더하기

```
...: t1 = (1, 2, 'a', 'b')
...: t2 = (3, 4)
...: t3 = t1 + t2
...: print(t3)
...:
(1, 2, 'a', 'b', 3, 4)
```

튜플을 더하는 방법을 보여 주는 예입니다. `t1`, `t2` 튜플의 요소값이 바뀌는 것은 아니지만, `t1`, `t2` 튜플을 더하여 새로운 튜플 `t3`를 생성한 것입니다.

➤ 튜플 곱하기

```
...: t1 = (3, 4)
...: t2 = t1 * 3
...: print(t2)
...:
(3, 4, 3, 4, 3, 4)
```

예시는 튜플의 곱하기(반복) 보여 줍니다.

➤ 튜플 길이 구하기

```
...: t1 = (1, 2, 'a', 'b')
...: print(len(t1))
...:
4
```

예시는 튜플의 길이를 구하는 방법을 보여 줍니다.

※ 튜플은 요소값을 변경할 수 없기 때문에 `sort`, `insert`, `remove`, `pop`과 같은 내장 함수가 없습니다.

※ 프로그램이 실행되는 동안 요소값이 항상 변하지 않기를 바란다가거나 값이 바뀔까 걱정하고 싶지 않다면 주저하지 말고 튜플을 사용해야 합니다. 이와 반대로 수시로 그 값을 변화시켜야 할 경우라면 리스트를 사용해야 합니다. 실제 프로그램에서는 값이 변경되는 형태의 변수가 훨씬 많기 때문에 평균적으로 튜플보다 리스트를 더 많이 사용합니다.

■ 딕셔너리 자료형

사람은 누구든지 "이름" = "김철수", "생일" = "몇 월 며칠" 등과 같은 방식으로 그 사람이 가진 정보를 나타낼 수 있습니다. 파이썬은 영리하게도 이러한 대응 관계를 나타낼 수 있는 딕셔너리(dictionary) 자료형을 가지고 있습니다. 딕셔너리는 단어 그대로 '사전'이라는 뜻입니다. 즉 "people"이라는 단어에 "사람", "baseball"이라는 단어에 "야구"라는 뜻이 부합되듯이 딕셔너리는 Key와 Value를 한 쌍으로 가지는 자료형이다. 예컨대 Key가 "baseball"이라면 Value는 "야구"가 될 것입니다. 딕셔너리는 리스트나 튜플처럼 순차적으로(sequential) 해당 요소값을 구하지 않고 Key를 통해 Value를 얻습니다. 이것이 바로 딕셔너리의 가장 큰 특징입니다. baseball이라는 단어의 뜻을 찾기 위해 사전의 내용을 순차적으로 모두 검색하는 것이 아니라 baseball이라는 단어가 있는 곳만 펼쳐 보는 것입니다.

➤ 표현방법

Key와 Value의 쌍 여러 개가 {}로 둘러싸여 있다. 각각의 요소는 Key: Value 형태로 이루어져 있고 쉼표(,)로 구분되어 있습니다.

{Key1: Value1, Key2: Value2, Key3: Value3, ...} 이것이 딕셔너리의 기본 형태입니다.

```
...: dic = {'name': 'pey', 'phone': '010-9999-1234', 'birth': '1118'}
...: print(dic)
...:
{'name': 'pey', 'phone': '010-9999-1234', 'birth': '1118'}
```

Key는 각각 'name', 'phone', 'birth', 각각의 Key에 해당하는 Value는 'pey', '010-9999-1234', '1118'이 됩니다.

```
...: a = {1: 'hi'}
...: print(a)
...: b = {'a': [1, 2, 3]}
...: print(b)
...:
{1: 'hi'}
{'a': [1, 2, 3]}
```

Key로 정수값 1, Value로 문자열 'hi'를 사용한 예입니다. Value에 리스트도 넣을 수 있습니다.

➤ 딕셔너리 요소추가

```
...: a = {1: 'a'}
...: a[2] = 'b'
...: print(a)
...: a['name'] = 'pey'
...: print(a)
...: a[3] = [1, 2, 3]
...: print(a)
...:
{1: 'a', 2: 'b'}
{1: 'a', 2: 'b', 'name': 'pey'}
{1: 'a', 2: 'b', 'name': 'pey', 3: [1, 2, 3]}
```

{1: 'a'} 딕셔너리에 a[2] = 'b'와 같이 입력하면 딕셔너리 a에 Key와 Value가 각각 2와 'b'인 {2: 'b'} 딕셔너리 쌍이 추가됩니다.

딕셔너리 a에 {'name': 'pey'} 쌍이 추가했습니다.

Key는 3, Value는 [1, 2, 3]을 가지는 한 쌍이 또 추가되었습니다.

➤ 딕셔너리 요소삭제

del 함수를 사용해서 Key에 해당하는 {Key: Value} 쌍이 삭제된다.

```
...: a = {1: 'a', 2: 'b', 'name': 'pey', 3: [1, 2, 3]}
...: del a['name']
...: print(a)
...:
{1: 'a', 2: 'b', 3: [1, 2, 3]}
```

➤ 딕셔너리 사용 방법

리스트나 튜플, 문자열은 요소값을 얻고자 할 때 인덱싱이나 슬라이싱 기법 중 하나를 사용했습니다. 하지만 딕셔너리는 단 1가지 방법 뿐입니다.

```
...: grade = {'tom': 50, 'jane': 90}
...: print(grade['tom'])
...: print(grade['jane'])
...:
50
90
```

```
...: a = {1:'a', 2:'b'}
...: print(a[1])
...: print(a[2])
...:
a
b
```

Key를 사용해서 Value를 구하는 방법입니다. Key의 Value를 얻기 위해서는 '딕셔너리_변수_이름[Key]'를 사용해야 합니다.

a[1]이 의미하는 것은 리스트나 튜플의 a[1]과는 전혀 다릅니다. 딕셔너리 변수에서 [] 안의 숫자 1은 두 번째 요소를 나타내는 것이 아니라 Key에 해당하는 1을 나타냅니다.

➤ 딕셔너리 자료형 만들 때 주의사항

```
...: a = {1:'a', 1:'b'}  
...: print(a)  
...:  
{1: 'b'}
```

Key는 고유한 값이므로 중복되는 Key 값을 설정해 놓으면 하나를 제외한 나머지 것들이 모두 무시됩니다

```
a = {[1,2] : 'hi'}  
print(a)
```

```
TypeError: unhashable type: 'list'
```

Key에 리스트는 쓸 수 없습니다. 딕셔너리의 Key로 쓸 수 있느냐, 없느냐는 Key가 변하는(mutable) 값인지, 변하지 않는(immutable) 값인지에 달려 있습니다.

➤ 딕셔너리 관련 함수

```
...: a = {'name': 'tom', 'phone': '010-1111-1004', 'birth': '1212'}
...: print(a.keys())
...: print(list(a.keys()))
...:
dict_keys(['name', 'phone', 'birth'])
['name', 'phone', 'birth']
```

keys()함수는 딕셔너리 a의 Key만을 모아 dict_keys 객체를 리턴합니다.

dict_keys 객체를 리스트로 변환하려면 list() 함수를 예시처럼 이용합니다.

```
...: a = {'name': 'pey', 'phone': '010-9999-1234', 'birth': '1118'}
...: print(a.values())
...: print(list(a.values()))
...:
dict_values(['pey', '010-9999-1234', '1118'])
['pey', '010-9999-1234', '1118']
```

Key를 얻는 것과 마찬가지로 방법으로 Value만 얻고 싶다면 values() 함수를 사용하면 됩니다. Values() 함수를 호출하면 dict_values 객체를 리턴합니다.

```
...: a = {'name': 'pey', 'phone': '010-9999-1234', 'birth': '1118'}
...: print(a.items())
...: print(list(a.items()))
...:
dict_items([('name', 'pey'), ('phone', '010-9999-1234'), ('birth', '1118')])
[('name', 'pey'), ('phone', '010-9999-1234'), ('birth', '1118')]
```

Items() 함수는 Key와 Value의 쌍을 튜플로 묶은 값을 dict_items 객체로 리턴합니다.


```

...: a = {'name': 'pey', 'phone': '010-9999-1234', 'birth': '1118'}
...: a.clear()
...: print(a)
...:
{}

```

clear() 함수는 딕셔너리 안의 모든 요소를 삭제합니다.

```

...: a = {'name': 'tom', 'phone': '010-1111-1004', 'birth': '1212'}
...: print(a.get('name'))
...: print(a['name'])
...:
tom
tom

```

get(x) 함수는 x라는 Key에 대응되는 Value를 리턴합니다. a.get('name')은 a['name']을 사용했을 때와 동일한 결과값을 리턴합니다.

```

...: a = {'name': 'tom', 'phone': '010-1111-1004', 'birth': '1212'}
...: print(a.get('address'))
...:
None

```

```

a = {'name': 'tom', 'phone': '010-1111-1004', 'birth': '1212'}
print(a['address'])

```

```

KeyError: 'address'

```

다음 예제에서 볼 수 있듯이 딕셔너리에 존재하지 않는 키로 값을 가져오려고 할 경우 a.get('address')은 None을 리턴하여 오류가 발생하지 않고 a['address'] 방식은 오류를 발생시킵니다. 여기에서 None은 '거짓'이라는 뜻입니다.

```

...: a = {'name': 'tom', 'phone': '010-1111-1004', 'birth': '1212'}
...: print(a.get('address', 'Daegu'))
...:
Daegu

```

딕셔너리 안에 찾으려는 Key가 없을 경우, 디폴트 값을 대신 가져오게 하고 싶을 때는 get(x, '디폴트 값')을 사용하면 편리합니다.

```

...: a = {'name': 'tom', 'phone': '010-1111-1004', 'birth': '1212'}
...: print('name' in a)
...: print('address' in a)
...:
True
False

```

in을 이용하면 해당 Key가 딕셔너리 안에 있는지 조사하기 쉽습니다. 딕셔너리 안에 존재하지 않는 Key를 쓰면 거짓(False)을 리턴하고 존재하면 참(True)을 리턴합니다.

➤ dict_keys 객체, dict_values 객체, dict_items 객체

```

...: a = {'name': 'tom', 'phone': '010-1111-1004', 'birth': '1212'}
...: for i in a.keys():
...:     print(i)
...:
name
phone
birth

```

리스트를 사용하는 것과 별 차이는 없지만, 리스트 고유의 append, insert, pop, remove, sort 함수는 수행할 수 없습니다.

```

...: a = {'name': 'tom', 'phone': '010-1111-1004', 'birth': '1212'}
...: for i in a.values():
...:     print(i)
...:
tom
010-1111-1004
1212

```

뒤에서 배울 반복문을 통해서 dict_keys 객체, dict_values 객체, dict_items 객체들의 요소를 뽑아내는 법을 간단히 보고 뒤에서 더 자세히 배우겠습니다.

```

...: a = {'name': 'tom', 'phone': '010-1111-1004', 'birth': '1212'}
...: for i in a.items():
...:     print(i)
...:
('name', 'tom')
('phone', '010-1111-1004')
('birth', '1212')

```

■ 집합 자료형

집합(set)은 집합에 관련된 것을 쉽게 처리하기 위해 만든 자료형입니다.

➤ 표현방법

```
...: s = set()
...: print(s)
...:
set()
```

집합 자료형은 다음과 같이 set 키워드를 사용해 만들 수 있습니다.

```
...: s1 = set([1, 2, 3])
...: print(s1)
...: s2 = set("Python")
...: print(s2)
...:
{1, 2, 3}
{'P', 'y', 't', 'n', 'o', 'h'}
```

set()의 괄호 안에 리스트를 입력하여 만들거나 다음과 같이 문자열을 입력하여 만들 수도 있습니다.

➤ 집합 자료형의 특징

위에서 살펴본 set("Python")의 결과가 좀 이상하지 않은가요? 분명 "Python" 문자열로 set 자료형을 만들었는데 생성된 자료형에는 순서가 뒤죽박죽입니다. set에 다음과 같은 2가지 특징이 있습니다.

1. 순서가 없습니다.
2. 중복을 허용하지 않습니다.

```

...: s1 = set([1, 2, 3, 2, 3, 1, 2, 3, 1])
...: print(s1)
...:
{1, 2, 3}

```

```

...: data = [1, 2, 2, 3, 4, 4, 5]
...: unique_data = set(data)
...: print(unique_data)
...:
{1, 2, 3, 4, 5}

```

중복을 허용하지 않는 특징을 이용하여 유일한 요소만 뽑아내는 과정입니다.

➤ 변환

```

...: s1 = set([1, 2, 3])
...: l1 = list(s1)
...: print(l1)
...: print(l1[2])
...: t1 = tuple(s1)
...: print(t1)
...: print(t1[2])
...:
[1, 2, 3]
3
(1, 2, 3)
3

```

set 자료형은 순서가 없기(unordered) 때문에 인덱싱을 통해 요소값을 얻을 수 없습니다.

set 자료형에 저장된 값을 인덱싱으로 접근하려면 다음과 같이 리스트나 튜플로 변환한 후에 해야 합니다.

➤ 교집합, 합집합, 차집합 구하기

```
...: set1 = {1, 2, 3, 4}
...: set2 = {3, 4, 5, 6}
...: print(set1 & set2)
...: print(set1 | set2)
...: print(set1 - set2)
...: print(set2 - set1)
...:
{3, 4}
{1, 2, 3, 4, 5, 6}
{1, 2}
{5, 6}
```

2개의 set 자료형을 만든 후 교집합, 합집합, 차집합을 구해보겠습니다.

‘&’를 이용하면 교집합을 간단히 구할 수 있습니다.

‘|’를 사용하면 합집합을 구할 수 있습니다

‘-’를 사용하면 차집합을 구할 수 있습니다.

➤ 함수로 교집합, 합집합, 차집합 구하기

```
...: set1 = {1, 2, 3, 4}
...: set2 = {3, 4, 5, 6}
...: print(set1.intersection(set2))
...: print(set1.union(set2))
...: print(set1.difference(set2))
...: print(set2.difference(set1))
...:
{3, 4}
{1, 2, 3, 4, 5, 6}
{1, 2}
{5, 6}
```

intersection 함수를 사용해 차집합을 구합니다.

union 함수를 사용 사용해 차집합을 구합니다.

difference 함수를 사용해 차집합을 구합니다.

➤ 집합 자료형 관련 함수

```
...: set = {1, 2, 3, 4}
...: set.add(5)
...: print(set)
...: set.update([6, 7, 8])
...: print(set)
...: set.remove(2)
...: print(set)
...:
{1, 2, 3, 4, 5}
{1, 2, 3, 4, 5, 6, 7, 8}
{1, 3, 4, 5, 6, 7, 8}
```

Add함수는 1개의 값만 추가할 때 이용합니다.

Update 함수는 여러 개의 값을 한꺼번에 추가 할 때 이용합니다.

Remove 함수는 특정 값을 제거하고 싶을 때 이용합니다.

■ 논리 자료형

논리 자료형이란 참(True)과 거짓(False)을 나타내는 자료형입니다. 논리 자료형은 True, False 2가지 값을 가질 수 있습니다.

```
...: a = True
...: b = False
...: print(type(a))
...: print(type(b))
...:
<class 'bool'>
<class 'bool'>
```

1. True: 참을 의미.

2. False: 거짓을 의미

True나 False는 파이썬의 예약어로, true, false와 같이 작성하면 안 되고 첫 문자를 대문자로 작성해야 합니다.

➤ 사용방법

불 자료형은 조건문의 리턴값으로도 사용됩니다. 조건문에 대해서는 if 문에서 자세히 배우겠지만 잠시 살펴보고 넘어가겠습니다.

```
...: print(1 == 1)
...: print(2 > 1)
...: print(2 < 1)
...:
True
True
False
```

'1과 1이 같은가?'를 묻는 조건문입니다. 이런 조건문은 결과로 True 또는 False에 해당하는 불 자료형을 리턴한다. 1과 1은 같으므로 True를 리턴합니다.

2는 1보다 크므로 조건문은 참이고 True를 리턴합니다.

2는 1보다 작지 않으므로 조건문은 거짓이고 False를 리턴합니다.

➤ 자료형의 참과 거짓

```
...: print(bool('python'))
...: print(bool(''))
...: print(bool([1, 2, 3]))
...: print(bool([]))
...: print(bool(0))
...: print(bool(7))
...:
True
False
True
False
False
True
```

값	논리
"python"	참
" "	거짓
[1, 2, 3]	참
[]	거짓
1	참
0	거짓
None	거짓

➤ 응용

```
...: a = [1, 2, 3, 4]
...: while a:
...:     print(a.pop())
...:
4
3
2
1
```

니다. 결국 더 이상 끄집어 낼 것이 없으면 a가 빈 리스트([])가 되어 거짓이 됩니다. 따라서 while 문에서 조건문이 거짓이 되므로 while 문을 빠져나가게 됩니다. 이는 파이썬 프로그래밍에서 매우 자주 사용하는 기법 중 하나입니다.

```
...: if []:
...:     print("참")
...: else:
...:     print("거짓")
...:
거짓
```

while 문은 뒤에서 공부하겠지만, 간단히 알아보면 조건문이 참인 동안 조건문 안에 있는 문장을 반복해서 수행합니다. a.pop() 함수는 리스트 a의 마지막 요소를 끄집어 내는 함수이므로 리스트 안에 요소가 존재하는 한(a가 참인 동안) 마지막 요소를 계속 끄집어 낼 것입

[]는 앞의 표에서 볼 수 있듯이 비어 있는 리스트이므로 거짓입니다. 따라서 "거짓"이라는 문자열이 출력됩니다. if 문에 대해서 뒤에서 공부해 보겠습니다.

■ 변수

자료형을 저장하는 공간을 변수라고 합니다.

변수_이름 = 변수에_저장할_값

다른 프로그래밍 언어인 C나 JAVA에서는 변수를 만들 때 자료형의 타입을 직접 지정해야 하지만 파이썬은 변수에 저장된 값을 스스로 판단하여 자료형의 타입을 지정하기 때문에 더 편리합니다.

```
a = 1
b = "python"
c = [1, 2, 3]
```

우리는 앞에서 이미 변수를 사용해 왔고 다음 예와 같은 a, b, c를 '변수'라고 한다.

➤ 변수란

파이썬에서 사용하는 변수는 객체를 가리키는 것이라고도 말할 수 있습니다. 객체란 우리가 지금까지 보아 온 자료형의 데이터(값)와 같은 것을 의미하는 말입니다.

```
...: a = 1
...: b = "python"
...: c = [1, 2, 3]
...: print(id(a))
...: print(id(b))
...: print(id(c))
...:
140714974036776
2763140643504
2763160254528
```

옆에 예시 코드처럼 변수를 만들면 데이터(객체)가 자동으로 메모리에 생성되고 변수는 저장된 메모리의 주소를 가리키게 된다. 변수가 가리키는 메모리의 주소는 id() 함수를 이용하여 알 수 있습니다.

※ 메모리란 컴퓨터가 데이터를 기억하는 공간을 말한다.

➤ 리스트 복사

```
...: a = [1, 2, 3]
...: b = a
...: print(id(a))
...: print(id(b))
...:
2763160195008
2763160195008
```

```
...: a = [1, 2, 3]
...: b = a
...: a[1] = 4
...: print(a)
...: print(b)
...:
[1, 4, 3]
[1, 4, 3]
```

b 변수에 a 변수를 대입하면 어떻게 될까요? b와 a는 다른 걸까요? 결론부터 말하면 b는 a와 완전히 동일하다고 할 수 있습니다. 다만 [1, 2, 3]이라는 리스트 객체를 참조하는 변수가 a 변수 1개에서 b 변수가 추가되어 2개로 늘어났다는 차이만 있을 뿐입니다.

a 리스트의 두 번째 요소를 값 4로 바꾸었더니 a만 바뀌는 것이 아니라 b도 똑같이 바뀐 이유는 앞에서 살펴본 것처럼 a, b 모두 동일한 리스트를 가리키고 있기 때문입니다.

변수를 복사하면서 다른 주소를 가리키도록 만드는 2가지 방법이 있습니다.

```
...: a = [1, 2, 3]
...: b = a[:]
...: a[1] = 4
...: print(a)
...: print(b)
...: print(id(a))
...: print(id(b))
...:
[1, 4, 3]
[1, 2, 3]
2763160772864
2763160217280
```

첫번째 방법 [:] 이용

a 리스트 값을 바꾸더라도 b 리스트에는 아무런 영향이 없습니다.

```

...: from copy import copy
...: a = [1, 2, 3]
...: b = copy(a)
...: a[1] = 4
...: print(a)
...: print(b)
...: print(id(a))
...: print(id(b))
...:
[1, 4, 3]
[1, 2, 3]
2763160787136
2763160213056

```

두번째 방법 copy 모듈 이용

from copy import copy라는 처음 보는 형태의 문장이 나오는데, 이것은 뒤에 설명할 파이썬 모듈 부분에서 자세히 배웁니다. 지금은 copy 함수를 쓰기 위해서 사용하는 것이라고 알아두면 됩니다.

b = copy(a)는 b = a[:]과 동일합니다.

```

...: a = [1, 2, 3]
...: b = a.copy()
...: a[1] = 4
...: print(a)
...: print(b)
...: print(id(a))
...: print(id(b))
...:
[1, 4, 3]
[1, 2, 3]
2763159966592
2763160772416

```

※ 리스트 자료형의 자체 함수인 copy 함수를 사용해도 copy 모듈을 사용하는 것과 동일한 결과를 얻을 수 있습니다

➤ 변수를 만드는 다양한 방법

```
...: a, b = ('python', 'java')
...: print(a)
...: print(b)
...: (b, a) = 'python', 'java'
...: print(a)
...: print(b)
...:
python
java
java
python
```

튜플로 a, b에 값을 대입할 수 있습니다. 튜플은 괄호를 생략해도 됩니다.

```
...: [a, b] = ['python', 'life']
...: print(a)
...: print(b)
...: a = b = 'HTML'
...: print(a)
...: print(b)
...:
python
life
HTML
HTML
```

리스트로 변수를 만들 수도 있습니다. 여러 개의 변수에 같은 값을 대입할 수도 있습니다.

```
...: a = 'python'
...: b = 'java'
...: a, b = b, a
...: print(a)
...: print(b)
...:
java
python
```

예시와 같은 방법을 사용하여 두 변수의 값을 매우 간단하게 바꿀 수도 있습니다.

파이썬 제어문

```
...: money = True
...: if money:
...:     print("과자를 사라")
...: else:
...:     print("가게를 나와라")
...:
과자를 사라
```

수행하는 데 쓰는 것이 바로 if 문입니다.

money에 True를 대입했으므로 money는 참이다. 따라서 if 문 다음 문장이 수행되어 '과자를 사라'가 출력됩니다.

■ If 문 개요

프로그래밍을 할 때는 주어진 조건을 판단한 후 그 상황에 맞게 처리해야 할 경우가 생깁니다. 이렇듯 프로그래밍에서 조건을 판단하여 해당 조건에 맞는 상황을

➤ If문의 기본 구조

```
if 조건문:  
    조건문이_참이면_수행_할_코드1  
조건문이_참이면_수행_할_코드2  
    조건문이_참이면_수행_할_코드3
```

if와 else를 사용한 조건문의 기본 구조입니다.

조건문을 테스트해서 참이면 if 문 바로 다음

문장들을 수행하고 조건문이 거짓이면 else 문 다음 문장들을 수행하게 됩니다.

➤ 들여쓰기

들여쓰기를 잘못된 예시입니다.

```
...: money = True  
...: if money:  
...:     print("과자를 사라")  
...: else:  
...: print("가게를 나와라")  
...:  
Cell In[60], line 5  
    print("가게를 나와라")  
    ^  
IndentationError: expected an indented block after 'else' statement on line 4
```

if 문을 만들 때는 if 조건문 바로 다음 문장부터 if 문에 속하는 모든 문장에 들여쓰기(indentation)를 해야 합니다.

그렇지 않으면 다음과 같이 오류가 발생합니다.

➤ 조건문이란

if 조건문에서 '조건문'이란 참과 거짓을 판단하는 문장을 말합니다.

```
if 조건문:
    조건문이 참이면 수행 할 코드
    ...
else:
    조건문이 거짓이면 수행 할 코드
    ...
```

위에서 살펴본 과자 예시에서는 조건문은 money가 됩니다.

➤ 비교연산자

조건문에 쓰이는 비교연산자에 대해 알아보겠습니다.

비교연산자	설명
$x < y$	x가 y보다 작다.
$x > y$	x가 y보다 크다.
$x == y$	x와 y가 같다.
$x != y$	x와 y가 같지 않다.
$x \geq y$	x가 y보다 크거나 같다.
$x \leq y$	x가 y보다 작거나 같다.

```
...: x = 5
...: y = 3
...: print(x > y)
...: print(x < y)
...: print(x == y)
...: print(x != y)
...:
True
False
False
True
```

x에 5, y에3를 대입한 후

x > y 라는 조건문은 참이므로 True를 리턴합니다.

x < y 라는 조건문은 거짓이므로 False를 리턴합니다.

x == y 라는 조건문은 거짓이므로 False를 리턴합니다.

x != y 라는 조건문은 참이므로 True를 리턴합니다.

```
...: money= 4000
...: if money >= 3000:
...:     print("과자를 사라")
...: else:
...:     print("가게를 나와라")
...:
과자를 사라
```

조건문이 참이 되기 때문에 if 문 바로 다음 문장들을 수행하게 됩니다.

➤ and, or, not

조건을 판단하기 위해 사용하는 다른 연산자로는 and, or, not이 있습니다.

연산자	설명
x or y	x와 y 둘 중 하나만 참이어도 참이다.
x and y	x와 y 모두 참이어야 참이다.
not x	x가 거짓이면 참이다.

money는 2000이지만, card가 True이기 때문에 조건문이 참이 된다. 따라서 if 문에 속한 '과자를 사라' 문장이 출력됩니다.

```
...: money= 2000
...: card = True
...: if money >= 3000 or card:
...:     print("과자를 사라")
...: else:
...:     print("가게를 나와라")
...:
과자를 사라
```

➤ in, not in

영어 단어 in의 뜻이 ‘~안에’라는 것을 생각해 보면 다음 예가 쉽게 이해할 수 있습니다.

in	not in
x in 리스트	x not in 리스트
x in 튜플	x not in 튜플
x in 문자열	x not in 문자열

```
...: a = 1 in [1, 2, 3]
...: print(a)
...: b = 1 not in [1, 2, 3]
...: print(b)
...:
True
False
```

첫 번째 예는 [1, 2, 3]이라는 리스트 안에 1이 있는가?라는 조건문입니다. 1은 [1, 2, 3] 안에 있으므로 참이 되어 True를 리턴한다. 두 번째 예는 [1, 2, 3] 리스트 안에 1이 없는가?라는 조건문입니다. 1은 [1, 2, 3] 안에 있으므로 거짓이 되어 False를 리턴합니다.

```
...: a = 'a' in ('a', 'b', 'c')
...: print(a)
...: b = 'j' not in 'python'
...: print(b)
...:
True
True
```

튜플과 문자열에 in과 not in을 적용한 예입니다.

```

...: pocket = ['paper', 'coin', 'money']
...: if 'money' in pocket:
...:     print("과자를 사라")
...: else:
...:     print("가게를 나와라")
...:
과자를 사라

```

전에 이용하던 예시를 in을 이용하여 바꾸어 보았습니다.

➤ pass

```

...: pocket = ['paper', 'coin', 'money']
...: if 'money' in pocket:
...:     pass
...: else:
...:     print("가게를 나와라")
...:

```

가끔 조건문의 참, 거짓에 따라 실행할 행동을 정의할 때나 아무런 일도 하지 않도록 설정하고 싶을 때가 있습니다. 그 땐 pass 를 이용해보면 아무런 결과값도 나오지 않습니다.

➤ elif

```

...: pocket = ['paper', 'coin']
...: card = True
...: if 'money' in pocket:
...:     print("과자를 사라")
...: else:
...:     if card:
...:         print("과자를 사라")
...:     else:
...:         print("가게를 나와라")
...:
과자를 사라

```

if와 else만으로는 여러 조건을 판단하기 복잡해 집니다.

옆에 예시만 보더라도 언뜻 보기에 이해하기 어렵고 산만한 느낌이 듭니다. 이런 복잡함을 해결하기 위해 파이썬에서는 다중 조건 판단을 가능하게 하는 elif를 사용합니다.

```

...: pocket = ['paper', 'coin']
...: card = True
...: if 'money' in pocket:
...:     print("과자를 사라")
...: elif card:
...:     print("과자를 사라")
...: else:
...:     print("가게를 나와라")
...:

```

과자를 사라

elif는 이전 조건문이 거짓일 때 수행됩니다. 위에 코드와 비교해 보았을 때 더욱 직관적으로 조건들을 판단하기 쉬워집니다.

```

if 조건문:
    조건문이_참이면_수행_할_코드
    ...
elif 조건문:
    if_조건문이_거짓이고_elif_조건문이_참이면_수행_할_코드
    ...
else:
    if_조건문이_거짓이고_elif_조건문이_거짓이면_수행_할_코드
    ...

```

elif는 개수에 제한 없이 사용할 수 있습니다.

if, elif, else를 모두 사용할 때 기본 구조는 다음과 같습니다.

➤ If 문 한 줄로 적기

```

...: pocket = ['paper', 'coin', 'money']
...: if 'money' in pocket: print("과자를 사라")
...: else: print("가게를 나와라")
...:

```

과자를 사라

if 문 다음에 수행할 문장을 콜론(:) 뒤에 바로 적어도 가능합니다. else 문 역시 마찬가지입니다.

➤ 조건부 표현식

```
if score >= 60:
    message = "success"
else:
    message = "failure"
```

```
message = "success" if score >= 60 else "failure"
```

```
...: score = 75
...: message = "success" if score >= 60 else "failure"
...: print(message)
...:
success
```

score가 60 이상일 때 message에 "success", 아닐 경우에는 "failure"를 대입하는 코드입니다.

조건부 표현식을 이용하여 한 줄로 간단히 표현할 수 있습니다.

기본구조는 아래와 같습니다.

변수 = 조건문이_참인_경우의_값 if 조건문 else 조건문이_거짓인_경우의_값

■ While 반복문

문장을 반복해서 수행해야 할 경우 while 문을 사용합니다. 그래서 while 문을 ‘반복문’이라고도 부릅니다.

➤ 기본구조

```
while 조건문:
    수행할_코드1
    수행할_코드2
    수행할_코드3
...
```

while 문은 조건문이 참인 동안 while 문에 속한 문장들이 반복해서 수행됩니다.

```
...: treeHit = 0
...: while treeHit < 5:
...:     treeHit = treeHit + 1
...:     print(f"나무를 {treeHit}번 찍었습니다.")
...:     if treeHit == 5:
...:         print("나무 넘어갑니다.")
...:
나무를 1번 찍었습니다.
나무를 2번 찍었습니다.
나무를 3번 찍었습니다.
나무를 4번 찍었습니다.
나무를 5번 찍었습니다.
나무 넘어갑니다.
```

예에서 while 문의 조건문은 `treeHit < 5` 입니다. treeHit가 5보다 작은 동안 while 문에 포함된 문장들을 계속 수행합니다. while문 안의 문장을 보면 가장 먼저 `treeHit+1` 를 이용하여 treeHit 값이 계속 1씩 증가한다는 것을 알 수 있습니다. treeHit가 5가 되면 "나무 넘어갑니다."라는 문장을 출력합니다. 그이 후 `treeHit < 5` 조건문이 거짓이 되므로 while 문을 빠져나가게 됩니다.

treeHit	조건문	논리	수행하는 문장	While문
0	<code>0 < 5</code>	참	나무를 1번 찍었습니다.	반복
1	<code>1 < 5</code>	참	나무를 2번 찍었습니다.	반복
2	<code>2 < 5</code>	참	나무를 3번 찍었습니다.	반복
3	<code>3 < 5</code>	참	나무를 4번 찍었습니다.	반복
4	<code>4 < 5</code>	참	나무를 5번 찍었습니다. 나무 넘어갑니다.	반복
5	<code>5 < 5</code>	거짓		종료

```

...: prompt = """
...: 1. Add
...: 2. Del
...: 3. List
...: 4. Quit
...:
...: Enter number: """
...: number = 0
...: while number != 4:
...:     print(prompt)
...:     number = int(input())
...:

```

```

1. Add
2. Del
3. List
4. Quit

```

Enter number:

```
>? 1
```

```

1. Add
2. Del
3. List
4. Quit

```

Enter number:

```
>? 4
```

여러 가지 선택지 중 하나를 선택해서 입력 받는 예입니다.

number 변수에 0을 먼저 대입합니다. 이렇게 변수를 먼저 설정해 놓지 않으면 다음에 나올 while 문의 조건문인 number != 4 에서 변수가 존재하지 않는다는 오류가 발생합니다.

while 문을 보면 number가 4가 아닌 동안 prompt를 출력하고 사용자로부터 번호를 입력 받습니다. 다음 결과 화면처럼 사용자가 값 4를 입력하지 않으면 계속해서 prompt를 출력한다.

number = int(input()) 는 사용자의 숫자 입력을 받아들이는 함수이고 뒤에서 다루겠습니다.

4를 입력하면 조건문이 거짓이 되어 while 문을 빠져나가게 됩니다.

➤ while 문 강제로 빠져나가기

while 문은 조건문이 참인 동안 계속 while 문 안의 내용을 반복적으로 수행합니다. 하지만 강제로 while 문을 빠져나가고 싶을 때가 있습니다.

이렇게 코드를 강제로 멈추게 하는 것이 바로 break 문입니다.

```
...: coffee = 3
...: money = 300
...: while money:
...:     print("돈을 받았으니 커피를 줍니다.")
...:     coffee = coffee - 1
...:     print(f"남은 커피의 양은 {coffee}개입니다.")
...:     if coffee == 0:
...:         print("커피가 다 떨어졌습니다. 판매를 중지합니다.")
...:         break
...:
```

돈을 받았으니 커피를 줍니다.
남은 커피의 양은 2개입니다.
돈을 받았으니 커피를 줍니다.
남은 커피의 양은 1개입니다.
돈을 받았으니 커피를 줍니다.
남은 커피의 양은 0개입니다.
커피가 다 떨어졌습니다. 판매를 중지합니다.

예시 코드에서 money는 0이 아니기 때문에 항상 참입니다. 따라서 무한히 반복되는 무한 루프를 돌게 됩니다

while 문의 내용을 한 번 수행할 때마다 coffee의 개수가 1개씩 줄어들고 coffee가 0이 되면 coffee == 0 이 참이 되므로 if 문 다음 문장 "커피가 다 떨어졌습니다. 판매를 중지합니다."가 출력되고 break 문이 호출되어 while 문을 빠져나가게 됩니다.

➤ while 문의 맨 처음으로 돌아가기

while 문 안의 문장을 수행할 때 입력 조건을 검사해서 조건에 맞지 않으면 while 문을 빠져나갑니다. 그런데 프로그래밍을 하다 보면 while 문을 빠져나가지 않고 while 문의 맨 처음(조건문)으로 다시 돌아가게 만들고 싶은 경우가 생기게 됩니다. 이때 사용하는 것이 바로 continue 문입니다.

```
...: a = 0
...: while a < 10:
...:     a = a + 1
...:     if a % 2 == 0: continue
...:     print(a)
...:
1
3
5
7
9
```

1~10 사이에서 홀수만 출력하는 것을 while 문을 사용하였습니다. a가 10보다 작은 동안 a는 1만큼씩 계속 증가하고 a % 2 == 0 이 참이 되는 경우는 a가 짝수일 때입니다. 즉, a가 짝수이면 continue 문을 수행합니다. 이 continue 문은 while 문의 맨 처음인 조건문(a < 10)으로 돌아가게 하는 명령어입니다. 따라서 a가 짝수이면 print(a) 문장은 수행되지 않을 것입니다.

➤ 무한 루프

무한 루프란 무한히 반복한다는 의미입니다. 우리가 사용하는 일반 프로그램 중에서 무한 루프 개념을 사용하지 않는 프로그램은 거의 없습니다. 그만큼 자주 사용한다는 뜻입니다.

```
while True:
    수행할_코드1
    수행할_코드2
    ...
```

조건문이 항상 참이 되고 while 문 안에 있는 문장들은 무한히 수행될 것입니다. [Ctrl+C]로 빠져 나갈 수 있습니다.

■ For문

파이썬의 직관적인 특징을 가장 잘 보여 주는 것이 바로 이 for 문입니다. while 문과 비슷한 반복문인 for 문은 문장 구조가 한눈에 들어온다는 장점이 있습니다.

➤ 기본구조

```
for 변수 in 리스트(또는 튜플, 문자열):  
    변수_포함한_수행할_코드1  
    변수_포함한_수행할_코드2  
    ...
```

리스트나 튜플, 문자열의 첫 번째 요소부터 마지막 요소까지 차례로 변수에 대입되어 코드가 수행됩니다.

```
...: list = ['tom', 'jane', 'tim']  
...: for i in list:  
...:     print(i)  
...:  
tom  
jane  
tim
```

리스트의 첫 번째 요소인 'tom'이 먼저 i 변수에 대입된 후 print(i) 문장을 수행한다. 다음에 두 번째 요소 'jane'가 i 변수에 대입된 후 print(i) 문장을 수행하고 리스트의 마지막 요소까지 이것을 반복합니다.

➤ 다양한 for문의 사용

```
...: a = [(1,2), (3,4), (5,6)]  
...: for (first, second) in a:  
...:     print(first + second)  
...:  
3  
7  
11
```

예시는 a 리스트의 요소값이 튜플이기 때문에 각각의 요소가 자동으로 (first, last) 변수에 대입됩니다.

➤ for문의 응용

문제 : 총 5명의 학생이 시험을 보았는데 시험 점수가 60점 이상이면 합격이고 그렇지 않으면 불합격이다. 합격인지, 불합격인지 결과를 보여 주시오.

```
...: scores = [90, 25, 67, 45, 80]
...: number = 0
...: for score in scores:
...:     number = number + 1
...:     if score >= 60:
...:         print(f"{number}번 학생은 합격입니다.")
...:     else:
...:         print(f"{number}번 학생은 불합격입니다.")
...:
1번 학생은 합격입니다.
2번 학생은 불합격입니다.
3번 학생은 합격입니다.
4번 학생은 불합격입니다.
5번 학생은 합격입니다.
```

먼저 학생 5명의 시험 점수를 리스트로 표현해 봅시다.

1번 학생은 90점이고 5번 학생은 80점입니다. 이런 점수를 차례로 검사해서 합격했는지, 불합격했는지 통보해 주는 프로그램을 만들어 보겠습니다.

각각의 학생에게 번호를 붙여 주기 위해 number 변수를 사용하였습니다. 점수 리스트에서 차례로 점수를 꺼내어 score라는 변수에 대입하고 for 문 안의 문장들

을 수행합니다. 먼저 for 문이 한 번씩 수행될 때마다 number는 1씩 증가합니다.

이 프로그램을 실행하면 score가 60 이상일 때 합격 메시지를 출력하고 60을 넘지 않을 때 불합격 메시지를 출력합니다

➤ for 문과 continue 문

```
...: scores = [90, 25, 67, 45, 80]
...: number = 0
...: for score in scores:
...:     number = number + 1
...:     if score < 60:
...:         continue
...:     else:
...:         print(f"{number}번 학생은 합격입니다.")
...:
1번 학생은 합격입니다.
3번 학생은 합격입니다.
5번 학생은 합격입니다.
```

위에 문제에서 60점 이상인 사람에게는 축하 메시지를 보내고 나머지 사람에게는 아무런 메시지도 전하지 않는 프로그램을 만들어 보겠습니다.

점수가 60점 이하인 학생인 경우에는 `score < 60` 이 참이 되어 `continue` 문이 수행됩니다. 따라서 축하 메시지를 출력하는 부분인 `print` 문을 수행하지 않고 `for` 문의 처음으로 돌아가게 됩니다.

➤ range 함수

`for` 문은 숫자 리스트를 자동으로 만들어 주는 `range` 함수와 함께 사용하는 경우가 많습니다. 다음은 `range` 함수의 간단한 사용법입니다.

```
...: a = range(1, 11)
...: print(a)
...: b = range(10)
...: print(b)
...:
range(1, 11)
range(0, 10)
```

`range(시작_숫자, 끝_숫자)` 형태를 사용하는데, 이때 끝 숫자는 포함되지 않습니다.

`range(10)`은 0부터 10 미만의 숫자를 포함하는 `range` 객체를 만들어 줍니다.

```

...: add = 0
...: for ele in range(1, 11):
...:     add = add + ele
...: print(add)
...:

```

55

for와 range 함수를 사용하면 1부터 10까지 더하는 것을 다음과 같이 쉽게 구현할 수 있습니다.

➤ for와 range를 이용한 구구단

```

...: for i in range(2,10):
...:     for j in range(1, 10):
...:         print(i * j, end="\t")
...:     print(" ")
...:

```

```

2  4  6  8  10 12 14 16 18
3  6  9  12 15 18 21 24 27
4  8  12 16 20 24 28 32 36
5  10 15 20 25 30 35 40 45
6  12 18 24 30 36 42 48 54
7  14 21 28 35 42 49 56 63
8  16 24 32 40 48 56 64 72
9  18 27 36 45 54 63 72 81

```

예를 보면 for 문을 두 번 사용했다. 1번 for 문에서 2부터 9까지의 숫자가 차례대로 i에 대입되고 i가 처음 2일 때 2번 for 문을 만나게 된다. 2번 for 문에서 1부터 9까지의 숫자 가 j에 대입되고 그 다음 문장인 print(i*j, end="\t ")를 수행합니다.

따라서 i가 2일 때 2 * 1, 2 * 2, 2 * 3, ... 2 * 9까지 차례대로 수행되며 그 값을 출력하게 됩니다. 그 다음으로 i가 3일 때 역시 2일 때와 마찬가지로 수행될 것이고 i

가 9일 때까지 계속 반복됩니다.

print 문의 end 매개변수에는 줄 바꿈 문자(\n)가 기본값으로 설정되어 있습니다. 그래서 print(i*j, end=" ")와 같이 print 함수에 end 파라미터를 설정한 이유는 해당 결과값을 출력할 때 다음 줄로 넘기지 않고 그 줄에 계속 출력하기 위해서 입니다. 그 다음에 이어지는 print(" ")는 2단, 3단 등을 구분하기 위해 사용했습니다.

➤ 리스트 컴프리헨션 사용하기

```
...: a = [1,2,3,4]
...: result = []
...: for num in a:
...:     result.append(num * 3)
...: print(result)
...:
[3, 6, 9, 12]
```

예제에서는 a 리스트의 각 항목에 3을 곱한 결과를 result 리스트에 담았습니다.

[표현식 **for** 항목 **in** 반복_가능_객체 **if** 조건문]

리스트 컴프리헨션의 기본 구조입니다.

```
...: a = [1,2,3,4]
...: result = [num * 3 for num in a]
...: print(result)
...:
[3, 6, 9, 12]
```

리스트 컴프리헨션을 사용하면 다음과 같이 좀 더 간단하게 작성할 수 있습니다.

if 조건문' 부분은 생략할 수 있다.

```
...: a = [1,2,3,4]
...: result = [num * 2 for num in a if num % 2 == 0]
...: print(result)
...:
[4, 8]
```

[1, 2, 3, 4] 중에서 짝수에만 2를 곱하여 담고 싶다면 리스트 컴프리헨션 안에 'if 조건문'을 사용하면 된다.

```
...: result = [x*y for x in range(2,10)
...:           for y in range(1,10)]
...: print(result)
...:
[2, 4, 6, 8, 10, 12, 14, 16, 18, 3, 6, 9, 12, 15, 18, 21, 24, ...]
```

구구단의 모든 결과를 리스트에 담고 싶다면 다음과 같이 간단하게 구현할 수도 있습니다.

파이썬의 입출력

■ 함수

입력 값을 가지고 어떤 일을 수행한 후 그 결과물을 내어 놓는 것이 바로 함수가 하는 일입니다

➤ 함수를 사용해야 하는 이유

프로그래밍을 하다 보면 코드를 반복해서 작성하고 있는 자신을 발견할 때가 종종 있습니다. 이때가 바로 함수가 필요한 때입니다. 즉, 반복되는 부분이 있을 경우, '반복적으로 사용되는 가치 있는 부분'을 한 문치로 묶어 '어떤 입력 값을 주었을 때 어떤 결과값을 리턴해 준다'라는 식의 함수로 작성하는 것입니다. 함수를 사용하는 또 다른 이유는 프로그램을 기능 단위의 함수로 분리해 놓으면 프로그램 흐름을 일목요연하게 볼 수 있기 때문입니다. 마치 공장에서 원재료가 여러 공정을 거쳐 하나의 완제품이 되는 것처럼 프로그램에서도 입력한 값이 여러 함수를 거치면서 원하는 결과값을 내는 것을 볼 수 있습니다. 이렇게 되면 프로그램 흐름도 잘 파악할 수 있고 오류가 어디에서 나는지도 쉽게 알아차릴 수 있습니다.

➤ 파이썬 함수의 구조

```
def 함수이름(매개변수):  
    수행할_코드1  
    수행할_코드2  
    ...
```

def는 함수를 만들 때 사용하는 예약어이며, 함수 이름은 함수를 만드는 사람이 임의로 만들 수 있습니다. 매개변수는 함수에 입력으로 전달되는 값을 받는 변수입니다.

```
def add(a, b):  
    return a + b
```

예시의 함수의 이름은 add이고 입력으로 2개의 값을 받고 리턴값(출력값)은 2개의 입력값을 더한 값입니다.

직접 add 함수를 사용해보겠습니다.

변수 a에 3, b에 4를 대입한 후 앞에서 만든 add 함수에 a와 b를 입력값으로 넣어 줍니다. 그리고 변수 c에 add 함수의 리턴값을 대입하면 print(c)로 c의 값을 확인할 수 있습니다.

```
...: def add(a, b):  
...:     return a + b  
...: a = 3  
...: b = 4  
...: c = add(a, b)  
...: print(c)  
...:
```

7

➤ 매개변수와 인수

매개변수(parameter)와 인수(arguments)는 혼용해서 사용하는 용어이므로 잘 기억해야합니다. 매개변수는 함수에 입력으로 전달된 값을 받는 변수, 인수는 함수를 호출할 때 전달하는 입력값을 의미합니다.

a, b는 매개변수 입니다.

3, 4는 인수 입니다.

```
...: def add(a, b):  
...:     return a + b  
...: print(add(3, 4))  
...:
```

7

➤ 입력값과 리턴값에 따른 함수의 형태

함수의 형태는 입력값과 리턴값의 존재 유무에 따라 4가지 유형으로 나뉩니다.

```
def add(a, b):  
    result = a + b  
    return result
```

✓ 입력값과 리턴값이 있는 함수가 일반적인 함수

```
...: a = add(5, 7)  
...: print(a)  
...:  
12
```

다음은 일반적인 함수의 전형적인 예입니다.

```
...: def python():  
...:     return 'good'  
...: a = python()  
...: print(a)  
...:  
good
```

✓ 입력값이 없고 리턴값이 있는 함수

이처럼 입력값이 없고 리턴값만 있는 함수는 변수에
담아서 다음과 같이 사용한다.

```
...: def add(a, b):  
...:     print(f"{a}, {b}의 합은 {a+b}입니다.")  
...: add(3, 4)  
...:  
3, 4의 합은 7입니다.
```

✓ 입력값이 있고 리턴값이 없는 함수

리턴값이 없는 함수는 다음과 같이 사용합니다.

```
...: a = add(3, 4)  
...:  
...: print(a)  
...:  
3, 4의 합은 7입니다.  
None
```

a 값으로 None이 출력되었습니다. add 함수처럼 리
턴값이 없을 때 a 변수에 None을 리턴합니다. None을
리턴한다는 것은 리턴값이 없다는 것입니다

```
...: def say():
...:     print('Hi')
...:
...: say()
...:
Hi
```

➤ 함수의 리턴값은 언제나 하나

```
...: def add_and_mul(a,b):
...:     return a + b, a * b
...: result = add_and_mul(3,4)
...: print(result)
...:
(7, 12)
```

```
...: result1, result2 = add_and_mul(3, 4)
...: print(result1)
...: print(result2)
...:
7
12
```

✓ 입력값과 리턴값이 없는 함수

이 함수를 사용하는 방법은 단 1가지입니다.

add_and_mul은 2개의 입력 인수를 받아 더한 값과 곱한 값을 리턴하는 함수입니다.

리턴값은 두개인데 리턴값을 받아들이는 변수는 result 하나만 쓰였지만 오류는 발생하지 않았습니다. add_and_mul 함수의 리턴값 a+b와 a*b는 튜플값 하나인 (a+b, a*b)로 리턴됐기 때문입니다.

값을 분리하여 받고 싶다면 함수를 다음과 같이 호출하면 됩니다.

```

...: def add_and_mul(a,b):
...:     return a + b
...:     return a * b
...:
...: result = add_and_mul(2, 3)
...: print(result)
...:

```

5

return 문을 2번 사용하면 2개의 리턴값을 돌려 주지 않을까 하고 생각했지만 기대하는 결과는 나오지 않습니다. 두 번째 return 문인 return a * b는 실행되지 않았다는 뜻입니다. 즉, 함수는 return 문을 만나는 순간, 리턴값을 돌려 준 다음 함수를 빠져나가게 됩니다.

➤ 함수 안에서 선언한 변수의 효력 범위

함수 안에서 사용할 변수의 이름을 함수 밖에서도 동일하게 사용한다면 어떻게 될지 알아보겠습니다.

```

...: a = 1
...: def vartest(a):
...:     a = a + 1
...:
...: vartest(a)
...: print(a)
...:

```

1

먼저 a라는 변수를 생성하고 1을 대입했습니다. 그리고 입력으로 들어온 값에 1을 더해 주고 결과값을 리턴하지 않는 vartest 함수를 선언했습니다. 그리고 vartest 함수에 입력값으로 a를 주었다. 마지막으로 a의 값을 print(a)로 출력했다. 과연 어떤 값이 출력될까요?

vartest 함수에서 매개변수 a의 값에 1을 더했으므로 2가 출력될 것 같지만, 프로그램 소스를 작성해서 실행해 보면 결과값은 1이 나옵니다. 그 이유는 함수 안에서 사용하는 매개변수는 함수 안에서만 사용하는 '함수만의 변수'이기 때문이다

매개변수 a는 함수 안에서만 사용하는 변수일 뿐, 함수 밖의 변수 a와는 전혀 상관없다는 뜻입니다.

➤ 함수 안에서 함수 밖의 변수를 변경하는 방법

```
...: a = 1
...: def vartest(a):
...:     a = a + 1
...:     return a
...:
...: a = vartest(a)
...: print(a)
...:
```

2

첫 번째 방법은 return을 사용하는 방법입니다. 1을 더한 값을 리턴하도록 변경했습니다. 따라서 a = vartest(a)라고 작성하면 a에는 vartest 함수의 리턴값이 대입됩니다. 여기에서도 물론 vartest 함수 안의 a 매개 변수는 함수 밖의 a와는 다른 것입니다.

```
...: a = 1
...: def vartest():
...:     global a
...:     a = a + 1
...:
...: vartest()
...: print(a)
...:
```

2

두 번째 방법은 global 명령어를 사용하는 방법입니다. vartest 함수 안의 global a 문장은 함수 안에서 함수 밖의 a 변수를 직접 사용하겠다는 뜻입니다.

하지만 프로그래밍을 할 때 global 명령어는 사용하지 않는 것이 좋습니다. 함수는 독립적으로 존재하는 것이 좋기 때문입니다. 외부 변수에 종속적인 함수는 그다

지 좋은 함수가 아닙니다. 따라서 되도록 global 명령어를 사용하는 이 방법은 피하고 첫 번째 방법을 사용하기를 권합니다.



■ 사용자 입출력

우리들이 사용하는 대부분의 완성된 프로그램은 사용자 입력에 따라 그에 맞는 출력을 내보냅니다. 대표적인 예로 게시판에 글을 작성한 후 [확인] 버튼을 눌러야만(입력) 우리가 작성한 글이 게시판에 올라가는(출력) 것을 들 수 있습니다.

이미 함수 부분에서 입출력이 어떤 의미를 가지는지 알아보았습니다. 지금부터는 좀 더 다양한 입출력 방법에 대해서 알아보겠습니다.

➤ input 사용하기

input은 사용자가 키보드로 입력한 모든 것을 문자열로 저장합니다.

```
...: a = input()
...:
...: print(a)
...:
>? |
```

```
...: a = input()
...:
...: print(a)
...:
```

```
>? python|
```

'python'을 입력하면 변수 a에 문자열로 저장이 됩니다.

```
...: a = input()
...:
...: print(a)
...:
```

```
>? python
python
```

a를 print 해보면 사용자가 입력했던 python을 출력합니다.

➤ 문구를 띄워 사용자 입력 받기

```
...: a = input('숫자를 입력하세요: ')
...:
...: print(a)
...:
```

```
숫자를 입력하세요: >? 1004
1004
```

input()의 괄호 안에 안내 문구를 입력하여 프롬프트를 띄워 주면 됩니다.

```
...: print(type(a))
<class 'str'>
```

input은 입력되는 모든 것을 문자열로 취급하기 때문에 1004 도 숫자가 아닌 문자열로 인식합니다.

➤ print 자세히 알기

지금까지 우리가 사용한 print 문의 용도는 데이터를 출력하는 것이었습니다.

```
...: print("life" "is" "too short")
...: print("life"+"is"+"too short")
...: print("life", "is", "too short")
...:
lifeistoo short
lifeistoo short
life is too short
```

예시에서 첫번째와 두번째는 완전히 동일한 결과값을 출력합니다. 즉, 따옴표로 둘러싸인 문자열을 연속해서 쓰면 + 연산을 한 것과 같습니다.

쉼표(,)를 사용하면 문자열을 띄어 쓸 수 있습니다.

```
...: for i in range(3):
...:     print(i)
...: for i in range(3):
...:     print(i, end=' ')
...:
0
1
2
0 1 2
```

한 줄에 결과값을 계속 이어서 출력하려면 매개변수 end를 사용해 끝 문자를 지정해야 합니다.

end 매개변수의 초기값은 줄 바꿈(\n) 문자입니다.

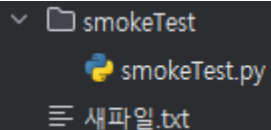
■ 파일 읽고 쓰기

이번에는 파일을 통한 입출력 방법에 대해 알아보겠습니다. 여기에서는 파일을 새로 만든 다음 프로그램이 만든 결과값을 새 파일에 적어 보고 또 파일에 적은 내용을 읽고 새로운 내용을 추가하는 방법도 알아보겠습니다.

➤ 파일 생성하기

```
파일_객체 = open(파일_이름, 파일_열기_모드)
```

```
...: f = open("새파일.txt", 'w')
...: f.close()
...:
```



smokeTest
smokeTest.py
새파일.txt

기본코드의 형태입니다.

파일을 생성하기 위해 파이썬 내장 함수 open을 사용합니다. open 함수는 다음과 같이 '파일 이름'과 '파일 열기 모드'를 입력값으로 받고 결과값으로 파일 객체를 리턴합니다. 마지막에는 항상 close로 닫아줍니다.

새파일이 생성된 것을 확인할 수 있습니다.

파일열기모드	설명
r	읽기 모드: 파일을 읽기만 할 때 사용한다.
w	쓰기 모드: 파일에 내용을 쓸 때 사용한다.
a	추가 모드: 파일의 마지막에 새로운 내용을 추가할 때 사용한다.

파일을 쓰기 모드로 열면 해당 파일이 이미 존재할 경우 원래 있던 내용이 모두 사라지고 해당 파일이 존재하지 않으면 새로운 파일이 생성됩니다. 위 예에서는 디렉터리에 파일이 없는 상태에서 '새파일.txt' 파일을 쓰기 모드인 'w'로 열었기 때문에 '새파일.txt'라는 이름의 새로운 파일이 현재 디렉터리에 생성되었습니다.

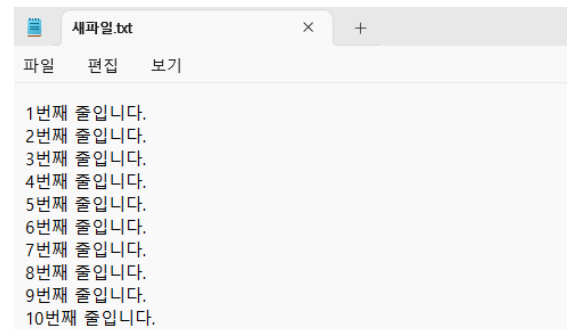
f.close()는 열려 있는 파일 객체를 닫아 주는 역할을 합니다. 사실 이 문장은 생략해도 됩니다. 프로그램을 종료할 때 파이썬 프로그램이 열려 있는 파일의 객체를 자동으로 닫아 주기 때문이다. 하지만 close()를 사용해서 열려 있는 파일을 직접 닫아 주는 것이 좋습니다. 쓰기모드로 열었던 파일을 닫지 않고 다시 사용하려고 하면 오류가 발생하기 때문입니다.

```
f = open("C:/document/새파일.txt", 'w')
f.close()
```

앞에 경로를 지정해주면 지정경로에 생성됩니다.

➤ 파일을 쓰기 모드로 열어 내용 쓰기

```
...: f = open("새파일.txt", 'w')
...: for i in range(1, 11):
...:     data = f"{i}번째 줄입니다.\n"
...:     f.write(data)
...: f.close()
...:
```



반복문을 통해 내용을 작성하고 확인해 보았습니다.

➤ readline으로 파일 읽기

```
...: f = open("새파일.txt", 'r')
...: line = f.readline()
...: print(line)
...: f.close()
...:
1번째 줄입니다.
```

새파일.txt의 가장 첫 번째 줄이 화면에 출력됩니다.

무한루프 반복문을 이용하여 모든 줄을 출력해보는 실습을 해보도록 합시다.

➤ readlines으로 파일 읽기

```
...: f = open("새파일.txt", 'r')
...: lines = f.readlines()
...: print(lines)
...: f.close()
...:
['1번째 줄입니다.\n', '2번째 줄입니다.\n', '3번째 줄입니다.\n', '4번째
```

readlines 함수는 파일의 모든 줄을 읽어서 각각의 줄을 요소로 가지는 리스트를 리턴합니다.

```
...: f = open("새파일.txt", 'r')
...: lines = f.readlines()
...: for line in lines:
...:     print(line)
...: f.close()
...:
1번째 줄입니다.
```

2번째 줄입니다.

3번째 줄입니다.

For 반복문을 통해 출력하면 print의 줄 바꿈(\n)과 요소 안에 줄 바꿈(\n)으로 인해서 줄 바꿈이 두 번 일어나는 부분도 고쳐보겠습니다.

```
...: f = open("새파일.txt", 'r')
...: lines = f.readlines()
...: for line in lines:
...:     line = line.strip()
...:     print(line)
...: f.close()
...:
```

1번째 줄입니다.

2번째 줄입니다.

3번째 줄입니다.

4번째 줄입니다.

파일을 읽을 때 줄 끝의 줄 바꿈(\n) 문자를 제거하고
사용해야 할 경우가 많습니다. 다음처럼 strip 함수를 사
용하면 줄 바꿈 문자를 제거할 수 있습니다.

➤ read 함수로 파일읽기

```
...: f = open("새파일.txt", 'r')
...: data = f.read()
...: print(data)
...: f.close()
...:
```

1번째 줄입니다.

2번째 줄입니다.

3번째 줄입니다.

4번째 줄입니다.

5번째 줄입니다.

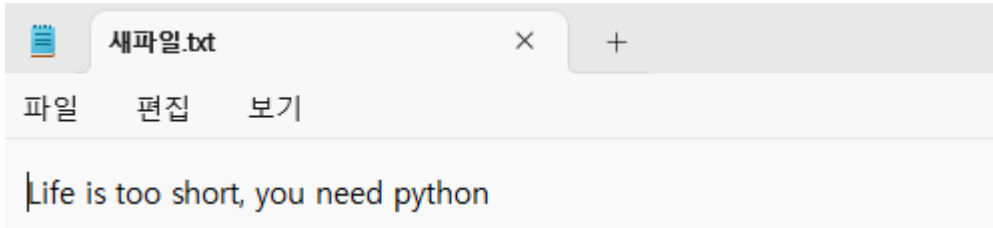
➤ f.read()는 파일의 내용 전체를 문자열로 리턴합니다

➤ with문 사용하기

```
...: with open("새파일.txt", "w") as f:  
...:     f.write("Life is too short, you need python")  
...:
```

파일을 열면(open) 항상 닫아(close) 주어야 합니다. 이렇게 파일을 열고 닫는 것을 자동으로 처리하는 방법은 파이썬의 with 문을 이용하는 것 입니다.

with 문을 사용하면 with 블록(with 문에 속해 있는 문장)을 벗어나는 순간, 열린 파일 객체 f가 자동으로 닫힙니다.



+ 모듈

모듈이란 함수나 변수 또는 클래스를 모아 놓은 파이썬 파일입니다. 모듈은 다른 파이썬 프로그램에서 불러와 사용할 수 있도록 만든 파이썬 파일 이라고도 할 수 있습니다. 우리는 파이썬으로 프로그래밍을 할 때 매우 많은 모듈을 사용합니다. 다른 사람들이 이미 만들어 놓은 모듈을 사용할 수도 있고 우리가 직접 만들어 사용할 수도 있습니다. 여기에서는 모듈을 어떻게 만들고 사용할 수 있는지 알아보겠습니다.

➤ 모듈 만들기

```
mod1.py ×
1 def add(a, b):
2     return a + b
3
4 def sub(a, b):
5     return a-b
```

```
mod1.py
modulTest.py
```

add와 sub 함수만 있는 파일 mod1.py를 만들고 저장을 합니다. 이 mod1.py 파일이 바로 모듈이라고 할 수 있습니다.

모듈을 이용할 파이썬 파일을 같은 디렉토리에 위치시킵니다.

➤ 모듈 불러오기

import 모듈_이름 을 이용하여 불러 옵니다.

import는 이미 만들어 놓은 파이썬 모듈을 사용할 수 있게 해 주는 명령어입니다.

```
modulTest.py ×
1 import mod1
2 print(mod1.add(3, 4))
3 print(mod1.sub(4, 2))
```

mod1.py 파일에 있는 add 함수를 사용하기 위해서는 mod1.add처럼 모듈 이름 뒤에 도트 연산자(.)를 붙이고 함수 이름을 쓰면 된다.

```
C:\Users\JW\anaconda3\python.exe C:\Users\JW\Project\DataScience\smokeTest\modulTest.py
7
2
```

인터프리터를 실행해 보면 아래와 같이 결과가 출력이 됩니다.

➤ 모듈의 함수 이름으로만 사용하기

mod1.add, mod1.sub처럼 쓰지 않고 add, sub처럼 모듈 이름 없이 함수 이름만 쓰고 싶은 경우도 있을 수 있습니다. 이럴 때는 다음과 같이 사용하면 됩니다.

from 모듈_이름 import 모듈_함수

```
modulTest.py ×  
1 from mod1 import add  
2 print(add(3, 4))
```

위와 같이 함수를 직접 import하면 모듈 이름을 붙이지 않고 바로 해당 모듈의 함수를 쓸 수 있습니다.

```
C:\Users\JW\anaconda3\python.exe C:\Users\JW\Project\DataScience\smokeTest\modulTest.py  
7
```

결과도 잘 출력됩니다.

```
from mod1 import add, sub  
  
from mod1 import *
```

선표(,)로 필요한 함수를 불러올 수 있습니다.

* 문자는 '모든 것'이라는 뜻이고 mod1 모듈의 모든 함수를 불러와 사용하겠다는 뜻입니다.

예외처리

프로그램을 만들다 보면 수없이 많은 오류를 만나게 됩니다. 물론 오류가 발생하는 이유는 프로그램이 잘못 동작하는 것을 막기 위한 파이썬의 배려입니다. 이번에는 파이썬에서 오류를 처리하는 방법에 대해서 알아보겠습니다.

■ 다양한 오류

```
f = open("없는파일", 'r')
```

없는 파일을 열려고 시도하면 FileNotFoundError 오류가 발생합니다.

```
FileNotFoundError: [Errno 2] No such file or directory: '없는파일'
```

```
4 / 0
```

0으로 나누려고 하니 ZeroDivisionError 오류가 발생합니다.

```
ZeroDivisionError: division by zero
```

```
a = [1, 2, 3]
```

```
a[3]
```

없는 요소값을 가리키면 IndexError 오류가 발생합니다.

```
IndexError: list index out of range
```

■ try-except 문

```
...: a = [1, 2, 3]
...: try:
...:     a[3]
...: except :
...:     print("오류")
...:
...: a = [1, 2, 3]
...: try:
...:     a[3]
...: except IndexError :
...:     print("오류")
...:
...: a = [1, 2, 3]
...: try:
...:     a[3]
...: except IndexError as e :
...:     print(e)
...:
오류
오류
list index out of range
```

➤ try-except만 쓰는 방법

오류의 종류에 상관없이 오류가 발생하면 except 블록을 수행합니다.

➤ 발생 오류만 포함한 except 문

오류가 발생했을 때 except 문에 미리 정해 놓은 오류와 동일한 오류일 경우에만 except 블록을 수행한다는 뜻입니다.

➤ 발생 오류와 오류 변수까지 포함한 except 문

두 번째 경우에서 오류의 내용까지 알고 싶을 때 사용하는 방법입니다.

내장함수

우리는 이미 몇 가지 내장 함수를 배웠습니다. `print`, `del`, `type` 등이 바로 그것입니다. 이러한 파이썬 내장 함수는 파이썬 모듈과 달리 `import`가 필요하지 않기 때문에 아무런 설정 없이 바로 사용할 수 있습니다.

활용 빈도가 높고 중요한 함수 위주로 먼저 살펴보도록 하겠습니다.

➤ `abs`

```
...: print(abs(3))
...: print(abs(-3))
...: print(abs(-3.14))
...:
3
3
3.14
```

`abs(x)`는 어떤 숫자를 입력 받았을 때 그 숫자의 절댓값을 리턴하는 함수입니다.

➤ `all`

`all(x)`는 `x`의 요소가 모두 참이면 `True`, 거짓이 하나라도 있으면 `False`를 리턴합니다.

```
...: print(all([1, 2, 3]))
...: print(all([1, 2, 3, 0]))
...: print(all([]))
...:
True
False
True
```

리스트 `[1, 2, 3]`은 모든 요소가 참이므로 `True`

리스트 `[1, 2, 3, 0]` 중에서 요소 `0`은 거짓이므로 `False`

`all`의 입력 인수가 빈 값인 경우에는 `True`

➤ divmod

```
...: print(divmod(7, 3))
...: print(divmod(8, 2))
...:
(2, 1)
(4, 0)
```

divmod(a, b)는 2개의 숫자 a, b를 입력으로 받는다. 그리고 a를 b로 나눈 몫과 나머지를 튜플로 리턴합니다.

➤ filter

```
...: def positive(x):
...:     return x > 0
...: print(list(filter(positive, [1, -3, 2, 0, -5, 6])))
...:
[1, 2, 6]
```

filter 함수는 첫 번째 인수로 함수, 두 번째 인수로 그 함수에 차례로 들어갈 데이터를 받습니다. 요소가 순서대로 함수에 들어가 리턴값이 참인 것만 리턴합니다.

list 함수는 filter 함수의 리턴값을 리스트로 출력하기 위해 사용했습니다.

➤ int

int(x)는 문자열 형태의 숫자나 소수점이 있는 숫자를 정수로 리턴하는 함수입니다.

```
...: print(int('3'))
...: print(int(3.14))
...:
3
3
```

➤ len

```
...: print(len("python"))
...: print(len([1,2,3]))
...: print(len((1, 'a')))
...:
6
3
2
```

len(s)는 입력값 s의 길이(요소의 전체 개수)를 리턴하는 함수입니다.

➤ list

```
...: print(list("python"))
...: print(list([1,2,3]))
...:
['p', 'y', 't', 'h', 'o', 'n']
[1, 2, 3]
```

list(iterable)은 반복 가능한 데이터를 입력 받아 리스트로 만들어 리턴하는 함수입니다

➤ map

```
...: def two_times(x):
...:     return x * 2
...: print(list(map(two_times, [1, -3, 2, 0, -5, 6])))
...:
[2, -6, 4, 0, -10, 12]
```

map(f, iterable)은 함수(f)와 반복 가능한 데이터를 입력으로 받습니다. map은 입력 받은 데이터의 각 요소에 함수 f를 적용한 결과를 리턴하는 함수입니다.

map 함수의 결과를 리스트로 출력하기 위해 list 함수를 사용합니다. map 함수는 map 객체를 리턴하기 때문입니다.

➤ max

```
...: print(max([1, 2, 3]))
...: print(max("python"))
...:
3
y
```

max(iterable)은 인수로 반복 가능한 데이터를 입력 받아 그 최댓값을 리턴하는 함수입니다.

➤ min

```
...: print(min([1, 2, 3]))
...: print(min("python"))
...:
1
h
```

min(iterable)은 max 함수와 반대로, 인수로 반복 가능한 데이터를 입력 받아 그 최솟값을 리턴하는 함수입니다.

➤ range

```
...: print(list(range(1, 10, 2)))
...: print(list(range(0, -10, -1)))
...:
[1, 3, 5, 7, 9]
[0, -1, -2, -3, -4, -5, -6, -7, -8, -9]
```

리를 말합니다.

인수가 한 개인 경우와 두 개인 경우는 공부를 했었습니다.

인수가 세 개인 경우 세 번째 인수는 숫자 사이의 거

➤ round

```
...: print(round(4.6))
...: print(round(5.678, 2))
...:
5
5.68
```

round 함수는 숫자를 입력 받아 반올림해 리턴하는 함수입니다. 두 번째 인수는 반올림하여 표시하고 싶은 소수점의 자릿수를 의미합니다.

➤ sorted

```
...: print(sorted([3, 1, 2]))
...: print(sorted(['a', 'c', 'b']))
...: print(sorted("zero"))
...:
[1, 2, 3]
['a', 'b', 'c']
['e', 'o', 'r', 'z']
```

sorted(iterable)는 입력 데이터를 정렬한 후 그 결과를 리스트로 리턴하는 함수입니다.

리스트 자료형에도 sort 함수가 있습니다. 하지만 리스트 자료형의 sort 함수는 리스트 객체 그 자체를 정렬만

할 뿐, 정렬된 결과를 리턴하지는 않는다.

```
...: a = [1, 4, 3, 2]
...: a.sort()
...: print(a)
...: b = [1, 4, 3, 2]
...: print(sorted(b))
...: print(b)
...:
[1, 2, 3, 4]
[1, 2, 3, 4]
[1, 4, 3, 2]
```

쉽게 이야기하면 sort() 함수를 사용하면 원본 리스트가 정렬됩니다.

sorted() 함수는 원본 리스트를 변경하지 않고 새로운 정렬된 리스트를 리턴합니다.

파이썬 표준 라이브러리

세계의 파이썬 고수들이 만든 유용한 프로그램을 모아 놓은 것이 바로 **파이썬 표준 라이브러리**입니다. '라이브러리'는 '도서관'이라는 뜻 그대로 원하는 정보를 찾아보는 곳입니다. 모든 라이브러리를 다 알 필요는 없고 어떤 일을 할 때 어떤 라이브러리를 사용해야 한다는 정도만 알면 됩니다. 이를 위해 어떤 라이브러리가 존재하고 어떻게 사용하는지 알아야 합니다. 자주 사용되고 꼭 알아 두면 좋은 라이브러리를 중심으로 하나씩 살펴보겠습니다.

➤ datetime.date

```
...: import datetime
...: day1 = datetime.date(2024, 6, 27)
...: day2 = datetime.date(2024, 8, 31)
...:
...: diff = day2 - day1
...: print(diff.days)
...:
```

65

datetime.date는 연, 월, 일로 날짜를 표현할 때 사용하는 함수입니다.

연, 월, 일을 인수로 받는 날짜객체를 다음과 같이 뺄셈으로 두 날짜 사이의 일수를 쉽게 구할 수 있습니다

➤ time

시간과 관련된 time 모듈에는 함수가 매우 많습니다

```
...: import time
...: print((time.time()))
...: print(time.localtime(time.time()))
...: print( time.asctime(time.localtime(time.time())) )
...: print(time.ctime())
...:
1717950496.970544
time.struct_time(tm_year=2024, tm_mon=6, tm_mday=10, tm_hour=1, tm_min=28, tm_sec=16, tm_wday=0, tm_yday=162, tm_isdst=0)
Mon Jun 10 01:28:16 2024
Mon Jun 10 01:28:16 2024
```

time.time()은 UTC(협정 세계 표준시)를 사용하여 현재 시간을 실수 형태로 리턴하는 함수입니다.

time.localtime은 time.time()이 리턴한 실수값을 사용해서 연, 월, 일, 시, 분, 초, ... 의 형태로 바꾸어 주는 함수입니다.

time.asctime은 time.localtime가 리턴된 튜플 형태의 값을 인수로 받아서 날짜와 시간을 알아보기 쉬운 형태로 리턴하는 함수입니다.

time.ctime()은 time.asctime(time.localtime(time.time()))을 간단하게 표시하는 함수입니다.

➤ random

random은 난수(임의의 수)를 발생시키는 모듈입니다. random과 randint 함수에 대해 알아보겠습니다.

```
...: import random
...: print(random.random())
...: print(random.randint(1, 10))
...: print(random.randint(1, 55))
...:
0.9589291144692151
6
1
In [23]:
...: import random
...: print(random.random())
...: print(random.randint(1, 10))
...: print(random.randint(1, 55))
...:
0.7185352225953864
4
17
```

random.random() 은 0.0에서 1.0 사이의 실수 중에서 난수 값을 리턴하는

random.randint 함수는 파이썬의 내장 모듈인 random 에서 제공하는 함수로, 지정된 범위 내에서 정수 하나를 무작위로 반환합니다. 이 함수는 두 개의 인자를 받으며, 이 두 인자 사이의 모든 정수 중에서 무작위로 하나를 선택하여 반환합니다. 양 끝 값도 포함됩니다.

➤ pickle

pickle은 객체의 형태를 그대로 유지하면서 파일에 저장하고 불러올 수 있게 하는 모듈입니다

pickle.dump로 저장한 파일을 pickle.load를 사용해서 딕셔너리 객체(data) 상태 그대로 불러옵니다.

➤ OS

os 모듈은 환경 변수나 디렉터리, 파일 등의 OS 자원을 제어할 수 있게 해 주는 모듈입니다.

함수	설명
os.mkdir(디렉터리)	디렉터리를 생성한다.
os.rmdir(디렉터리)	디렉터리를 삭제한다. 단, 디렉터리가 비어 있어야 삭제할 수 있다.
os.remove(파일)	파일을 지운다.
os.rename(src, dst)	src라는 이름의 파일을 dst라는 이름으로 바꾼다.

➤ json

JSON 파일을 읽을 때는 이 예처럼 json.load(파일_객체)를 사용합니다. 이렇게 load() 함수는 읽은 데이터를 딕셔너리 자료형으로 리턴합니다. 이와 반대로 딕셔너리 자료형을 JSON 파일로 생성할 때는 다음처럼 json.dump(딕셔너리, 파일_객체)를 사용합니다.

※ 소스코드 참조: 점프 투 파이썬

<https://wikidocs.net/book/1>