

2021년도

ICT 이노베이션스퀘어 확산 사업

[인공지능 : 기초부터 실전까지]

▶ 1차수~6차수 강의 교안

※ 본 교안은 강의 수강 용도로만 사용 가능합니다.
상업적 이용을 일절 금함.



1

파이썬 프로그래밍 언어

왜 프로그래밍 언어를 배워야 할까?

❖ 다양한 응용 프로그램(애플리케이션 혹은 앱)

- 스마트폰: 전화, 문자, 사진 촬영 등
- 컴퓨터: 오피스 프로그램, 웹 브라우저 등

❖ 응용 프로그램은 프로그래밍 언어를 이용해 제작

❖ 프로그래밍: 컴퓨팅 기기(컴퓨터나 스마트폰 등)가 어떤 일을 하도록 명령을 내리는 작업

❖ 프로그래밍 언어: 프로그래밍을 하기 위한 언어

❖ 코딩(Coding): 프로그래밍 언어를 활용해 특정 목적의 프로그램을 만드는 것

프로그래밍 언어란?

❖ 컴퓨터에 명령을 내리려고 만든 언어

❖ 초기 프로그래밍

- 컴퓨터의 중앙처리장치(CPU) 같은 하드웨어에 전기 신호를 직접 주기 위한 0과 1로 이뤄진 명령의 나열
- 기계어 (Machine language): 2진 숫자(0과 1)로만 이뤄진 명령
- 어셈블리어(Assembly language)

고급어와 저급어

❖ 고급어(High-level language)

- 하드웨어에 대한 지식이 없는 사람이 좀 더 잘 이해할 수 있고 작성할 수 있는 프로그래밍 언어
- 베이직(BASIC), 포트란(FORTRAN), C, C++, C#, 자바(Java), 파이썬(Python), 루비(Ruby), 펄(Perl), 루아(Lua), R 등
- 고급어로 작성된 코드도 바로 실행될 수 없음: 컴퓨터가 해석할 수 있는 기계어로 바꿔야 함

❖ 저급어(Low-level language): 어셈블리어

컴파일드 언어와 인터프리티드 언어

❖ 컴파일드 언어(Compiled Language)

- 다수의 명령어로 이뤄진 소스코드를 한 번에 기계어로 번역해서 실행 파일을 만듦

❖ 인터프리티드 언어(Interpreted Language)

- 소스코드를 한 줄씩 기계어로 번역해서 실행 결과를 보여줌
- 스크립트(Script)언어라고도 함

왜 파이썬인가?

- ❖ 1991년 귀도 반 로섬(Guido van Rossum)이 발표
- ❖ 인터프리티드 언어
- ❖ 특징
 - 배우기가 쉬움
 - 무료
 - 방대한 라이브러리
 - 어느 운영체제에서도 사용 가능
- ❖ 파이썬을 사용하는 곳
 - <https://wiki.python.org/moin/OrganizationsUsingPython>



2

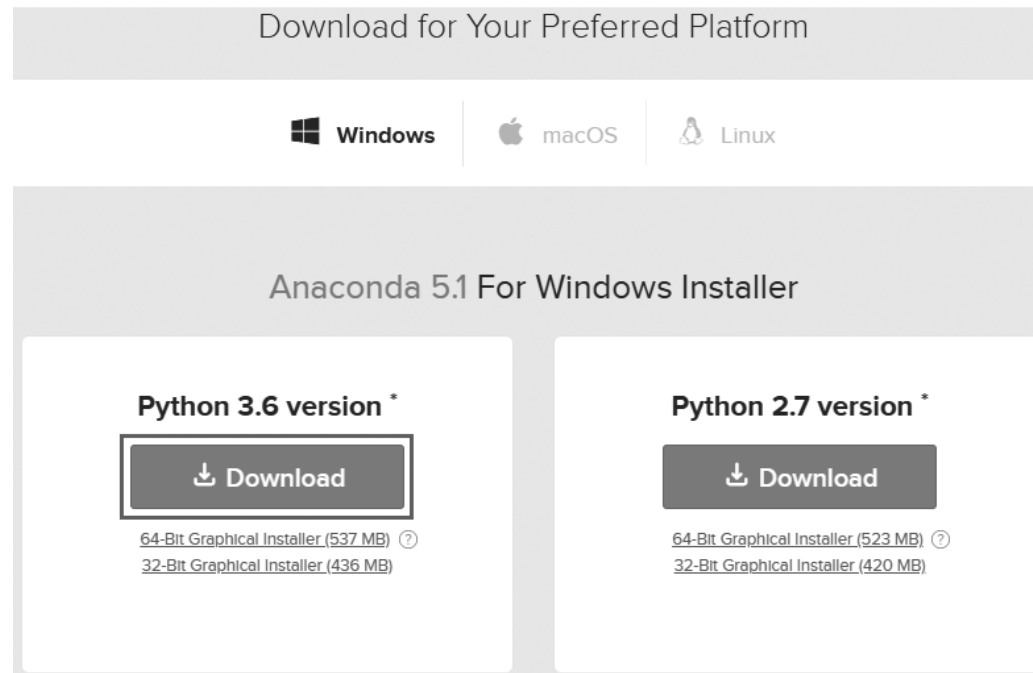
파이썬 프로그래밍 언어

파이썬 개발 환경 설치

❖ 파이썬 공식 배포판: <https://www.python.org>

❖ 아나콘다 배포판

- 파이썬 기본 프로그램과 함께 많이 사용하는 패키지와 통합 개발 환경을 한번에 설치
- 다운로드 페이지: <http://continuum.io/downloads>



파이썬 2.x와 3.x의 차이점

❖ 현재 파이썬 2.x와 3.x 모두 많이 사용됨

❖ 파이썬 3.x

- 파이썬 2.x를 개선한 차기 버전
- 구조와 구문이 일부 변경돼 서로 완벽하게 호환되지 않음

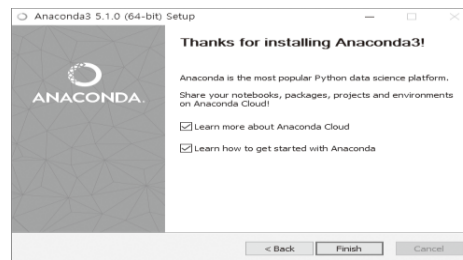
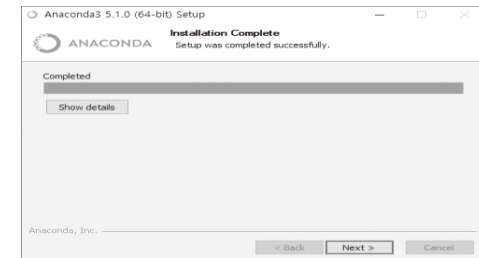
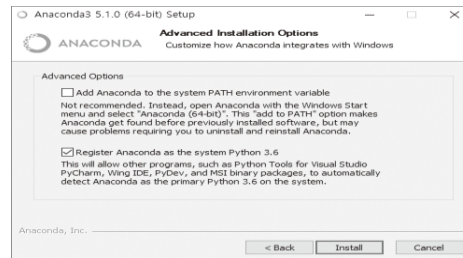
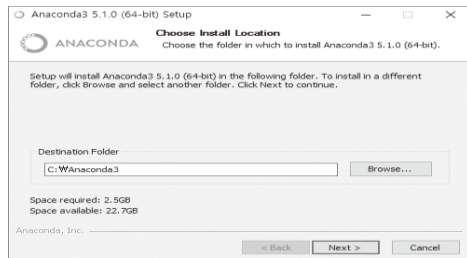
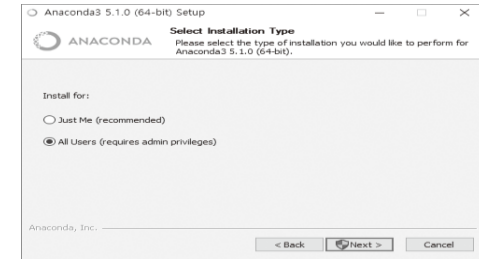
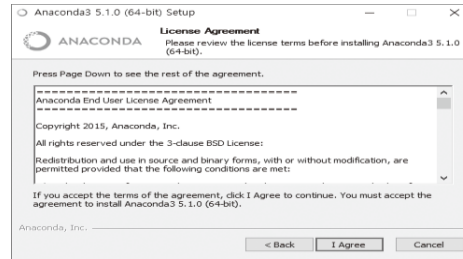
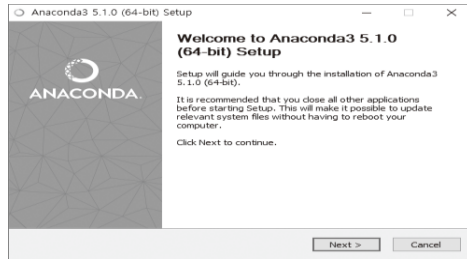
❖ 새로운 버전이 나왔음에도 파이썬 2.x가 많이 이용되는 이유

- 파이썬 2.x를 이용해 작성된 코드가 아직도 존재
- 파이썬 라이브러리 가운데 파이썬 2.x만 지원하는 경우가 있음

❖ 파이썬 3.x에서 바뀐 점

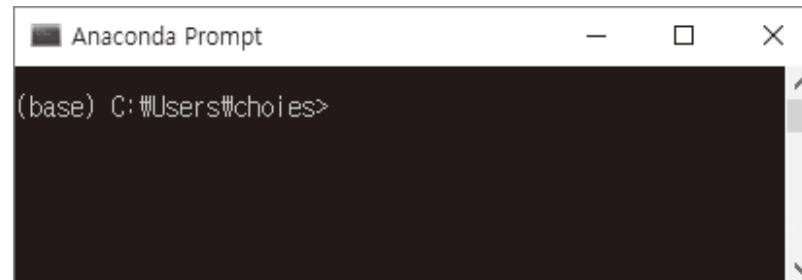
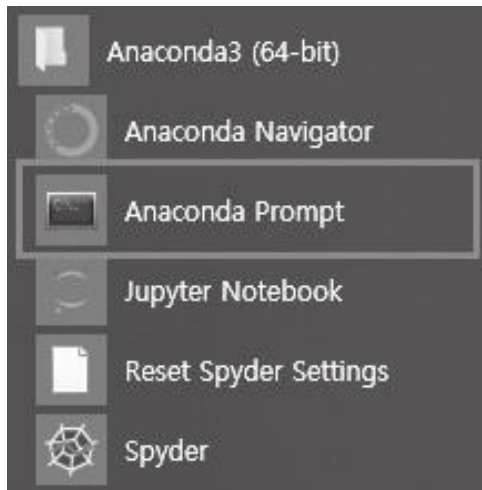
- print가 함수 형태로 사용되어 `print('Python')`과 같이 작성
 - 파이썬 2.x에서는 `print 'Python'`
- 유니코드(Unicode) 지원

아나콘다 설치



첫 번째 코드 작성

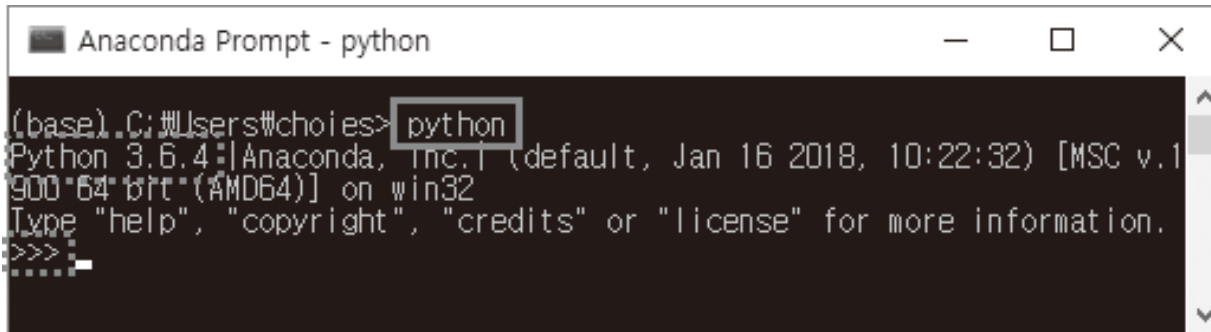
❖ 시작 버튼 클릭 후 [Anaconda Prompt]를 선택



첫 번째 코드 작성

❖ 실행된 명령 프롬프트에 'python' 이라고 입력

- 파이썬 코드를 입력할 수 있는 파이썬 콘솔 프로그램이 실행
- 파이썬 콘솔은 파이썬 개발 환경 중 가장 기본이 되는 개발 환경

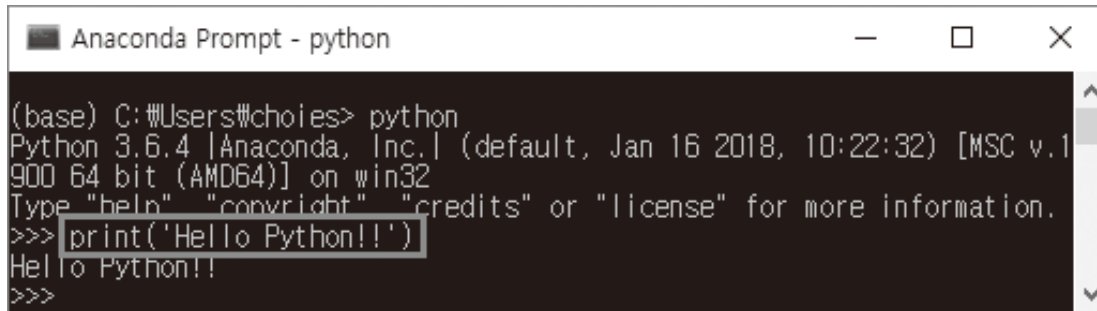


```
Anaconda Prompt - python
(base) C:\Users\choies> python
Python 3.6.4: [Anaconda, Inc.] (default, Jan 16 2018, 10:22:32) [MSC v.1900 64-bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

- ❖ 'Python 3.6.4' 는 현재 설치된 파이썬의 버전이 '3.6.4' 임을 의미
- ❖ '>>>' 는 파이썬 인터프리터 프롬프트(혹은 파이썬 프롬프트)

첫 번째 코드 작성

- ❖ `print('Hello Python!!')`을 입력하고 Enter 키를 누르면 `'Hello Python!!'`을 출력



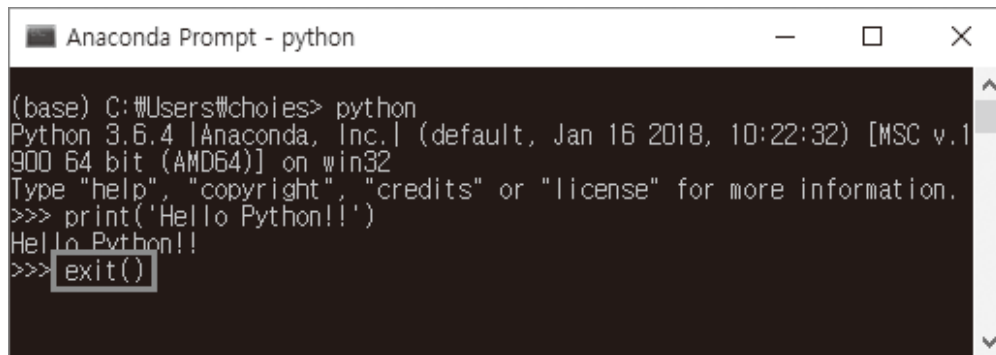
```
Anaconda Prompt - python
(base) C:\Users\choies> python
Python 3.6.4 |Anaconda, Inc.| (default, Jan 16 2018, 10:22:32) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print('Hello Python!!')
Hello Python!!
>>>
```

- ❖ 코드를 한 줄 입력한 후에는 마지막으로 Enter 키를 반드시 눌러야 함
- ❖ `print()` 함수는 괄호 안의 내용을 출력

첫 번째 코드 작성

❖ 파이썬 콘솔 프로그램을 종료

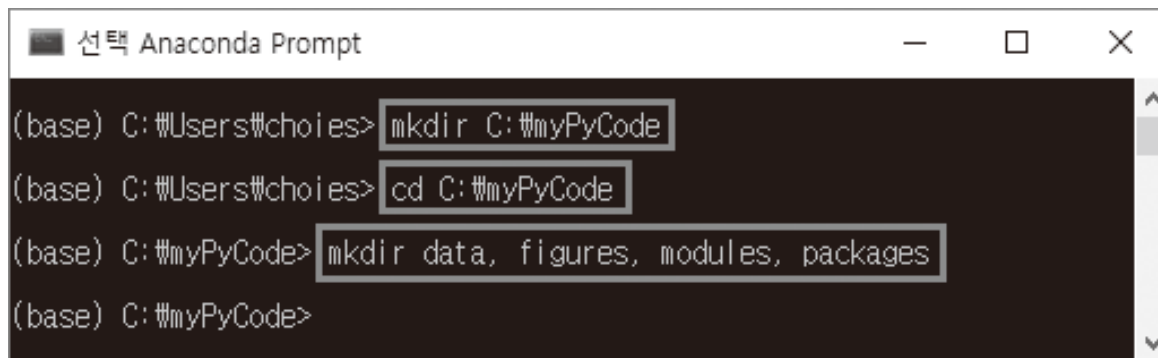
- 프롬프트에 `exit()`를 입력하고 Enter 키를 누르거나
- Ctrl + Z(키보드의 Ctrl 키를 누른 상태에서 Z 키를 누름)를 입력하고 Enter 키를 누름



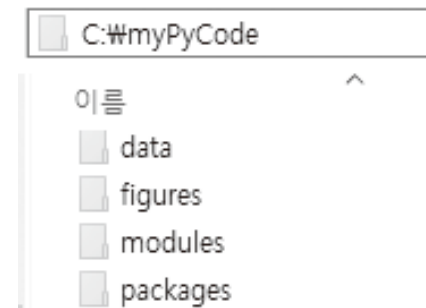
```
Anaconda Prompt - python
(base) C:\Users\choies> python
Python 3.6.4 |Anaconda, Inc.| (default, Jan 16 2018, 10:22:32) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print('Hello Python!!')
Hello Python!!
>>> exit()
```

코드 저장

- ❖ 작성한 파이썬 코드를 나중에 다시 사용하려면 텍스트 편집 프로그램(즉, 텍스트 편집기)을 이용해 컴퓨터에 저장해야 함
- ❖ 파이썬 코드는 일반 텍스트 파일이므로 일반 텍스트 편집기를 이용해 코드를 작성하고 저장
- ❖ 진행을 위해 작업 폴더를 만들고 코드를 저장
 - 'C:\myPyCode' 폴더
 - 'data', 'figures', 'modules', 'packages' 폴더도 생성

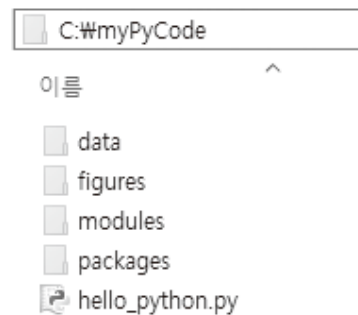
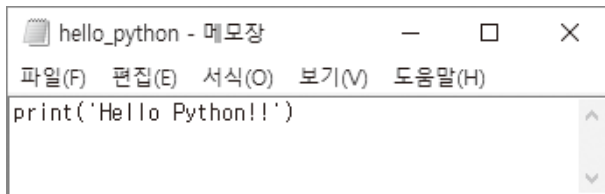


```
선택 Anaconda Prompt
(base) C:\Users\choies> mkdir C:\myPyCode
(base) C:\Users\choies> cd C:\myPyCode
(base) C:\myPyCode> mkdir data, figures, modules, packages
(base) C:\myPyCode>
```



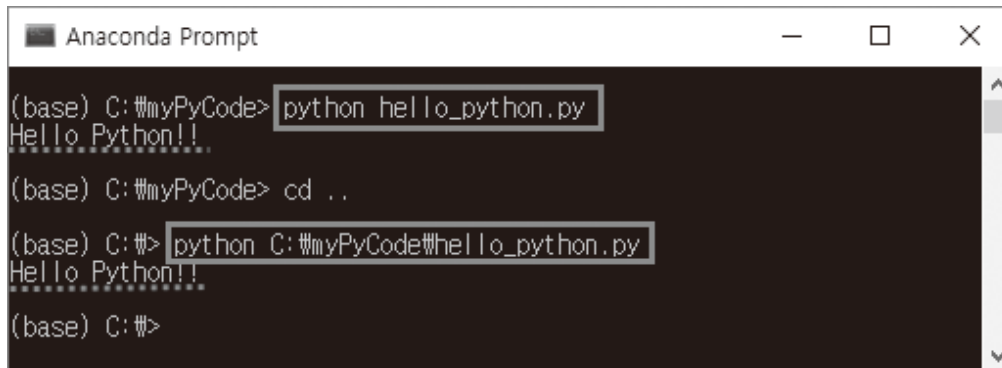
코드 저장

- ❖ 윈도우 메모장에서 `print('Hello Python!!')`를 입력하고 파일 이름을 `'C:\myPyCode'` 폴더에 `hello_python.py`로 저장



코드 실행

- ❖ 파이썬 코드 파일(확장자가 .py인 파일)은 명령 프롬프트에서 'python 파일명.py'를 입력해서 실행
- ❖ 앞에서 저장한 파이썬 코드는 'python hello_python.py'를 입력
- ❖ 명령 프롬프트의 위치가 파이썬 코드 파일이 있는 폴더가 아니라면 'python C:\myPyCode\hello_python.py'처럼 파일명 앞에 파일의 경로를 지정



```
Anaconda Prompt
(base) C:\myPyCode> python hello_python.py
Hello Python!!

(base) C:\myPyCode> cd ..

(base) C:\> python C:\myPyCode\hello_python.py
Hello Python!!

(base) C:\>
```

통합 개발 환경에서 코딩하기

❖ 통합 개발 환경

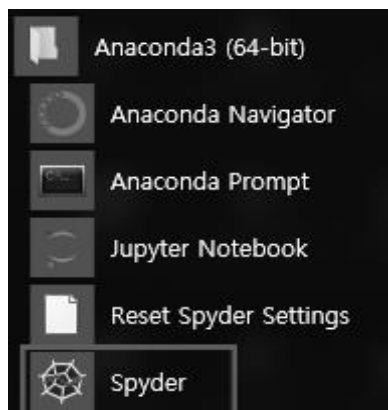
- 텍스트 편집기와 파이썬 개발 환경이 하나의 프로그램에서 동작

❖ Spyder 통합 개발 환경: 아나콘다 배포판에 포함돼 있음

- IPython 콘솔(Console)과 내장 편집기(Editor)가 통합

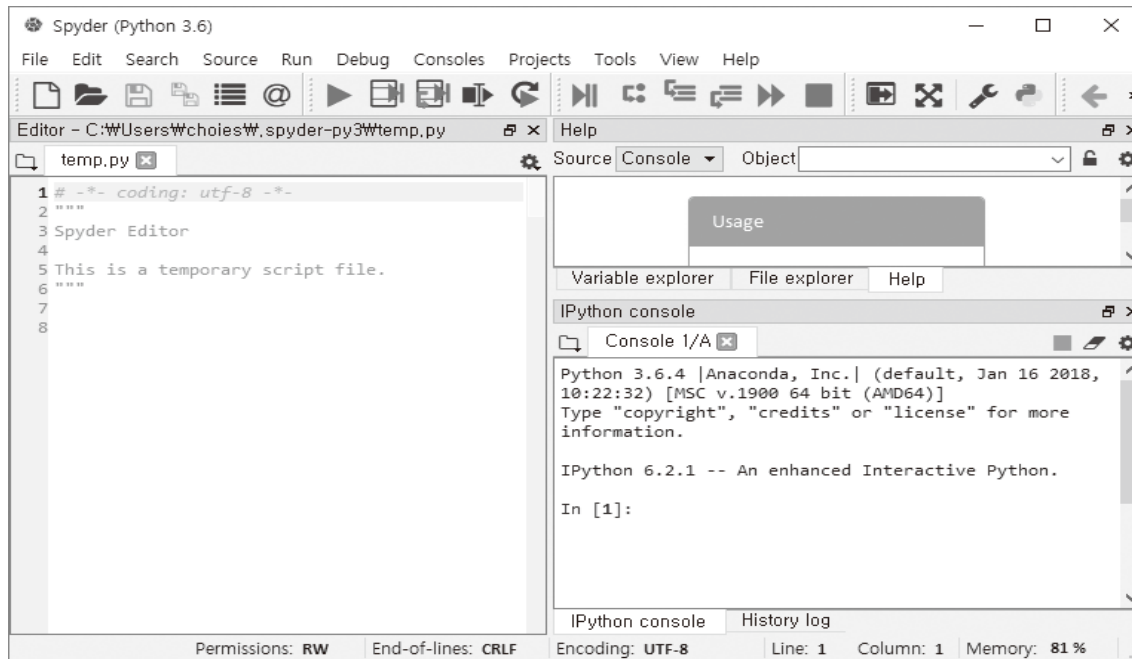
❖ Spyder 실행 및 설정

- 아나콘다 메뉴에서 [Spyder]를 클릭



통합 개발 환경에서 코딩하기

❖ Spyder 통합 개발 환경

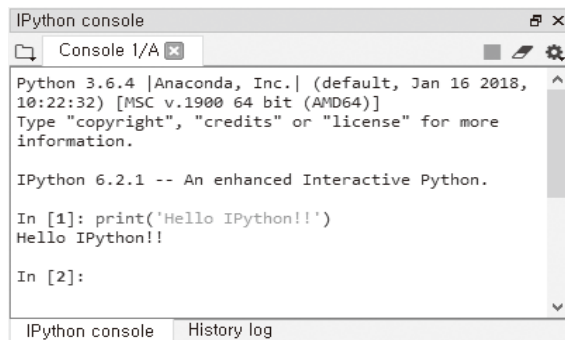


❖ IPython 콘솔의 프롬프트는 'In [1]:'

- 대괄호 안의 숫자는 코드를 입력할 때마다 1씩 증가

통합 개발 환경에서 코딩하기

- ❖ 'In [1]:' 다음에 `print('Hello IPython!!')`를 입력해서 실행한 결과



```
Python 3.6.4 |Anaconda, Inc.| (default, Jan 16 2018, 10:22:32) [MSC v.1900 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 6.2.1 -- An enhanced Interactive Python.

In [1]: print('Hello IPython!!')
Hello IPython!!

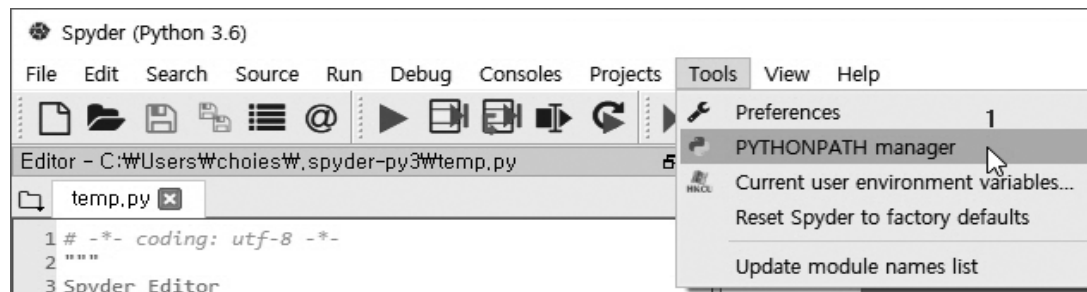
In [2]:
```

- ❖ IPython에서 `print`를 입력하면 글자색이 자동으로 바뀜
 - IPython이 파이썬 내장 명령어를 인식해서 글자색을 변경

통합 개발 환경에서 코딩하기

❖ Spyder의 PYTHONPATH Manager를 이용해 'C: myPyCode' 내의 modules과 packages 폴더를 PYTHONPATH 환경 변수에 설정

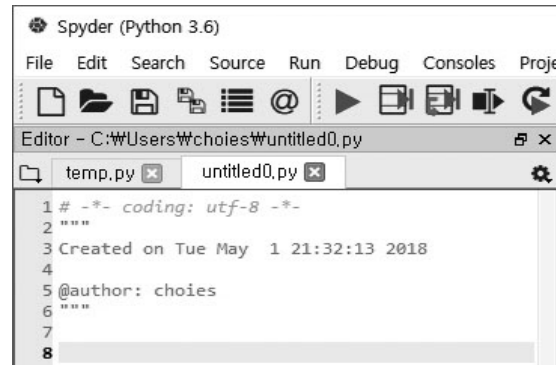
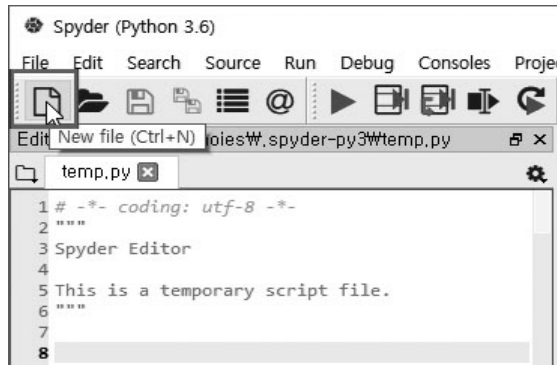
- Spyder 메뉴 중 [Tools] 클릭 후 [PYTHONPATH Manager] 클릭



- [Add path] 버튼을 클릭
- 'Select Directory' 팝업 창이 나오면 'C: myPyCode modules' 폴더와 'C: myPyCode packages' 폴더 선택
- [Synchronize ...] 버튼을 클릭한 후 [Yes] 버튼 클릭
- [Close] 버튼 클릭
- Spyder 종료 후 다시 시작

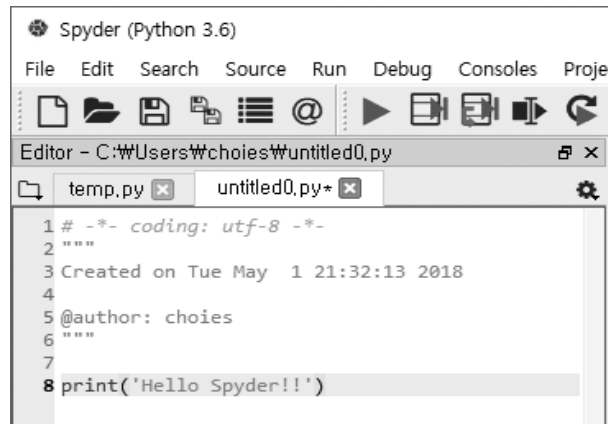
Spyder 에디터에서 코드 작성하기

- ❖ Spyder에서 키보드로 Ctrl + N을 누르거나 마우스로 상단의 New file 아이콘을 클릭
- ❖ 새 파일이 열리고 문자 인코딩('utf-8') 형식, 파일 생성 날짜, 파일을 생성한 현재 사용자 정보가 자동으로 앞에 삽입됨



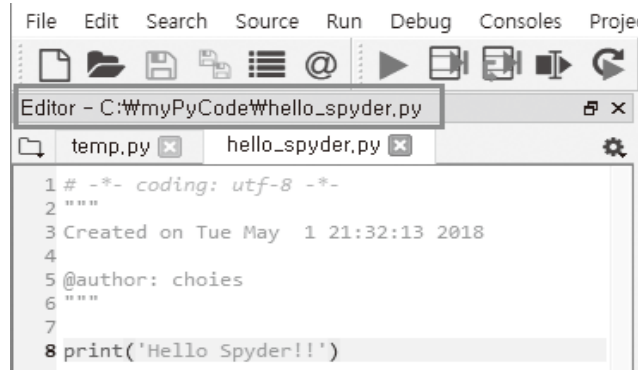
Spyder 에디터에서 코드 작성하기

❖ 새로 생성한 파이썬 파일에 `print('Hello Spyder!!')` 입력



```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Tue May 1 21:32:13 2018
4
5 @author: choies
6 """
7
8 print('Hello Spyder!!')
```

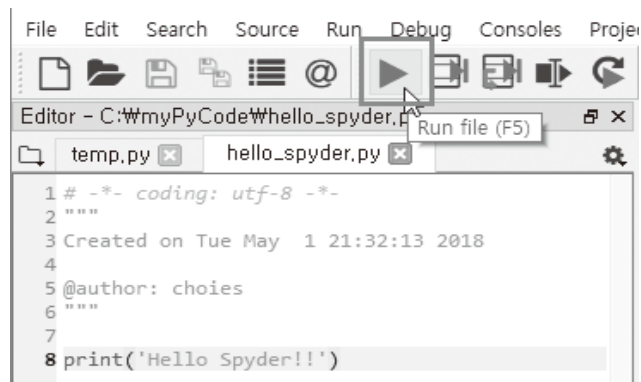
❖ 키보드로 `Ctrl + S`를 입력하거나 마우스로 [Save file] 아이콘을 클릭한 후 작업 폴더인 `'C:\myPyCode'` 폴더에 파일 이름을 `hello_spyder.py`로 저장



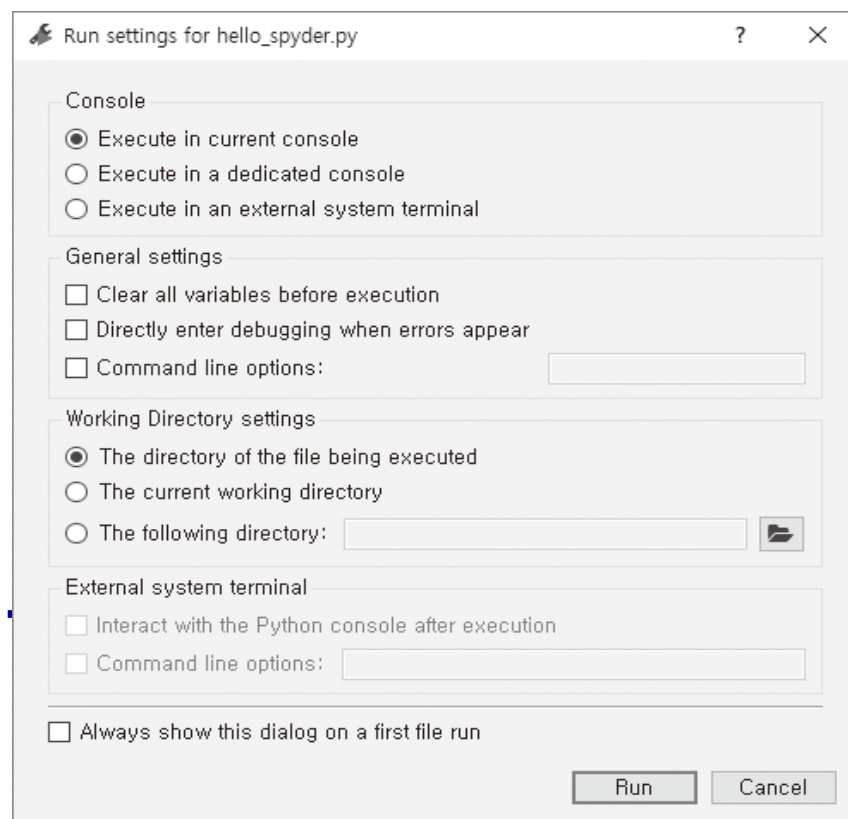
```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Tue May 1 21:32:13 2018
4
5 @author: choies
6 """
7
8 print('Hello Spyder!!')
```


Spyder 에디터에서 코드 작성하기

❖ F5 키를 누르거나 [Run file(F5)] 아이콘을 클릭

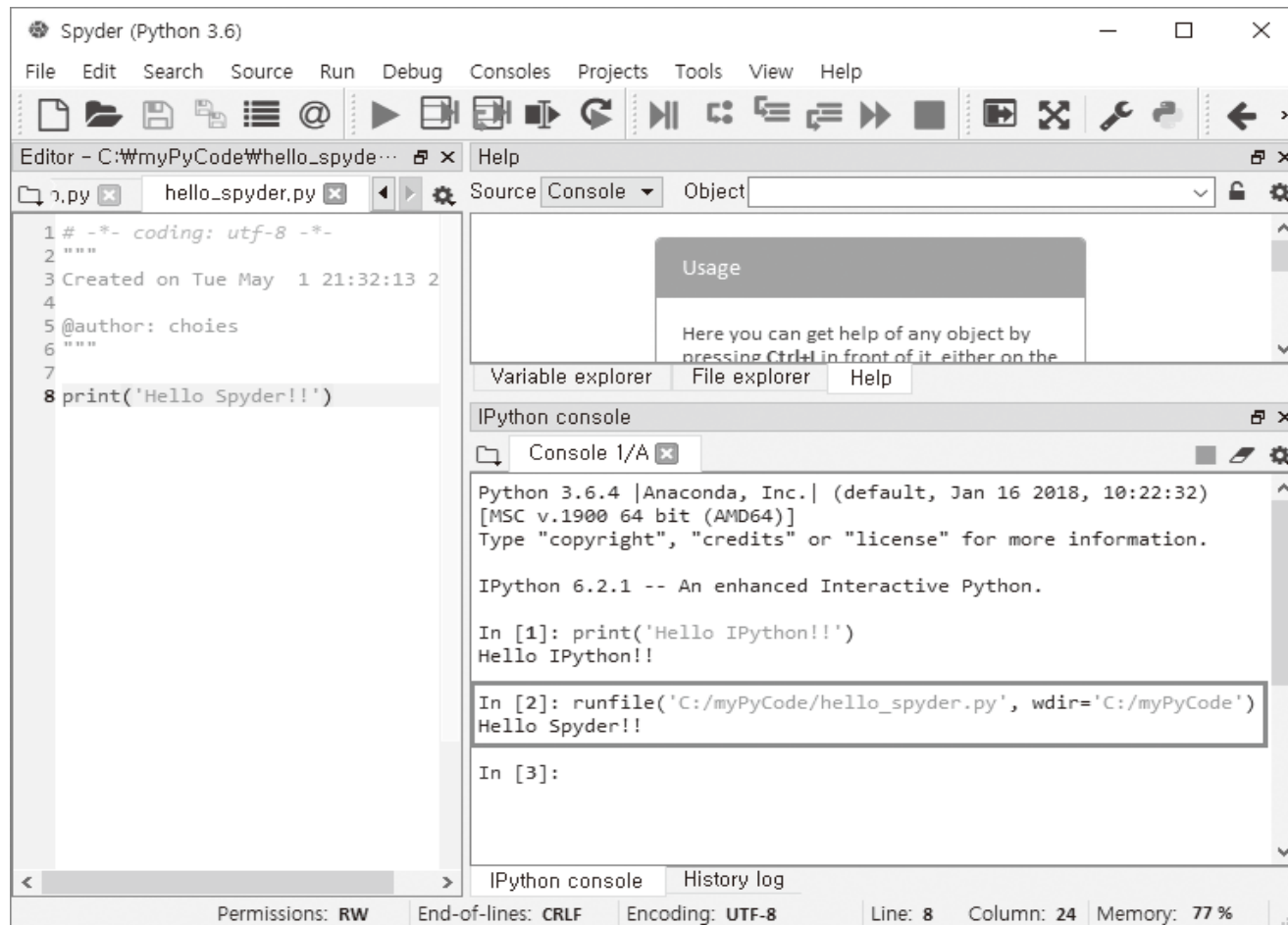


❖ Spyder에서 저장된 코드를 처음 실행하는 경우 나타나 설정 창에서 하단의 [Run] 버튼을 클릭



Spyder 에디터에서 코드 작성하기

- ❖ 모든 것이 정상적으로 수행되면 IPython 콘솔에서 'hello_spyder.py' 파일을 실행하고 실행 결과가 표시됨



주피터 노트북에서 코딩하기

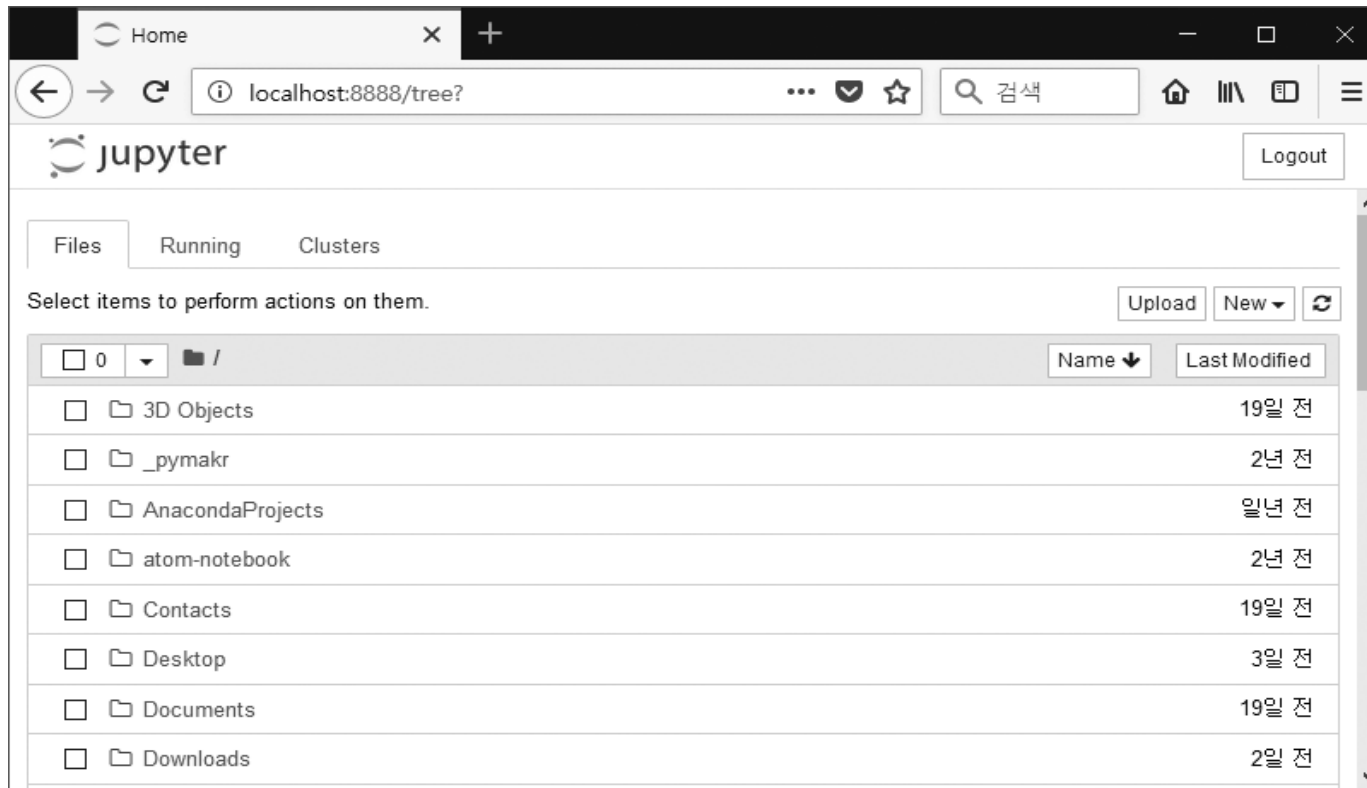
❖ 주피터 노트북(Jupyter Notebook)

- 코드 작성 및 실행뿐만 아니라 코드 설명을 위한 문서 작성을 편리하게 할 수 있는 웹 응용 프로그램

❖ 주피터 노트북 실행

- 아나콘다 메뉴에서 [Jupyter Notebook]을 클릭해서 실행
- 주피터 서버가 시작되고 기본 브라우저의 새 창에서 주피터 노트북이 열림
- Home이 주피터 노트북의 시작점

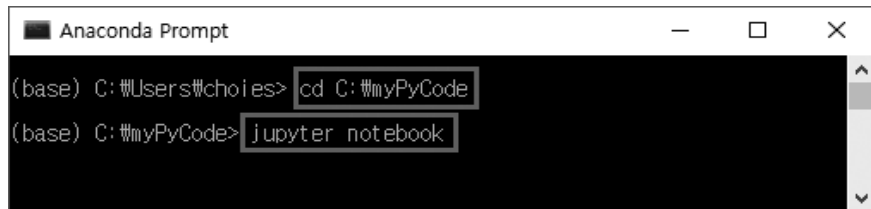
주피터 노트북에서 코딩하기



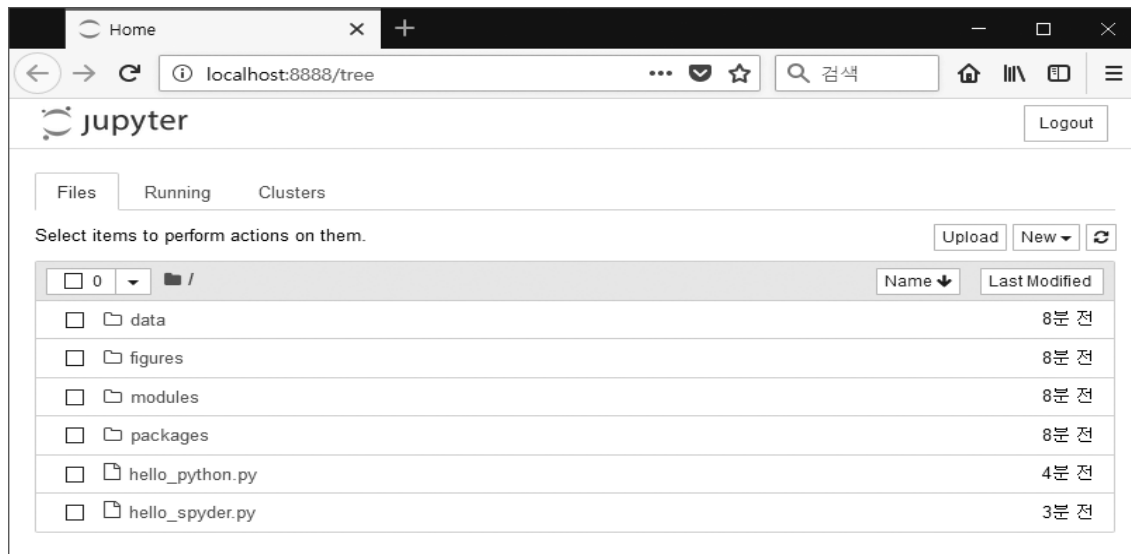
- ❖ 주피터 노트북의 시작 폴더는 'C: Users 사용자명'
- ❖ 주피터 노트북에서 새로운 노트북을 생성하면 이 폴더나 그 아래의 폴더에만 저장 가능

주피터 노트북에서 코딩하기

- ❖ 아나콘다 메뉴에서 [Anaconda Prompt]를 클릭
- ❖ 다음과 같이 주피터 노트북을 실행
 - 프롬프트에 'cd C: myPyCode'를 입력해 작업 폴더로 이동
 - 프롬프트에 'jupyter notebook'을 입력해 주피터 노트북 실행

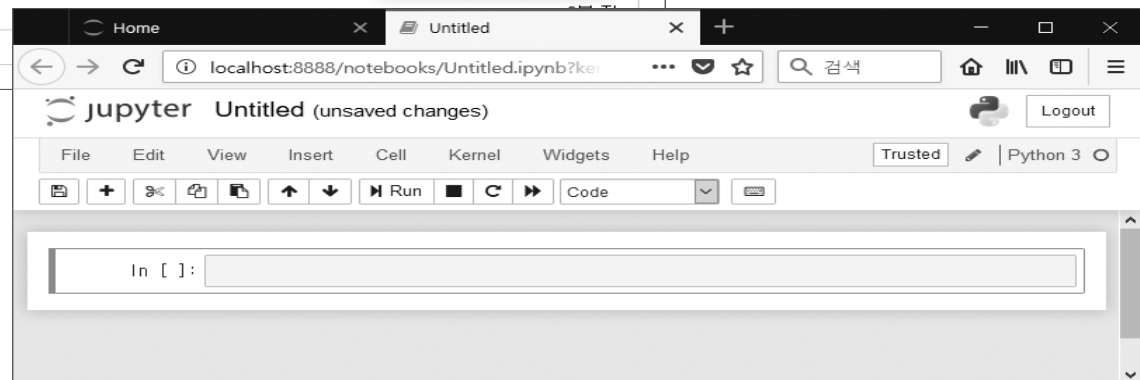
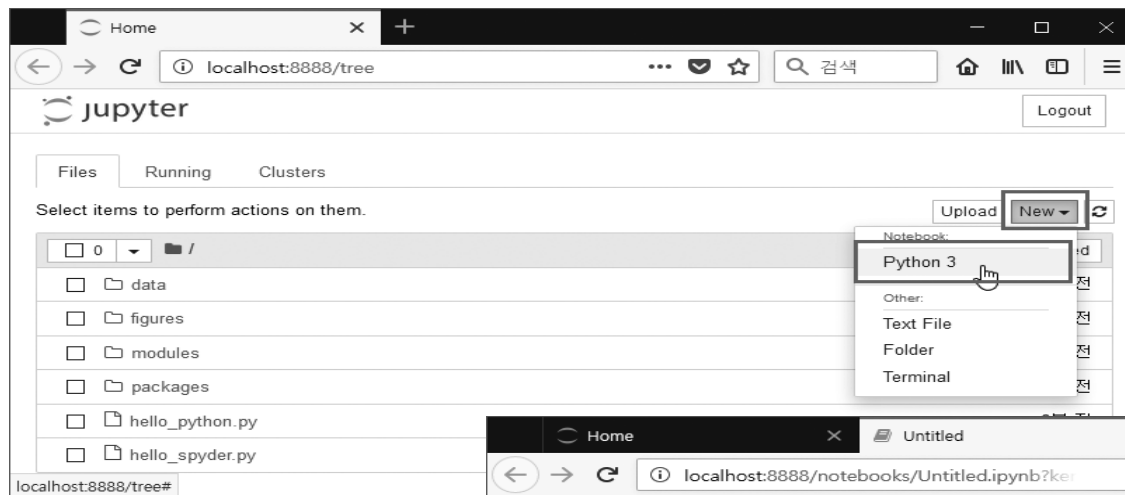


```
Anaconda Prompt
(base) C:\Users\choies> cd C:\myPyCode
(base) C:\myPyCode> jupyter notebook
```



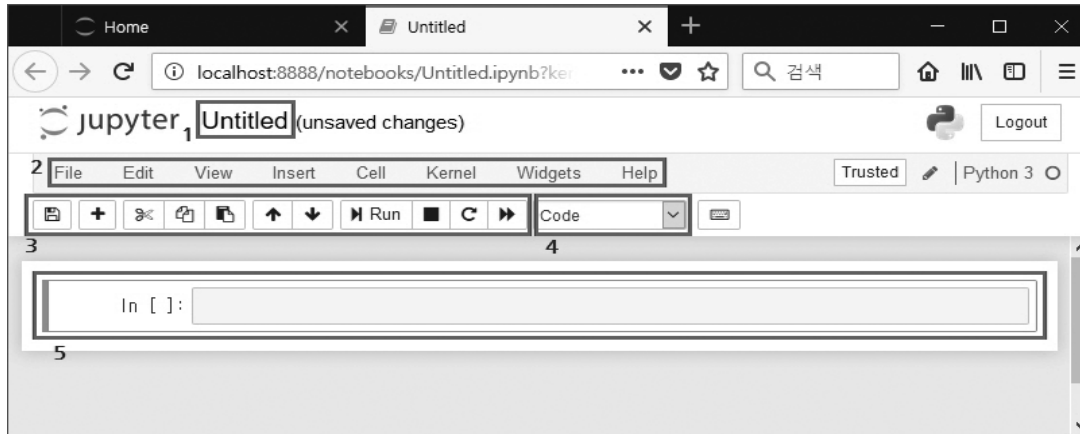
주피터 노트북에서 코딩하기

- ❖ 노트북을 생성하면 'C:\myPyCode' 폴더에 저장
- ❖ 실행된 주피터 노트북의 오른쪽 위에 [New] → [Python 3]를 차례대로 클릭

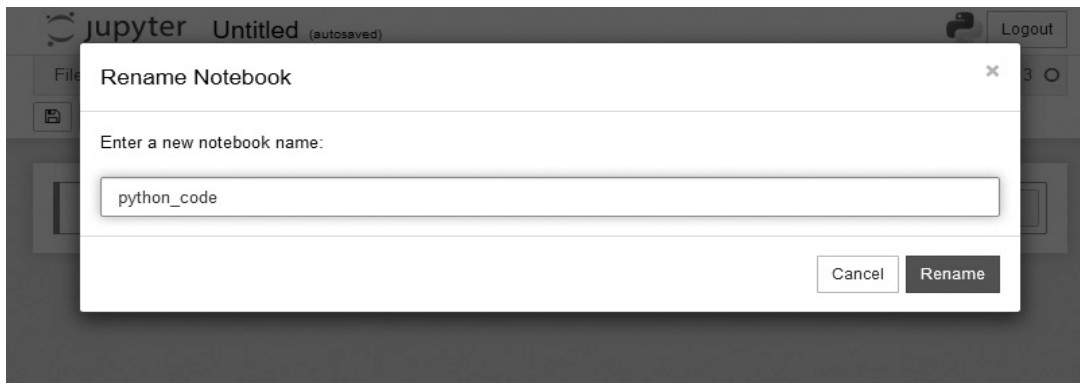


주피터 노트북 사용법

❖ 주피터 노트북의 사용자 메뉴



❖ 노트북 제목(파일 이름): 노트북을 새로 생성하면 임의로 제목이 지정. 노트북 제목을 클릭해 노트북



주피터 노트북 사용법

- ❖ 메뉴 바: 노트북의 모든 메뉴를 선택
- ❖ 툴 바: 메뉴 바에서 많이 사용하는 기능을 아이콘으로 제공
- ❖ 셀 타입
 - Code: 코드를 작성하기 위한 타입
 - Markdown: 마크다운 형식의 문서를 작성하기 위한 타입
- ❖ 셀: 파이썬 코드와 문자를 입력하는 셀
 - 코드 셀: 셀 앞에 `'In []:'` 이 표시됨. 코드를 입력하고 실행하면 [] 안에는 숫자가 나타나며, 입력한 순서대로 숫자가 1씩 증가
 - 코드 셀의 안쪽에 편집할 수 있는 영역을 편집 영역, 그 밖의 영역을 편집 외 영역이라고 함

주피터 노트북 사용법

❖ 편집 모드와 명령 모드

- 편집 모드(Edit mode): 코드나 문서를 작성하기 위한 모드

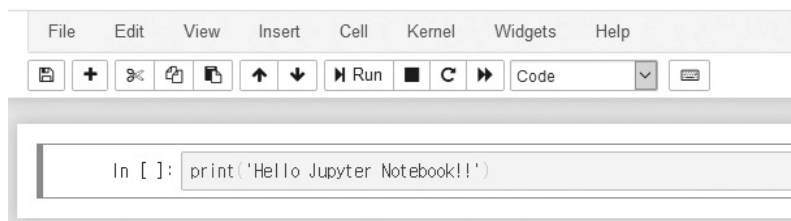
```
In [ ]: print("Jupyter Notebook: Edit mode")
```

- 명령 모드(Command mode): 셀을 다루기 위한 모드

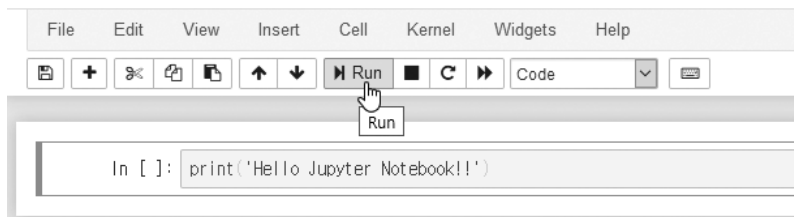
```
In [ ]: print("Jupyter Notebook: Command mode")
```

주피터 노트북에서 코드 작성

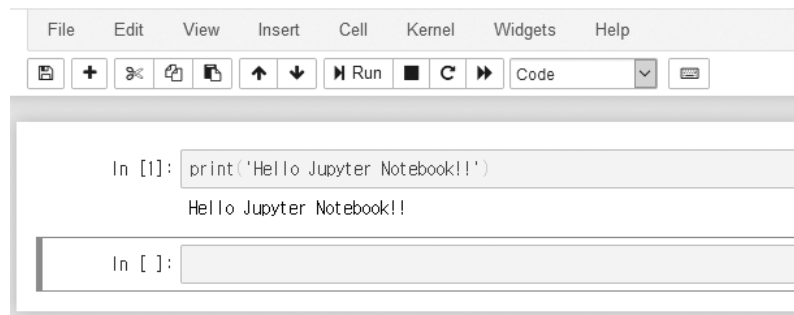
- ❖ 생성한 노트북의 첫 번째 셀 편집 영역을 마우스로 클릭해 선택하고 `print('Hello Jupyter Notebook!!')`을 입력



- ❖ 셀툴 바에서 [셀 실행] 아이콘을 클릭하거나 Shift + Enter를 누름

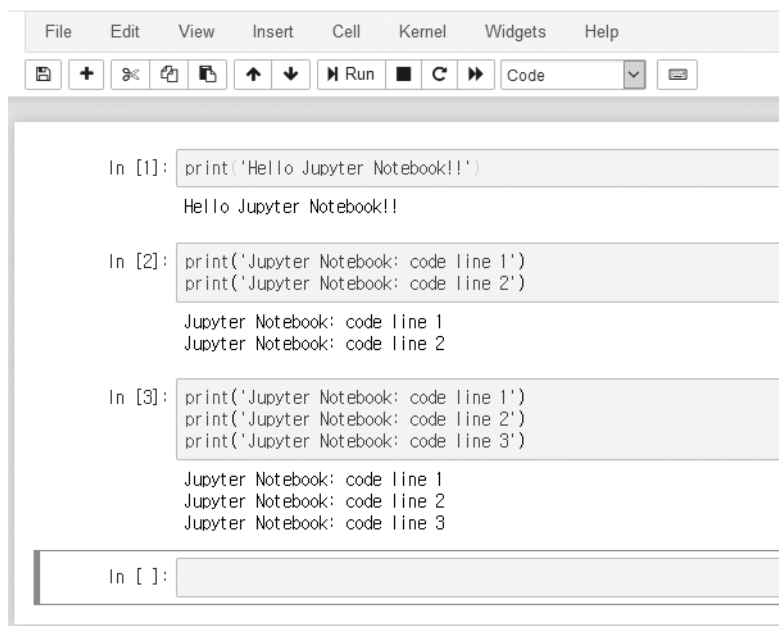


- ❖ 코드가 실행되고 그 아래 새로운 코드 셀이 추가됨



주피터 노트북에서 코드 작성

- ❖ 코드 셀에는 코드를 여러 줄 입력할 수 있음. 코드 셀에 코드를 여러 줄 입력해 실행하면 위에서부터 순차적으로 코드가 모두 실행됨



The screenshot displays the Jupyter Notebook interface with a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for adding, deleting, and running cells. Three code cells are shown, each with its input and output:

```
In [1]: print('Hello Jupyter Notebook!!')
Hello Jupyter Notebook!!
```

```
In [2]: print('Jupyter Notebook: code line 1')
print('Jupyter Notebook: code line 2')
Jupyter Notebook: code line 1
Jupyter Notebook: code line 2
```

```
In [3]: print('Jupyter Notebook: code line 1')
print('Jupyter Notebook: code line 2')
print('Jupyter Notebook: code line 3')
Jupyter Notebook: code line 1
Jupyter Notebook: code line 2
Jupyter Notebook: code line 3
```

At the bottom, there is an empty code cell labeled "In []:".

3

파이썬을 계산기처럼 이용하기



간단한 사칙 연산

❖ 주피터 노트북을 실행해 새 노트북을 준비

❖ 파이썬에서의 사칙 연산

- 더하기(+), 빼기(-), 곱하기(*), 나누기(/) 기호를 이용

❖ 덧셈과 뺄셈

```
In: 1 + 1
```

```
Out: 2
```

```
In: 5 - 2
```

```
Out: 3
```

- 파이썬에서는 숫자와 연산자 사이의 공백은 무시
 - '5-2'나 '5 -2'나 '5 - 2'나 모두 결과가 같음

간단한 사칙 연산

❖ 곱셈과 나눗셈

In: 15 * 2

Out: 30

In: 10 / 2

Out: 5.0

- 나눗셈의 결과가 5가 아니라 5.0
 - 파이썬(3.x 버전)에서 나눗셈 연산은 실수로 처리
 - 파이썬 버전 2.x에서는 정수 나누기 정수(즉, 정수/정수)의 경우 결과가 정수로 처리됨

간단한 사칙 연산

❖ 실수 연산

In: $1.2 + 5.3$

Out: 6.5

In: $3.5 - 5.0$

Out: -1.5

In: $1.4 * 2$

Out: 2.8

In: $5 / 2$

Out: 2.5

❖ 연산 기호가 두 개 이상일 경우

- 일반적인 연산 규칙을 따름
- 연산에서 괄호는 '('와 ')'를 이용

간단한 사칙 연산

❖ 복합 연산

In: $2 + 3 * 4$

Out: 14

In: $3 / 2 * 4 - 5 / 2$

Out: 3.5

In: $10 / 5 + (5 - 2) * 2$

Out: 8.0

In: $(5 * 4 - 15) + ((5 - 2) * (9 - 7))$

Out: 11

type() 함수

- ❖ 자료의 형식(데이터 타입)을 알려줌
- ❖ type() 함수의 인자로 정수를 입력하면 int를, 실수를 입력하면 float를 결과로 돌려줌
- ❖ type() 함수에 정수와 실수를 입력한 예

```
In: type(3)
```

```
Out: int
```

```
In: type(1.2)
```

```
Out: float
```

거듭제곱과 나머지

❖ 거듭제곱(Power) 표현

$$A^n = A \times A \times A \dots \times A$$

❖ 2⁵의 값을 구하고자 할 때

In: 2 * 2 * 2 * 2 * 2

Out: 32

❖ 거듭제곱을 위한 연산자 **

In: 2 ** 5

Out: 32

거듭제곱과 나머지

❖ 실수에 대한 거듭제곱 계산

```
In: 1.5 ** 2  
Out: 2.25
```

- 이때 거듭제곱의 지수도 정수일 필요가 없음

❖ 나머지

- 퍼센트 기호(%)를 이용: 나머지 연산자(Modulo operator)

```
In: 13 % 5  
Out: 3
```

❖ 몫

- 정수 나누기 연산자(//) 이용

```
In: 13 // 5  
Out: 2
```

산술 연산자

연산자 기호	의미	예	결과
+	더하기	$5 + 2$	7
-	빼기	$5 - 2$	3
*	곱하기	$5 * 2$	10
/	나누기	$5 / 2$	2.5
**	거듭제곱	$5 ** 2$	25
%	나머지	$5 \% 2$	1
//	몫	$5 // 2$	2

과학적 표기법

❖ 아주 큰 수나 작은 수를 다뤄야 할 때 사용(과학이나 공학 분야)

- 거듭제곱 연산자를 이용할 경우

```
In: 3 * 10 ** 8  
Out: 300000000
```

❖ 10의 거듭제곱(즉, 10^n)의 경우 en으로 편리하게 입력

- 3×10^8 를 입력할 때

```
In: 3e8  
Out: 300000000.0
```

과학적 표기법

- ❖ 숫자에 들어간 0의 개수에 따라서 en 형식 출력 여부가 결정
- ❖ 1e15를 입력하면 과학적 표기법으로 표시하지 않고, 1e16을 입력하면 과학적 표기법으로 표시
- ❖ 1e-4를 입력하면 과학적 표기법으로 표시하지 않고, 1e-5를 입력하면 과학적 표기법으로 표시
- ❖ 0의 개수에 따라 en 형식 출력 여부가 결정되는 예

```
In: 1e15  
Out: 1000000000000000.0
```

```
In: 1e16  
Out: 1e+16
```

```
In: 1e-4  
Out: 0.0001
```

```
In: 1e-5  
Out: 1e-05
```

진수 표현과 변환

❖ 다양한 진법의 예

- 시간을 표현할 때 : 60초가 1분이고 60분은 1시간(60진법)
- 오전과 오후는 각각 12시간이고, 1년도 12달이므로 12진법

❖ 컴퓨터 프로그래밍: 2진법(0과 1), 8진법, 16진법

❖ 파이썬에서는 10진법 외에 2진법, 8진법, 16진법으로 숫자를 입출력하며 변환하는 방법을 제공

- 10진수 외에 2진수, 8진수, 16진수를 입력하기 위해서는 숫자 앞에 각각 '0b', '0o', '0x'를 붙임

진수 표현과 변환

- 10진수 17을 각각 10진수, 2진수, 8진수, 16진수로 입력한 예

In: 17

Out: 17

In: 0b10001

Out: 17

In: 0o21

Out: 17

In: 0x11

Out: 17

- ❖ 파이썬에서는 2진수, 8진수, 16진수 형태로 숫자를 입력해도 기본적으로 10진수로 출력
- 숫자를 각 진법에 따라 표현해야 할 경우 2진수, 8진수, 16진수로 변환하는 함수를 활용: bin(), oct(), hex()

진수 표현과 변환

❖ 10진수 17을 2진수, 8진수, 16진수로 출력하는 예

```
In: bin(17)
Out: '0b10001 '
```

```
In: oct(17)
Out: '0o21 '
```

```
In: hex(17)
Out: '0x11'
```

❖ 주의사항

- 출력 결과가 작은 따옴표(' ') 안에 있는데 이것은 출력 결과가 숫자가 아니라 문자열임을 나타냄
- bin(), oct(), hex() 함수를 이용한 출력 결과는 산술 연산에 이용할 수 없음
- 진수 변환은 연산이 모두 끝난 후에 수행

진수 표현과 변환

❖ 서로 다른 진수의 연산 결과를 10진수, 2진수, 8진수, 16진수로 변환해 출력한 예

```
In: 0b10 * 0o10 + 0x10 - 10  
Out: 22
```

```
In: bin(0b10 * 0o10 + 0x10 - 10)  
Out: '0b10110'
```

```
In: oct(0b10 * 0o10 + 0x10 - 10)  
Out: '0o26'
```

```
In: hex(0b10 * 0o10 + 0x10 - 10)  
Out: '0x16'
```

논리 연산 및 비교 연산

❖ 조건에 따라 결과가 달라지는 다양한 경우

❖ 논리 연산(logical operation)

- 다양한 조건에 따라 코드가 다르게 실행되도록 작성
- 어떤 조건을 만족하는 참(True)과 만족하지 않는 거짓(False)을 이용

❖ 논리 연산은 불린 연산(Boolean operation)이라고도 함

- 논리 연산을 위한 데이터 타입: 불(bool)
- 불 데이터 타입에는 논리 참(True) 혹은 논리 거짓(False)이 있음
- 참(True) 혹은 거짓(False)을 입력할 때 참은 True, 거짓은 False를 입력

논리 연산 및 비교 연산

❖ 불 데이터 타입을 입력한 예

```
In: print(True)
```

```
Out: True
```

```
In: print(False)
```

```
Out: False
```

❖ True나 False의 데이터 타입은 type() 함수를 이용해 확인 가능

```
In: type(True)
```

```
Out: bool
```

❖ 불 데이터의 경우 논리 연산만 가능

- 논리곱(and): 두 개의 불 데이터가 모두 참일 때만 참이고 나머지는 거짓
- 논리합(or): 두 개의 불 데이터 중 하나라도 참이면 참이고 둘 다 거짓이면 거짓
- 논리부정(not): 하나의 불 데이터가 참이면 거짓이고 거짓이면 참

논리 연산 및 비교 연산

❖ 진리표

논리 연산자	의미	활용 예	설명
and	논리곱	A and B	A와 B 모두 참일 때만 참이고, 나머지는 거짓
or	논리합	A or B	A와 B 중 하나라도 참이면 참이고, 둘다 거짓일 때 거짓
not	논리 부정	not A	A가 참이면 거짓이고, 거짓이면 참

❖ 논리 연산자를 이용한 진리표

A	B	A and B	A or B	not A
False	False	False	False	True
False	True	False	True	True
True	False	False	True	False
True	True	True	True	False

논리 연산 및 비교 연산

❖ 논리 연산(and, or, not)의 예

```
In: print(True and False)
    print(True or False)
    print(not True)
```

```
Out: False
     True
     False
```

논리 연산 및 비교 연산

❖ 비교 연산

- 비교 연산의 결과는 불 데이터로 출력됨. 따라서 비교 연산은 논리 연산과 함께 이용하는 경우가 많음
- 비교 연산자

비교 연산자	의미	활용 예	설명
==	같다	$a == b$	a는 b와 같다
!=	같지 않다	$a != b$	a는 b와 같지 않다
<	작다	$a < b$	a는 b보다 작다
>	크다	$a > b$	a는 b보다 크다
<=	작거나 같다	$a \leq b$	a는 b보다 작거나 같다
>=	크거나 같다	$a \geq b$	a는 b보다 크거나 같다

논리 연산 및 비교 연산

❖ 비교 연산자를 이용해 연산한 예

```
In: print(5 == 3)
    print(5 != 3)
    print(5 < 3)
    print(5 > 3)
    print(5 <= 3)
    print(5 >= 3)
```

```
Out: False
     True
     False
     True
     False
     True
```


논리 연산 및 비교 연산

❖ 비교 연산과 논리 연산을 혼합한 연산

- 비교 연산자의 우선순위가 논리 연산자의 우선순위보다 높음

❖ In: `print(1 > 0 and -2 < 0)`
Out: `True`

- ❖ 비교 연산인 '`1 > 0`'과 '`-2 < 0`'을 먼저 수행하는데 결과는 모두 `True`이고, 이 연산의 결과를 이용해 논리곱(`and`) 연산을 수행
- ❖ 괄호와 연산이 함께 있으면 괄호의 우선순위가 높음. 괄호가 여러 겹 있을 때는 가장 안쪽 괄호부터 먼저 계산

```
In: print((3 < 0) and ((-5 > 0) and (1 > 5)))  
    print((3 > 0) or ((-5 > 0) and (1 > 5)))  
    print(((3 > 0) or (-5 > 0)) and ((4 > 8) or ( 3 < 0)))  
  
Out: False  
     True  
     False
```



4

변수와 자료형

변수

❖ 변수를 사용하지 않는 예: 12340의 1/2, 1/4, 1/5 구하기

In: 12340 * 1/2

Out: 6170.0

In: 12340 * 1/4

Out: 3085.0

In: 12340 * 1/5

Out: 2468.0

변수

❖ 변수(variable)

- 숫자와 같은 자료(data)를 넣을 수 있는 상자

❖ 변수명(변수 이름)

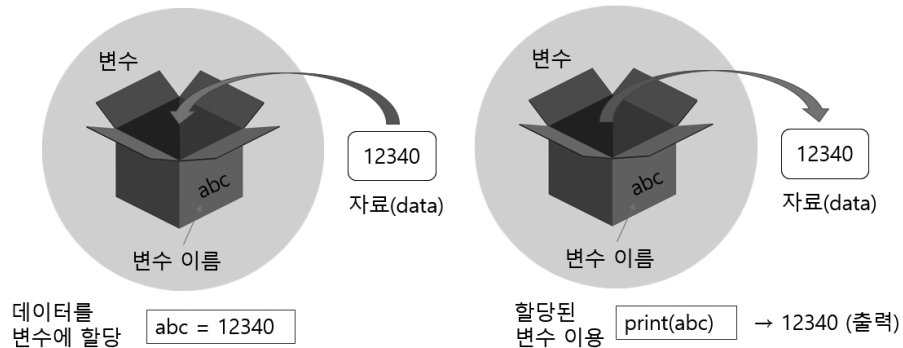
- 상자에 붙인 이름. 변수는 컴퓨터의 임시 저장 공간(메모리)에 저장됨

❖ 파이썬에서는 등호(=)를 이용해 변수에 자료를 할당

- '변수명 = data'
- 자료가 숫자라면 data에 숫자를 쓰면 되고 문자열이라면 문자열을 씀
- 변수명과 등호, 등호와 data 사이의 공백은 무시
- 데이터가 할당된 변수를 이용하려면 그냥 변수명만 쓰면 됨

변수

❖ 변수에 데이터를 할당하고 그 변수를 활용하는 예



❖ 데이터를 변수에 할당하고 활용하는 예

- 숫자 12340을 변수명이 abc인 변수에 할당하고 활용

```
In: abc = 12340  
    print(abc)
```

```
Out: 12340
```

변수

❖ 변수명으로 변수에 할당된 값을 출력

```
In: abc
```

```
Out: 12340
```

❖ 변수를 사용하는 예: 12340의 1/2, 1/4, 1/5 구하기

```
In: print(abc * 1/2)  
    print(abc * 1/4)  
    print(abc * 1/5)
```

```
Out: 6170.0
```

```
     3085.0
```

```
     2468.0
```



- 일일이 숫자를 입력하지 않아도 됨
- 각 연산에 변수 abc가 숫자 12340으로 대치돼 계산됨

변수명을 만드는 규칙

- ❖ 변수명은 문자, 숫자, 밑줄 기호(_)를 이용해 만듦
- ❖ 숫자로 시작하는 변수명을 만들 수 없음
- ❖ 대소문자를 구분
- ❖ 공백을 포함할 수 없음
- ❖ 밑줄 이외의 기호는 변수에 이용할 수 없음
- ❖ 예약어(Reserved word)는 변수명으로 이용할 수 없음

문자열

❖ 문자의 나열을 의미

❖ 파이썬에서는 따옴표로 둘러싸인 문자의 집합

❖ 문자열 만들기

- 문자열을 표시하기 위해 문자열 시작과 끝에 큰따옴표(")나 작은따옴표(')를 지정
- 둘 중 어떤것을 사용해도 되지만 양쪽에는 같은 기호를 이용

❖ 문자열 표시에 큰따옴표와 작은따옴표를 이용한 예

```
In: print("String Test")  
Out: String Test
```

```
In: print('String Test')  
Out: String Test
```


문자열

❖ 문자열을 변수에 저장한 후 print() 함수로 출력하는 예

```
In: string1 = "String Test 1"  
    string2 = 'String Test 2'  
    print(string1)  
    print(string2)
```

```
Out: String Test 1  
     String Test 2
```

❖ 변수에 할당한 문자열의 타입 확인

```
In: type(string1)  
Out: str
```

```
In: type(string2)  
Out: str
```

문자열

❖ 문자열 안에 큰따옴표나 작은따옴표 포함하기

```
In: string3 = 'This is a "double" quotation test'
    string4 = "This is a 'single' quotation test"
    print(string3)
    print(string4)
```

```
Out: This is a "double" quotation test
     This is a 'single' quotation test
```

- ❖ 문자열에 큰따옴표와 작은따옴표를 모두 포함하고 싶거나 문장을 여러 행 넣고 싶거나 입력한 그대로 출력하고 싶을 때
 - 문자열 전체를 삼중 큰따옴표(""")나 삼중 작은따옴표(''')로 감쌈

문자열

```
In: long_string1 = '''[삼중 작은따옴표를 사용한 예]
파이썬에는 삼중 따옴표로 여러 행의 문자열을 입력할 수 있습니다
.큰따옴표(")와 작은따옴표(')도 입력할 수 있습니다. '''
```

```
long_string2 = """[삼중 큰따옴표를 사용한 예]
파이썬에는 삼중 따옴표로 여러 행의 문자열을 입력할 수 있습니다
.큰따옴표(")와 작은따옴표(')도 입력할 수 있습니다."""
```

```
print(long_string1)
print(long_string2)
```

```
Out: [삼중 작은따옴표를 사용한 예]
파이썬에는 삼중 따옴표로 여러 행의 문자열을 입력할 수 있습니다
.큰따옴표(")와 작은따옴표(')도 입력할 수 있습니다.
[삼중 큰따옴표를 사용한 예]
파이썬에는 삼중 따옴표로 여러 행의 문자열을 입력할 수 있습니다
.큰따옴표(")와 작은따옴표(')도 입력할 수 있습니다.
```

문자열

❖ 문자열 다루기

- 더하기 연산자(+)와 곱하기 연산자(*)를 이용 가능
- 더하기 연산자: 문자열끼리 연결(concatenation)해 문자열을 하나로 만듦
- 곱하기 연산자: 곱한 만큼 문자열을 반복

```
In: a = 'Enjoy '  
    b = 'python!'  
    c = a + b  print(c)
```

```
Out: Enjoy python!
```

```
In: print(a * 3)
```

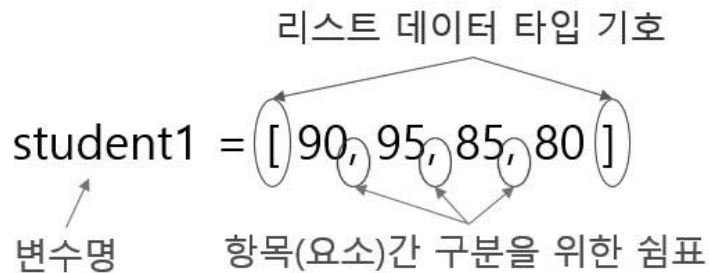
```
Out: Enjoy Enjoy Enjoy
```

리스트(List)

- ❖ 숫자, 문자열, 불 데이터 타입은 데이터를 하나씩만 처리 가능
- ❖ 때때로 데이터를 묶어 놓으면 처리하기가 편할 때가 있음
- ❖ 리스트(List)는 여러 개의 데이터를 묶어서 처리할 때 편리

리스트 만들기

- ❖ 대괄호([])를 이용해 만듦
- ❖ 리스트를 만들 때 각 항목의 데이터 타입은 같지 않아도 됨
- ❖ 데이터는 입력한 순서대로 지정되며 항목은 콤마(,)로 구분
- ❖ 대괄호 안에 아무것도 쓰지 않으면 빈 리스트가 만들어짐
- ❖ 리스트를 만드는 예



In: # 1번 학생의 국어, 영어, 수학, 과학 점수가 각각 90,95,85,80

```
student1 = [90,95,85,80]
```

```
student1
```

Out: [90, 95, 85, 80]

리스트 만들기

❖ 리스트 타입의 데이터가 할당된 변수(리스트 변수)의 구조

student1[0] [1] [2] [3]
↑ ↑ ↑ ↑
student1 = [90, 95, 85, 80]

❖ 리스트에서 각 항목은 '변수명[i]'로 지정

```
In: student1[0]
```

```
Out: 90
```

```
In: student1[1]
```

```
Out: 95
```

```
In: student1[-1]
```

```
Out: 80
```

리스트 만들기

❖ 리스트의 특정 항목을 변경할 경우

- '변수명[i] = new_data'를 이용

```
In: student1[1] = 100 # 두 번째 항목에 새로운 데이터 할당
    student1
```

```
Out: [90, 100, 85, 80]
```

❖ 리스트에 문자열을 입력하는 예

```
In: myFriends = ['James', 'Robert', 'Lisa', 'Mary']
    myFriends
```

```
Out: ['James', 'Robert', 'Lisa', 'Mary']
```


리스트 만들기

❖ 리스트 변수 myFriends에서 세 번째, 네 번째, 마지막 항목을 지정하는 예

```
In: myFriends[2]  
Out: 'Lisa'
```

```
In: myFriends[3]  
Out: 'Mary'
```

```
In: myFriends[-1]  
Out: 'Mary'
```

❖ 숫자, 문자열, 불, 리스트를 혼합한 형태의 리스트

```
In: mixedList = [0, 2, 3.14, 'python', 'program', True, myFriends]  
mixedList
```

```
Out: [0, 2, 3.14, 'python', 'program', True, ['James', 'Robert', 'Lisa', 'Mary']]
```

리스트 다루기

❖ 리스트 더하기와 곱하기

- 더하기: 두 리스트 연결
- 곱하기: 리스트를 곱한 수만큼 반복

❖ 리스트 더하기

```
In: list_con1= [1,2,3,4]
    list_con2 = [5,6,7,8]
    list_con = list_con1 + list_con2 # 리스트 연결

    print(list_con)
```

```
Out: [1, 2, 3, 4, 5, 6, 7, 8]
```

❖ 리스트 곱하기

```
In: list_con1= [1,2,3,4]
    list_con = list_con1 * 3 # 리스트 반복
    print(list_con)
```

```
Out: [1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4]
```

리스트 다루기

❖ 리스트 중 일부 항목 가져오기

- 인덱스의 범위를 지정해 리스트 중 일부 항목을 가져옴

```
리스트[i_start:i_end]
```

```
리스트[i_start:i_end:i_step]
```

- `i_start`, `i_end`, `i_step`은 각각 인덱스의 시작, 끝, 스텝(증가 단계)
- '`i_start`'에서 '`i_end - 1`'까지의 리스트를 반환
- `i_start`를 생략하면 인덱스는 0으로 간주
- `i_end`를 생략하면 인덱스는 마지막이라고 간주

리스트 다루기

❖ 리스트 중 일부 항목 가져오기

- 인덱스의 범위를 지정해 리스트 중 일부 항목을 가져오는 예

```
In: list_data = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
    print(list_data)
    print(list_data[0:3])
    print(list_data[4:8])
    print(list_data[:3])
    print(list_data[7:])
    print(list_data[::2])
```

```
Out: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
      [0, 1, 2]
      [4, 5, 6, 7]
      [0, 1, 2]
      [7, 8, 9]
      [0, 2, 4, 6, 8]
```

리스트 다루기

❖ 리스트에서 항목 삭제하기

- `del 리스트[i]`

```
In: print(list_data)
    del list_data[6]
    print(list_data)
```

```
Out: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
     [0, 1, 2, 3, 4, 5, 7, 8, 9]
```

❖ 리스트에서 항목의 존재 여부 확인하기

- 항목 `in` 리스트
- 리스트에 항목이 있으면 `True`, 없으면 `False`를 반환

```
In: list_data1 = [1, 2, 3, 4, 5]
    print(5 in list_data1)
    print(6 in list_data1)
```

```
Out: True
     False
```

리스트 다루기

❖ 리스트 메서드 활용하기

■ 변수명.메서드이름()

리스트 메서드	설명	사용 예
append()	리스트에서 항목 하나를 맨 마지막에 추가	myFriends.append('Thomas')
insert()	리스트에서 특정 위치에 항목을 삽입	myFriends.insert(1, 'Paul')
extend()	리스트에서 항목 여러 개를 맨 마지막에 추가	myFriends.extend(['Laura', 'Betty'])
remove()	입력값과 첫 번째로 일치하는 항목을 리스트에서 삭제	myFriends.remove('Laura')
pop()	리스트의 마지막 항목을 제거한 후에 반환	popFriend = myFriends.pop()
index()	리스트에서 인자와 일치하는 첫 번째 항목의 위치를 반환	indexFriend = myFriends.index('Lisa')
count()	리스트에서 인자와 일치하는 항목의 개수를 반환	countFriend = myFriends.count('Mary')
sort()	숫자나 문자열로 구성된 리스트 항목을 순방향으로 정렬	myFriends.sort()
reverse()	리스트 항목을 끝에서부터 역순으로 정렬	myFriends.reverse()

튜플(Tuple)

- ❖ 리스트와 유사하게 데이터 여러 개를 하나로 묶는 데 이용
- ❖ 튜플의 항목은 숫자, 문자열, 불, 리스트, 튜플, 세트, 딕셔너리 등으로 만들 수 있음
- ❖ 튜플의 속성은 리스트와 유사
- ❖ 튜플 데이터는 한번 입력(혹은 생성)하면 그 이후에는 항목을 변경할 수 없음
- ❖ 튜플 만들기
 - 소괄호('()')를 사용하거나 괄호를 사용하지 않고 데이터를 입력

```
In: tuple1 = (1,2,3,4)
    tuple1
```

```
Out: (1, 2, 3, 4)
```

튜플 만들기

❖ 튜플의 요소 지정

```
In: tuple1[1]
```

```
Out: 2
```

❖ 소괄호를 사용하지 않고 튜플 생성

```
In: tuple2 = 5,6,7,8  
    print(tuple2)
```

```
Out: (5, 6, 7, 8)
```

❖ 인자가 하나만 있는 튜플 생성

```
In: tuple3 = (9,) # 반드시 콤마(,) 필요  
    tuple4 = 10, # 반드시 콤마(,) 필요  
    print(tuple3)  
    print(tuple4)
```

```
Out: (9,)  
     (10,)
```


튜플 다루기

- ❖ 한번 생성된 튜플은 요소를 변경하거나 삭제할 수 없음
- ❖ 한 번 생성한 후에 요소를 변경할 필요가 없거나 변경할 수 없도록 하고 싶을 때 주로 이용
- ❖ 튜플의 요소를 변경하고자 하는 예

```
In: tuple5 = (1,2,3,4)
    tuple5[1] = 5 # 한번 생성된 튜플의 요소는 변경되지 않음

Out: -----
      TypeError                                Traceback (most recent call last)
<ipython-input-91-883687f4b7db> in <module>()  1 tuple5 =
    (1,2,3,4)
----> 2 tuple5[1] = 5 # 한번 생성된 튜플의 요소는 변경되지 않음

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

세트(Set)

- ❖ 수학의 집합 개념을 구현한 데이터 타입
- ❖ 데이터의 순서가 없고 데이터를 중복해서 쓸 수 없음
- ❖ 교집합, 합집합, 차집합을 구하는 메서드를 사용할 수 있음
- ❖ 세트 만들기
 - 중괄호('{ }')로 데이터를 감싸서 만듦

```
In: set1 = {1, 2, 3}
    set1a = {1, 2, 3, 3}
    print(set1)
    print(set1a)
```

```
Out: {1, 2, 3}
      {1, 2, 3}
```

세트의 교집합, 합집합, 차집합 구하기

❖ 세트의 교집합, 합집합, 차집합 메서드

메서드	기호 표시	사용 예
교집합(intersection)	$A \cap B$	<code>A.intersection(B)</code>
합집합(union)	$A \cup B$	<code>A.union(B)</code>
차집합(difference)	$A - B$	<code>A.difference(B)</code>

❖ 집합 메서드 사용 예

```
In: A = {1, 2, 3, 4, 5}          # Set A
    B = {4, 5, 6, 7, 8, 9, 10}   # Set B
    A.intersection(B)           # 집합 A에 대한 집합 B의 교집합( $A \cap B$ )
```

```
Out: {4, 5}
```

```
In: A.union(B) # 집합 A에 대한 집합 B의 합집합( $A \cup B$ )
```

```
Out: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

```
In: A.difference(B) # 집합 A에 대한 집합 B의 차집합( $A - B$ )
```

```
Out: {1, 2, 3}
```

세트의 교집합, 합집합, 차집합 구하기

❖ 집합 연산자 이용

❖ 교집합, 합집합, 차집합을 위한 세트 연산자: '&', '|', '-'

```
In: A = {1, 2, 3, 4, 5}      # Set A
    B = {4, 5, 6, 7, 8, 9, 10} # Set B
    A & B                    # 집합 A에 대한 집합 B의 교집합( $A \cap B$ )
```

```
Out: {4, 5}
```

```
In: A | B # 집합 A에 대한 집합 B의 합집합( $A \cup B$ )
```

```
Out: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

```
In: A - B # 집합 A에 대한 집합 B의 차집합( $A - B$ )
```

```
Out: {1, 2, 3}
```

리스트, 튜플, 세트 간 타입 변환

❖ list(), tuple(), set()을 이용해 서로 변환

```
In: a = [1,2,3,4,5]
```

```
In: type(a)
```

```
Out: list
```

```
In: b = tuple(a)
```

```
    b
```

```
Out: (1, 2, 3, 4, 5)
```

```
In: type(b)
```

```
Out: tuple
```

```
In: c = set(a)
```

```
    c
```

```
Out: {1, 2, 3, 4, 5}
```

```
In: type(c)
```

```
Out: set
```

```
In: list(b)
```

```
Out: [1, 2, 3, 4, 5]
```

```
In: list(c)
```

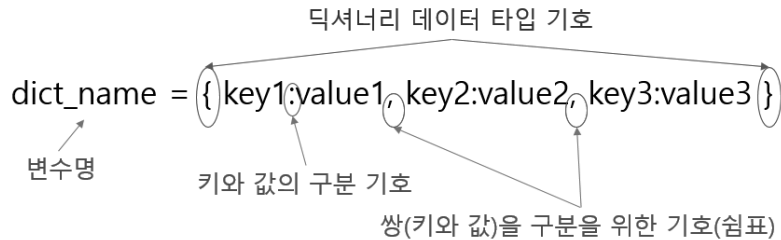
```
Out: [1, 2, 3, 4, 5]
```

딕셔너리

- ❖ 사전의 구성: 표제어가 있고 그에 대한 설명. 표제어만 찾으면 그에 대한 설명을 전부 확인할 수 있음
- ❖ 파이썬의 딕셔너리도 사전과 유사하게 구성돼 있으며 사전의 표제어와 설명은 파이썬에서 각각 키(key)와 값(value)에 해당
- ❖ 키와 값이 항상 쌍으로 구성. 따라서 키를 알면 그에 해당하는 값을 쉽게 알 수 있음
- ❖ 딕셔너리는 인덱스 대신 키를 이용해 값을 다룸
- ❖ 딕셔너리의 키는 임의로 지정한 숫자나 문자열이 될 수 있으며, 값으로는 어떤 데이터 타입도 사용 가능
- ❖ 딕셔너리 만들기
 - 데이터 전체를 중괄호({ })로 감싸서 만듦.
 - 키와 값의 구분은 콜론(:)으로 함
 - 키와 값으로 이뤄진 각 쌍은 콤마(,)로 구분

딕셔너리

❖ 딕셔너리의 구조와 생성 방법



❖ 딕셔너리의 키와 값이 모두 문자열인 예

```
In: country_capital = {"영국": "런던", "프랑스": "파리", "스위스": "베른", "호주": "멜  
버른", "덴마크": "코펜하겐"}  
  
country_capital
```

```
Out: {'덴마크': '코펜하겐', '스위스': '베른', '영국': '런던', '프랑스': '파리', '호주': '멜  
버른'}
```

❖ 특정 키의 값만 가져오는 예

```
In: country_capital["영국"]
```

```
Out: '런던'
```

딕셔너리 다루기

❖ 딕셔너리에 데이터 추가하기

```
In: country_capital["독일"] = "베를린"  
country_capital
```

```
Out: {'덴마크': '코펜하겐', '독일': '베를린', '스위스': '베른', '영국': '런던', '프랑스': '파리', '호주': '멜버른'}
```

❖ 딕셔너리의 데이터 변경하기

```
In: country_capital["호주"] = "캔버라"  
country_capital
```

```
Out: {'덴마크': '코펜하겐', '독일': '베를린', '스위스': '베른', '영국': '런던', '프랑스': '파리', '호주': '캔버라'}
```

❖ 딕셔너리의 데이터 삭제하기

```
In: del country_capital["덴마크"]  
country_capital
```

```
Out: {'독일': '베를린', '스위스': '베른', '영국': '런던', '프랑스': '파리', '호주': '캔버라'}
```


딕셔너리 메서드 활용하기

딕셔너리 메서드	설명	사용 예
keys()	딕셔너리에서 키 전체를 리스트 형태로 반환	dict_data.keys()
values()	딕셔너리에서 값 전체를 리스트 형태로 반환	dict_data.values()
items()	딕셔너리에서 키와 값의 쌍을 (키, 값)처럼 튜플 형태로 반환	dict_data.items()
update(dict_data2)	딕셔너리에 딕셔너리 데이터('dict_data2') 추가	dict_data.update(dict_data2)
clear()	딕셔너리의 모든 항목 삭제	dict_data.clear()

5

제어문



제어문

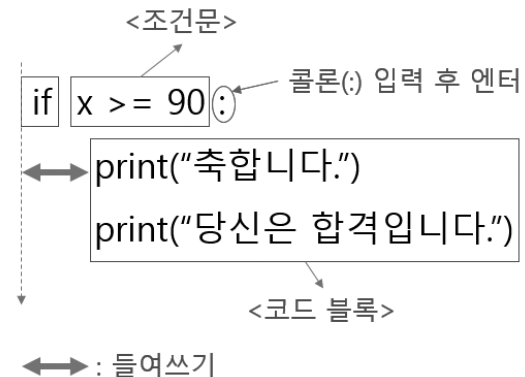
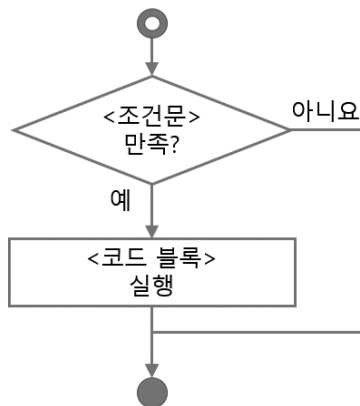
- ❖ 조건에 따라서 명령을 다르게 수행
- ❖ 특정한 조건을 만족할 때까지 계속 반복
- ❖ 제어문: 코드의 진행 순서를 바꾸는 구문
 - 조건문: 조건을 검사해 분기하는 구문
 - 반복문: 어떤 구간이나 조건을 만족하는 동안 코드의 일부분을 반복

조건에 따라 분기하는 if 문

- ❖ 지정한 조건에 따라 다르게 분기해 명령을 수행
- ❖ 조건의 만족 여부에 따라서 코드 수행 결과가 달라짐
- ❖ 단일 조건에 따른 분기(if)

```
if <조건문>:  
    <코드 블록>
```

❖ 조건문의 기본 구조



조건에 따라 분기하는 if 문

❖ 비교 연산자

비교 연산자	의미	활용 예	설명
==	같다	a == b	a는 b와 같다
!=	같지 않다	a != b	a는 b와 같지 않다
<	작다	a < b	a는 b보다 작다
>	크다	a > b	a는 b보다 크다
<=	작거나 같다	a <= b	a는 b보다 작거나 같다
>=	크거나 같다	a >= b	a는 b보다 크거나 같다

❖ 논리 연산자

논리 연산자	의미	활용 예	설명
and	논리곱	A and B	A와 B가 모두 참이면 참이고 그 외에는 거짓
or	논리합	A or B	A와 B 중 하나라도 참이면 참이고 둘 다 거짓이면 거짓
not	논리부정	not A	A가 참이면 거짓이고 거짓이면 참

조건에 따라 분기하는 if 문

❖ 변수 x의 값이 90보다 크거나 같으면 'Pass'를 출력하는 예

```
In: x = 95  
if x >= 90:  
    print("Pass")
```

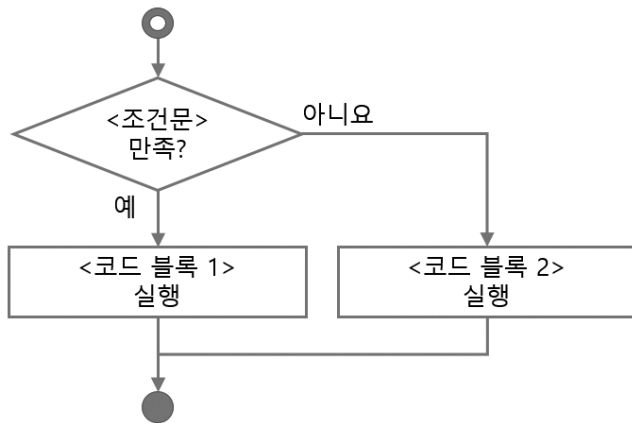
Out: Pass

단일 조건 및 그 외 조건에 따른 분기

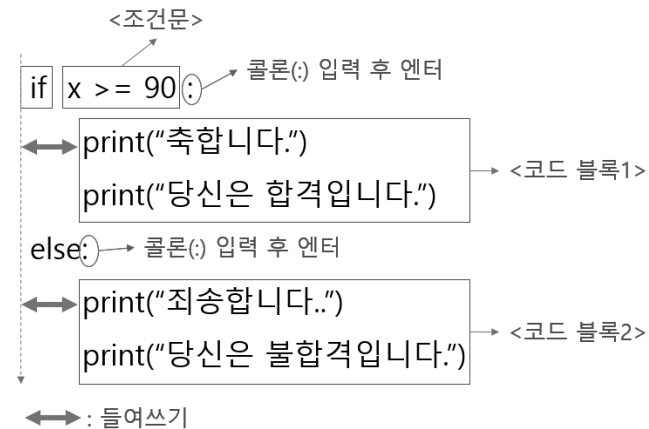
- ❖ <조건문>의 만족 여부에 따라 코드를 다르게 수행하려면 'if ~ else' 구조의 조건문을 이용

```
if <조건문>:  
    <코드 블록 1>  
else:  
    <코드 블록 2>
```

- ❖ if ~ else 문의 흐름도



- ❖ if ~ else 문의 사용 예

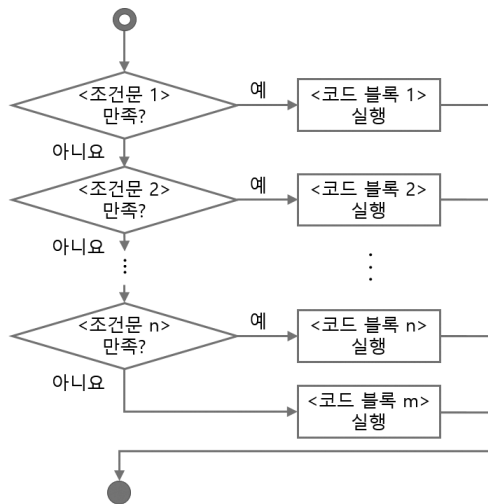


여러 조건에 따른 분기

❖ 'if ~ elif ~ else' 조건문을 이용

```
if <조건문 1>:  
    <코드 블록 1>  
elif <조건문 2>:  
    <코드 블록 2>  
...  
elif <조건문 n>:  
    <코드 블록 n>  
else:  
    <코드 블록 m>
```

❖ if ~ elif ~ else 문의 흐름도



```
In: x = 85  
if x >= 90:  
    print("Very good")  
elif (x >= 80) and (x < 90):  
    print("Good")  
else:  
    print("Bad")
```

Out: Good

중첩 조건에 따른 분기

❖ 중첩 조건문

- 조건문 안에 또 다른 조건문을 사용한 구조

```
if <조건문 1>:  
    if <조건문 1-1>:  
        <코드 블록 1-1>  
    else:  
        <코드 블록 1-2>  
elif <조건문 2>:  
    <코드 블록 2>  
else:  
    <코드 블록 3>
```

❖ 중첩 조건문을 사용한 예

```
if x >= 90: → <조건문 1>  
    if x == 100: → <조건문 1-1>  
        print("완벽합니다.") → <코드 블록 1-1>  
    else:  
        print("훌륭합니다.") → <코드 블록 1-2>  
elif 80 <= x < 90: → <조건문 2>  
    print("잘했습니다.") → <코드 블록 2>  
else:  
    print("노력이 필요합니다.") → <코드 블록 3>  
↕ : 들여쓰기
```

중첩 조건에 따른 분기

❖ 중첩 조건문을 사용한 코드

```
In: x = 100
    if x >= 90:
        if x==100 :
            print("Perfect")
        else:
            print("Very Good")
    elif (x >= 80) and (x < 90):
        print("Good")
    else:
        print("Bad")
```

Out: Perfect

for 문

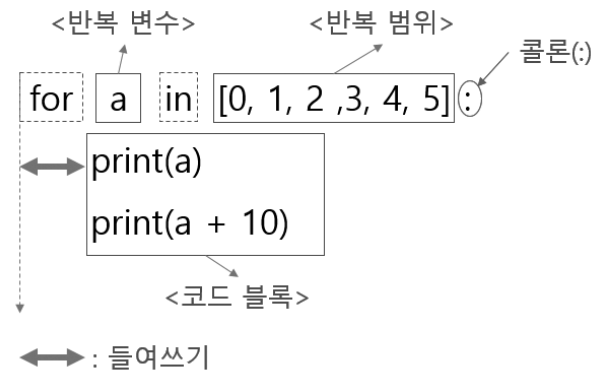
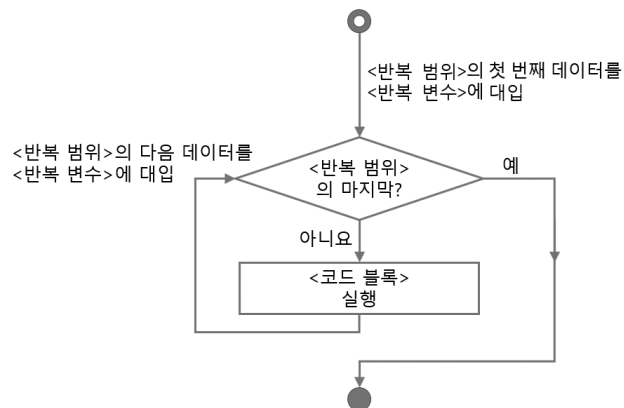
❖ 반복문

- 작업을 반복적으로 수행하는 구문
- for 문과 while 문

❖ for 문의 구조

```
for <반복 변수> in <반복 범위>:  
    <코드 블록>
```

❖ for 문의 흐름도



for 문

❖ 반복 범위 지정

- 리스트와 range() 함수를 이용

❖ 리스트 이용

```
In: for a in [0, 1, 2, 3, 4, 5]:  
    print(a)
```

```
Out: 0
```

```
1
```

```
2
```

```
3
```

```
4
```

```
5
```

❖ 문자열 리스트 이용

```
In: myFriends = ['James', 'Robert', 'Lisa', 'Mary'] # 리스트를 변수에 할당  
    for myFriend in myFriends: print(myFriend)
```

```
Out: James
```

```
Robert Lisa Mary
```

for 문

❖ range() 함수 이용

```
range(start, stop, step)
```

- start는 범위의 시작 지점, stop은 범위의 끝 지점, step은 증감의 크기

❖ range() 함수의 사용 예

```
In: print(range(0, 10, 1))
```

```
Out: range(0, 10)
```

```
In: print(list(range(0, 10, 1)))
```

```
Out: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
In: for a in range(0, 6, 1): print(a)
```

```
Out: 0
```

```
1
```

```
2
```

```
3
```

```
4
```

```
5
```

for 문

❖ range() 함수에서 step이 1인 경우 생략 가능

```
range(start, stop)
```

❖ step이 1이고 start가 0인 경우 start 역시 생략 가능

```
range(stop)
```

❖ range() 함수 사용 예

```
In: print(list(range(0, 10, 1)))  
    print(list(range(0, 10)))  
    print(list(range(10)))
```

```
Out: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]  
      [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]  
      [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

for 문

❖ 중첩 for 문

```
for <반복 변수 1> in <반복 범위 1>:  
    for <반복 변수 2> in <반복 범위 2>:  
        <코드 블록>
```

❖ 중첩 for 문의 예

```
In: x_list = ['x1', 'x2']  
    y_list = ['y1', 'y2']  
  
    print("x y")  
    for x in x_list:  
        for y in y_list:  
            print(x,y)
```

```
Out: x y  
     x1 y1  
     x1 y2  
     x2 y1  
     x2 y2
```

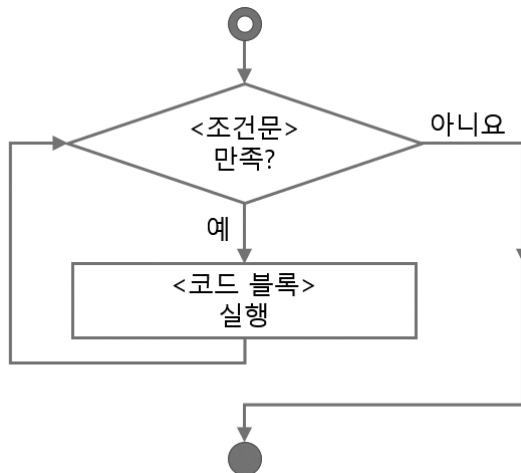
while 문

❖ 조건에 따라 반복 여부를 결정

❖ while 문의 구조

```
while <조건문>:  
    <코드 블록>
```

❖ while 문의 흐름도



while 문

❖ while 문의 사용 예

```
In: i = 0      # 초기화
    sum = 0    # 초기화

    print("i sum")
    while (sum < 20):  # 조건 검사
        i = i + 1      # i를 1씩 증가
        sum = sum + i  # sum과 현재 i를 더해서 sum을 갱신
        print(i, sum)  # i와 sum을 출력
```

Out: i sum

1 1

2 3

3 6

4 10

5 15

6 21

while 문

❖ 무한 반복 while 문

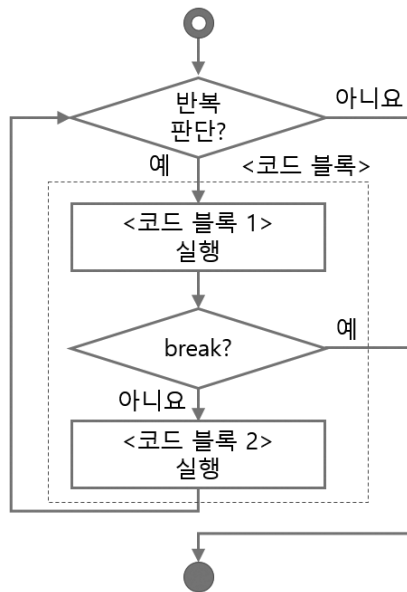
- 코드 블록을 무조건 계속 반복하라고 명령을 내려야 할 때

```
while True:  
    print("while test")
```

break와 continue

❖ 반복문을 빠져나오는 break

- 반복문에서 break 사용 시 흐름



```
In: k=0
while True:
    k = k + 1 # k는 1씩 증가

    if(k > 3): # k가 3보다 크면
        break # while 문을 빠져나옴

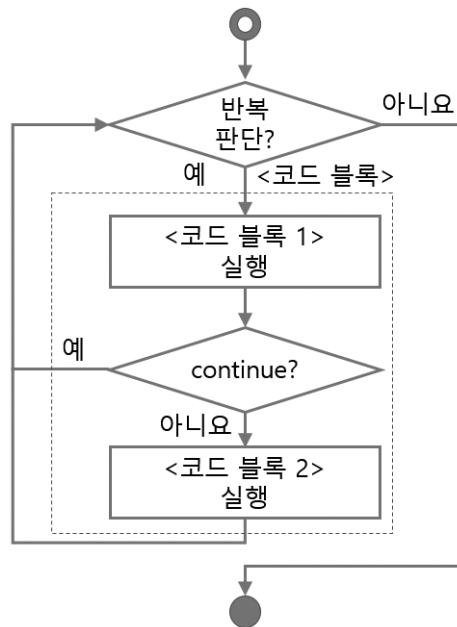
    print(k)    # k 출력
```

```
Out: 1
     2
     3
```

break와 continue

❖ 다음 반복을 실행하는 continue

- 반복문에서 continue 사용 시 흐름



```
In: for k in range(5):  
    if(k == 2):  
        continue  
  
    print(k)
```

```
Out: 0  
     1  
     3  
     4
```

간단하게 반복하는 한 줄 for 문

❖ 컴프리헨션(comprehension)

- 리스트 컴프리헨션
- 세트 컴프리헨션
- 딕셔너리 컴프리헨션

❖ 리스트 컴프리헨션의 기본 구조

```
[<반복 실행문> for <반복 변수> in <반복 범위>]
```

❖ 리스트 컴프리헨션의 예

```
In: numbers = [1,2,3,4,5]
    square = [i**2 for i in numbers]
    print(square)
```

```
Out: [1, 4, 9, 16, 25]
```

❖ 조건문을 포함한 리스트 컴프리헨션

```
[<반복 실행문> for <반복 변수> in <반복 범위> if <조건문>]
```

간단하게 반복하는 한 줄 for 문

❖ 조건문을 포함한 리스트 컴프리헨션의 예

```
In: numbers = [1,2,3,4,5]
    square = [i**2 for i in numbers if i>=3]

    print(square)
```

```
Out: [9, 16, 25]
```

감사합니다

2020 ※ 본 교안은 강의 수강 용도로만 사용 가능합니다.
상업적 이용을 일절 금함.