1. Jupyter Notebook에서 code 실행하기
  1)Notebook 신규 작성하기
    -Anaconda3 (64-bit) > Anaconda Prompt에서 다음의 명령을 실행하여 Jupyter Notebook을 실행한다.
      $ jupyter notebook
    -OS에 초기 설정된 Web browser가 실행되어 Jupyter Notebook의 Home 화면이 표시된다.
    -먼저 Notebook을 작성한다.
    -[New] > [Notebook:Python 3]을 선택한다.
    -'Untitled'라고 표시되어 있는 곳은 Notebook의 명칭이다.
    -저장되는 Notebook의 file명으로도 사용할 수 있다.
    -명칭은 변경이 가능하다.
    -명칭을 변경하려면 'Untitled'라고 표시되어 있는 곳을 선택해서 title을 'HelloJupyter'로 변경한다.
    -[Rename] button을 click하면 또는 Enter를 누르면 명칭이 변경된다.

  2)Code 실행하기
    -In [ ]:로 표시되어 있는 입력란이 code cell이다.
    -Code cell에 Python의 code를 입력한다.
      print('Hello Jupyter!!')
    -Code cell을 선택한 상태에서 Shift+Enter를 누르면 code를 실행할 수 있다.
      --Python code를 작성하면 Python의 문법 부분이 highlight(Syntax Highlight)된다.
      --print() 함수가 실행 가능하다.
      --실행 결과의 출력 내용이 표시된다.
    -다시 처음 작성한 Code cell를 선택하고 계속 Ctrl+Enter를 누를 때마다 아래로 code cell이 내려가지 않으면서 In [숫자]: 의 숫자는 계속 증가한다.
    -굳이 print()를 사용하지 않아도 문자열을 입력하고 계속 Ctrl+Enter를 누를 때마다 실행결과가 Out [숫자] : 의 숫자가 증가한다.
    -추가된 새로운 code cell에 오히려 print() 함수의 괄호가 닫혀있지 않은 부정 코드 (print('Hello Jupyter!!')를 입력해서 실행해 본다.
    -그러면 error 내용이 code cell 바로 아래에 표시된다.
      File "<ipython-input-17-a0d2abf11254>", line 1
        (print("Hello Jupyter!!'"")
                                  ^
      SyntaxError: unexpected EOF while parsing
    -Error 출력은 Python이 출력하는 error message가 강조된 것으로, 문법상의 문제를 의미하는 SyntaxError가 출력되고 있다.
    -Error message는 code 내용에서 잘못된 부분을 찾아볼 수 있는 hint가 된다.
    -다음과 같은 cycle을 반복하면서 programming 하는 환경을 'Interactive 환경' 또는 '대화식 환경'이라고 부른다.
      --code를 code cell에 작성한다.
      --coce를 실행하여 결과를 확인한다.
      --error가 발생하거나 의도하지 않은 결과가 출력된 경우 code 내용을 재검토한다.
      --다음 code cell에 code를 작성한다.
    -Interactive 환경은 Notebook의 중요한 기능 중의 하나이다.

  3)변수나 함수를 정의해서 이용하기
    -Code cell에 변수나 함수의 정의를 입력한다.
      your_name = 'Jupyter'

      def hello(name):
        print('Hello {0} !!' .format(name))

```
      hello(your_name)
```
-실행하면 다음 문자열이 출력된다.
```
      Hello Jupyter !!
```
-한번 정의한 변수나 함수는 다른 code cell에서 이용 가능하다.
-다른 cell에 아래와 같이 입력하면 Hello Notebook !!라고 출력된다.
```
      hello('Notebook')
```
-code를 작성한 것만으로는 변수나 함수를 정의한 것과 같이 되지 않으므로 한 번 code cell을 실행할 필요가 있다.
-실행/미실행은 code cell 왼쪽의 In [] 안의 숫자가 들어 있는지 없는지로 판별할 수 있다.
-비어 있는 경우 그 code cell은 실행되지 않는다.
-정의되지 않는 변수나 함수를 이용하면 NameError가 된다.
-이것도 Python에서 정의되지 않은 변수나 함수를 이용하려는 경우의 동작과 동일하다.

4)Python의 표준 library 이용하기
-Notebook에서도 Python의 표준 library나 third-party package를 이용할 수 있다.
```
      import math

      math.sqrt(4)
```

5)DocString의 표시
-Graph명이나 함수명의 끝에 '?'를 붙여서 실행하면 'DocString'을 참조할 수 있다.
-방금 사용한 표준 library math에 포함되어 있는 sqrt() 함수에 '?'를 붙여 실행한 결과로 하단에 'DocString'이 표시된다.
```
      math.sqrt?
      ----------------------------------------------------------------
      Signature: math.sqrt(x, /)
      Docstring: Return the square root of x.
      Type:      builtin_function_or_method
```

-DocString은 Python의 module이나 class, 함수의 설명문(comment)이다.
-DocString의 작성 방법은 PEP 257(https://www.python.org/dev/peps/pep-0257)에서 정의되어 있다.
-이용하는 표준 library나 third-party package의 graph나 함수의 사용방법, 인수에 관한 정보를 얻고 싶은 경우에 Docstring을 참조한다.
-Notebook 상에서 스스로 정의한 함수에 Docstring을 기술해 두면 동일한 방법으로 참조할 수 있다.

6)변수의 상태 확인
-변수에 '?'를 붙이면 변수의 상태를 확인할 수 있다.
-다음과 같이 변수 your_name에 문자열 'Jupyter'가 대입되어 있는 것을 확인할 수 있다.
```
      your_name?
      ----------------------------------------------------------------
      Type:        str
      String form: Jupyter
      Length:      7
      Docstring:
      str(object='') -> str
      str(bytes_or_buffer[, encoding[, errors]]) -> str

      Create a new string object from the given object. If encoding or
      errors is specified, then the object must expose a data buffer
      that will be decoded using the given encoding and error handler.
```

```
      Otherwise, returns the result of object.__str__() (if defined)
      or repr(object).
      encoding defaults to sys.getdefaultencoding().
      errors defaults to 'strict'.

  7)Tooltip 표시
    -도움 기능을 이용하는 다른 방법이 있다.
    -대상의 끝부분에 입력 cursor를 놓은 상태에서 Shift+Tab을 누르면 tooltip이 표시된다.
    -'?'을 입력할 필요가 없으므로 빠르게 도움 기능을 이용할 수 있다.
      your_name<-----여기에 Shift+Tab을 누른다.

  8)Code의 자동완성 기능 이용하기
    -Jupyter Notebook의 code cell에서는 입력 중인 code에 대해 자동완성 기능을 이용할 수 있다.
    -Tab key를 누르면 자동완성 기능이 실행된다.
    -자동완성 후보가 다양한 경우에는 목록 형태로 표시되고 하나의 경우에는 바로 입력이 완성된다.
    -예를 들어, 'math.s'까지 입력하고 Tab을 누르면 'math.sin', 'math.sinh', 'math.sqrt'등 3가지가 표시된다.
    -3개의 후보 중에서 선택하는 것도 가능한데, 'math.sq'까지 입력하면 'math.sqrt' 하나만으로 좁혀져 바로 자동완성된다.
    -자동완성 기능은 Python의 예약어나 표준 library 이외에도 사용자가 정의한 등급이나 방식, 변수 등에 대해서도 유효하다.
    -Code 자동완성은 code 입력 속도를 향상시킬 뿐만 아니라 입력 실수를 예방하는 효과도 있다.


2. Jupyter Notebook Interface
  1)Menu
    -Menu에는 File, Edit, View, Insert, Cell, Kernel, Widgets, Help가 있다.
    -File menu
      --New Notebook : 신규로 Notebook을 작성한다. 그 시점에 Kernel을 선택한다.
      --Open : Dashboard view를 신규 tab에 연다.
      --Make a Copy : 현재 Notebook을 복사하여 신규 tab에서 복사한 Notebook을 연다.
      --Rename : Notebook 명칭을 변경한다.
      --Save and Checkpoint : 현재 상태를 file로 저장하고 checkpoint를 작성한다.
      --Revert tyo Checkpoint : 지정한 checkpoint 시점의 상태로 복원한다.
      --Print Preview : 인쇄에 적합한 layout의 Notebook을 연다.
      --Download as : 현재 Notebook을 다양한 형식으로 download할 수 있다.
      --Close and Halt : 현재 Notebook을 종료한다.

    -Edit menu
      --주로 code cell을 편집할 때 사용한다.
      --Cut Cells : Active code cell을 잘라서 내용을 clipboard에 저장한다.
      --Copy Cells : Active code cell의 내용을 clipboard에 저장한다.
      --Paste Cells Above : 복사 또는 자른 code cell을 Active code cell의 상위에 도입한다.
      --Paste Cells Below : 복사 또는 자른 code cell을 Active code cell의 하위에 도입한다.
      --Paste Cells & Replace : 복사한 code cell의 내용으로 Active code cell의 내용을 덮어쓴다.
      --Delete Cells : Active code cell을 삭제한다.
      --Undo Delete Cells : 최근의 Delete Cells 동작을 취소한다.
      --Split Cell : Active code cell의 cursor 행을 경계로 cell을 분할한다.
      --Merge Cell Above : Active code cell을 하나 위에 cell과 결합한다.
      --Merge Cell Below : Active code cell을 하나 밑에 cell과 결합한다.
      --Merge Cell Up : Active code cell의 위치를 하나 위로 이동한다.
```

--Merge Cell Down : Active code cell의 위치를 하나 밑으로 이동한다.
--Edit Notebook Metadata : NoteBook metadata (JSON file)을 편집한다.
--Find And Replace : 검색과 치환을 실행하기 위한 확인 dialog를 연다.
--Cut Cell Attachments : Image 첨부에 관한 기능이다.
--Copy Cell Attachements : Image 첨부에 관한 기능이다.
--Paste Cell Attachments : Image 첨부에 관한 기능이다.
--Insert Image : Markdown mode의 cell에 image를 도입한다.

-Split Cell 이용 예
--Split Cell은 입력 cursor의 위치에서 code를 분할하는 기능이다.
--다음은 함수 hello()의 정의와 실행이 기술된 code로, 하나의 code cell에 입력되어 있다고 할 수 있다.

```
def hello(name):
    print('Hello {0} !!' .format(name))

hello('Jupyter')
```

--hello('Jupyter')의 1행 위에 cursor를 둔 상태로 Edit menu의 'Split Cell'을 click하면 cursor 행을 경계로 code가 2개의 cell로 분할된다.
--import 문장이나 함수의 정의를 독립한 code cell로 만들거나 길어진 code를 분할하기 위해서 이용할 수 있다.
--신규 code cell을 추가해 잘라서 붙여넣기하는 것보다 재빠르게 실행할 수 있어서 편리하다.
--관련 기능으로 'Merge Cell Above'와 'Merge Cell Below'가 있다.
--전자는 선택 중인 code cell과 하나 위의 code cell을 결합해서 하나의 code cell을 만든다.
--후자는 하나 밑의 code cell과 결합한다.

-View menu
--Notebook의 각 part 표시를 새롭게 바꿀 때 이용한다.
--Toggle Header : header를 보여주거나 숨긴다.
--Toggle Toolbar : toolbar를 보여주거나 숨긴다.
--Toggle Line Number : code cell 행 번호를 보여주거나 숨긴다.
--Cell Toolbar : cell toolbar의 유형을 선택한다.

-Toggle Line Numbers
--'Toogle Line Numbers'에서 code cell의 행 번호를 보여주거나 숨길 수 있다.
--행 번호는 code cell마다 1부터 시작된다.

-Cell Toolbar
--Cell Toolbar submenu는 각 code cell의 toolbar 유형을 선택할 수 있다.
--None : 초기 설정 상태이다.
--Edit Metadata : code cell의 metadata를 편집한다.
--Raw Cell Forma : cell type을 Raw NBConvert에 지정한 경우에 이용 가능한 항목이다. 보통은 이용하지 않는다.
--Slideshow : Slide Show mode에 관한 설정 항목을 각 cell에 표시한다.
--Attachements : code cell이 Markdown type인 경우 첨부 가능한 image에 관한 정보를 편집한다.
--Tags : code cell에 대해 tag를 부여할 수 있다.

-Tags
--Code cell의 toolbar 'Tags'를 유효하게 하면 code cell에 대해 tag를 부여할 수 있게 된다.
--Tag는 Notebook의 동작에는 영향이 없다.
--Jupyter Notebook의 주변 tool에서 Notebook을 이용할 때 tag 값을 취득해서 이용할 수 있다.
--예를 들어, Notebook을 HTML이나 다른 format으로 변환하는 tool 'nbconvert'에서는 tag의 내용에

따라 출력하는 template을 구분해서 쓰는 방법이 있다.(http://nbconvert.readthedocs.io/en/latest/customizing.html#Templates-using-cell-tags)

-Insert menu
--code cell을 위, 아래로 삽입할 때 이용한다.
--Insert Cell Above : Active Cell 위에 빈 cell을 삽입한다.
--Insert Cell Below : Active Cell 밑에 빈 cell을 삽입한다.

-Cell menu
--code cell을 실행할 때 사용한다.
--Run Cells : Active Cell을 실행한다.
--Run Cells and Select Below : Active Cell을 실행하고 focus를 하나 밑의 cell로 이동한다.
--Run Cells and Insert Below : Active Cell을 실행하고 하나 밑에 빈 cell을 삽입한다.
--Run All : Notebook 내의 cell을 제일 위에서부터 순서대로 실행한다.
--Run All Above : Notebook 내의 cell을 제일 위에서부터 순서대로, Active Cell을 하나 위의 cell까지 실행한다. Active Cell은 실행되지 않는다.
--Run All Below : Active Cell에서 Notebook 내의 가장 밑까지 cell을 실행한다. Active Cell은 실행되지 않는다.
--Cell Type : cell type을 변경하는 menu이다. Code, MarkDown, Raw NBConvert에서 선택한다.
--Current Outputs : code cell 실행 결과의 표시에 관한 menu이다.
--All Output : Notebook 전체 code cell 실행 결과의 표시에 관한 menu이다.

-Current Outputs / All Output
--Current Outputs과 All Output submenu는 code cell 실행 결과의 표시에 관한 설정이다.
--Current Outputs은 선택되어 있는 code cell의 실행 결과에 대해, All Output은 Notebook 전체 code cell의 실행 결과에 영향을 준다.
--Toggle : 실행 결과를 화면에 보여주거나 숨긴다.
--Toggle Scrolling : 실행 결과가 긴 경우, 초기 설정은 scrollbar와 함께 결과가 출력된다. scrollbar 이용 유무를 바꿀 수 있다.
--Clear : 실행 결과를 삭제한다.

-Kernel menu
--Notebook 상에서 실행된 Pytyon을 실행할 때 이용한다.
--Interrupt : 실행 중인 처리를 중단한다.
--Restart : Notebook을 재실행한다.
--Restart & Clear Output : Notebook을 재실행하고 모든 code cell의 실행 결과를 삭제한다.
--Restart & Run All : Notebook을 재실행하고 모든 code cell을 순서대로 실행한다.
--Reconnect : Kernel에 재접속한다.
--Shutdown : Kernel을 종료한다.
--Change Kernel : Kernel을 변경한다.

-Kernel이란?
--Kernel이라는 것은 OS의 설명에 자주 등장하는 용어로 OS의 기본 기능을 담당하는 SW 부분을 가리킨다.
--Kernel은 program의 상태 관리나 HW의 자원 관리를 담당한다.
--Jupyter Notebook에서의 kernel은 우선 IPython kernel을 가리킨다.
--Jupyter Notebook 단일로는 Web Application의 interface를 사용자에게 제공하는 것에 지나지 않는다.
--Jupyter Notebook의 interface와 program 언어인 Python과의 중간 역할을 IPython kernel이 맡고 있다.
--간단히 정리해보면, Jupyter Notebook 상에 programming 언어의 기능을 제공하는 것이 Kernel이라

고 말할 수 있다.
--Project Jupyter의 공식적인 유지관리를 하고 있는 kernel은 IPython kernel이며, 최소 구성의 Jupyter Notebook에서는 IPhthon만 이용할 수 있다.
--IPython kernel 이외에도 사용자가 다른 community가 공개한 kernel을 이용할 수 있다.
--예를 들어, 인기 언어중 하나인 Ruby용 kernel IRuby(https://github.com/SciRuby/iruby), 분석 용도에 focus를 맞추면서 높은 처리 성능을 자랑하는 Julia 언어(https://julialang.org/)용 kernel IJulia(https://github.com/JuliaLang/IJulia)가 있다.
--Project Jupyter의 GitHub에 있는 Wiki에는 80개 이상의 kernel list가 게재되어 있다.
https://github.com/jupyter/jupyter/wiki/Jupyter-kernels

-Interrupt
--Interrupt는 실행 중인 code cell이 있는 경우 처리를 중단한다.
--구체적으로 Python의 내장 예외 처리인 KeyboardInterrupt가 실행된다.
--아래의 code는 loop마다 5초동안 sleep하는 code로 실행완료까지 시간을 필요로 한다.

```
import time

for i in range(10) :
    print(i)
    time.sleep(5)
```

--code 실행 중 [Kernel] > [Interrupt]를 click하면 처리가 중단된다.
--사용자의 interrupt에 따라 처리가 중단된 것을 나타내는 예외처리 KeyboardInterrupt가 발생한다.

```
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-1-85f8a4efffef> in <module>
     3 for num in range(10):
     4     print(num)
----> 5     time.sleep(5)

KeyboardInterrupt:
```

-Widgets menu
--Notebook 상에서 광범위한 interactive 기능을 추가하는 Widget에 관한 조작을 실행한다.

-Help menu
--Jupyter Notebook을 이용하기 위한 각종 문서의 link집이 제공되어 있다.
--Keyboard shortcut의 확인과 변경에 관한 menu도 있다.

2)Toolbar
-Menu의 하단에 배치되어 있는 것이 toolbar이다.
-toolbar는 이미 설명한 Notebook menu의 단축키로서 동작한다.
-Menu에서 이루어지는 조작 중에서 자주 이용하는 기능들이 toolbar에 등록되어 있다.
-다음은 순서대로 설명한 것이다.
--저장 : Notebook file을 덮어쓰고 저장한다.
--Code cell의 추가 : 선택 중인 code cell의 하부에 신규 code cell을 도입한다.
--잘라내기 : Active 상태인 code cell을 잘라낸다.
--복사 : Active 상태인 code cell을 복사한다.
--붙여넣기 : 잘라내기 또는 복사한 code cell을 선택 중인 code cell의 하부에 삽입한다.
--위 화살표 : Active 상태인 code cell을 하나 위로 이동시킨다.
--아래 화살표 : Active 상태인 code cell을 하나 아래로 이동시킨다.
--실행 : Active 상태인 code cell을 실행한다. keyboard shortcut s+q와 같은 효과이다.

--중단 : code cell이 실행 중인 경우 실행을 중지한다.
--재실행 : kernel을 재실행한다. 재실행 전에 확인 대화상자가 표시된다.
--Code Cell Type : code cell type을 지정한다.
--명령 Palett : Notebook에 대해 실시 가능한 여러 가지의 명령을 확인하면서 실행할 수 있다.

-Code Cell Type
--Code : code를 실행하는 type이다.
--Markdown : Markdown을 기술할 수 있는 type이다.
--Raw NBConvert : Notebook format 변경tool 'nbconvert'를 실행할 때 출력하는 내용을 기술할 수 있는 type이다.
--Heading : code cell을 Markdown type으로 변경한 후, code cell의 선두에 표제어를 나타내는 #을 도입한다.

3. Jupyter Notebook의 keyboard shortcut

1)Edit mode와 command mode

-Notebook의 단축키는 편집 mode용과 명령 mode용이 각각 준비되어 있다.
-편집 mode는 어느 code cell이 편집 중인 상태를 가리키며 단축key는 편집 중인 code cell에 대해 작용한다.
-Code cell의 테두리가 녹색이 된 상태가 편집 mode이다.
-Code cell이 편집중인 아닌 상태를 명령 mode라고 한다.
-명령 mode에서는 Active(편집 중이 아닌 점에 주의) code cell에 파란색 테두리가 생긴다.
-명령 mode에서 편집 mode로의 이행은 Enter key, 그 반대는 Esc key를 누른다.

2)단축key 목록 표시하기

-Notebook의 keyboard shortcut은 [Help] > [Keyboard Shortcuts]로 목록이 표시된다.
-또한 명령 mode에서 H를 눌러 단축key 목록을 표시할 수 있다.

3)편리한 단축key

-편집 mode
--Shift + Enter : Active Code Cell을 실행하고 하나 밑의 cell을 선택한다. 마지막 code cell인 경우 실행 후에 code cell을 도입한다. 이 단축key는 명령 mode에서도 유효하다.
--Ctrl + Enter : Active Code Cell을 실행한다. 이 단축key는 명령 mode에서도 유효하다.
--Alt + Enter : Active Code Cell을 실행하고 하나 밑의 code cell을 도입한다. 이 단축key는 명령 mode에서도 유효하다.
--Ctrl + Z : Undo이다. 직전의 편집 내용을 취소한다.
--Ctrl + Shift + Z : Redo이다. Undo로 취소한 내용을 다시 실행한다.
--Tab : 입력 중인 code의 보완처리를 실행한다. 행의 선두인 경우 indent를 도입한다.
--Shift + Tab : tooltip을 표시한다. method나 함수의 정보를 확인하기 위해 이용할 수 있다. 변수에 사용하면 변수의 상태를 확인할 수 있다.
--Esc : 편집 mode에서 명령 mode로 바꾼다.

-명령 mode
--↑: 하나 위의 cell을 선택
--↓: 하나 밑의 cell을 선택
--A : Active Code Cell의 위에 신규 cell을 도입
--B : Active Code Cell의 밑에 신규 cell을 도입
--Ctrl + S : Notebook file을 저장
--Shift + L : 모든 code cell의 줄 번호를 보여주거나 숨긴다.
--D, D : (D뒤에 한번 더 입력)선택 중인 code cell을 삭제한다.
--Z : 삭제한 code cell을 하나 복원한다.
--Y : Active Code Cell을 code type으로 한다. code를 기술하는 type이다.

--M : Active Code Cell을 Markdown type으로 한다.
--O, O : (O 뒤에 한번 더 입력)Kernel을 restart한다. restart하기 전에 확인 dialog가 표시된다.
--P : 명령 Palette를 연다. 명령 palette에서는 Notebook에 대한 다양한 조작을 실행할 수 있다.
--H : 단축key 목록을 표시한다. 표시중에 Enter로 비표시할 수 있다.
--Enter : 선택중인 code cell을 명령 mode에서 편집 mode로 바꾼다.

4. Jupyter Notebook의 저장과 재사용
1)Python 표준의 interative mode나 IPython 환경에서는 사용자가 입력 또는 실행한 code를 저장하고 재사용하는 데에 요령이 필요하다.
2)하지만 Jupyter Notebook에서는 text editor를 이용하는 느낌으로 Notebook의 내용을 저장할 수 있다.
3)Notebook file(.ipynb file) 구조
-Notebook을 새로 작성하면 .ipynb라는 확장자 file이 만들어진다.
-처음 Notebook을 작성하는 경우에는 Untitled.ipynb라는 file이 만들어진다.
-이것이 Notebook의 실체이며, Notebook file이라고 부른다.
-또한 .ipynb라는 확장자는 Jupyter Notebook이 IPython Notebook이라는 제품명이었을 때의 흔적이다.
-Notebook file에는 Notebook의 상태나 입력한 내용이 JSON 형식으로 기록되어 있다.
-JSON은 text editor로 열리거나 편집할 수 있는 것을 의미한다.
-Source code 이외에 출력 결과나 실행 횟수 등의 정보가 기술되어 있어 Notebook file의 정보를 바탕으로 Notebook이 구성되어 있는 것을 알 수 있다.

4)File 저장하기
-Notebook을 새로 작성한 시점에 Notebook file이 하나 만들어진다.
-또한 Notebook의 제목을 변경하면 file명도 같이 변경된다.
-File명은 Notebook Home화면에서 확인할 수 있다.
-Notebook file을 덮어쓰기 저장하는 것은 [File] > [Save and Checkpoint]를 선택 또는 Ctrl + S이다.
-Text Editor에서 문서나 Source code를 기술하는 경우와 같이 꾸준히 저장한다.

5)Auto Save 기능
-Notebook은 120초마 Notebook의 상태를 덮어쓰기 저장하는 Auto Save 기능이 기본으로 설정되어 있다.
-File의 저장 상황은 header에 표시되어 있다.
-Auto Save에 의해 저장된 경우, 'autosaved'라고 표시된다.
-Auto Save 기능은 무효로 하거나 저장 간격을 지정하는 것이 가능하다.

6)Checkpoint
-Code의 입력이나 실행 유무와 같은 Notebook의 어느 시점에서의 논리적인 중단점이며, 그 때의 처리 상태를 완전하게 보존하고 후에 그 시점에서 처리를 재개 할 수 있도록 한 point를 말한다.
-해당 관리 구조는 .ipynb_checkpoints/ directory 안에 .ipynb file을 일시적으로 저장하는 형태로 구현되어 있다.
-작성하는 방법은 명시적으로 Notebook을 저장한다.
-Checkpoint가 추가됨과 동시에 Notebook 자체도 저장된다.
-Auto Save 기능이 실행되는 경우에는 Notebook에 file이 저장되지만 checkpoint는 추가되지 않는다.
-즉, Auto Save 기능이 실행된 후 복원하는 경우 마지막으로 Checkpoint가 추가된 시점의 상태로 복원되므로 주의가 필요하다.
-Checkpoint의 복원은 [File] menu > [Revert to Checkpoint]를 click해서 표시된 시점을 선택한다.

7)File 불러와서 다시 이용하기
-저장한 file을 불러와서 다시 이용할 수 있다.
-[File] > [Open]
-기술한 code의 내용뿐만 아니라 code cell을 실행할지, 하지 않을지 실행한 결과의 출력 내용을 포함하여 저장한 상태에서 재개할 수 있다.

5. Markdown과 수식의 이용
  1)Jupyter Notebook에서는 Python code뿐만 아니라 Markdown이나 수식의 기술, 표시가 가능하다.
  2)Markdown이란
    -Markdown은 문서 기술에 대한 서식을 정한 '경량 Markup 언어'의 하나이다.
    -https://daringfireball.net/projects/markdown
    -경량 Markup 언어에는 여러 가지 종류가 존재한다.
    -Wiki system 'MediaWiki'의 'Wiki markup(https://www.mediawiki.org/wiki/MediaWiki)' 혹은 BBCode 등이 있다.
    -Python 관련 문서 작성시에는 자주 'reStructuredText(reST)라는 경량 markup언어가 채용된다.
    -'Asciidoc(http://asciidoctor.org/docs/what-is-asciidoc/)'은 기술 문서의 집필에 사용되는 경우도 있다.
    -여러 가지 종류가 존재하는 경량 markup 언어인 Markdown은 그 중에서도 특히 서식이 간단하고 알기 쉬워서 많은 tool이나 service에 채용되었다.
    -간단하다는 것은 기능이 적다는 것이다.
    -그 때문에 Markdown에 기능을 추가한 확장 기법이 다수 고안되었다.
    -목록이나 정의 list등의 기능을 추가한 'PHP Markdown Extra(https://michelf.ca/projects/php-markdown/extra/)'와 문서 format의 상호 변환 tool인 'pandoc(http://pandoc.org/)'에 실제 설치되어 있는 'pandoc 확장 Markdown'등이 그 예이다.
    -확장 기법의 난립이 문제시 된 적도 있고 기법의 재통일을 노린 'CommmonMark(http://commonmark.org/)'라는 기법도 새로 만들어졌다.

  3)Jupyter Notebook에서 채용된 GitHub Flavored Markdown
    -Jupyter Notebook은 수 많은 Markdown 확장 기법의 하나로 GitHub Flavored Markdown(이하 GFM)을 채용하고 있다.
    -GFM은 명칭대로 GitHub가 주체가 되어 사양을 책정한 기법이다.
    -Issue나 README file등 GitHub 상에 있는 기능이 GFM을 support하고 있다.
    -GFM은 사실상 표준이라고 할 수 있다.
    -Markdown 및 GFM의 상세한 내용은 GitHub의 공개 문서 'Mastering Markdown(https://guides.githu.bom/features/mastering-markdown/#GitHub-flavored-markdown)'을 참조한다.

  4)Markdown 기술하기
    -Jupyter Notebook에서 Markdown을 이용해 보자.
    -초기 설정에서 code cell은 code type으로 되어 있다.
    -'Markdown'으로 변경하여 대상 code cell을 Markdown type으로 변경할 수 있다.

```
# 이것은 표제이다.

이것은 단락이다.

- 조항 쓰기도
- 이렇게 표현할 수 있다.

테이블도 기술할 수 있다.

Package | version
--------|-----------
pandas  | 0.19.2
matplotlib | 2.0.0
```

411
412 -기술한 Markdown을 Notebook에 HTML로 표시할 때는 code cell을 실행한다.
413 -실행 방법은 Python의 code를 사용한 경우와 같다.
414 -실행 결과, HTML 표시로 교체된다.
415
416 5)Notebook 설명 comment로 Markdown 이용하기
417 -Markdown은 Python code와 함께 사용하면 효과적이다.
418 ## 표준 Library 이용 방법
419
420 이 Notebook에서는 Python의 표준 Library의 이용 방법을 설명한다.
421
422 표준 Library를 이용할 때는 'import'문을 기술한다.
423 ---
424 In [ ] : import math
425 ---
426 'import'한 'math' module을 이용한다.
427 ---
428 In [ ] : print(math.sqrt(4))
429
430 -위 code를 실행한다.
431 -또한, Markdown type에서는 Python code를 실행할 수 없다.
432 -Code cell이 어느 type인지 확인한다.
433 -Source code 안의 comment로 설명하는 것보다 Markdown으로 기술하는 것이 보다 풍부한 표현의 문서를 Notebook에 더할 수 있다.
434 -Markdown으로 설명이 더해진 Notebook을 기술 설명용의 자료로 작성, 공개되는 경우도 많다.
435
436 6)수식 기술하기
437 -Jupyter Notebook은 수식을 표현할 수 있다.
438 -수식의 rendering에는 MathJax(https://www.mathjax.org/)라는 JavaScript library를 이용하고 있다.
439 -MathJax가 support하고 있는 형식은 여러 개가 있는데, 여기에서는 LaTeX 서식의 수식을 기술하는 방법을 설명한다.
440 -수식을 기술할 때에는 Markdown을 사용하는 경우와 같이 code cell을 Markdown type으로 설정한다.
441
442 ## 수식 표시
443
444 이 Code cell에는 다음과 같이 서식을 표현할 수 있다.
445
446 \begin{align}
447 \sqrt{2x-1}+(3+x)^3$
448 \end{align}
449
450 $$ E = mc ^ 2 $$
451
452 인라인에서도 오른쪽에 쓰여진 것처럼 $\sqrt{a^2+b^2}$ 기술할 수 있다.
453
454 7)Image 첨부하기
455 -Markdown type code cell에 image를 drag & drop하여 image를 첨부할 수 있다.
456
457 ## Jupyter 기동 후 Home 화면이다.
458

459 ![jupyter-home.png](attachment:jupyter-home.png) <----아무 image를 drag & drop 한다.
460
461 -Code cell을 실행하면 image가 표시된다.
462 -Code cell에 첨부한 image는 첨부 후에 표시된 text '![jupyter-home.png](attachment:jupyter-home.png)'을 삭제해도 표시 상 보이지 않게 되는 것일 뿐, code cell 내에서는 image data가 남게 된다.
463 -Image를 삭제할 때에는 [View] > [Cell Toolbar] > [Attachements]를 click해서 code cell 오른쪽 상단의 [Edit Attachements]를 click하고 휴지통 표시를 click한 후에 [Apply]를 click한다.
464
465
466 6. Magic 명령어 이용
467 1)Magic Commands는 IPython의 kernel에서 제공되는 구조이다.
468 2)Matplotlib의 graph 작성 방법을 지정하는 것과 같이 Notebook의 동작과 관련 있는 기능 이외에 directory 이동하기, file 일람 표시하기 등의 utility 기능을 제공한다.
469 3)Shell(명령 prompt)의 이용방법
470 -Magin 명령에 대해서 설명하기 전에 Notebook 상에서 OS의 shell을 실행하는 방법을 소개한다.
471 -제일 앞에 '!'를 붙이고 계속해서 명령을 입력한다.
472 -예를 들어, macOS의 환경에서 UNIX 명령 cat을 이용해서 file의 내용을 출력할 때 다음과 같이 입력한다.
473
474 !cat /etc/hosts
475
476 -File 조작이나 확인 등의 처리를 terminal의 기동 없이 Notebook 상에서 처리할 수 있으므로 편리하다.
477 -어떤 명령이 실행 가능한 지는 OS에 따라 다르다.
478 -Windows에서는 cat 명령어가 없기 때문에 실행할 수 없다.
479 -대신 Windows에서는 type명령을 사용한다.
480
481 !type C:\Windows\System32\drivers\etc\hosts
482
483 4)Magic 명령의 기본
484 -Magic 명령은 선두에 %를 입력한 후에 계속해서 명령을 입력하는 서식이다.
485 -현재의 directory를 출력하는 magic 명령은 아래와 같다.
486
487 %pwd
488
489 -Magic 명령을 입력한 cell을 실행하면 아래와 같은 결과가 표시된다.
490
491 Out [8] : 'C:\\Users\\Instructor'
492
493 -Magic 명령의 %pwd는 shell을 이용한 '!pwd'와 다른 점이 없는 것처럼 보인다.
494 -하지만 '!'를 이용한 명령 실행이 OS의 shell에 의존하는 것에 비해 magic 명령은 IPython kernel에서 제공되는 기능에 의존한다.
495 -즉, Windows에서는 !pwd는 실행되지 않지만, %pwd는 실행된다.
496 -실제로 %pwd는 내부적으로 실행되고 있다.
497
498 import os
499
500 os.getcwd()
501
502 -표준 library os module의 getcwd() 함수는 현재 directory의 정보를 알려주는 함수이다.
503 -그 때문에 %pwd 명령에서 directory의 정보를 취득할 수 있다.

-Magic 명령이 어떤 형태의 값을 알려주는 경우 Python 함수를 호출한 경우와 같이 return 값을 변수에 대입할 수 있다.

```
working_dir = %pwd
working_dir
```

-working_dir에는 현재 directory 경로('C:\\Users\\Instructor')가 문자열로 대입된다.

5)편리한 magic 명령

-%time

--Python의 실행 시간을 측정하는 magic 명령이다.
--예를 들어 다음과 같이 sum()의 실행 속도를 측정하는 예를 보자.
--측정 대상이 되는 code 앞에 magic 명령어 %time을 입력하고 cell을 실행한다.
--그러면 sum() 함수의 실행 결과 앞에 실행 시간에 관한 정보가 표시된다.

```
%time sum(range(100000))
--------------------------------------------------
Wall time: 69 ms
499999500000
```

--몇 개의 수치가 출력되는데 program 시작에서 종료까지 걸린 시간(Wall time)에 주목하면 좋다.
--CPU times: 행에 표시되어 있는 user는 'user CPU time'으로 program 자체의 실행에 필요한 시간을 나타낸다.
--sys는 'system CPU time'을 나타내고 OS system 요구에 필요한 시간에 해당한다.
--File의 읽고 쓰기에 필요한 시간이 대표적인 system CPU time이다.

-%timeit

--아래는 %time과 비슷한 명령의 실행 예이다.

```
%timeit sum(range(100000))
----------------------------------------
53.7 ms ± 7.94 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

--이 명령어는 여러 번 시행한 결과의 측정 값을 요약해서 나타낸 점이 %time과 다르다.
--위의 결과를 보면 10번 되풀이한 처리를 7번 시행한 경우의 최소값 53.7 ms ± 7.94ms이 출력되어 있다.
--Loop 횟수 및 시행횟수는 아래와 같이 옵션으로 지정 가능하다.

```
# Loop 횟수 : 2,000회, 시행 횟수 : 5회로 지정
%timeit -n 2000 -r 5 sum(range(10000))
----------------------------------------------------------
375 µs ± 13.9 µs per loop (mean ± std. dev. of 5 runs, 2000 loops each)
```

--여러 행의 Python code에 대해서 %timeit을 사용하는 경우 아래와 같이 magic 명령을 code cell 전체에 적용한다.

```
%%timeit -n 1000 -r 3

for i in range(1000) :
  i * 2
------------------------------
100 µs ± 27.7 µs per loop (mean ± std. dev. of 3 runs, 1000 loops each)
```

552
553 --Code cell 전체에 적용할 때는 선두의 %를 2개 연속해서 %%라고 한다.
554 --이것은 'Cell magics'이라고 불린다.
555 --지금까지 설명한 %가 하나인 경우에는 'Line magics'라고 불려고 구별된다.
556 --%timeit는 하나의 명령에 적용한다.
557 --반면, %%timeit에서는 입력한 cell의 code 전체가 대상이 된다.
558
559 -%history
560 --Code cell의 실행 이력을 목록으로 취득하는 magic 명령이다.
561 --과거 이력을 알고 싶은 필요성이 적을지도 모르지만, Notebook에서는 terminal과 달리 실행한 명령을 그대로 표시하고 있다.
562 --다만, 단일 code cell에서 parameter를 변경해서 반복 실행하는 경우가 있어 직전에 실행한 명령을 확인하고 싶을 때 이용할 수 있다.
563 --Code cell에 'sum(range(100))'이라고 입력해 실행하고 code 를 'sum(range(1000))'으로 변경해 다시 한 번 실행해 둔다.
564 --계속해서 다음 code cell에서 다음과 같이 %history 명령을 실행하면 최근에 실행한 명령이 새로운 순서로 다음과 같이 취득된다.
565
566 ```
567 # 최근 3개의 이력 취득
568 %history -l 3
569 ------------------------------
570 for i in range(1000):
571   i * 2
572 sum(range(100))
573 sum(range(1000))
574 ```
575 -%ls
576 --UNIX의 ls와 같은 동작을 하는 현재 directory의 file 목록을 취득하는 magic 명령이다.
577 --Data 분석 process에서는 이용하는 file의 file명을 code 안에 입력하는 경우가 많다.
578 --%ls 명령은 file명 확인을 위해 이용할 수 있다.
579 --File 목록의 취득 결과는 아래와 같다.
580
581 ```
582 Volume in drive C has no label.
583 Volume Serial Number is E663-E346
584
585 Directory of C:\Users\Instructor
586
587 01/09/2019 11:18 PM   <DIR>   .
588 01/09/2019 11:18 PM   <DIR>   ..
589 01/08/2019 01:02 AM   <DIR>   .anaconda
590 01/06/2019 11:57 PM   <DIR>   .atom
591 12/07/2018 12:19 AM   4,944 .bash_history
592 10/06/2018 11:36 PM   <DIR>   .codeintel
593 ....
594 ....
595 ```
596 --Magic 명령의 기본에서 설명한 것처럼 !ls와 $ls는 다르다.
597 --Magic 명령 %ls는 OS의 종별을 판단해서 내부에서 실행하는 명령을 적절하게 가려 쓴다.
598 --macOS의 경우에는 ls 명령이 이용되고 Windows에서는 dir 명령이 이용된다.

-%autosave

599 --Notebook에서는 Notebook의 상태를 정기적으로 file에 기록하는 Auto Save 기능이 있다.
600 --Auto Save는 초기 설정에서는 120초에 한 번의 빈도로 실행된다.
601 --%autosave 명령을 이용하면 Auto Save의 빈도를 변경할 수 있다.
602 --Auto Save의 빈도가 너무 높거나 낮다고 느끼는 경우에는 변경한다.
603 --값을 0으로 지정하면 Auto Save를 무효로 할 수 있다.
604
605 # Auto Save를 60초에 한번 실행하기
606 %autosave 60
607 ---
608 Autosaving every 60 seconds
609
610 -%matplotlib
611 --이것은 graph 작성 package인 'Matplotlib'에 대한 설정을 실행하는 magic 명령이다.
612 --이것은 아래와 같이 graph를 작성하는 code보다 먼저 지정한다.
613 --가장 처음 code cell에 입력하는 것이 좋다.
614
615 %matplotlib inline
616
617 import numy as np
618 import matplotlib.pyplot as plt
619
620 x = np.arange(0, 10, 0.2)
621 y = np.sin(x)
622
623 fig = plt.figure()
624 ax = fig.add_subplot(111)
625
626 ax.plot(x, y)
627 plt.show
628
629 -%matplotlib inline
630 --inline을 설정한 경우 code cell의 바로 아래에 graph가 출력된다.
631
632 -%matplotlib tk
633 --tk를 설정한 경우 별도 창에 interactive한 graph가 출력된다.
634 --Tk는 GUI application을 개발하기 위한 도구이다.
635 --IPython kernel에서는 Tk의 Python interface
tkinter(https://docs.python.org/3/library/tkinter.html)가 이용되고 있다.
636 --별도 창에서 표시된 graph는 확대나 축소가 가능하다.
637 --그림을 저장하는 menu도 제공하고 있다.
638
639 %matplotlib tk
640
641 import numy as np
642 import matplotlib.pyplot as plt
643
644 x = np.arange(0, 10, 0.2)
645 y = np.sin(x)
646
647 fig = plt.figure()
648 ax = fig.add_subplot(111)

```
        ax.plot(x, y)
        plt.show
```

-%matplotlib notebook
--notebook을 설정한 경우 inline을 설정한 경우와 마찬가지로 code cell 바로 아래에 graph가 출력된다.
--단순히 graph를 출력하였던 inline과 다르게 동적으로 graph를 움직이거나 저장할 수 있는 interactive graph이다.
--또한, 한 번 설정한 출력 방법을 변경할 때에는 kernel의 재시동을 실시한다.
--Kernel 재시동은 Notebook의 menu에서 실행할 수 있다.

7. Jupyter Notebook 공유 방법
1)누군가와 자신의 Notebook을 공유하고 싶을 경우 어떻게 하면 될까?
2)Notebook file의 실체는 text file이다.
3)따라서 받는 쪽 computer에 Jupyter Notebook 환경이 있는 경우, Notebook file을 넘겨주면 Notebook을 재사용하는 것이 가능하다.
4)Jupyter Notebook이 도입되어 있지 않은 환경이나 internet에 Notebook을 공개하고 싶을 경우 다른 여러 가지 방법이 있다.
5)여기에서는 GitHub와 nbviewer를 이용하는 방법을 소개한다.

6)GitHub에 Notebook 공개하기
-GitHub(https://github.com)는 분산형 Version 관리 System(Distributed Version Control Systems)인 'Git'를 base로 한 SW Source code를 관리하기 위한 service이다.
-개인이나 기업이 보유하는 source code 관리 외에 OpenSource SW의 source code 관리와 공개에도 이용되고 있다.
-GitHub는 Notebook file 표시를 기본으로 지원하고 있다.
-그 때문에 gitHub repository에 push하는 것만으로 Notebook 내용을 공유할 수 있다.
-Code cell의 출력 내용도 포함해서 공유하고 싶은 경우, 한 번 code cell을 실행한 후에 Notebook을 저장하고 GitHub에 배치할 필요가 있는 점에 주의한다.
-실행되지 않은 code cell의 내용은 출력 결과에 포함되어 있지 않다.
-Notebook code cell의 In [] : 및 Out []:의 번호는 cell이 실행될 때마다 수치가 합산된다.
-여러 번 실행하거나 거슬러 올라가 실행하는 것에 따라 큰 수치가 되거나 code cell의 순번과 번호의 순번이 일치하지 않는 상태가 되는 경우가 있다.
-동작상의 문제는 없지만 번호를 정리하고 싶을 때는 Kernel을 재실행한다.
-Kernel 재실행과 code cell의 재실행에는 [Kernel] > [Restart & Run All]을 이용한다.
-Code cell의 배치순서대로 번호가 1부터 numbering 되고, 전체 code cell이 한번씩 실행된다.
-Notebook 전체를 재실행하는 것은 code의 과실을 찾는 유효한 방법이기 때문에 Notebook의 공유 전에 'Restart & Run All'을 실행해 두면 좋다.

7)nbviewer
-nbviewer는 Notebook file을 정적인 HTML로서 출력하기 위한 SW이다.
-OSS로서의 Project Jupyter 산하에서 관리되고 있다.
-https://github.com/jupyter/nbviewer
-nbviewer의 환경을 독자적으로 구축하는 것도 가능하지만 service되고 있는 'nbviewer.jypyter.org'을 이용하는 것이 편리하다.
-https://nbviewer.jupyter.org/
-nbviewer는 GitHub 상의 Notebook file URL을 지정하면 Notebook의 내용을 HTML로 출력해 주는 service이다.