

Python Tutorial 3

lambda, function, class

Haesun Park, haesunrpark@gmail.com,

1. lambda
2. function
3. decorator
4. class
5. design pattern
6. jupyter notebook
7. example

lambda

람다(lambda) 함수 정의

- 보통 익명 함수(Anonymous function)으로도 불리우며 한 곳에서만 사용되어 이름이 불필요하고 비교적 간단한 함수
- 익명함수를 변수에 할당하여 파라메타로 전달 (파이썬은 함수이름을 파라메타로 전달 가능합니다)
- *lambda 파라메타: 표현식*

Perl:

```
(sub { return $x*$x })->(10);
```

Javascript:

```
(function(x){return x*x;})(10)
```

PHP:

```
array_map(function($x) { return  
$x*$x; }, [10])
```

Python:

```
(lambda x: x*x)(10)
```

맵(map)과 함께 사용하기

- 맵(map) 함수를 사용하여 리스트 컴프리헨션과 유사한 작업을 수행할 수 있습니다.
- *map(함수, 스트링/리스트/튜플/딕셔너리/셋)*
- 맵은 함수의 리턴 값을 모아서 리스트로 만들어 줍니다.

```
>>> [x*x for x in range(5)]  
[0, 1, 4, 9, 16]
```

```
>>> map(lambda x: x*x, range(5))  
<map at 0x106da4128>
```

```
>>> list(map(lambda x: x*x, range(5)))  
[0, 1, 4, 9, 16]
```

```
>>> sqr = lambda x: x*x  
>>> list(map(sqr, range(5)))  
[0, 1, 4, 9, 16]
```

```
>>> def sqr(x):  
    return x*x  
>>> list(map(sqr, range(5)))  
[0, 1, 4, 9, 16]
```

필터(filter)와 함께 사용하기

- `filter(함수, 스트링/리스트/튜플/딕셔너리/셋)`
- 필터는 함수의 리턴 값이 참인 것만 리스트로 만들어 줍니다.

```
>>> def odd(p):  
    return [i for i in p if i%2]  
>>> odd(range(10))  
[1, 3, 5, 7, 9]
```

```
>>> def odd(k):  
    return k % 2  
>>> list(filter(odd, range(10)))  
[1, 3, 5, 7, 9]
```

```
>>> list(filter(lambda x: x%2, range(10)))  
[1, 3, 5, 7, 9]
```

0	1	2	3	4	5	6	7	8	9	
0	1	0	1	0	1	0	1	0	1	← 필터 함수의 결과 값
[1,		3,		5,		7,		9]	← 필터 함수의 결과가 참인 원소로만 구성함

리듀스(reduce)와 함께 사용하기

- *reduce*(함수, 스트링/리스트/튜플/딕셔너리/셋)
- 리듀스는 파라메타의 값들을 모두 누적하여 적용합니다.

```
>>> from functools import reduce  
>>> reduce(lambda x, y: x+y, range(10))  
45
```

(((((1+2)+3)+4)+5)+6)+7)+8)+9)

```
>>> reduce(lambda x, y: x+y, 'abcde')  
'abcde'
```

((('a'+b')+c')+d')+e')

```
>>> reduce(lambda x, y: y+x, 'abcde')  
'edcba'
```

('e'+('d'+('c'+('b'+a'))))

함수 포인터 처럼 사용하기

- 두개의 람다 함수를 리스트에 할당하고 선택적으로 사용할 수 있습니다.
- 함수명을 리스트에 할당하여 사용하는 것과 동일합니다.

```
>>> func_choice = [lambda x, y: x**y,  
lambda x, y: x/y]  
>>> func_choice[0].__class__  
function  
  
>>> val1 = 10  
>>> val2 = 2  
>>> condition = 1  
>>> func_choice[condition](val1, val2)  
5.0  
>>> condition = 0  
>>> func_choice[condition](val1, val2)  
100
```

```
>>> def f1(x, y): return x**y  
>>> def f2(x, y): return x/y  
>>> func_choice = [f1, f2]  
  
>>> val1 = 10  
>>> val2 = 2  
>>> condition = 1  
>>> func_choice[condition](val1, val2)  
5.0  
>>> condition = 0  
>>> func_choice[condition](val1, val2)  
100
```


function

함수의 정의

- 파이썬에서 함수는 오브젝트 중의 하나입니다.
- 입력값을 받아 선언된 함수 내부의 코드를 실행하고 하나의 리턴값을 되돌립니다.
- return 문에서 리턴값을 여러개 나열할 경우 호출자에게 튜플로 전달됩니다.
- *def 함수이름(파라메타):*
 파이썬 문장

```
>>> def factorial(n):  
    sum = 1  
    for i in range(n, 1, -1):  
        sum *= i  
    return sum
```

```
>>> factorial(5)  
120
```

파라메타

인자

위치 인자

- 함수를 호출할 때 나열된 파라메타 순서대로 인자를 할당합니다.
- return 문은 함수 내부 어디에서나 사용될 수 있습니다.

```
>>> def factorial(n, stop):  
    if n == 0:  
        return n, 1  
    elif n < 0:  
        return n, None  
    sum = 1  
    for i in range(n, stop, -1):  
        sum *= i  
    return n, sum
```

```
>>> factorial(5, 1)  
(5, 120)  
>>> factorial(0, 5)  
(0, 1)  
>>> factorial(-5, 1)  
(-5, None)
```

키워드 인자

- 함수를 호출할 때 지정된 키워드 인자를 파라메타에 할당합니다.
- 키워드 인자와 위치 인자를 혼용할 경우에는 위치 인자가 앞서 나열되어야 합니다.

```
>>> def factorial(n, stop):  
    if n == 0:  
        return n, 1  
    elif n < 0:  
        return n, None  
    sum = 1  
    for i in range(n, stop, -1):  
        sum *= i  
    return n, sum
```

```
>>> factorial(n=5, stop=1)  
(5, 120)  
>>> factorial(stop=1, n=5)  
(5, 120)  
>>> factorial(stop=1, 5)  
SyntaxError  
>>> factorial(5, stop=1)  
(5, 120)
```

인자 기본값

- 인자가 제공되지 않을 때 파라메타에 지정된 기본값이 사용됩니다.
- 기본값이 제공되는 파라메타는 함수 선언 맨 뒤에 와야 합니다.

```
>>> def factorial(n, stop=1):  
    if n == 0:  
        return n, 1  
    elif n < 0:  
        return n, None  
    sum = 1  
    for i in range(n, stop, -1):  
        sum *= i  
    return n, sum
```

```
>>> factorial(n=5, stop=1)  
(5, 120)  
>>> factorial(n=5)  
(5, 120)  
>>> factorial(5)  
(5, 120)
```

```
>>> def factorial(n=5, stop):  
    pass  
SyntaxError
```

재귀 함수

- 재귀 함수를 사용할 때는 리턴값의 갯수에 주의해야 합니다.
- 종료 조건을 반드시 두어야 합니다.

```
>>> def factorial(n, stop=1):  
    if n == 0:  
        return n, 1  
    elif n < 0:  
        return n, None  
    sum = 1  
    for i in range(n, stop, -1):  
        sum *= i  
    return n, sum
```

```
>>> def factorial(n, stop=1):  
    if n < stop or n <= 0:  
        return 1  
    return n, n*factorial(n-1, stop)
```

```
>>> factorial(5, 4)  
(5, (4, 4, 4, 4, 4, 4, 4, 4, 4, 4))
```

```
5, 5 * factorial(4, 4)
```

```
5, 5 * (4, 4)
```

```
5, (4, 4, 4, 4, 4, 4, 4, 4, 4, 4)
```

```
>>> def factorial(n, stop=1):  
    if n < stop or n <= 0:  
        return 1  
    return n * factorial(n-1, stop)
```

익명의 위치 인자

- 함수 선언시 *(asterisk)가 인자 이름 앞에 있으면 여러개의 위치 인자에 대응합니다.
- 필수적이지 않은 인자들을 관리하는 도구로 많이 사용됩니다.
- 관습적으로 args 명칭을 사용합니다.
- 익명의 위치 인자는 튜플로 전달 됩니다.

```
>>> def menu(soup, main, *args):  
    print("Soup is %s" % soup)  
    print("Main is %s" % main)  
    for other in args:  
        print("Other is %s" % str(other))
```

```
>>> menu('mushroom', 'steak', 'wine', 100)  
Soup is mushroom  
Main is steak  
Other is wine  
Other is 100
```

```
>>> menu('egg', 'wine', 100)  
Soup is mushroom  
Main is wine  
Other is 100
```

익명의 키워드 인자

- 함수 선언시 `**`(asterisk) 두개가 인자 이름 앞에 있으면 여러개의 키워드 인자에 대응합니다.
- 관습적으로 `kwargs` 명칭을 사용합니다.
- 익명의 위치 인자는 딕셔너리로 전달 됩니다.

```
>>> def menu(*args, **kwargs):  
    for other in args:  
        print("Order is %s" % str(other))  
    for k in kwargs:  
        print("Menu is %s: %s" % (k, kwargs[k]))
```

```
>>> menu('mushroom', drink='wine', tip=100)  
Order is mushroom  
Menu is drink: wine  
Menu is tip: 100
```


변수의 범위-1

- 함수내 변수이름은 지역변수를 먼저 참조하고 전역변수를 찾습니다.
- 지역변수는 전역변수의 값에 영향을 미치지 않습니다.

```
>>> a = 'dog'
```

```
>>> id(a)
```

```
4410123464
```

```
>>> def f1():
```

```
    print(id(a))
```

전역변수 a

```
>>> f1()
```

```
4410123464
```

dog

```
>>> a = 'dog'
```

```
>>> id(a)
```

```
4410123464
```

```
>>> def f2():
```

```
    a = 'cat'
```

```
    print(id(a))
```

지역변수 a

```
>>> f2()
```

```
4325227968
```

```
>>> id(a)
```

전역변수 a

```
4410123464
```

dog

cat

변수의 범위-2

- 지역변수가 생성이 될 때 참조가 먼저 일어나면 에러가 발생합니다.
- global 키워드로 전역변수를 명시적으로 참조할 수 있습니다.
- 전역변수를 수정할 필요가 있을 때는 함수 파라메타나 리턴값으로 처리하는 것이 좋습니다.

```
>>> a = 'dog'
```

```
>>> id(a)
```

```
4410123464
```

```
>>> def f3():
```

```
    print(id(a))
```

```
    a = 'cat'
```

```
>>> f3()
```

```
UnboundLocalError
```

dog

cat

지역변수 a

```
>>> a = 'dog'
```

```
>>> id(a)
```

```
4410123464
```

```
>>> def f4():
```

```
    global a
```

```
    print(id(a))
```

```
    a = 'cat'
```

```
    print(id(a))
```

```
>>> f4()
```

```
4410123464
```

```
4325227968
```

```
>>> id(a)
```

```
4325227968
```

dog

cat

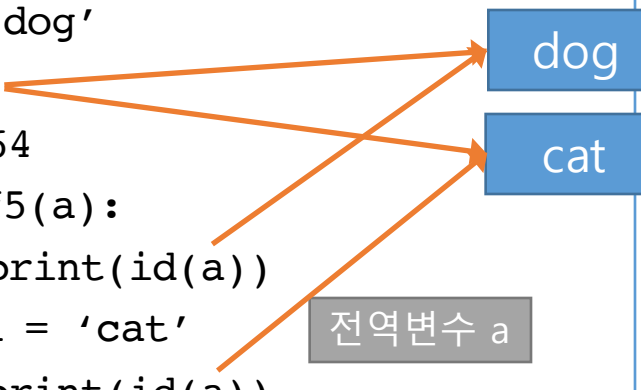
전역변수 a

변수의 범위-3

- 파라메타로 전달되는 것은 오브젝트를 가리키는 참조입니다.
- call-by-object-reference: 오브젝트 레퍼런스가 call-by-value로 전달됩니다.

```
>>> a = 'dog'
>>> id(a)
4410123464
>>> def f5(a):
    print(id(a))
    a = 'cat'
    print(id(a))

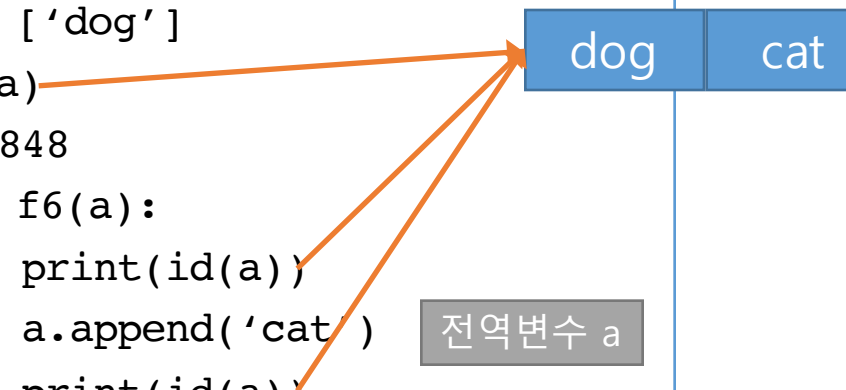
>>> f5(a)
4410123464
4325227968
>>> id(a)
4325227968
```



The diagram illustrates the state of variable `a` and the function `f5`. Variable `a` is shown pointing to the object 'dog' (memory address 4410123464). Inside the function `f5`, the parameter `a` (labeled '전역변수 a') points to the object 'cat' (memory address 4325227968). Arrows indicate the mapping from the code to the objects.

```
>>> a = ['dog']
>>> id(a)
4407771848
>>> def f6(a):
    print(id(a))
    a.append('cat')
    print(id(a))

>>> f6(a)
4407771848
4407771848
>>> a
['dog', 'cat']
```



The diagram illustrates the state of variable `a` and the function `f6`. Variable `a` is shown pointing to a list object containing 'dog' (memory address 4407771848). Inside the function `f6`, the parameter `a` (labeled '전역변수 a') also points to the same list object. After the function call, the list is modified to contain both 'dog' and 'cat'. Arrows indicate the mapping from the code to the objects.

내부함수

- 함수안에 함수를 정의할 수 있습니다.
- 내부함수는 외부함수 밖에서 찾을 수 없습니다.
- 동일한 이름을 가진 다른 외부함수 보다 내부함수를 먼저 찾습니다.

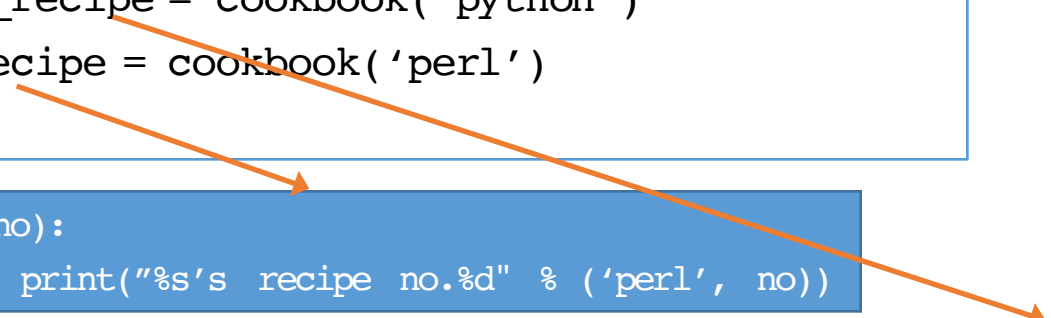
```
>>> def outer():  
    def inner():  
        print("inner")  
    inner()  
>>> outer()  
inner
```

```
>>> def inner():  
    print("global inner")  
>>> def outer():  
    def inner():  
        print("inner")  
    inner()  
>>> outer()  
inner
```

클로저

- 내부함수를 이용하여 동적으로 함수를 생성합니다.
- 외부함수에서 넘겨진 인자를 기억합니다.
- 전역변수 사용을 피하고 작은 클래스의 역할을 대신합니다.
- 자바스크립트와 루비에서 널리 사용됩니다.

```
>>> def cookbook(name):  
    def recipe(no):  
        print("%s's recipe no.%d" % (name, no))  
    return recipe  
  
>>> python_recipe = cookbook('python')  
>>> perl_recipe = cookbook('perl')
```



```
>>> python_recipe(1)  
python's recipe no.1  
>>> perl_recipe(2)  
perl's recipe no.2  
>>> id(python_recipe)  
4407710720  
>>> id(perl_recipe)  
4407657608
```

```
def recipe(no):  
    print("%s's recipe no.%d" % ('perl', no))
```

```
def recipe(no):  
    print("%s's recipe no.%d" % ('python', no))
```

제너레이터

- 제너레이터는 반복문에 사용될 수 있는 range와 유사한 함수입니다.
- 큰 사이즈의 리스트를 미리 만들지 않고 요청할 때마다 엘리먼트를 생성합니다.
- return 대신 yield 문을 사용합니다.

```
>>> def gaus_dist(n):  
    while n > 0:  
        yield random.random()  
        n -= 1  
>>> gd = gaus_dist(5)  
>>> for i in gd:  
    print(i)  
0.33179007798016014  
0.7001361002199015  
0.8208898097755184  
0.3130969547016207  
0.9497738374045565
```

```
>>> gd = gaus_dist(5)  
>>> next(gd)  
0.5344178248682653  
>>> next(gd)  
0.39623666632511123  
>>> sum(gd)  
1.5102706419901075
```

독스트링(docstring)

- 함수 정의시 첫라인에 쓰인 주석은 독스트링으로 특별하게 관리됩니다.
- 함수 설명이나 파라메타, 리턴 값에 대한 설명을 기록합니다.

```
>>> def sample(a = 0):  
    '''this is sample function'''  
    pass  
>>> sample?  
Signature: sample(a=0)  
Docstring: this is sample function  
File:      ~/Github/python-tutorial/<ipython-input-307-c359cd57cbbf>  
Type:      function  
>>> help(sample)  
Signature: sample(a=0)  
Docstring: this is sample function  
File:      ~/Github/python-tutorial/<ipython-input-307-c359cd57cbbf>  
Type:      function
```

decorator

함수에 기능 추가

- 데코레이터는 인자로 함수를 넘겨 받아 새로운 함수를 리턴하는 함수입니다.
- 기존의 함수를 변경시키지 않고 새로운 기능을 추가할 때 사용합니다.
- 함수선언 위에 '@' 표시와 함께 함수 선언시 쓸 수 있습니다.

```
>>> def print_name(first, last):
    return 'My Name is %s, %s' % (last, first)
>>> def p_decor(func):
    def func_wrapper(*args, **kwargs):
        text = func(*args, **kwargs)
        return "<p>%s</p>" % text
    return func_wrapper
>>> p_name = p_decor(print_name)
>>> p_name('jobs', 'steve')
'<p>My Name is steve, jobs</p>'
```

```
>>> def p_decor(func):
    def func_wrapper(*args, **kwargs):
        text = func(*args, **kwargs)
        return "<p>%s</p>" % text
    return func_wrapper
>>> @p_decor
    def print_name(first, last):
        return 'My Name is %s, %s' % (last, first)
>>> print_name('jobs', 'steve')
'<p>My Name is steve, jobs</p>'
```

파라메타 전달

- 데코레이터에 파라메타를 추가하기 위해 클로저의 기능을 활용합니다.
- 여러개의 데코레이터를 중첩해서 사용할 수 있습니다.

```
>>> def html_tag(tag):
    def p_decor(func):
        def func_wrapper(*args, **kwargs):
            text = func(*args, **kwargs)
            return "<%s>%s</%s>" % (tag, text, tag)
        return func_wrapper
    return p_decor

>>> @html_tag('div')
def print_name(first, last):
    return 'My Name is %s, %s' % (last, first)

>>> print_name('jobs', 'steve')
'<div>My Name is steve, jobs</div>'
```

```
>>> @html_tag
    @jquery_lib
    @blank_strip
    def print_name(tag):
        ...
```

wrapper 감추기

- 데코레이터를 사용하면 함수의 원래 이름이 바뀌어 집니다.
- `functools.wraps` 를 사용하여 함수 정보를 바꾸어 줍니다.

```
>>> def html_tag(tag):
    def p_decor(func):
        def func_wrapper(*args, **kwargs):
            text = func(*args, **kwargs)
            return "<%s>%s</%s>" % (tag, text, tag)
        return func_wrapper
    return p_decor

>>> @html_tag('div')
def print_name(first, last):
    '''p tagging for your name'''
    return 'My Name is %s, %s' % (last, first)

>>> print_name('jobs', 'steve')
'<div>My Name is steve, jobs</div>'

>>> print_name.__name__
'func_wrapper'

>>> print_name.__doc__
```

```
>>> from functools import wraps

>>> def html_tag(tag):
    def p_decor(func):
        @wraps(func)
        def func_wrapper(*args, **kwargs):
            text = func(*args, **kwargs)
            return "<%s>%s</%s>" % (tag, text, tag)
        return func_wrapper
    return p_decor

>>> @html_tag('div')
def print_name(first, last):
    '''div tagging for your name'''
    return 'My Name is %s, %s' % (last, first)

>>> print_name.__name__
'print_name'

>>> print_name.__doc__
'p tagging for your name'
```

class

클래스 기본

- 인스턴스 변수는 `__init__` 메소드 안에서 초기화하는 것이 좋습니다.
- 인스턴스 메소드의 첫 인자는 `self`(인스턴스 자신)입니다.
- `_(underscore)` 두개로 시작하는 함수나 변수는 네임 망글링(mangling)을 합니다.

```
>>> class Person(object):  
  
    def __init__(self, name, gender):  
        self.name = name  
        self.gender = gender  
  
    def has_right(self):  
        return True
```

인스턴스 변수

인스턴스 메소드

```
>>> class Man(Person):  
    pass  
  
>>> tom = Man('tom', 'M')  
>>> tom is Man  
False  
>>> isinstance(tom, Man)  
True
```

프라이빗 함수

```
>>> isinstance(tom, Person)  
True  
>>> isinstance(Man, Person)  
True
```

```
>>> class Woman(Person):  
  
    def __init__(self, name, weight):  
        super(Woman, self).__init__(name, 'F')  
        self.__weight = weight  
  
    def __get_weight(self):  
        return self.__weight  
  
>>> jane = Woman('jane', 50)  
>>> jane._Woman__get_weight()  
50
```

getter/setter 메소드

- @property 와 @name.setter 데코레이터를 사용하여 getter/setter 메소드를 구현합니다.

```
>>> class Person(object):  
  
    def __init__(self, name, gender):  
        self.name = name  
        self.gender = gender  
  
>>> jane = Person('jane', 'F')  
>>> jane.name  
'jane'
```

```
>>> class Person(object):  
  
    def __init__(self, name, gender):  
        self.__hidden_name = name  
        self.gender = gender  
  
    @property  
    def my_name(self):  
        return self.__hidden_name  
  
    @my_name.setter  
    def my_name(self, str):  
        self.__hidden_name = str  
  
>>> jane = Person('jane', 'F')  
>>> jane.my_name  
'jane'  
>>> jane.my_name = 'suji'  
>>> jane.my_name  
'suji'
```

클래스 변수, 클래스 메소드

- 클래스 변수는 클래스 바디에 선언하고 보통 메소드 선언보다 앞서서 기술합니다.
- 클래스 변수는 인스턴스와 클래스를 통해 접근할 수 있습니다.
- 클래스 메소드는 @classmethod 데코레이터를 사용하고 첫번째 인자는 클래스 자신입니다.
- 클래스 메소드는 인스턴스를 만들지 않고 클래스 내부 자료를 수정할 수 있습니다.

```
>>> class Person(object):
```

```
    population = 0
```

```
    def __init__(self, name, gender):
```

```
        self.name = name
```

```
        self.gender = gender
```

```
        Person.population += 1
```

```
    @classmethod
```

```
    def increase(cls):
```

```
        cls.population += 1
```

클래스 변수

클래스 메소드

```
>>> jane = Person('jane', 'F')
```

```
>>> jane.population
```

```
1
```

```
>>> tom = Person('tom', 'M')
```

```
>>> Person.population
```

```
2
```

```
>>> Person.increase()
```

```
>>> jane.population
```

```
3
```

정적 메소드

- 정적 메소드는 @staticmethod 데코레이터를 사용합니다.
- 정적 메소드는 인스턴스나 클래스 인자를 제공받지 않습니다.
- 정적 메소드는 클래스나 인스턴스의 자료를 변경하지 않으며 유틸리티 목적으로 사용됩니다.

```
>> class Person(object):  
  
    population = 0  
  
    def __init__(self, name, gender):  
        self.name = name  
        self.gender = gender  
        Person.population += 1  
  
    @classmethod  
    def increase(cls):  
        cls.population += 1  
  
    @staticmethod  
    def desc():  
        print('Person is Animal')
```

정적 메소드

```
>>> jane = Person('jane', 'F')  
>>> jane.population  
1  
>>> tom = Person('tom', 'M')  
>>> Person.population  
2  
>>> Person.increase()  
>>> jane.poplution  
3  
>>> Person.desc()  
'Person is Animal'  
>>> jane.desc()  
'Person is Animal'
```

인스턴스도 정적 메소드를
호출합니다.

같은 이름, 다른 종류 메소드

- 인스턴스 메소드와 클래스/정적 메소드를 같은 이름으로 혼용하여 사용하면 안됩니다.
- 같은 이름을 사용해야 하는 경우 인스턴스 메소드의 이름을 변수에 할당하여 우회합니다.

```
>> class Person(object):  
  
    def __init__(self, name, gender):  
        self.name = name  
        self.gender = gender  
        self.desc = self.__desc  
  
    def __desc(self):  
        print('%s is Person' % self.name)  
  
    @staticmethod  
    def desc():  
        print('Person is Animal')
```

```
>>> jane = Person('jane', 'F')  
>>> jane.desc()  
jane is Person  
>>> Person.desc()  
Person is Animal
```

독스트링(docstring)

- 클래스 정의시 첫라인에 쓰인 주석은 독스트링으로 특별하게 관리됩니다.
- 메소드, 클래스 변수에 대한 설명을 기록합니다.

```
>>> class Man(Person):
    '''this is subclass of Person'''
    pass

>>> Man?
Init signature: Man(name, gender)
Docstring:      this is subclass of Person
Type:           type
>>> help(Man)
Help on class Man in module __main__:
class Man(Person)
|   this is subclass of Person
|   Method resolution order:
|       Man
|       Person
|       builtins.object
|   Methods inherited from Person:
|   __init__(self, name, gender)
...
```

design pattern

<https://github.com/faif/python-patterns>

Singleton(Borg)

- 여러개의 오브젝트가 하나의 상태를 공유합니다.

```
>>> class Person(object):
    __shared_state = {}
    def __init__(self, name):
        self.__dict__ = self.__shared_state
        self.name = name

>>> tom = Person('tom')
>>> jane = Person('jane')
>>> tom is jane
False
>>> tom.name
jane
>>> jane.name
jane
```

```
>>> class Person(object):
    __instance = None

    def __new__(cls, *args, **kwargs):
        if not cls.__instance:
            cls.__instance = super(Person, cls).__new__(cls)
        return cls.__instance

    def __init__(self, name):
        self.name = name

>>> tom = Person('tom')
>>> jane = Person('jane')
>>> tom is jane
True
>>> tom.name
jane
>>> jane.name
jane
```


Strategy

- 클래스에게 메소드를 주입하여 선택한 기능을 수행하게 합니다.

```
>>> def lower(str):
    print(str.lower())
>>> def upper(str):
    print(str.upper())
>>> class Person(object):
    def __init__(self, func=None):
        if func is not None:
            self.exe = func

    def exe(self, name):
        print(name)

>>> lw_person = Person(lower)
>>> lw_person.exe('Jane')
jane
>>> up_person = Person(upper)
>>> up_person.exe('Jane')
JANE
```



Adapter

- 클래스의 메소드를 바꾸어 인터페이스를 단일화 합니다.

```
>>> class Person(object):
    def speak(self):
        print('hello')
>>> class Dog(object):
    def bark(self):
        print('walwal')
>>> class Adapter(object):
    def __init__(self, obj, **kwargs):
        self.obj = obj
        self.__dict__.update(kwargs)

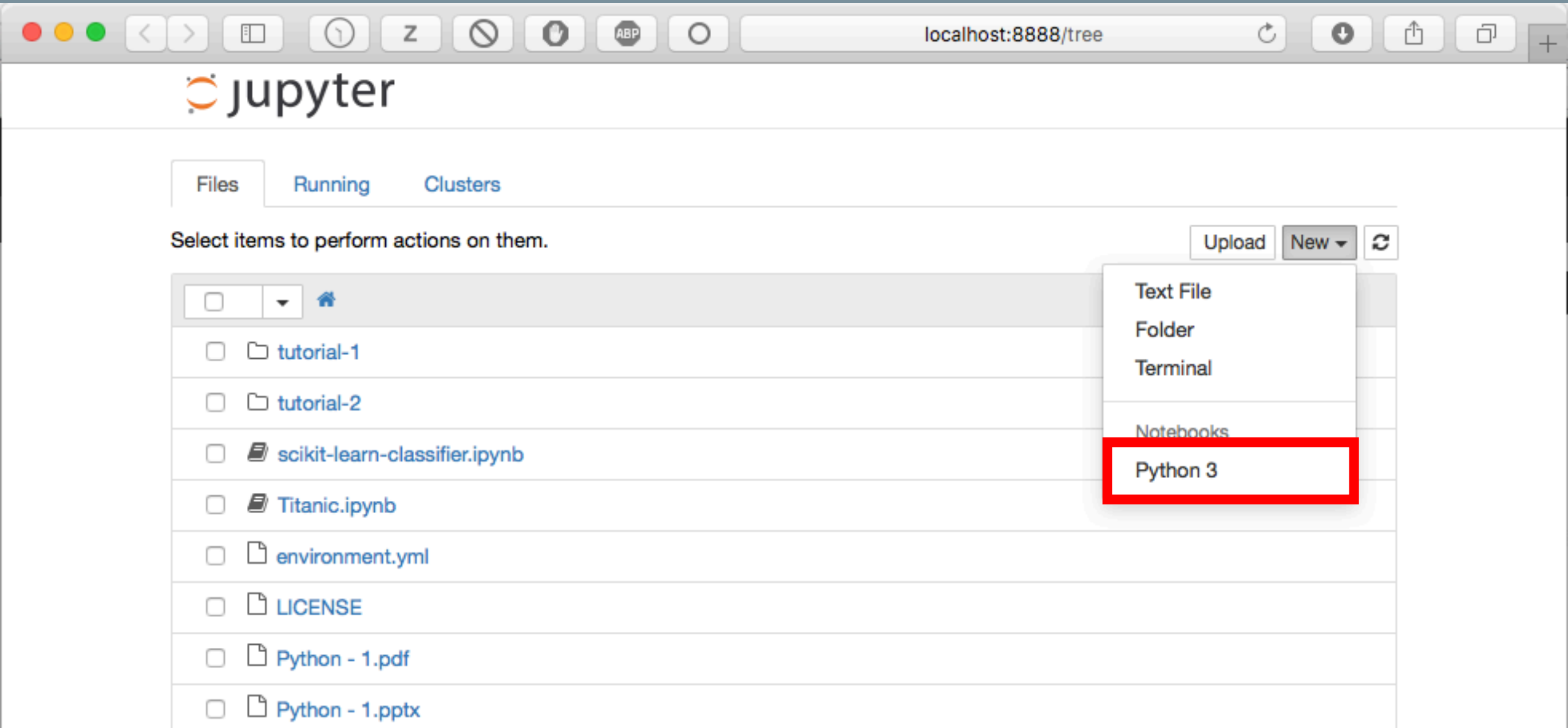
    def __getattr__(self, attr):
        return getattr(self.obj, attr)
>>> person = Person()
>>> person.speak()
hello
>>> dog = Dog()
>>> dog.bark()
walwal
```

```
>>> dog = Adapter(dog, make_noise=dog.bark)
>>> person = Adapter(person, make_noise=person.speak)
>>> dog.bark()
walwal
>>> dog.make_noise()
walwal
>>> person.speak()
hello
>>> person.make_noise()
hello
```

Demo

jupyter notebook

새로운 노트북 만들기



The image shows a web browser window at localhost:8888/tree displaying the JupyterLab interface. The 'Files' tab is active, showing a list of files and folders. A dropdown menu is open from the 'New' button, showing options: Text File, Folder, Terminal, Notebooks, and Python 3. The 'Python 3' option is highlighted with a red rectangle.

localhost:8888/tree

jupyter

Files Running Clusters

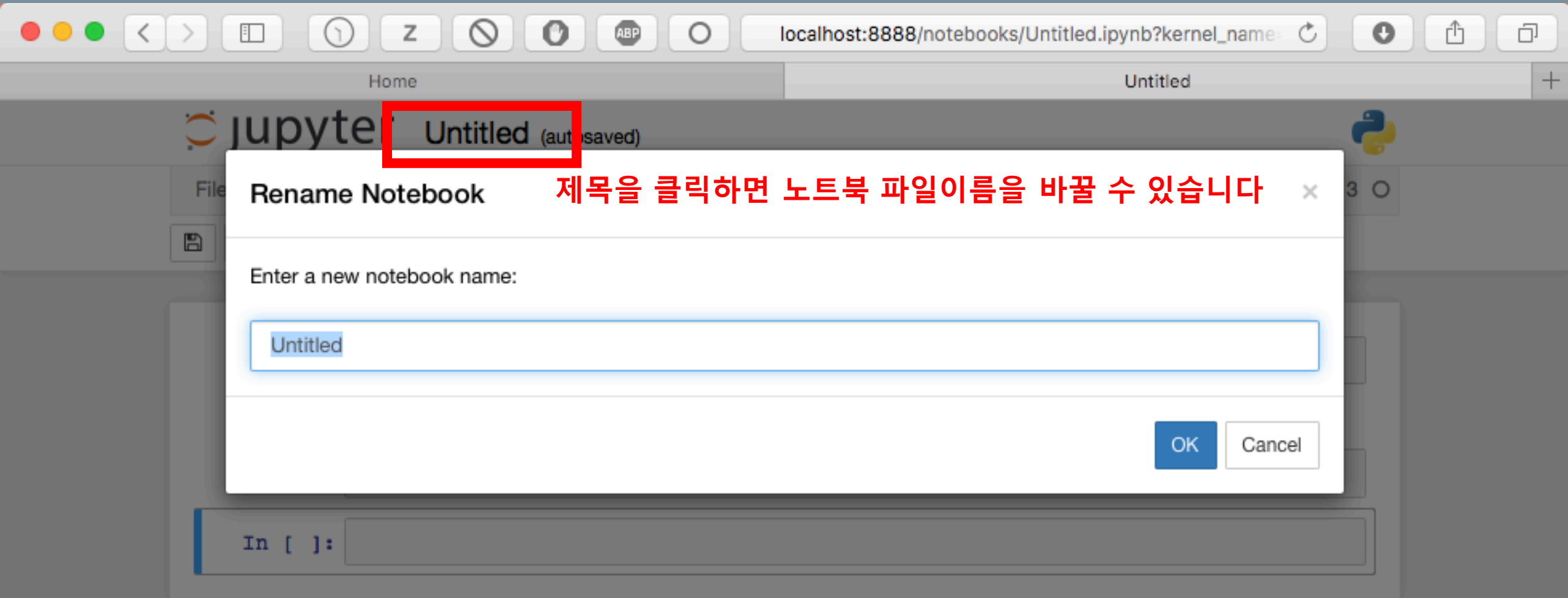
Select items to perform actions on them.

Upload New ↕

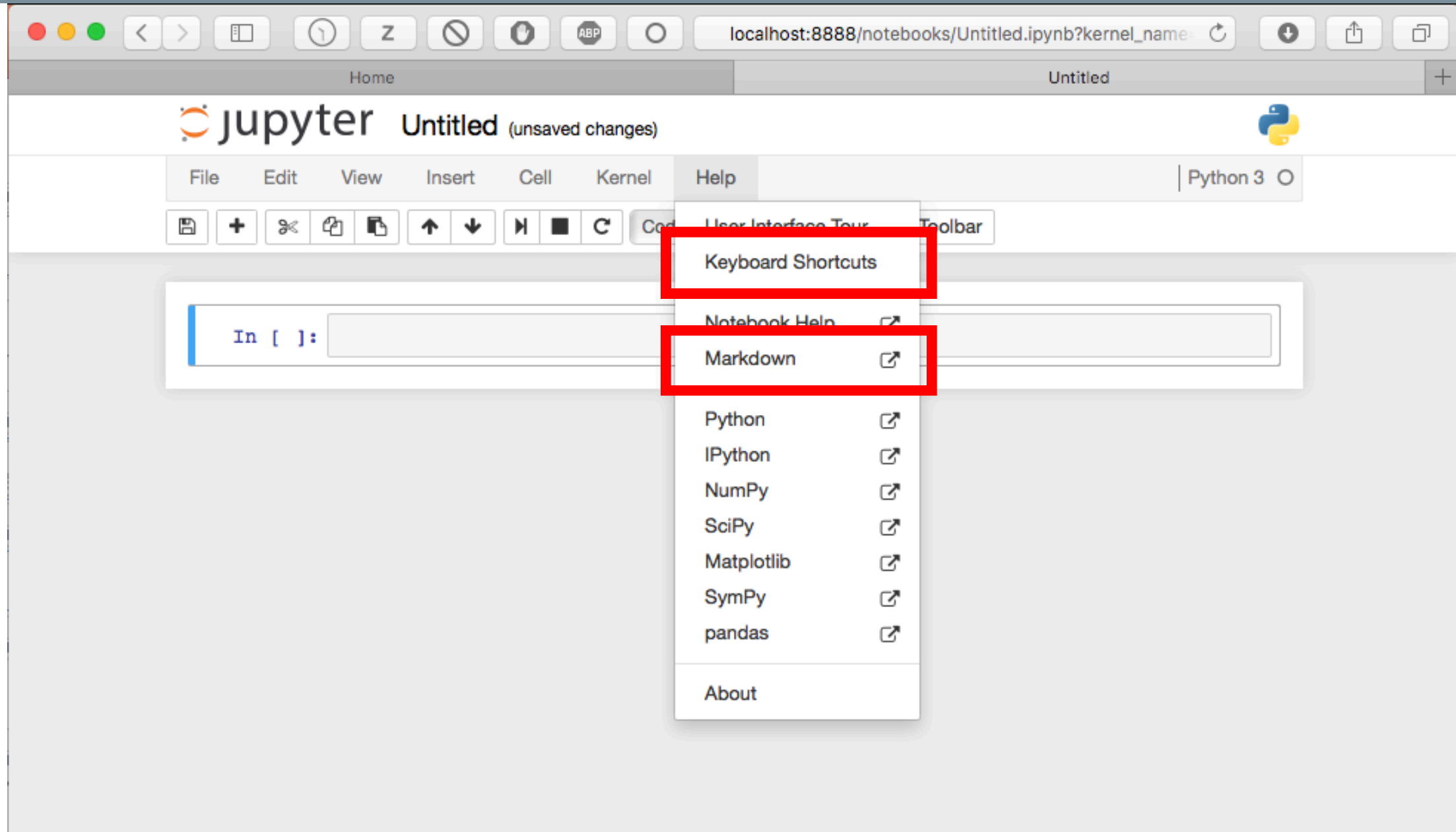
- ☐ Home
- ☐ tutorial-1
- ☐ tutorial-2
- ☐ scikit-learn-classifier.ipynb
- ☐ Titanic.ipynb
- ☐ environment.yml
- ☐ LICENSE
- ☐ Python - 1.pdf
- ☐ Python - 1.pptx

Text File
Folder
Terminal
Notebooks
Python 3

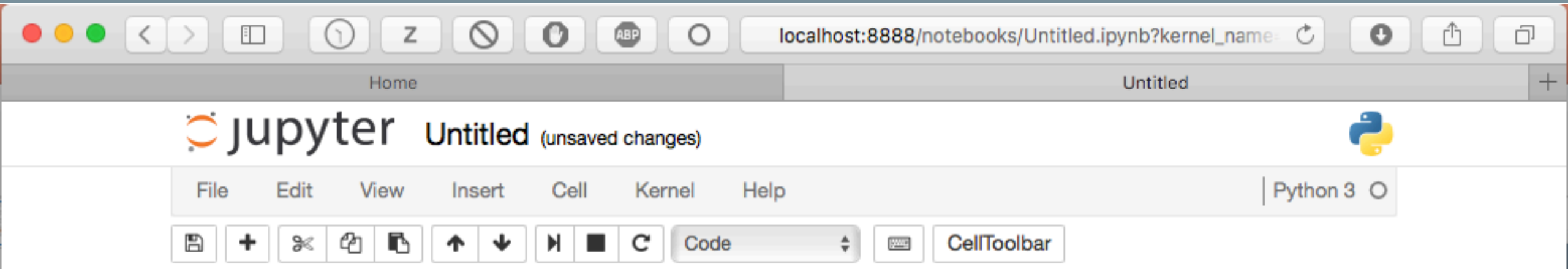
노트북 이름 바꾸기



키보드, 막다운 도움말



셀 조작 핫키



셀을 편집하려면 Enter

In [1]: 3+5

Out[1]: 8

셀의 내용을 실행시키려면 Shift+Enter

코드 셀을 막다운 셀로 바꾸려면 m, 그 반대는 y

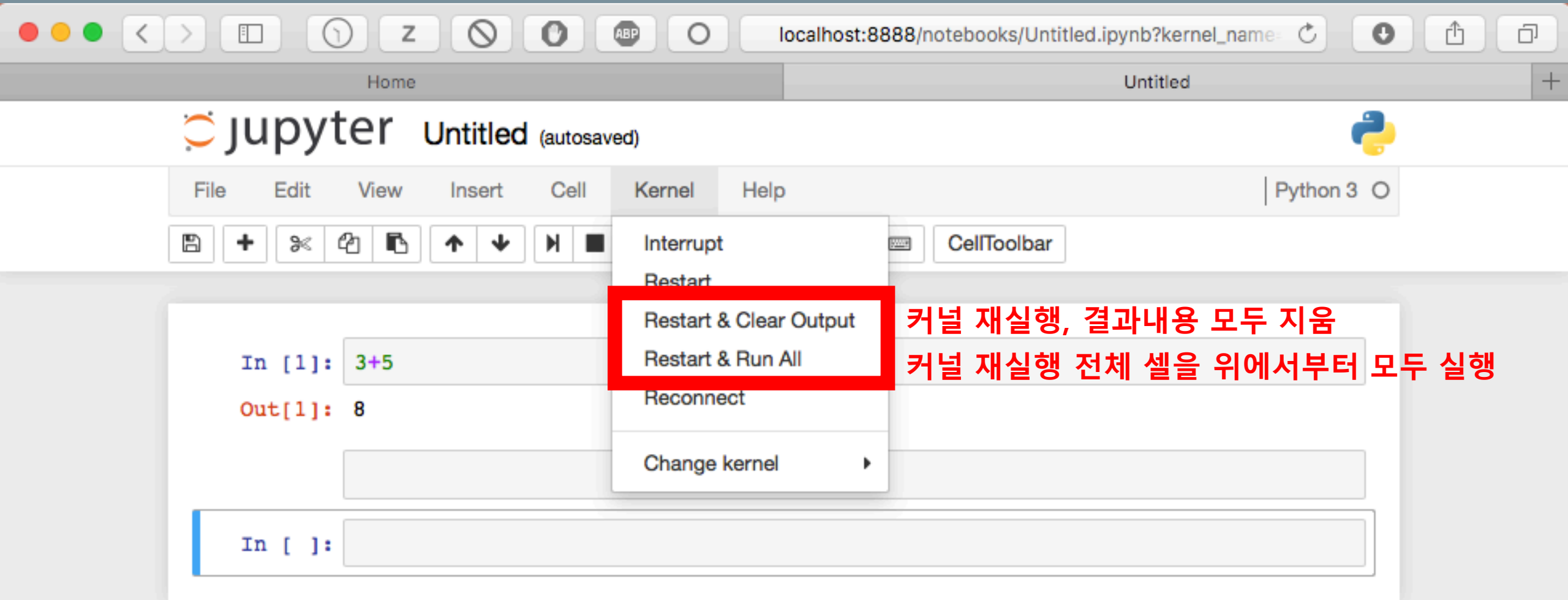
In []:

현재 셀 아래 새 셀을 추가하려면 b, 위에 추가하려면 a

현재 셀을 복사할때 c, 붙여넣을 땐 v

현재 셀을 오려낼땐 x, 지울땐 dd

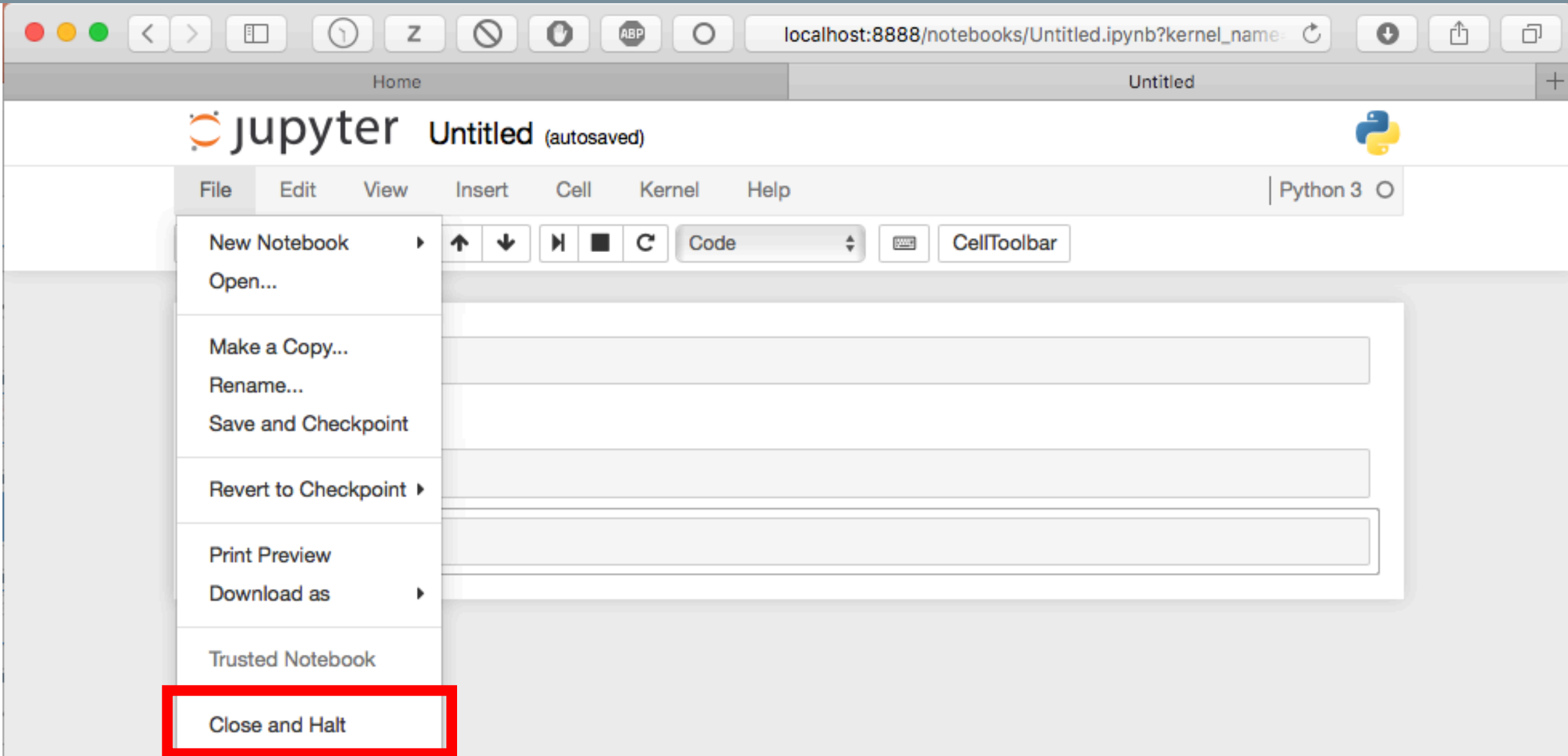
커널 재시작 및 실행



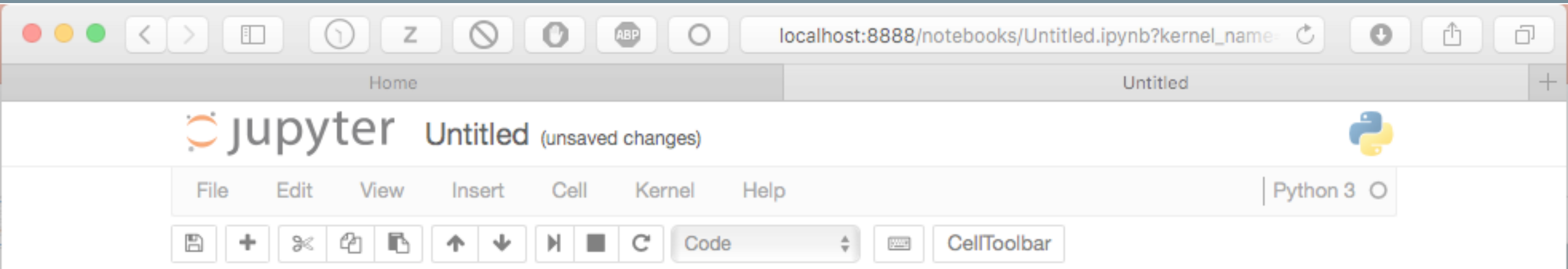
The screenshot shows the Jupyter Notebook interface. The browser address bar indicates the URL is `localhost:8888/notebooks/Untitled.ipynb?kernel_name=`. The Jupyter logo and "Untitled (autosaved)" are visible. The menu bar includes File, Edit, View, Insert, Cell, Kernel, and Help. The Kernel menu is open, showing options: Interrupt, Restart, Restart & Clear Output, Restart & Run All, Reconnect, and Change kernel. The "Restart & Clear Output" and "Restart & Run All" options are highlighted with a red box. The notebook content shows a code cell with `In [1]: 3+5` and an output cell with `Out[1]: 8`. A new code cell is at the bottom with `In []:`.

Kernel 재실행, 결과내용 모두 지움
Kernel 재실행 전체 셀을 위에서부터 모두 실행

노트북 종료



오브젝트 지속성



In [1]: 3+5

Out[1]: 8

In []:

- 노트북 셀의 오브젝트(변수)는 노트북이 종료되면 모두 사라집니다.
- 다시 노트북을 열었을 때 출력 값은 보이지만 오브젝트는 메모리에 없습니다.
- 오브젝트를 다시 만드려면 해당 셀을 다시 실행해야 합니다.
- 노트북 실행중에 상위 셀을 다시 실행하면 그로 인해 하위 오브젝트가 무효화 될 수 있습니다.

SciPy2016



NumPy
Base N-dimensional
array package



SciPy library
Fundamental
library for scientific
computing



Matplotlib
Comprehensive 2D
Plotting



IPython
Enhanced
Interactive Console



Sympy
Symbolic
mathematics



pandas
Data structures &
analysis

<https://github.com/jupyter/jupyterlab>

Pre-Alpha Jupyter Lab Demo

127.0.0.1:8888/lab

File Notebook Editor Terminal Console Help

Files

- design 4 days ago
- examples a month ago
- git-hooks 20 days ago
- images 14 days ago
- jupyterlab 3 hours ago
- jupyterlab.egg-info 4 days ago
- lib 40 minutes ago
- node_modules 40 minutes ago
- scripts a month ago
- src an hour ago
- test an hour ago
- tutorial 7 days ago
- typings an hour ago
- CONTRIBUTING.md 15 days ago
- jupyter-plugins-dem.. a month ago
- jupyter_plugins.png a month ago
- LICENSE a month ago
- MANIFEST.in a month ago
- package.json an hour ago
- README.md 5 days ago
- readthedocs.yml 20 days ago
- setup.py 21 days ago
- tslint.json 24 days ago

Commands

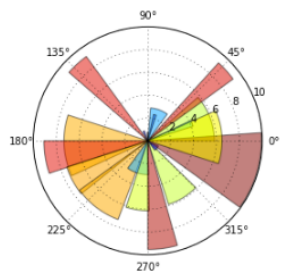
Untitled.ipynb

A simple polar plot

An example taken [from the matplotlib gallery](#):

```
In [1]: %matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

N = 20
theta = np.linspace(0.0, 2 * np.pi, N, endpoint=False)
radii = 10 * np.random.rand(N)
width = np.pi / 4 * np.random.rand(N)
ax = plt.subplot(111, projection='polar')
bars = ax.bar(theta, radii, width=width, bottom=0.0)
for r, bar in zip(radii, bars):
    bar.set_facecolor(plt.cm.jet(r / 10.))
    bar.set_alpha(0.5)
```



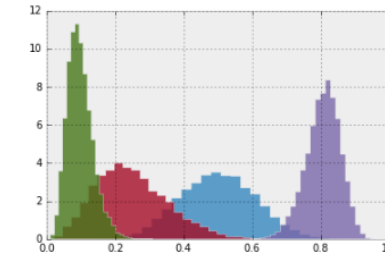
Python 3.5.2 [Continuum Analytics, Inc.] (default, Jul 2 2016, 17:52:12)
Type "copyright", "credits" or "license" for more information.

IPython 5.1.0.dev -- An enhanced Interactive Python.
? -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help -> Python's own help system.
object? -> Details about 'object', use 'object??' for extra details.

```
In [1]: %matplotlib inline
from numpy.random import beta
import matplotlib.pyplot as plt
plt.style.use('bmh')

def plot_beta_hist(a, b):
    plt.hist(beta(a, b, size=10000), histtype="stepfilled",
             bins=25, alpha=0.8, normed=True)
    return

plot_beta_hist(10, 10)
plot_beta_hist(4, 12)
plot_beta_hist(50, 12)
plot_beta_hist(6, 55)
```



In []:

Terminal 1

```
1 [|||||] 18.1%
2 [|||||] 5.0%
3 [|||||] 15.6%
4 [|||||] 5.0%
Mem [|||||] 5987/8192MB
Swp [|||||] 2487/3072MB

Tasks: 305 total, 1 running
Load average: 2.29 2.07 2.09
Uptime: 4 days, 21:59:11

PID USER PRI NI VIRT RES SHR S CPU% MEM% TIME+ Command
32374 fperez 31 0 2389M 2048 0 R 0.0 0.0 0:00.00 htop
1 root 0 0 0 0 0 0 0.0 0.0 0:00.00 (launchd)
46 root 0 0 0 0 0 0 0.0 0.0 0:00.00 (syslogd)
47 root 0 0 0 0 0 0 0.0 0.0 0:00.00 (UserEventAgent)

F1 Help F2 Setup F3 Search F4 Invert F5 Tree F6 Sort By F7 Filter F8 Hide F9 All F10 Quit
```

Setup & Example

Github Download

The screenshot shows the GitHub interface for the repository `rickiepark/python-tutorial`. The browser address bar displays the URL `https://github.com/rickiepark/python-tutorial`. The repository page includes navigation links (Personal, Open source, Business, Explore, Pricing, Blog, Support), a search bar, and buttons for 'Sign in' and 'Sign up'. The repository name is `rickiepark / python-tutorial`, with 1 watch, 0 stars, and 0 forks. The 'Code' tab is selected, showing the repository description 'python code & jupyter notebooks for learning'. Below this, statistics show 6 commits, 1 branch, 0 releases, and 1 contributor. A 'Branch: master' dropdown and a 'New pull request' button are visible. The 'Find file' button and a 'Clone or download' button are also present. The commit history shows a single commit by `rickiepark` titled '테스트 코드 정리' (Organizing test code), with the latest commit hash `8dc6907` made 7 days ago. A file named `tutorial-1` is listed under this commit, also dated 7 days ago.

python code & jupyter notebooks for learning

6 commits 1 branch 0 releases 1 contributor

Branch: master New pull request Find file Clone or download

rickiepark 테스트 코드 정리 Latest commit 8dc6907 7 days ago

tutorial-1 테스트 코드 정리 7 days ago

내 문서 밑으로 압축 해제

학생 점수 관리

- Student 와 GradeBooks 클래스가 있습니다.
 - Student 클래스는 학생 이름으로 초기화되고 학생마다 고유번호를 발급합니다.
 - Gradebooks는 과목 이름으로 초기화 합니다.
 - Gradebooks는 학생과 점수를 인스턴스 변수로 관리합니다.
 - Gradebooks는 학생과 점수를 입력받아 누적하여 저장합니다.
 - Gradebooks는 모든 학생의 리스트를 리턴하고 가장 높은 점수를 가진 학생을 찾습니다.
-
- 각 클래스와 메소드에 독스트링을 추가합니다.
 - Student 클래스에서 고유번호를 위한 클래스 변수가 오염되지 않도록 숨깁니다.
 - Gradebooks에 한 학생의 복수개 점수를 입력받을 수 있도록 메소드를 추가합니다.
 - 과목 클래스를 만들고 Gradebooks에 title 대신 과목 오브젝트를 이용하도록 변경합니다.

Q&A

Any Question: haesunrpark@gmail.com

github URL: <https://github.com/rickiepark/python-tutorial>

This slide available at
<http://tensorflowkorea.wordpress.com>