Упражнение 1

Команден интерпретатор IPython. Основни възможности на интерпретатора. Възможности на средата Jupyter Notebook за разработване на интерактивни приложения

Има два начина за инсталиране на IPython на компютър. Ако вече имате инсталиран Python, можете да използвате *pip* (мениджър на пакети на Python), за да инсталирате IPython, като използвате следната команда:

pip install ipython

IPython може да се инсталира и с помощта на дистрибуцията на Python Anaconda. Когато Python се инсталира чрез Anaconda, автоматично се инсталира IPython. Ако Python се инсталира чрез Miniconda (по-малка версия на Anaconda), IPython ще трябва да се инсталира с помощта на conda (мениджър на пакети на Anaconda за пакети за наука за данни) със следната команда:

conda install ipython

След инсталация на IPython, зад аможе да се използва трябва да се въведе следната команда в терминала на компютъра:

ipython

След като се напише тази команда, терминала предоставя няколко подробности за IPython, като версията, описание на IPython и някои команди, които може да се въведат при започване (?, % Quickref, help, object?).

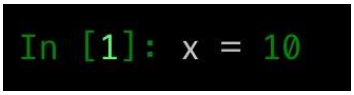
1. Изпълняване на команди на Native Shell

Ако трябва да се стартира интерпретатора на Python (използвайки python) и след това да се напише cd след зареждането на интерпретатора, на терминала ще се появи грешка. Тази грешка възниква, защото интерпретаторът на Python не разпознава командата. Командата cd е присъща на терминала на компютъра, но не е и на интерпретатора на Python. IPython включва поддръжка на команди на собствената обвивка като cd, ls, история на командите и др.

2. Синтаксис на IPython

Едно от първите неща, които се забелязват при IPython, е, че той осигурява подчертаване на синтаксиса, което означава, че използва цвят за разграничаване на части от кода на Python. Въведете x = 10 в терминала и забележете как IPython подчертава

кода с различни цветове. Това подчертаване на синтаксиса е подобрение спрямо интерпретатора по подразбиране на Python и прави кода много по-лесен за четене в терминала.



IPython автоматично предоставя отстъп, когато се пише Python код в интерпретатора.

Пример:
Стартирайте интерпретатора на IPython, като напишете ipython втерминала.
След това въведете следния код и натиснете „Enter" (или „Return" на клавиатурата):

for x in range(10):

Забележете, че след като натиснете „Enter" (или „Return") на клавиатурата, IPython автоматично предоставя необходимите отстъпи (четири интервала) на следващия ред код.
На следващия ред въведете следния код:

print x

За да стартирате кода, натиснете „Enter" (или „Return") два пъти. (Второто „Enter" информира IPython, че пишете код и инструктира IPython да изпълни кода.)

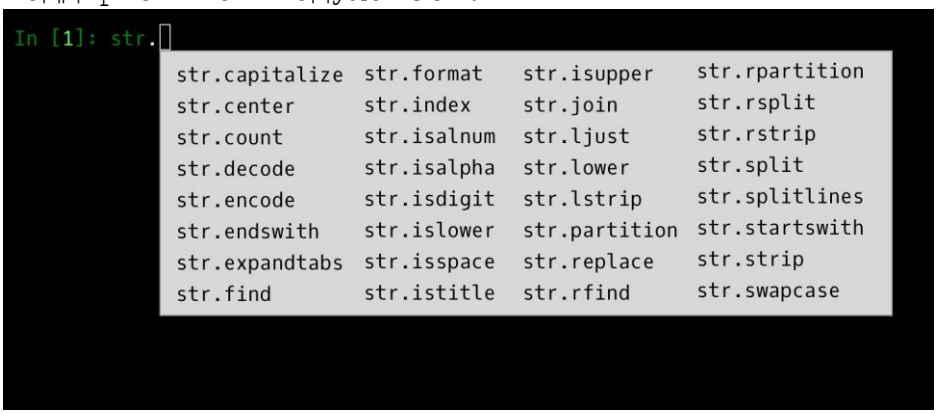
Резултат:

3. Завършване на раздел

IPython осигурява попълване на раздели. Например str модулът предоставя някои полезни методи, които могат да се използват за низове.

Пример:
Стартирайте IPython на терминал. Въведете следния код и след това натиснете „Tab" на клавиатурата:
str.

След натискане на TAB трябва да се визуализира списък с методи, поддържани от модула str.



Може да се използват клавишите със стрелки нагоре и надолу на клавиатурата, за да навигирате през методите и да изберете този, който искате да използвате, или можете да започнете да въвеждате името на метода, който искате да използвате, и да го завършите, като използвате „Tab" . Това е огромно подобрение спрямо интерпретатора по подразбиране!

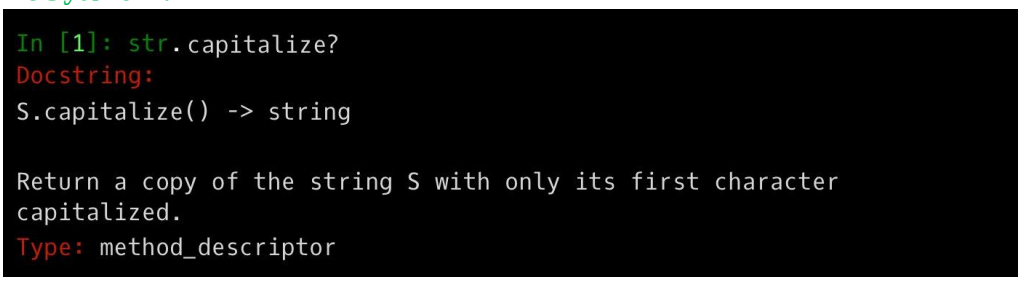
4. Документация

Автозавършването на команда е полезно, защото предоставя списък с всички възможни методи, които конкретен модул съдържа.

Пример:
Въведете следната команда в IPython и натиснете „Enter" (или „Return") на клавиатурата:

```
str.capitalize?
```

Резултат:



Може да се използва ? в края на почти всяка команда в IPython, за извеждане на повече информация за командата (или метод, променлива и т.н.).

4.1. Достъп до документация с ?

Пример:

За да се види документацията на вградената функция len, може да се направи следното:

```
In [1]: help(len)
Help on built-in function len in module builtins:

len(...)
    len(object) -> integer

    Return the number of items of a sequence or mapping.
```

В зависимост от вашия редактор, тази информация може да се показва като вграден текст или в отделен изскачащ прозорец.

Тъй като намирането на помощ за обект е толкова често и полезно, IPython въвежда символ ? като стенография за достъп до тази документация и друга подходяща информация:

```
In [2]: len?
Type:        builtin_function_or_method
String form: <built-in function len>
Namespace:   Python builtin
Docstring:
len(object) -> integer

Return the number of items of a sequence or mapping.
```

Тази нотация работи почти за всичко, включително обектни методи:

```
In [3]: L = [1, 2, 3]
In [4]: L.insert?
Type:        builtin_function_or_method
String form: <built-in method insert of list object at 0x1024b8ea8>
Docstring:   L.insert(index, object) -- insert object before index
```

или дори самите обекти, с документацията от техния тип:

```
In [5]: L?
Type:        list
String form: [1, 2, 3]
Length:      3
Docstring:
list() -> new empty list
list(iterable) -> new list initialized from iterable's items
```

Важното е, че това ще работи дори за функции или други обекти, които създавате сами!

Пример: Дефиниране на функция "square":

```
In [6]: def square(a):
  ....:     """Return the square of a."""
  ....:     return a ** 2
  ....:
```

За да се създаде docstring за функцията се поставя низов литерал в първия ред. Тъй като doc низовете обикновено са множество редове се използва нотация с трите кавички на Python за многоредови низове.

Пример: Използване на ? за намиране на низ в документа:

```
In [7]: square?
Type:        function
String form: <function square at 0x103713cb0>
Definition:  square(a)
Docstring:   Return the square of a.
```

4.2. Достъп до изходния код с ??

IPython предоставя пряк път към изходния код с двойния въпросителен знак (??):

```
In [8]: square??
Type:        function
String form: <function square at 0x103713cb0>
Definition:  square(a)
Source:
def square(a):
    "Return the square of a"
    return a ** 2
```

За прости функции като тази двойният въпросителен знак може да даде бърз поглед върху детайлите.

5. Изследване на модули

Другият полезен интерфейс на IPython е използването на клавиша tab за автоматично попълване и изследване на съдържанието на обекти, модули и пространства от имена. В следващите примери ще използваме <TAB>, за да посочим кога трябва да се натисне клавишът Tab.

За визуализиране на списък с всички налични атрибути на обект, можете да се въведе името на обекта, последвано от точка (".") и клавиша Tab:

```
In [10]: L.<TAB>
L.append   L.copy     L.extend   L.insert   L.remove   L.sort
L.clear    L.count    L.index    L.pop      L.reverse
```

За да стесните списъка, можете да въведете първия знак или няколко знака от името и клавишът Tab ще намери съответстващите атрибути и методи:

```
In [10]: L.c<TAB>
L.clear  L.copy   L.count

In [10]: L.co<TAB>
L.copy   L.count
```

Ако има само една опция, натискането на клавиша Tab ще завърши реда за вас. Например следното незабавно ще бъде заменено с L.count:

```
In [10]: L.cou<TAB>
```

Въпреки че Python няма строго наложено разграничение между публични / външни атрибути и частни / вътрешни атрибути, по конвенция се използва предшестващо долно подчертаване за обозначаване на такива методи. За по-голяма яснота тези частни методи и специални методи са пропуснати от списъка по подразбиране, но е възможно да се изброят, като изрично се напише:

```
In [10]: L._<TAB>
L.__add__           L.__gt__            L.__reduce__
L.__class__         L.__hash__          L.__reduce_ex__
```

Пример: Намиране на всички възможни импорти в пакета itertools, които започват с co:

```
In [10]: from itertools import co<TAB>
combinations                    compress
combinations_with_replacement   count
```

Пример:
Визуализиране на налични импорти в системата (това ще се промени в зависимост от това кои скриптове и модули на трети страни са видими за сесията на Python):

```
In [10]: import <TAB>
Display all 399 possibilities? (y or n)
Crypto              dis                 py_compile
Cython              distutils           pyclbr
...                 ...                 ...
difflib             pwd                 zmq

In [10]: import h<TAB>
hashlib             hmac                http
heapq               html                husl
```

Попълването на раздела е полезно, ако знаете първите няколко знака на обекта или атрибута, който търсите, но няма голяма помощ, ако искате да съпоставите символите в средата или края на

думата. За този случай на използване, IPython предоставя средство за заместване на имена, използващи символа *.

Например може да се изброи всеки обект в пространството от имена, който завършва с Warning:

```
In [10]: *Warning?
BytesWarning                  RuntimeWarning
DeprecationWarning            SyntaxWarning
FutureWarning                 UnicodeWarning
ImportWarning                 UserWarning
PendingDeprecationWarning     Warning
ResourceWarning
```

Забележете, че знакът * съответства на всеки низ, включително празния низ.

Пример:
По същия начин, да предположим, че търсим низ метод, който съдържа думата find някъде в името си. Можем да се търси по този начин:

```
In [10]: str.*find*?
str.find
str.rfind
```

6. Преки пътища

6.1. Преки пътища за навигация

Въпреки че използването на клавишите със стрелки наляво и надясно за придвижване назад и напред по линията е съвсем очевидно, има и други опции:

Ctrl-a Преместване на курсора в началото на реда
Ctrl-e Преместване на курсора в края на реда
Ctrl-b или клавиша със стрелка наляво Преместване на курсора назад с един символ
Ctrl-f или клавиша със стрелка надясно Преместване на курсора напред с един символ

6.2. Преки пътища за въвеждане на текст

Ctrl-d Изтриване на следващия знак в реда
Ctrl-k Изрязване на текст от курсора до края на реда
Ctrl-u Изрязване на текста от началото на реда до курсора
Ctrl-y Yank (т.е. поставяне) на текст, който преди това е бил изрязан
Ctrl-t Транспониране (т.е. превключване) на предишните два знака

6.3. Преки пътища за история на командите

Ctrl-p (или клавиша със стрелка нагоре) Достъп до предишна команда в историята
Ctrl-n (или клавиша със стрелка надолу) Достъп до следващата команда в историята
Ctrl-r Обратно търсене в историята на командите

Обратното търсене може да бъде особено полезно. В точка **4.1.** дефинирахме функция, наречена "square". Нека да извършим обратно търсене в нашата история на Python от нова обвивка на IPython и да намерим тази дефиниция отново. Когато натиснете Ctrl-r в терминала IPython, ще се изведе следния ред:

```
In [1]:
(reverse-i-search)`':
```

Ако се започне писане на символи в този ред, IPython ще попълни автоматично последната команда, ако има такава, която съответства на тези символи:

```
In [1]:
(reverse-i-search)`sqa': square??
```

Във всеки момент може да се добавят още знаци, за да се прецизира търсенето, или да се натисне Ctrl-r за търсене на друга команда, която съответства на заявката. Натискане на Ctrl-r още два пъти ще изведе:

```
In [1]:
(reverse-i-search)`sqa': def square(a):
    """Return the square of a"""
    return a ** 2
```

След като намерите командата, която търсите, натиснете Return и търсенето ще приключи. След това можем да използваме извлечената команда и да продължим с нашата сесия:

Клавишите Ctrl-p / Ctrl-n или клавишите със стрелки нагоре / надолу също могат да се използват за търсене в историята, но само чрез съвпадение на символи в началото на реда. Тоест, ако се напише def и след това се натисне Ctrl-p, той ще намери най-новата команда (ако има такава) в историята, която започва със символите def.

```
In [1]: def square(a):
    """Return the square of a"""
    return a ** 2

In [2]: square(2)
Out[2]: 4
```

6.4. Други преки пътища

Ctrl-l Изчистване на екрана на терминала
Ctrl-c Прекъсване на текущата команда на Python
Ctrl-d Излизане от сесията на IPython

По-специално Ctrl-c може да бъде полезен, когато по невнимание започнете много продължителна работа.

Въпреки че някои от дискутираните тук преки пътища може да изглеждат малко досадни в началото, те бързо стават автоматични с практиката.

7. Подобрения, които IPython добавя върху нормалния синтаксис на Python.

Подобренията в IPython са известни като „магически команди" и са с префикс от символа %. Тези магически команди са предназначени за кратко решаване на различни често срещани проблеми при стандартния анализ на данни. Вълшебните команди се предлагат в два варианта: *„магията на линиите"*, които се обозначават с един% префикс и действат на един ред на въвеждане, и *„магията на клетките"*, които се означават с двоен %% префикс и действат на множество редове на въвеждане.

7.1. Поставяне на блокове с код:%paste и %cpaste

При работа в интерпретатора на IPython, често срещан проблем е, че поставянето на многоредови кодови блокове може да доведе до неочаквани грешки, особено когато са включени маркери за отстъп и интерпретатор. Често срещан случай е намирането на примерен код на уебсайт и поставянето му в интерпретатора.

Например:

```
>>> def donothing(x):
...     return x
```

Кодът е форматиран, както би се появил в интерпретатора на Python. При поставяне в IPython, ще се получи грешка:

```
In [2]: >>> def donothing(x):
   ...:     ...     return x
   ...:
  File "<ipython-input-20-5a66c8964687>", line 2
    ...     return x
                ^
SyntaxError: invalid syntax
```

Интерпретаторът е объркан от допълнителните символи. Функцията %paste на IPython е създадена да обработва точно този тип многоредови, маркирани входове:

```
In [3]: %paste
>>> def donothing(x):
...     return x

## -- End pasted text --
```

Командата %paste едновременно въвежда и изпълнява кода, така че сега функцията е готова за използване:

```
In [4]: donothing(10)
Out[4]: 10
```

Команда с подобно изпълнение е %cpaste, която отваря интерактивен многоредов ред, в който може да се поставя код, който да се изпълнява в пакет:

```
In [5]: %cpaste
Pasting code; enter '--' alone on the line to stop or use Ctrl-D.
:>>> def donothing(x):
:...     return x
:--
```

7.2. Изпълняващ се външен код: %run

Пример:
Създаден е файл myscript.py със следното съдържание:

```
#-----------------------------------
# file: myscript.py

def square(x):
    """square a number"""
    return x ** 2

for N in range(1, 4):
    print(N, "squared is", square(N))
```

Можете да се изпълни IPython сесия, както следва:

```
In [6]: %run myscript.py
1 squared is 1
2 squared is 4
3 squared is 9
```

Обърнете внимание също така, че след стартиране на този скрипт, всички функции, дефинирани в него, са достъпни за използване в текущата IPython сесия:

```
In [7]: square(5)
Out[7]: 25
```

7.3. Изпълнение на времевия код: %timeit

Друг пример за полезна магическа функция е %timeit, който автоматично ще определи времето за изпълнение на едноредовия Python израз.

Пример:
Проверка на ефективността на разбирането на списък:

```
In [8]: %timeit L = [n ** 2 for n in range(1000)]
1000 loops, best of 3: 325 µs per loop
```

Предимството на %timeit е, че за кратки команди той автоматично изпълнява множество изпълнения, за да постигне по-стабилни резултати. За многоредови оператори добавянето на втори знак % ще превърне това в клетъчна магия, която може да обработва множество редове на въвеждане. Например, ето еквивалентната конструкция с for-loop:

```
In [9]: %%timeit
   ...: L = []
   ...: for n in range(1000):
   ...:     L.append(n ** 2)
   ...:
1000 loops, best of 3: 373 µs per loop
```

Вижда се, че разбирането на списъците е с около 10% по-бързо от еквивалентната конструкция за цикъл в този случай.

7.4. Помощ за магически функции:?,% Magic и% lsmagic

Подобно на нормалните функции на Python, магическите функции на IPython имат документи и тази полезна документация може да бъде достъпна по стандартния начин. Така например, за да се прочете документацията на %timeit magic трябва да се напише:

```
In [10]: %timeit?
```

Документацията за други функции може да бъде достъпна по подобен начин. За получаване на достъп до общо описание на наличните магически функции, включително някои примери, може да се напише това:

```
In [11]: %magic
```

За бърз и лесен списък на всички налични магически функции:

```
In [12]: %lsmagic
```

Jupyter

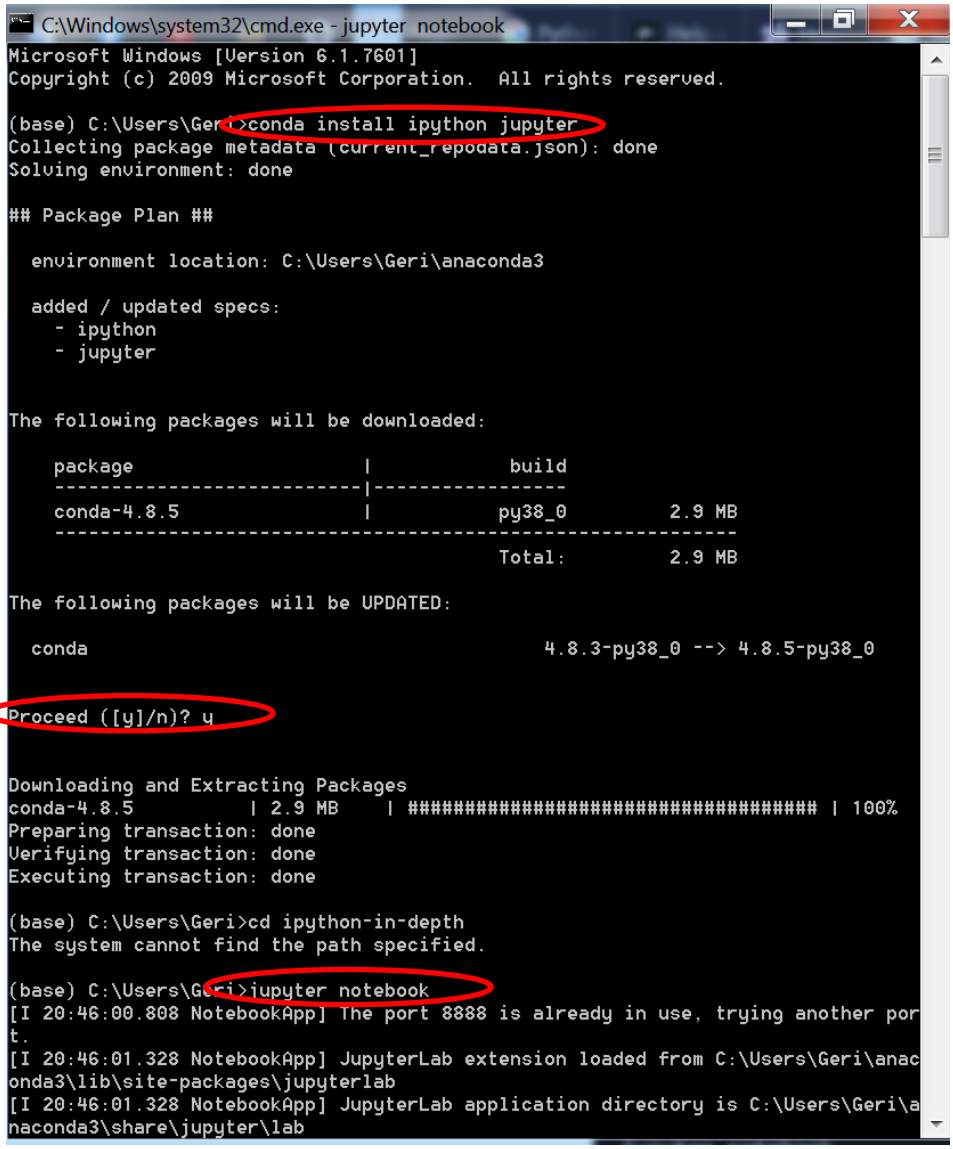
Jupyter предоставя интерактивен уеб-базиран интерфейс, който може да бъде извикан от браузър. Jupyter се използва за писане на програми на Python и създаване на вградени графики за визуализиране на данни.

1. Инсталиране и стартиране на Jupyter

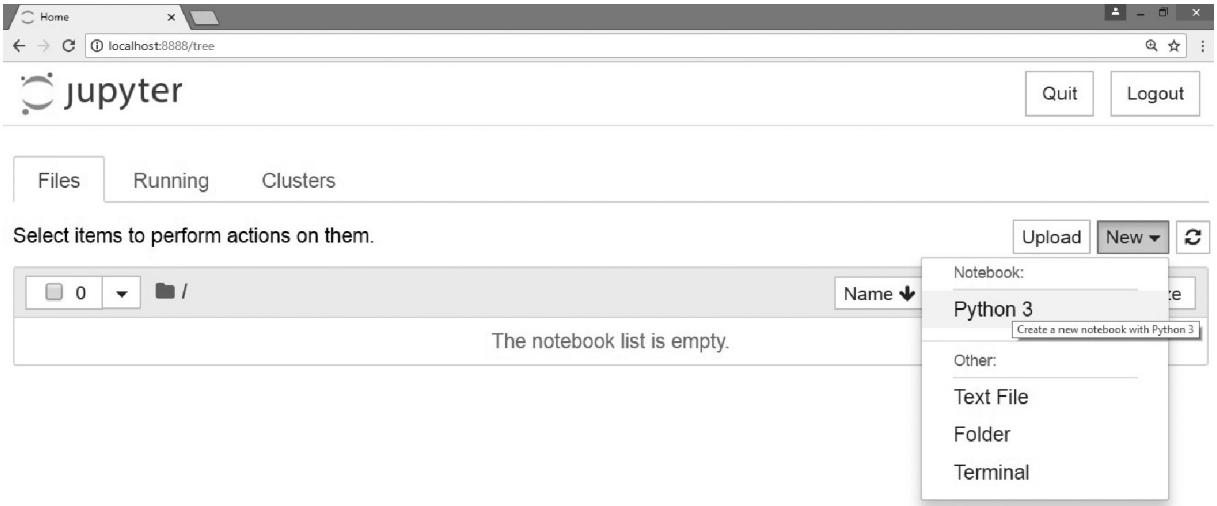
- *Инсталиране на Anaconda distribution*
- *Стартиране на Anaconda cmd.exe prompt и изпълнение на следните команди:*

  conda install ipython jupyter

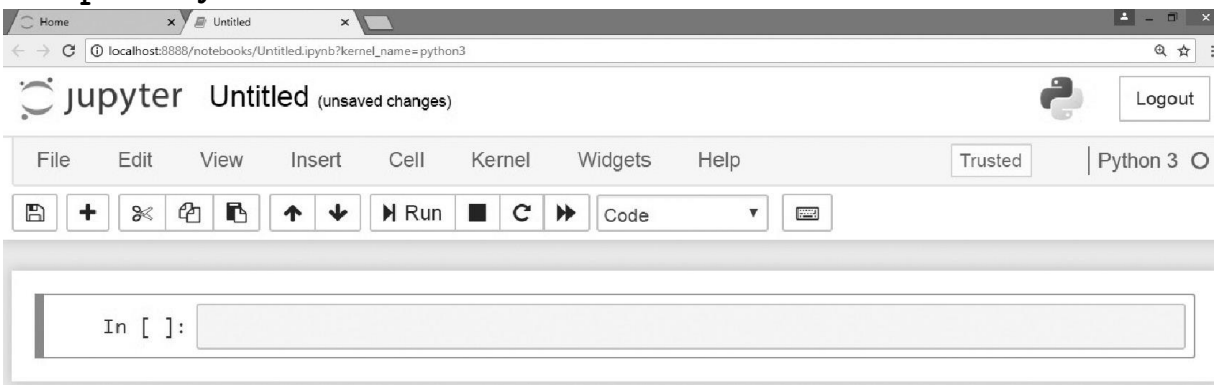
jupyter notebook



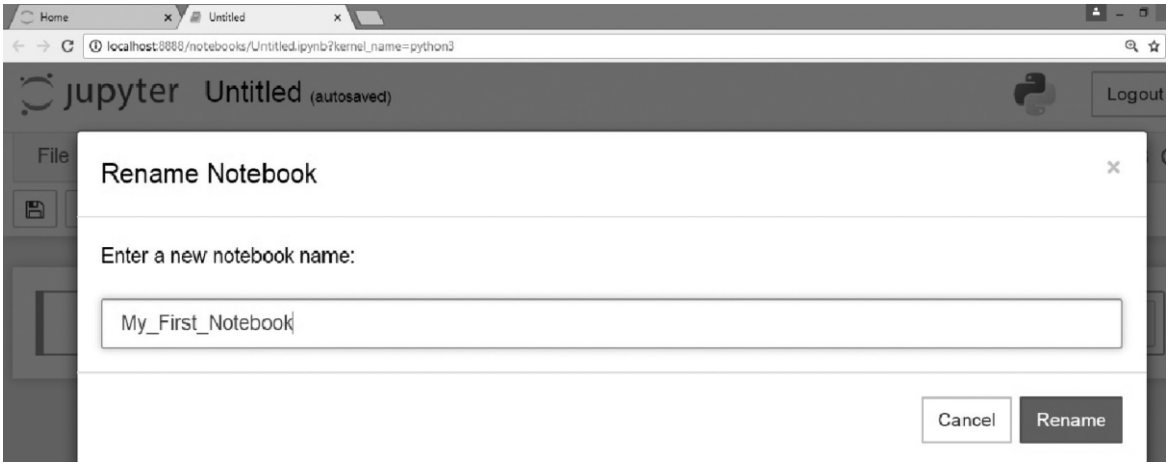
2. Създаване на нов Jupyter notebook



3. Преименуване на бележника

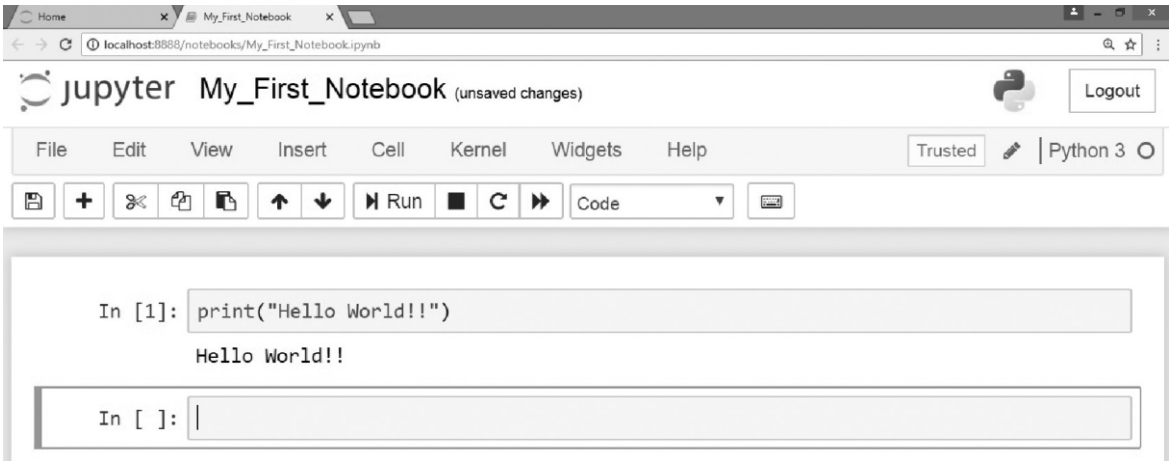


За да се преименува бележника, щракнете върху Untitled в горната част. Появява се прозорец Rename notebook . Въведете име за бележника. Нека назовем бележника като My_First_Notebook, след което се потвърждава с бутона Rename. Бележникът се записва като My_First_Notebook.



4. Писане на код в бележника

В бележника може да се въвежда Python код и той ще се появи в клетката. Изпълнението на кода в тази клетка може да се извърши, чрез стартиране с Run или натискане на клавишната комбинация Alt + Enter.



Задачи за изпълнение:

Задача 1:

Създайте Jupyter бележник с два файла:

- В първия файл се съдържа програма, която въвежда число и се извежда кубичната му стойност.
- Във втория файл се въвеждат два низа, след което се обединяват в трети. В третия низ се търси последователност ”no”.

Задача 2:

Използвайки магическите методи създайте масив с пет елемента. Изведете стойността на първият елемент, променете стойността му. Изтрийте третият елемент.