

Python in Data Science

Giảng viên: **TS. Nguyễn Văn Quyết**

Email: quyetict@gmail.com

Giảng viên

● Đào tạo:

- 2019: Tiến sĩ KHMT, Đại học Quốc gia Chonnam, Hàn Quốc
- 2013: Thạc sĩ CNTT, Đại học Bách Khoa Hà Nội
- 2009: Kỹ sư CNTT, Trường Đại học SPKT Hưng Yên

● Lĩnh vực nghiên cứu:

- Phân tích dữ liệu lớn (Big Data Analytics)
- Xử lý đồ thị lớn (Big Graph Processing)
- Tính toán song song (Parallel Computing)
- Các hệ thống phân tán (Distributed Systems)
- Kết nối vạn vật (Internet of Things)



- Tại sao sử dụng Python trong phân tích dữ liệu?
- IPython là gì?
- Cài đặt IPython
- Các lệnh trong IPython
- Lịch sử lệnh nhập và xuất dữ liệu
- Xử lý ngoại lệ

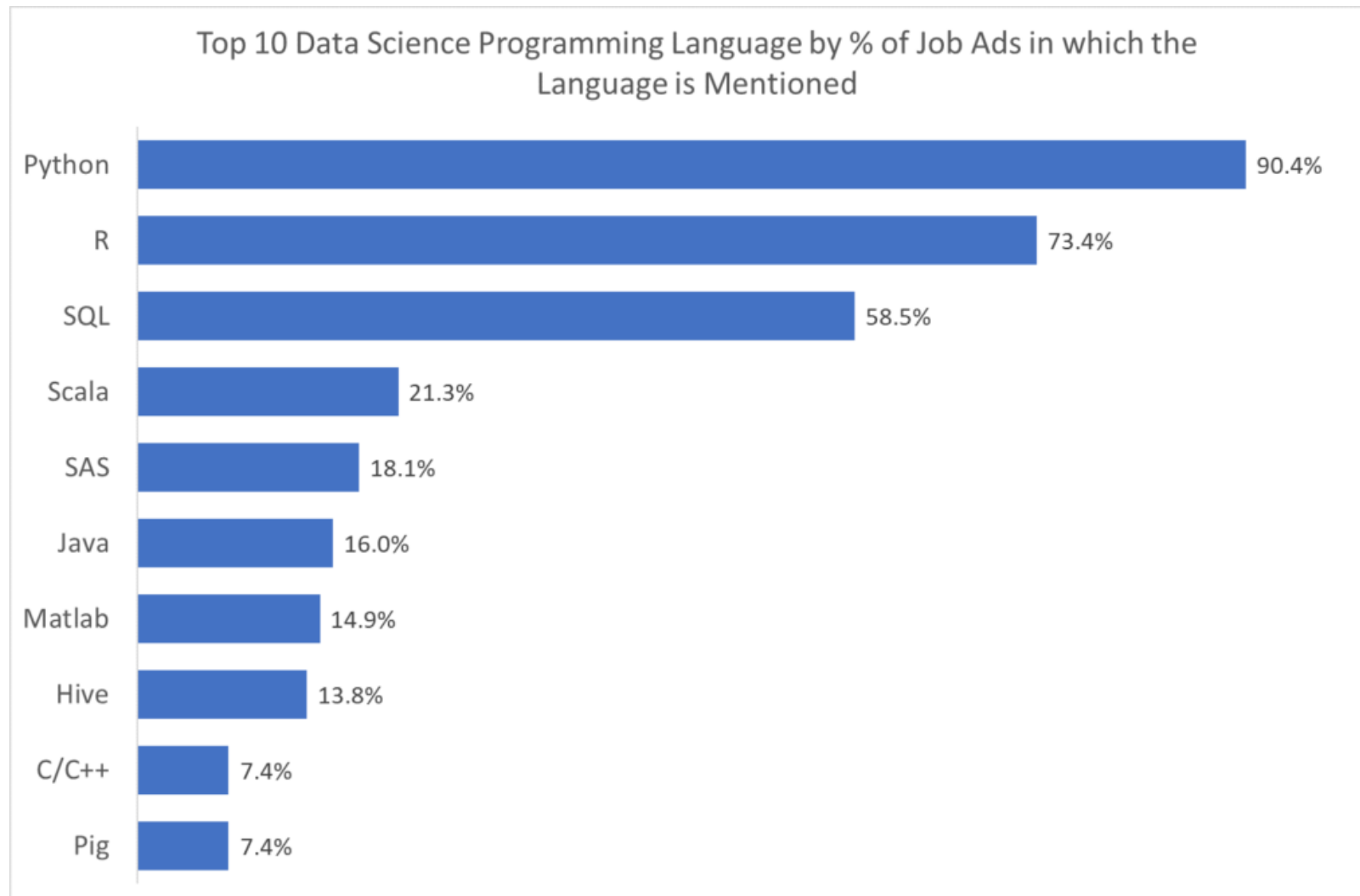
Tại sao sử dụng Python trong phân tích dữ liệu?



Tại sao sử dụng Python trong phân tích dữ liệu?

- Python là ngôn ngữ đơn giản, dễ sử dụng, dễ làm quen với người mới bắt đầu và có tính mềm dẻo cao.
- Python có một cộng đồng khổng lồ đóng góp lớn cho các thư viện mã nguồn mở của nó.
- Python có một lượng lớn thư viện hỗ trợ cho việc phân tích dữ liệu.
- Python là một ngôn ngữ độc lập về nền tảng, có thể được dùng trong nhiều hệ điều hành (Windows, Mac, Linux,...)
- Python có hai thư viện Matplotlib và Seaborn cung cấp nhiều công cụ và được sử dụng rộng rãi trong việc hình ảnh hóa (visualization) dữ liệu.

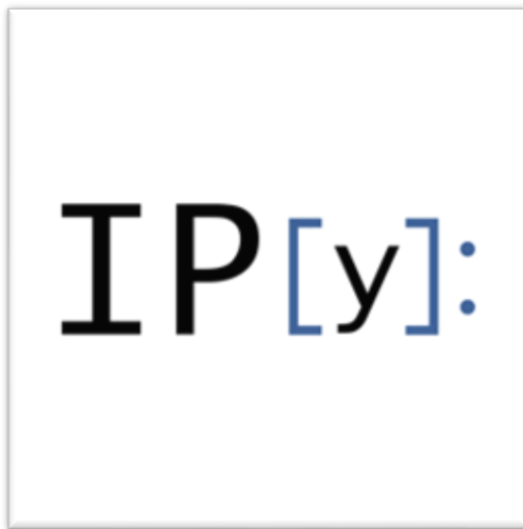
Tại sao sử dụng Python trong phân tích dữ liệu?



<https://towardsdatascience.com/team-r-or-team-python-2f8cf04310e6>

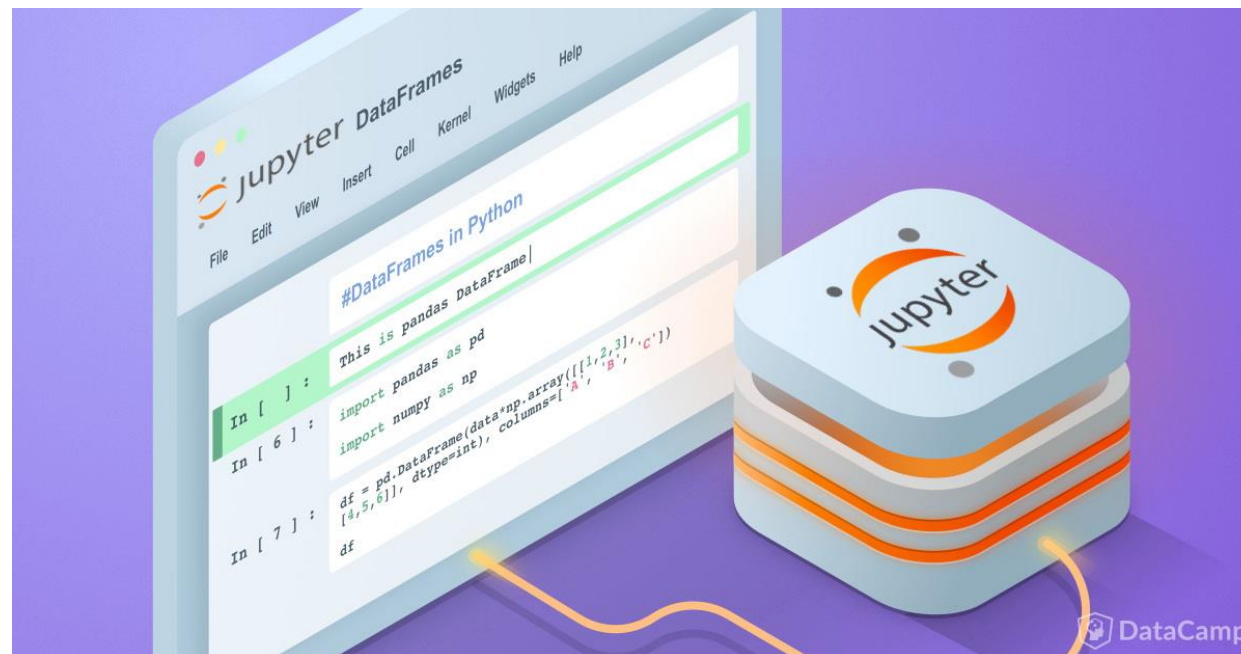
Giới thiệu về IPython

- IPython là một command shell giúp cho việc lập trình tương tác (interactive computing) với nhiều ngôn ngữ trong đó đặc biệt là Python.
- IPython được thiết kế nhằm tối đa hóa năng suất trong việc lập trình tương tác cũng như phát triển phần mềm.



Giới thiệu về Jupyter Notebook

- Jupyter Notebook là một ứng dụng web mã nguồn mở cho phép bạn tạo tài liệu chứa code, phương trình, hình ảnh, dữ liệu được trực quan hóa dựa trên ipython.
- Khi cài Jupyter Notebook, Ipython sẽ được mặc định cài theo.



Cài đặt IPython

- Sử dụng Anaconda để cài đặt và làm việc với Ipython và Jupyter Notebook.



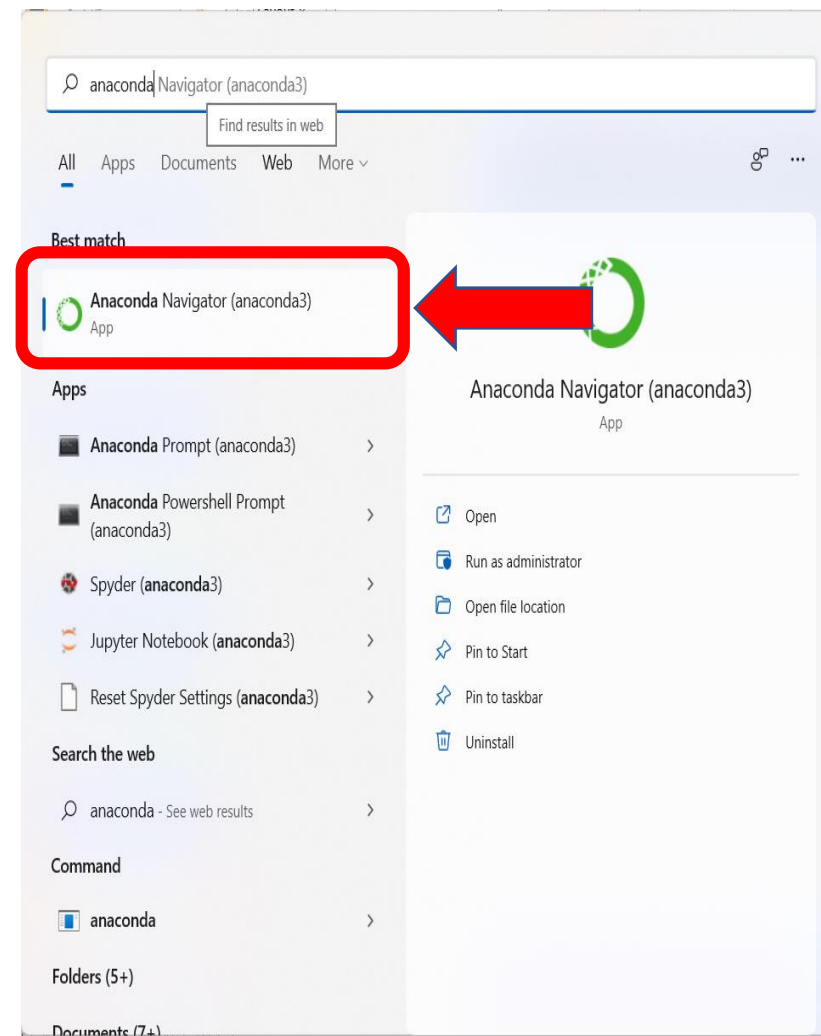
Anaconda là gì?

- Anaconda là một trình phân phối (distribution) của Python và R cho mục đích điện toán khoa học (scientific computing) như khoa học dữ liệu (data science), xử lý dữ liệu lớn (large-scale data processing),... nhằm đơn giản hóa quá trình quản lý gói (package management) và triển khai gói (package deployment).
- Truy cập [Anaconda | The World's Most Popular Data Science Platform](#) để có tải xuống.

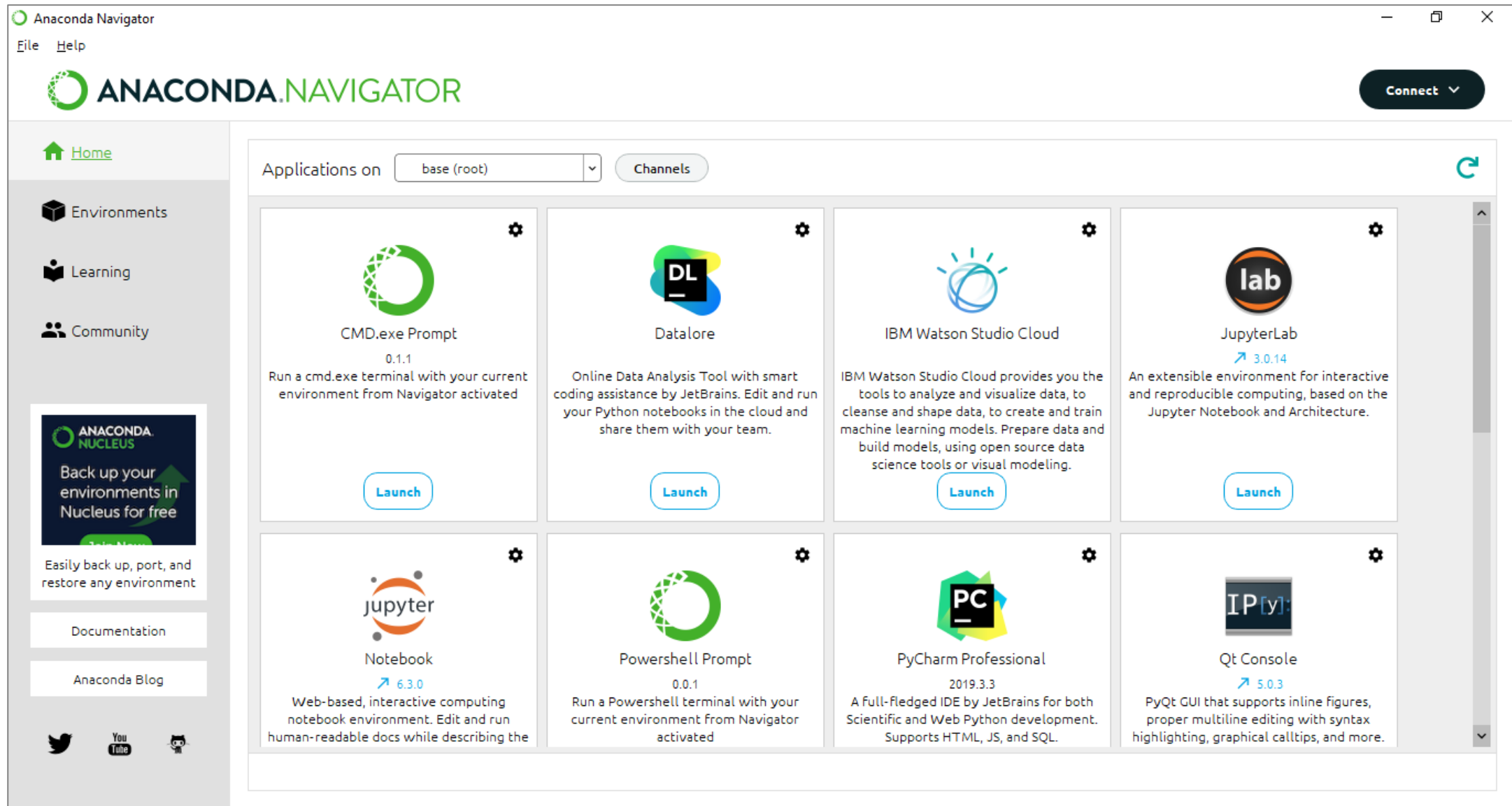


Trình điều hướng Anaconda (Anaconda Navigator)

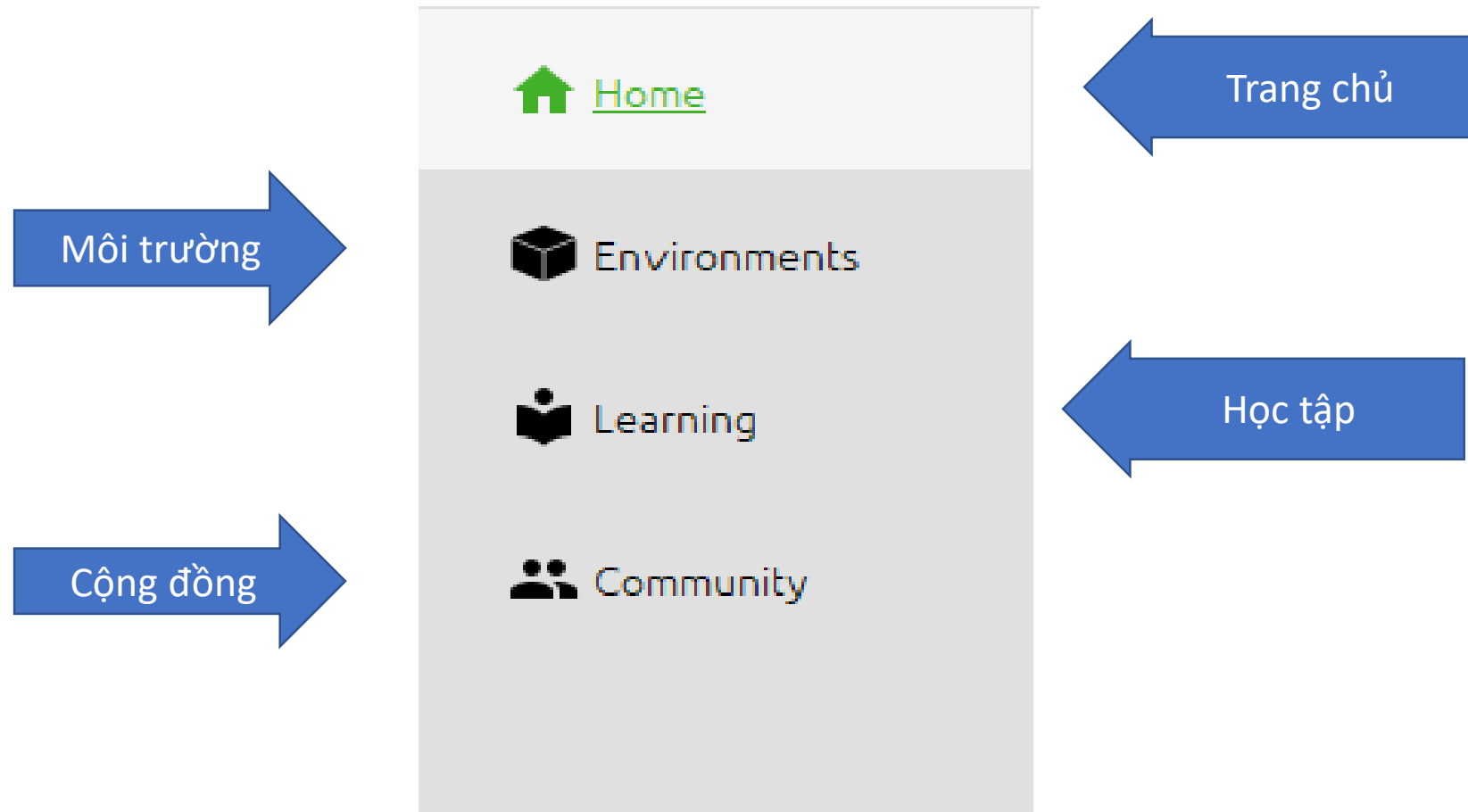
- Trình điều hướng Anaconda (Anaconda Navigator) là một trình giao diện đồ họa được thiết kế để làm việc với các tác vụ với Anaconda.
- Sau khi đã cài đặt Anaconda xong, chọn **Start => Search** rồi tìm kiếm Anaconda Navigator như hình bên để mở trình điều hướng Anaconda.



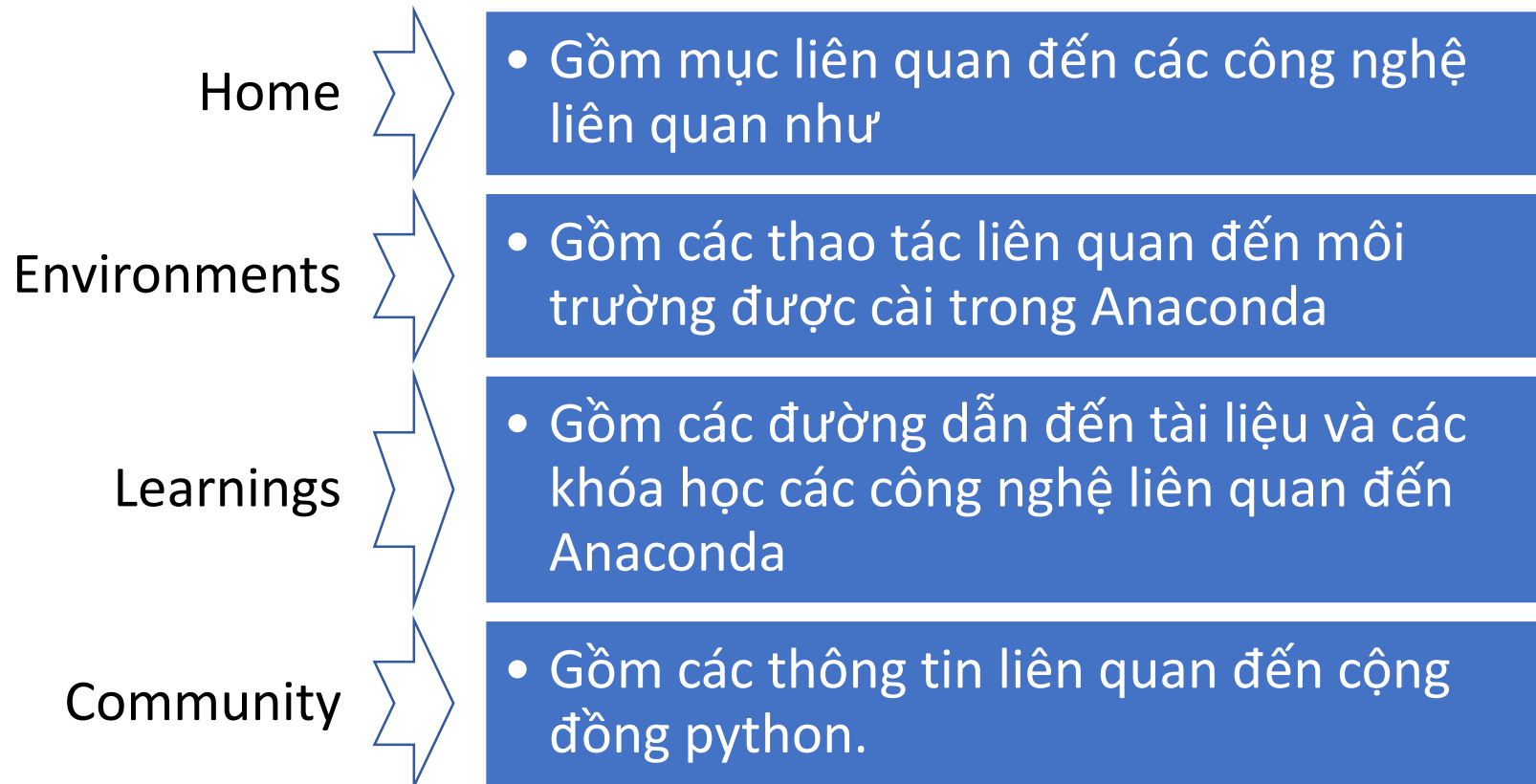
Giao diện của Anaconda



Các mục trong Anaconda Navigator



Các mục trong Anaconda Navigator



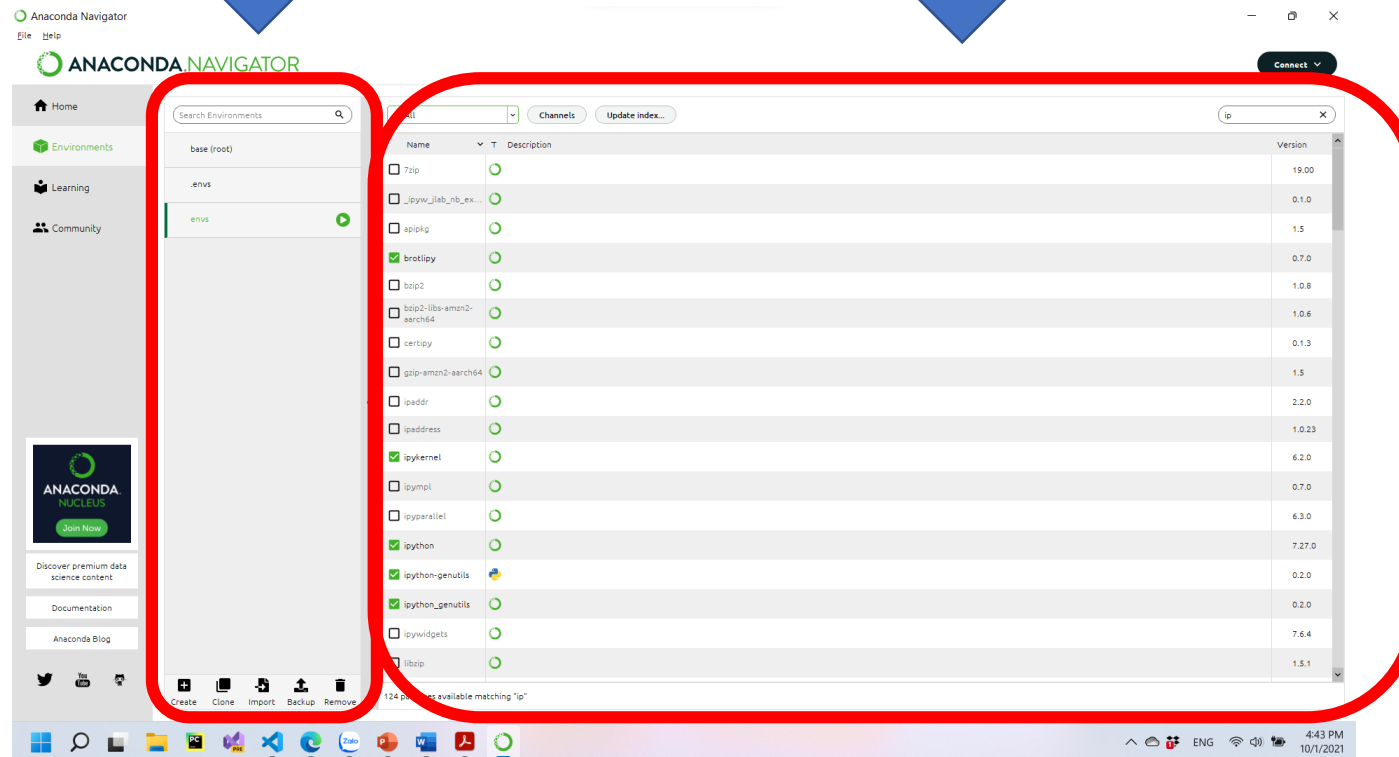
Tạo môi trường với Anaconda

- Môi trường Python là một tập hợp các thư viện (library), trình biên dịch (interpreter) và kịch bản (scripts) được cài đặt tách biệt với môi trường khác và môi trường cài đặt sẵn trong máy tính (Python được cài đặt với hệ điều hành).
- Môi trường được thiết kế để nhằm thống nhất các phiên bản gói (package version), tránh trường hợp chương trình không chạy được khi thay đổi phiên bản thư viện.

Môi trường trong Anaconda

Các môi trường được cài đặt với Anaconda

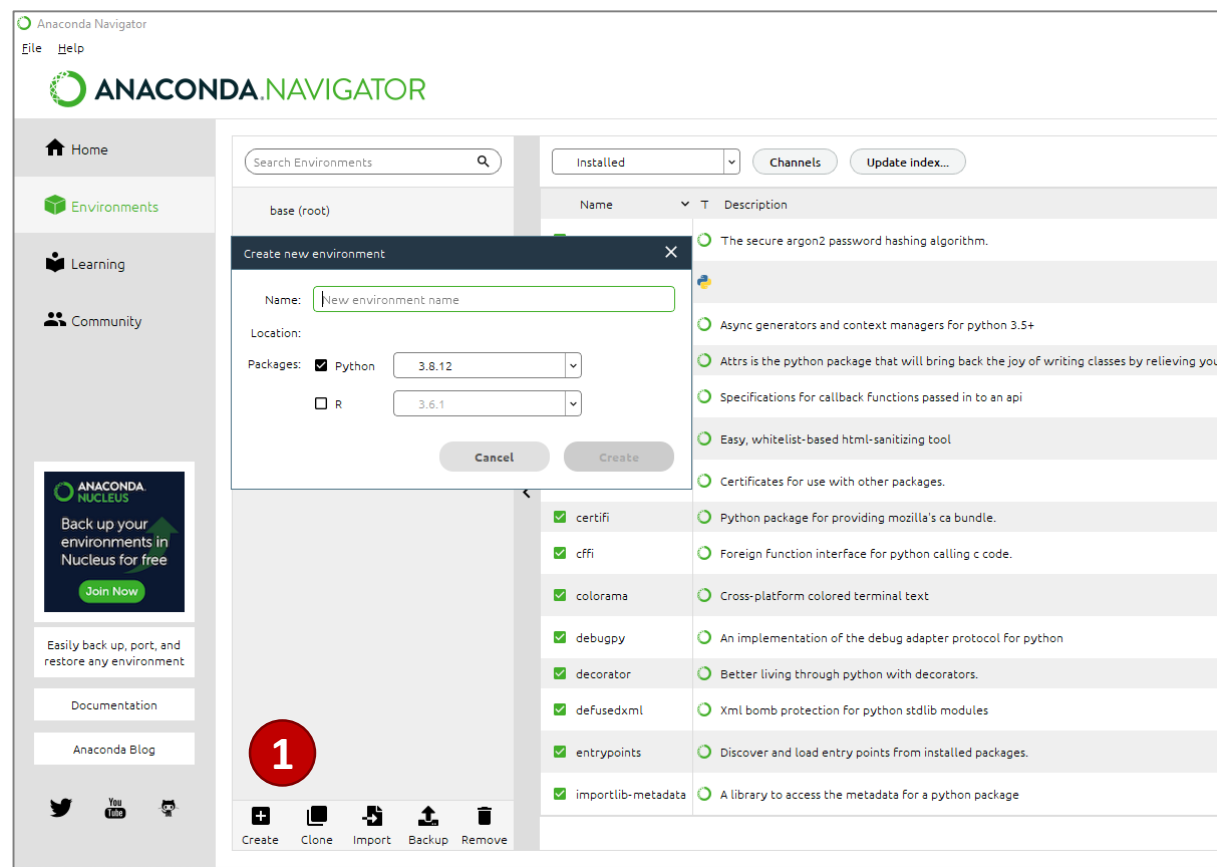
Danh sách các package được cài đặt trong môi trường tương ứng



Tạo môi trường bằng Anaconda

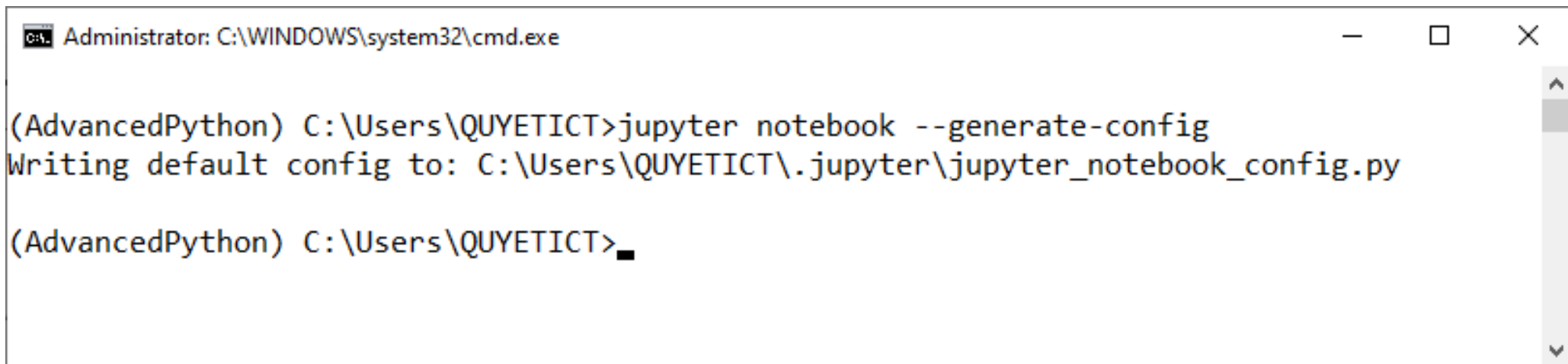
● Bước 1: Tạo môi trường có tên là **AdvancedPython**

● Click **Create**



Tạo môi trường bằng Anaconda

- Bước 2: Tạo file cấu hình mặc định cho môi trường
 - Click chuột trái vào biểu tượng kích hoạt môi trường **AdvancedPython**
 - Chọn **Open Terminal**
 - Gõ lệnh: **jupyter notebook --generate-config**



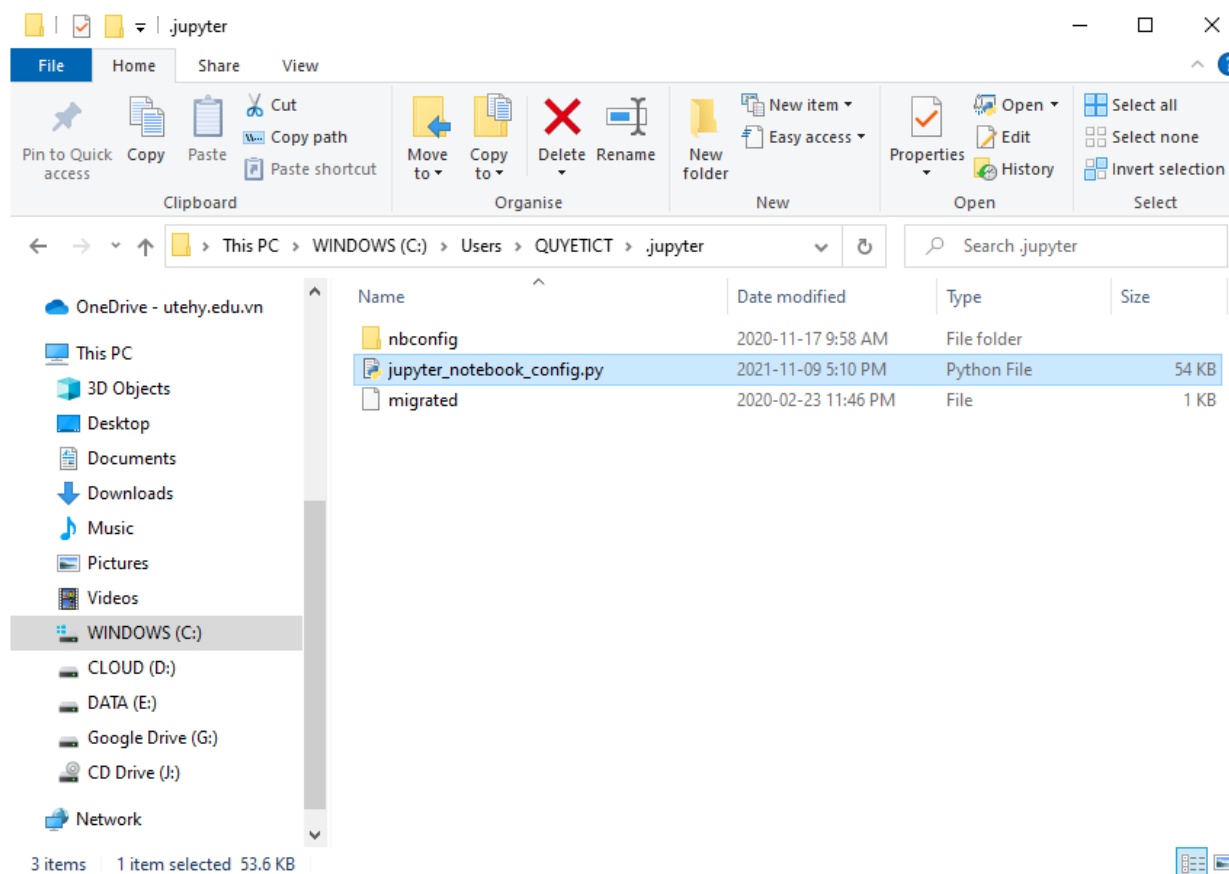
```
Administrator: C:\WINDOWS\system32\cmd.exe

(AdvancedPython) C:\Users\QUYETICT>jupyter notebook --generate-config
Writing default config to: C:\Users\QUYETICT\.jupyter\jupyter_notebook_config.py

(AdvancedPython) C:\Users\QUYETICT>_
```

Tạo môi trường bằng Anaconda

- Bước 4: Mở file **jupyter_notebook_config.py** để thiết lập thư mục làm việc mặc định của jupyter notebook



Tạo môi trường bằng Anaconda

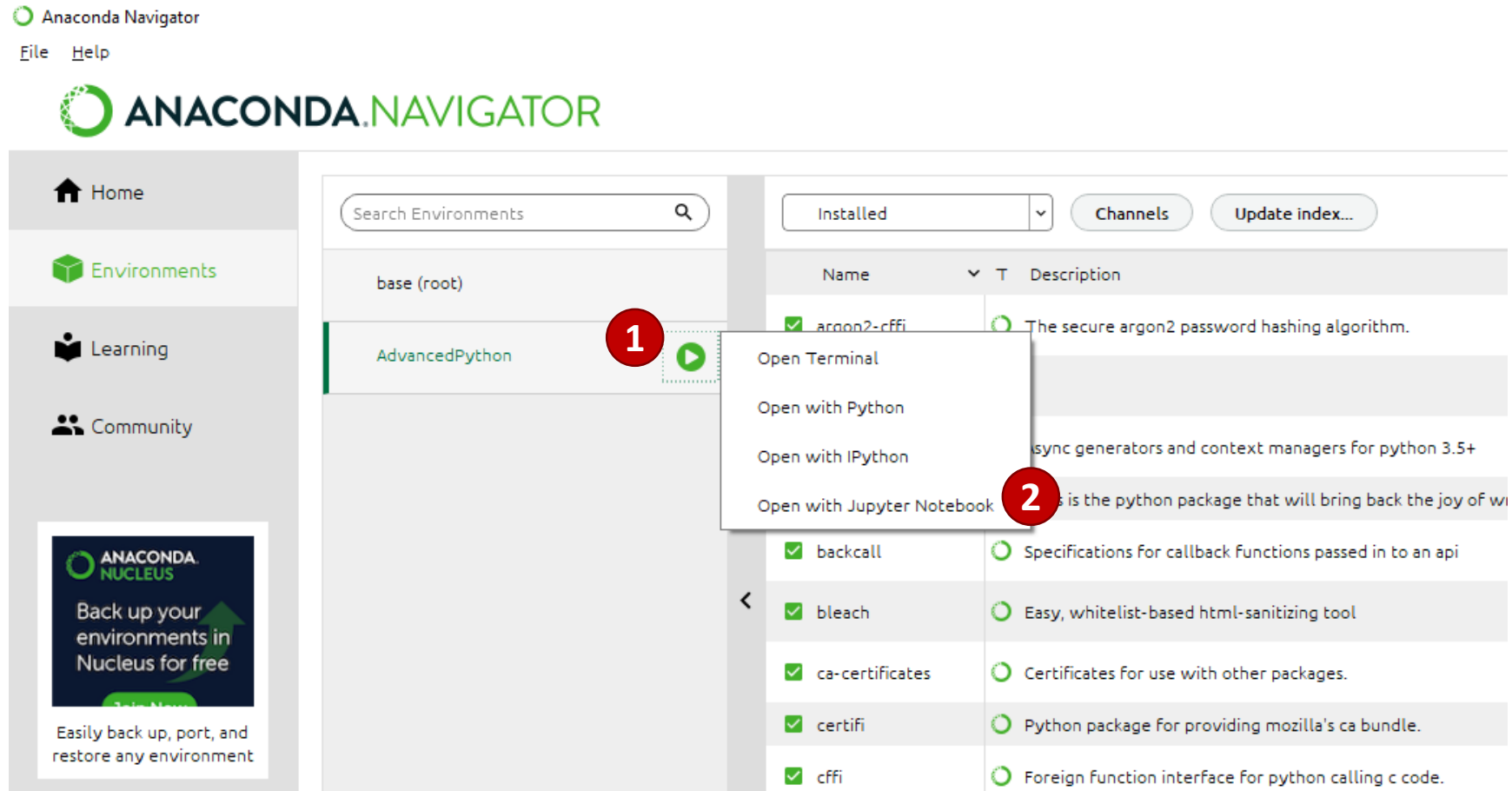
● Bước 5: Cấu hình thư mục mặc định cho môi trường

```
384
385  ## Dict of Python modules to load as notebook server extensions. Entry values can
386  #   be used to enable and disable the loading of the extensions. The extensions
387  #   will be loaded in alphabetical order.
388  #   Default: {}
389  # c.NotebookApp.nbserver_extensions = {}
390
391  ## The directory to use for notebooks and kernels.
392  #   Default: ''
393  c.NotebookApp.notebook_dir = 'D:\AdvancedPython'
394
395  ## Whether to open in a browser after starting.
396  #
397  #       The specific browser used is platform dependent and
398  #       determined by the python standard library `webbrowser`
399  #       module, unless it is overridden using the --browser
400  #       (NotebookApp.browser) configuration option.
401  #   Default: True
402  # c.NotebookApp.open_browser = True
```

● Bước 6: Tạo một thư mục mới để chứa project có tên **AdvancedPython** tại ổ D.

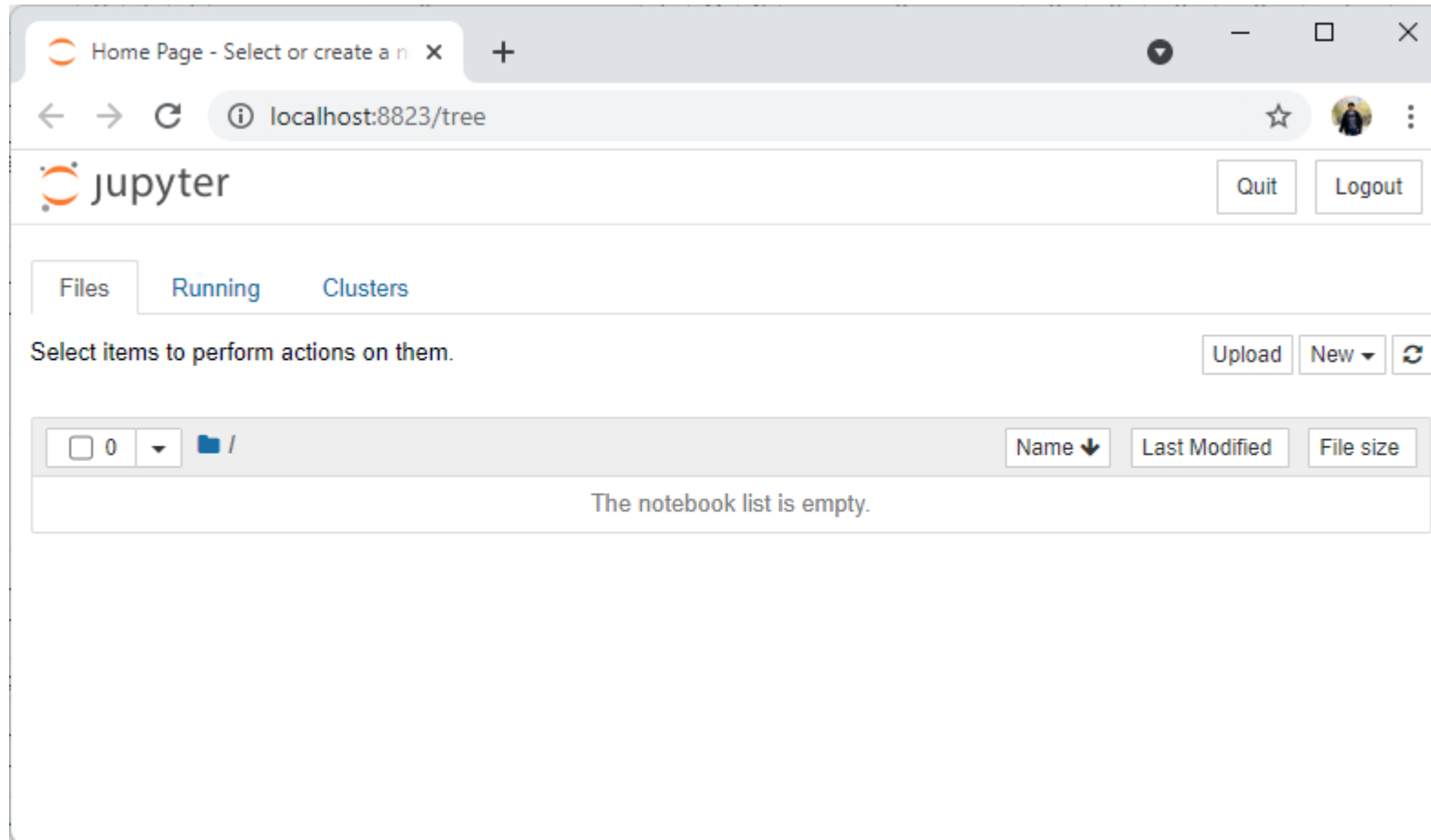
Giao diện Jupyter notebook

● Bước 7: Mở Jupyter Notebook trên trình duyệt



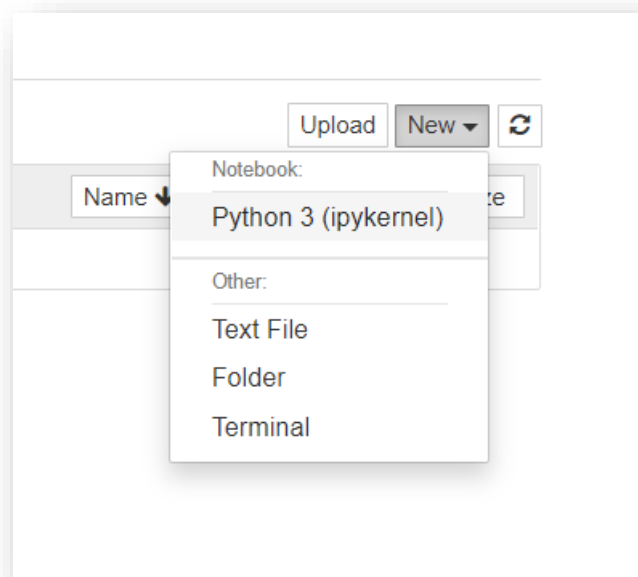
Giao diện Jupyter notebook

● Bước 7: Mở Jupyter Notebook trên trình duyệt



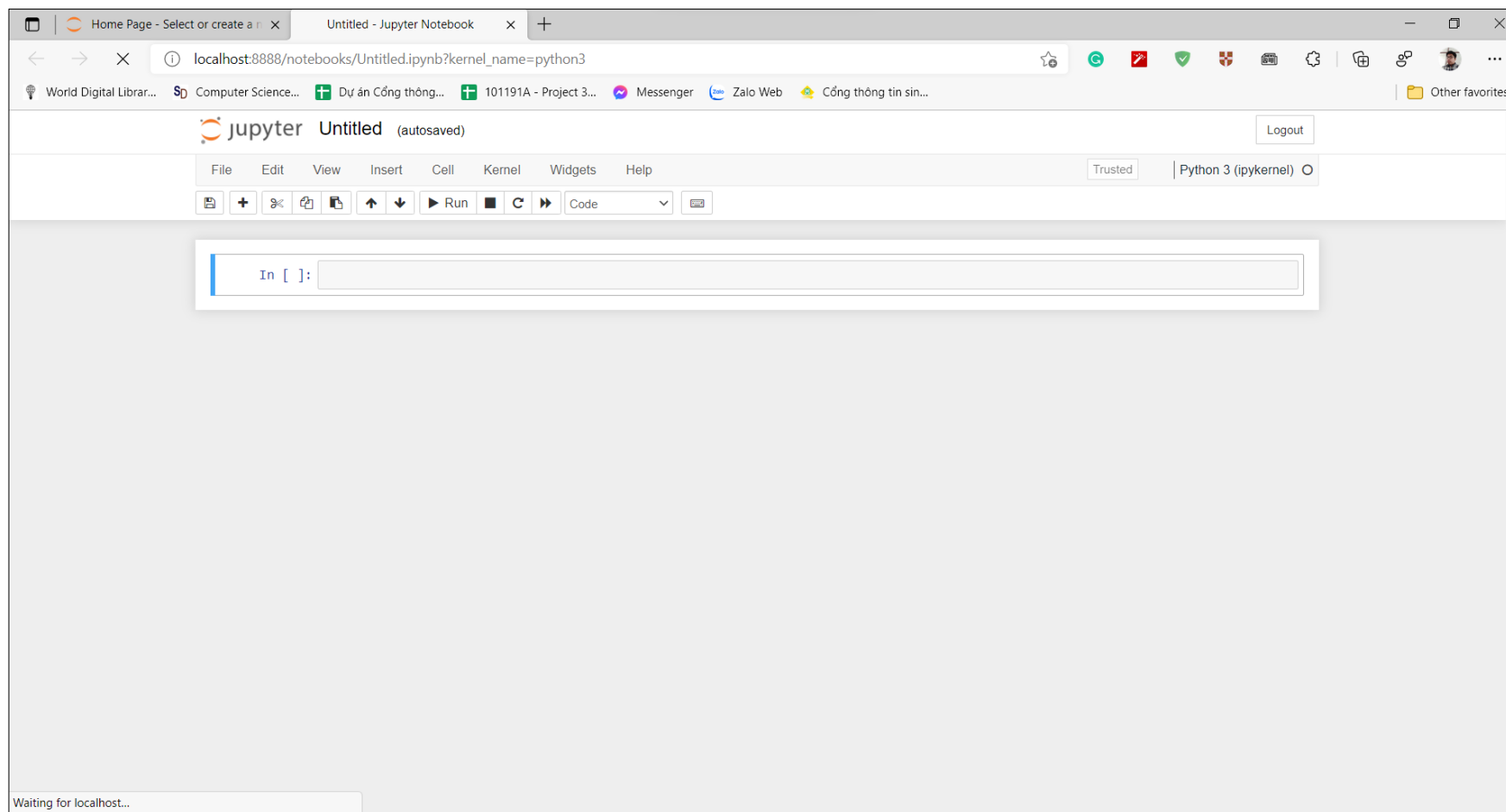
Viết chương trình Python bằng Jupyter Notebook

- Chọn **New => Python 3 (ipykernel)** để tạo một file python notebook mới.



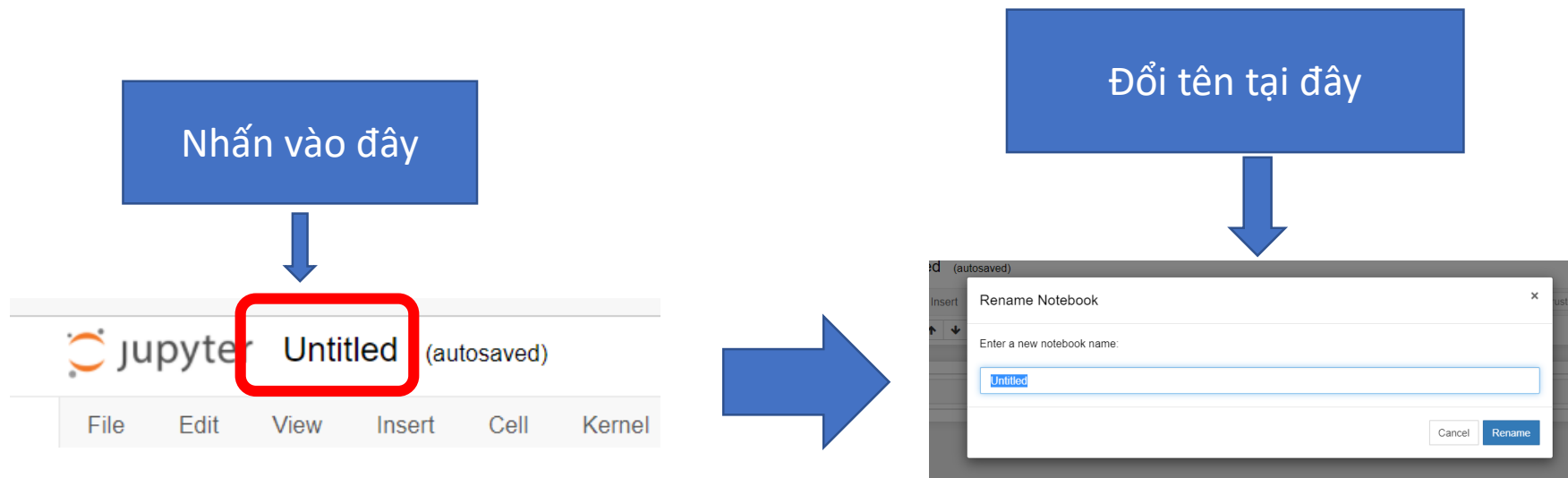
Giao diện của một Notebook

- File python notebook vừa được tạo sẽ có dạng như sau:



Đặt tên cho Notebook

- Mọi Notebook mới được tạo đều có tên mặc định là Untitled (Chưa đặt tên).
- Để đặt/đổi tên cho một notebook ta chọn tên của notebook trên đầu trang. Một hộp thoại hiện ra cho phép bạn sửa đổi tên theo ý muốn



Các file đang chạy trong Jupyter Notebook

- Các file màu xanh là các file đang chạy. Bạn có thể chuyển qua Running chọn Shut down để ngừng các file đang chạy.



The screenshot displays the Jupyter Notebook interface with the 'Running' tab selected. The top section, 'Select items to perform actions on them.', shows a list of files: 'envs' (16 hours ago) and 'KHMT.ipynb' (Running, 18 minutes ago, 588 B). The bottom section, 'Currently running Jupyter processes', shows a list of processes: 'terminals/1' (Shutdown) and 'KHMT.ipynb' (Python 3 (ipykernel), Shutdown, seconds ago).

Cell của một Notebook

- Cell: Một notebook bao gồm một chuỗi các cell. Có 3 loại cell được Jupyter Notebook hỗ trợ.
 - Code cell: Là kiểu cell được dùng nhiều nhất. Dùng để nhập và thực thi code.

```
In [3]: print("Xin chào! Tôi là sinh viên Khoa học máy tính")  
Xin chào! Tôi là sinh viên Khoa học máy tính
```

- Markdown Cell: Là loại cell dùng để hiển thị văn bản hoặc đánh văn bản. Khi chạy loại cell này sẽ hiển thị văn bản cho người sử dụng.

```
# Bài 1
```

Bài 1 ¶

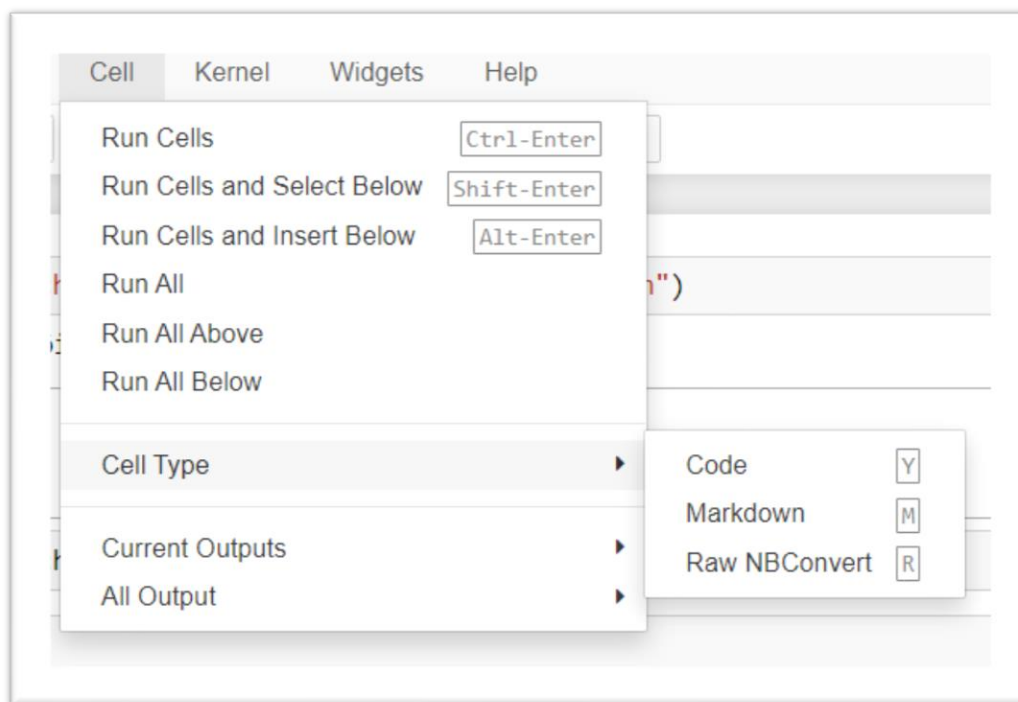
Cell của một Notebook

- RawNBConvert: Là loại cell dùng để hiển thị code, html. Nội dung được đưa vào trong RawNBConvert Cell sẽ không bị thay đổi khi chạy.

```
print("Xin chào! Chúng tôi là sinh viên khoa học máy tính")
```

Cell của một Notebook

- Để đổi loại cell của một Notebook ta chọn cell cần đổi rồi chọn **Cell => Cell Type** rồi chọn loại cell mong muốn.



Kernel (Nhân) trong Jupyter Notebook

- Kernel là một trình thông dịch và thực thi code trong Jupyter Notebook. Ipython Kernel được cài đặt sẵn cùng với Jupyter Notebook.
- Kernel chạy và liên kết các cell code với nhau chứ không chạy các cell riêng lẻ.

4. Các lệnh trong IPython

- IPython có nhiều lệnh đặc biệt được gọi là “Magic command” nhằm đơn giản hóa các công việc thông thường và giúp người dùng có thể dễ dàng điều chỉnh hành vi của hệ thống IPython.
- Một magic command là các lệnh có tiền tố là dấu “%”.
 - Ví dụ: Lệnh `%timeit` dưới đây được dùng để đo thời gian thực thi của một câu lệnh hay một cell.

```
In [10]: %timeit print("Chúng tôi là sinh viên khoa học máy tính")
Chúng tôi là sinh viên khoa học máy tính
Chúng tôi là sinh viên khoa học máy tính
Chúng tôi là sinh viên khoa học máy tính
Chúng tôi là sinh viên khoa học máy tính
Chúng tôi là sinh viên khoa học máy tính
Chúng tôi là sinh viên khoa học máy tính
Chúng tôi là sinh viên khoa học máy tính
Chúng tôi là sinh viên khoa học máy tính
Chúng tôi là sinh viên khoa học máy tính
Chúng tôi là sinh viên khoa học máy tính
Chúng tôi là sinh viên khoa học máy tính
Chúng tôi là sinh viên khoa học máy tính
Chúng tôi là sinh viên khoa học máy tính
Chúng tôi là sinh viên khoa học máy tính
Chúng tôi là sinh viên khoa học máy tính
Chúng tôi là sinh viên khoa học máy tính
92.7 µs ± 1.64 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

4. Các lệnh trong IPython

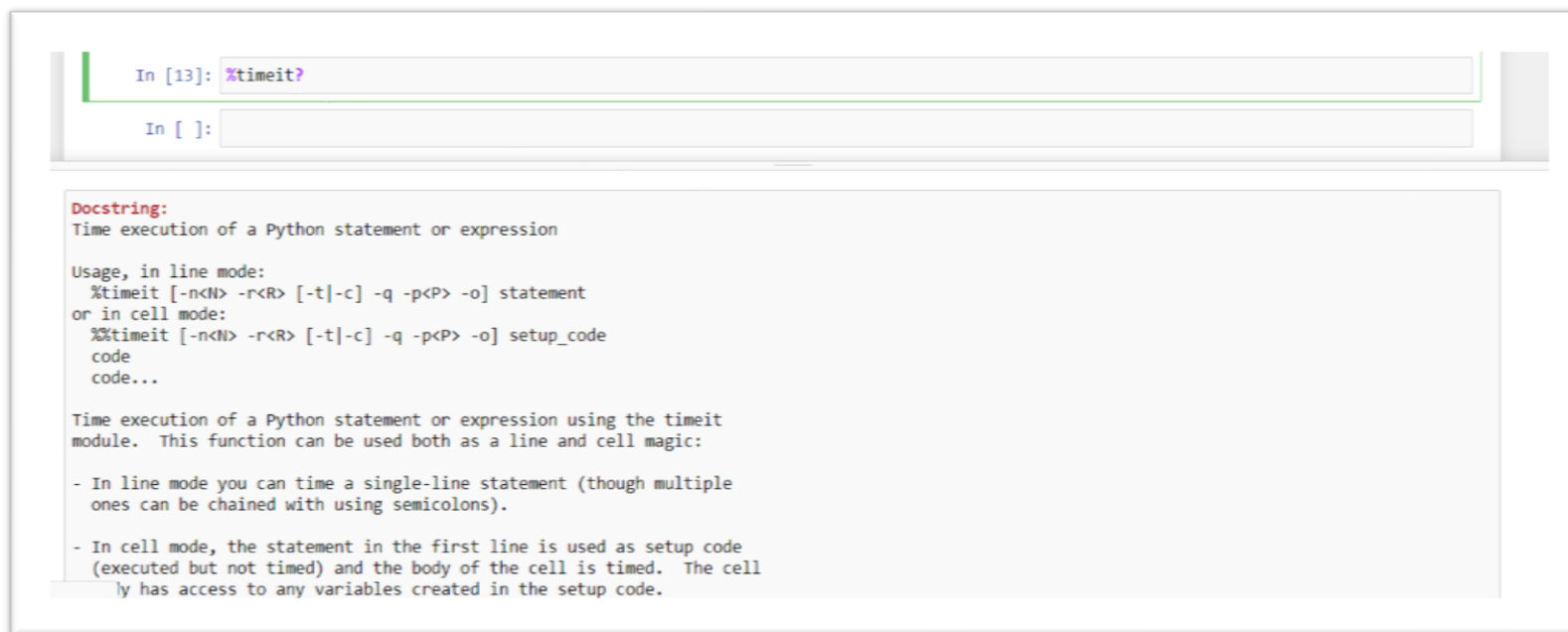
- Dấu “%” đơn cho phép lệnh thực hiện trên một dòng đơn còn dấu “%%” cho phép lệnh thực hiện trên toàn bộ cell.
 - Lệnh %%timeit khi chạy trên toàn bộ cell

```
In [16]: %%timeit
for i in range(0,100):
    print("Chúng tôi là sinh viên khoa học máy tính")

Chúng tôi là sinh viên khoa học máy tính
Chúng tôi là sinh viên khoa học máy tính
Chúng tôi là sinh viên khoa học máy tính
Chúng tôi là sinh viên khoa học máy tính
Chúng tôi là sinh viên khoa học máy tính
Chúng tôi là sinh viên khoa học máy tính
Chúng tôi là sinh viên khoa học máy tính
Chúng tôi là sinh viên khoa học máy tính
Chúng tôi là sinh viên khoa học máy tính
Chúng tôi là sinh viên khoa học máy tính
Chúng tôi là sinh viên khoa học máy tính
Chúng tôi là sinh viên khoa học máy tính
Chúng tôi là sinh viên khoa học máy tính
Chúng tôi là sinh viên khoa học máy tính
Chúng tôi là sinh viên khoa học máy tính
Chúng tôi là sinh viên khoa học máy tính
Chúng tôi là sinh viên khoa học máy tính
Chúng tôi là sinh viên khoa học máy tính
Chúng tôi là sinh viên khoa học máy tính
9.32 ms ± 287 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```


Hậu tố “?”

- Dùng dấu “?” đằng sau các câu lệnh để hiển thị chi tiết chức năng và cú pháp của nó.
 - Ví dụ 1: Dấu “?” sau lệnh %timeit hiển thị chức năng và cú pháp của nó.



```
In [13]: %timeit?

In [ ]:

Docstring:
Time execution of a Python statement or expression

Usage, in line mode:
%timeit [-n<N> -r<R> [-t|-c] -q -p<P> -o] statement
or in cell mode:
%%timeit [-n<N> -r<R> [-t|-c] -q -p<P> -o] setup_code
code
code...

Time execution of a Python statement or expression using the timeit
module. This function can be used both as a line and cell magic:

- In line mode you can time a single-line statement (though multiple
ones can be chained with using semicolons).

- In cell mode, the statement in the first line is used as setup code
(executed but not timed) and the body of the cell is timed. The cell
  ly has access to any variables created in the setup code.
```

Hậu tố “?”

- Ví dụ 2: Dùng dấu “?” sau lệnh print để hiển thị cú pháp cũng như chức năng của lệnh print.

```
In [14]: print?
```

```
In [ ]: |
```

Docstring:

```
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
```

Prints the values to a stream, or to sys.stdout by default.

Optional keyword arguments:

file: a file-like object (stream); defaults to the current sys.stdout.

sep: string inserted between values, default a space.

end: string appended after the last value, default a newline.

flush: whether to forcibly flush the stream.

Type: builtin_function_or_method

Hậu tố “??”

- Hậu tố “??” cho phép hiển thị mã nguồn (source code) của các magic command nếu có thể. Nếu không thể hiển thị mã nguồn, hậu tố “??” sẽ hiển thị chức năng và cú pháp của lệnh ý như hậu tố “?”.
- Dùng hậu tố “??” để hiển thị mã nguồn của lệnh magic command.

```
In [8]: %timeit??

In [ ]:
```

```
Source:
@skip_doctest
@no_var_expand
@line_cell_magic
@needs_local_scope
def timeit(self, line='', cell=None, local_ns=None):
    """Time execution of a Python statement or expression

    Usage, in line mode:
    %timeit [-n<N> -r<R>] [-t|-c] -q -p<P> -o] statement
    or in cell mode:
    %timeit [-n<N> -r<R>] [-t|-c] -q -p<P> -o] setup_code
    code
    code...

    Time execution of a Python statement or expression using the timeit
    module. This function can be used both as a line and cell magic:

    - In line mode you can time a single-line statement (though multiple
      lines can be obtained with the magic command)
```

Một số magic command phổ biến trong IPython

Lệnh	Mô tả
%magic	Hiển thị chi tiết mô tả các lệnh magic command sẵn có
%debug	Kích hoạt debugger của hệ thống
%paste	Thực thi lệnh code đã được format trong clipboard
%matplotlib	Thiết đặt tương tác với thư viện matplotlib
%run script.py	Chạy một file python có tên “script.py” trong python.
%time	Hiển thị thời gian chạy của một lệnh
%timeit	Chạy một dòng lệnh nhiều lần để tính toán chính xác thời gian chạy của một câu lệnh

5. Lịch sử nhập xuất trong Python

- Để đơn giản hóa cho việc nhập và xuất biến, Ipython lưu trữ tham chiếu đến hai biến nhập (input), giá trị mà người dùng nhập, và xuất (output), giá trị được trả về vào trong các biến đặc biệt.
- Hai giá trị trả về gần nhất được lần lượt lưu trong biến “_” (gạch dưới đơn) và biến “__” (gạch dưới kép).
 - Hai giá trị trả về gần nhất tại dòng out được trả về trong hai biến đặc biệt.

```
In [5]: a=14  
a
```

```
Out[5]: 14
```

```
In [6]: b=2  
b
```

```
Out[6]: 2
```

```
In [9]: print(str(_) + "+" + str(__))  
2+14
```

5. Lịch sử nhập xuất trong Python

- Nội dung các cell mà người dùng nhập vào (input) code cell của ipython sẽ được lưu trong các biến `_iX` (trong đó `X` là số thứ tự mà cell đó thực hiện trong chương trình) dưới dạng một chuỗi (string).
 - Ví dụ: Hình dưới thể hiện biến `_i9` lưu nội dung của cell được thực thi tại dòng cell thứ 9 dưới dạng một chuỗi.

```
In [9]: print(str(_) + "+" + str(__))
```

```
2+14
```

```
In [21]: _i9
```

```
Out[21]: 'print(str(_) + "+" + str(__))'
```

5. Lịch sử nhập xuất trong Python

- Tương ứng với mỗi biến chứa input của mỗi cell ta sẽ có một biến chứa output của cell đó, biến đó là `_X` (X là số thứ tự mà cell đó được thực thi khi chạy chương trình). Ví dụ `_i5` sẽ có một biến `_5` tương ứng.
 - Lưu ý: Các biến output chỉ còn được lưu khi còn chữ “Out” ở đầu các cell.

```
In [5]: a=14
a
Out[5]: 14

In [25]: _5
Out[25]: 14
```

5. Lịch sử nhập xuất trong python

- Một số magic command có thể được dùng để thao tác với biến input và output:
 - %reset: Xóa hết tất cả các biến nhập và xuất được lưu trong bộ nhớ.
 - %hist: Hiển thị lịch sử các biến input.
 - %xdel: gỡ bỏ tất cả tham chiếu đối với một đối tượng cụ thể khỏi hệ thống Ipython.

6. Xử lý ngoại lệ trong IPython

- Ngoại lệ (Exceptions) là một công cụ mà các lập trình viên sử dụng để miêu tả lỗi lớn cho một chương trình làm chương trình bị dừng lại hoặc không thể chạy tiếp.
- Ngoại lệ có thể xảy ra vì lỗi của chương trình, lỗi của người dùng hoặc đơn giản do một số yếu tố bất ngờ như mất kết nối Internet.
- Ngoại lệ bản chất là một đối tượng chứa thông tin về lỗi chương trình.
- Ngoại lệ thường được định nghĩa bằng kiểu (type) của nó kèm theo lời nhắn (message) miêu tả về lỗi chương trình.

Một số kiểu ngoại lệ trong python

- **SyntaxError** (Lỗi cú pháp): là lỗi mà lập trình viên gõ sai cú pháp.
- **AttributeError** (Lỗi thuộc tính): là lỗi mà lập trình viên cố gắng truy cập vào một thuộc tính của một object không tồn tại.
- **KeyError** (Lỗi từ khóa): là lỗi khi lập trình viên cố gắng truy cập vào một key không tồn tại trong dictionary.
- **TypeError** (Lỗi kiểu dữ liệu): lỗi xảy ra khi gán không đúng kiểu dữ liệu.
- **IOError** (Lỗi nhập xuất dữ liệu): Lỗi xảy ra khi python không thể truy cập vào file trong bộ nhớ.
- **ImportError** (Lỗi import): Là lỗi import thất bại.

Ngoại lệ

- Vì ngoại lệ là một object nên ta có thể khởi tạo nó như một instance của các object khác và có thể gán giá trị của nó cho một biến như hình dưới.

```
In [11]: a=Exception("Chương trình bị lỗi")
```

Raise ngoại lệ

- Một ngoại lệ có thể được raised (trồi lên) hoặc thrown (ném) khi xảy ra. Khi được raised, chương trình theo mặc định sẽ dừng lại và thông báo lỗi dựa theo ngoại lệ được raised. Từ khóa **raise** được sử dụng để raise exception.

```
In [11]: a=Exception("Chương trình bị lỗi")
```

```
In [12]: raise a
```

```
-----  
Exception                                Traceback (most recent call last)  
~\AppData\Local\Temp\ipykernel_11456\4052815451.py in <module>  
----> 1 raise a  
  
Exception: Chương trình bị lỗi
```

```
In [ ]:
```

Raise ngoại lệ

- Lý do mà từ khóa **raise** được sử dụng là vì các hàm trong python khi gọi lẫn nhau sẽ được đặt trong một ngăn xếp (stack) để xử lý từ dưới lên trên. Nếu một ngoại lệ xảy ra ở tầng dưới của stack, nó sẽ được đẩy lên các hàm ở tầng trên để mỗi hàm đều có cơ hội để xử lý nó. Việc đẩy lên được gọi là raise.
- Từ khóa **raise** chỉ được theo sau bởi một object loại **Exception** và các class được kế thừa của nó do vậy mọi loại ngoại lệ trong Python đều là Exception hoặc là các lớp kế thừa của nó.

Xử lý ngoại lệ trong python

- Khi ngoại lệ xảy ra, chương trình sẽ bị dừng lại (crash), để ngăn chặn việc này xảy ra, lập trình viên sẽ tiến hành xử lý ngoại lệ bằng cách đặt đoạn code cần xử lý ngoại lệ vào trong vòng try và xử lý từng loại ngoại lệ trong vòng except.

```
: try:  
    "Đoạn code cần được xử lý ngoại lệ"  
except TypeError as e:  
    "Chạy đoạn code cần được xử lý. Có thể có nhiều dòng except "  
finally:  
    "Đoạn code cuối cùng sẽ được chạy khi có ngoại lệ xảy ra bất kể ngoại lệ có được xử lý hay không"
```

Ví dụ về việc xử lý ngoại lệ

- Dưới đây là đoạn code xử lý ngoại lệ yêu cầu người dùng nhập vào một số. Chương trình sẽ yêu cầu người dùng nhập lại nếu giá trị nhập vào không phải là số nguyên.

```
: while(True):  
    print("Hãy nhập một số nguyên")  
    try:  
        a=int(input())  
        break  
    except:  
        print("Dữ liệu được nhập có định dạng không đúng! Xin hãy nhập lại!")
```

Q & A