Jupyter console

The [Jupyter console](https://github.com/jupyter/jupyter_console" \t "_blank), formerly known as IPython, is an enhanced Python interpreter. From our earlier missions, you may recall that by typing python on the command line, you get access to an interactive shell that lets you write and execute Python code. Jupyter console enhances this shell, and adds several niceties that make working with data easier.

Generally, it's useful to use the shell in situations where you need to quickly test some code you're writing. This happens frequently when you're writing data analysis scripts. It can also be used to quickly explore datasets and do basic analysis. Another use case is prototyping code before later saving it to a script file.

The main difference between Jupyter console and Jupyter notebook is that the console functions in interactive mode. Whenever you type a line of code, it is immediately executed, and you can see the results. If you want to write medium-length pieces of code, do deep exploration of a dataset to tell a story, the notebook is better. If you want to test out code you're writing, or run quick commands, the console is better.

The Jupyter project is in the midst of rebranding from IPython to Jupyter. Depending on the version of Jupyter you have installed, you can access the console by typing either jupyter console or ipythonat the command line.

Instructions

* Open the Jupyter console by typing ipython.
* Once you access it, you can run Python commands. Type print(10) to see what happens.
* Exit Jupyter console by typing exit.

Answer

/home/dq$ ipython

Python 3.4.3 (default, Nov 28 2017, 16:41:13)

Type "copyright", "credits" or "license" for more information.

IPython 4.2.0 -- 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**]: print(10)

10

In [**2**]: exit

**Getting help**

Jupyter console has a robust built-in help system. You can get help in several ways:

* You can type ? after starting the console. This will display help about Jupyter. You can exit by typing q.
* You can type %quickref. This is a **magic** that will tell you some useful commands. We'll talk more about Jupyter magics shortly.
* If you want information about a variable, just type the name of the variable, followed by ?. For information on the dq variable, you'd type dq?.
* Type help() to get access to Python help. This will enable you to get help on all the modules and functions currently available. You can quit by typing quit.
* If you want to use the Python help system to get information on a variable, type help(variable\_name). If you wanted help with the variable dq, you'd type help(dq).

Being able to get help will let you see which methods are allowed on which objects, and be able to better understand the capabilities of Jupyter console.

Instructions

* Open the Jupyter console by typing ipython.
* Assign the value 5 to the variable dq.
* Get help with the variable dq using both ? and help(dq).
* Type help(), and explore the Python help functionality. See if you can look up some modules.
* Exit Jupyter console by typing exit.

**Persistent sessions**

Just like with Jupyter notebook, Jupyter console starts a kernel session when you first load it. Every time you run code in the console, the variables are stored in the session. Any subsequent lines you execute can access those variables.

This functionality is extremely powerful, and allows you to execute longer scripts line by line and inspect the variables at each step.

Instructions

* Open the Jupyter console by typing ipython.
* Assign the value 5 to the variable dq.
* Assign the value dq \* 10 to the variable dq\_10.
* Exit Jupyter console by typing exit.

<https://ipython.readthedocs.io/en/stable/interactive/magics.html>

**Jupiter magics**

You may have used the %quickref Jupyter magic in the last screen. Magics are special Jupyter commands that always start with %. They enable you to access Jupyter-specific functionality, without Python executing your commands.

Some useful magics are:

* %run -- allows you to run an external Python script. Any variables in the script will be stored in the current kernel session.
* %edit -- opens a file editor. Any code you type into the editor will be executed by Jupyter when you exit the editor.
* %debug -- if there's an error in any of your code, running %debug afterwards will open an interactive debugger you can use to trace the error.
* %history -- shows you the last few commands you ran.
* %save -- saves the last few commands you ran to a file.
* %who -- print all the variables in the session.
* %reset -- resets the session, and removes all stored variables.

You can see a full list of magics [here](http://ipython.readthedocs.org/en/stable/interactive/magics.html).

You can use the %run, %who, and %debug magics to iteratively develop scripts with Jupyter console. Have your favorite editor open, and start writing a Python script. In a separate shell, open Jupyter console. As you get to checkpoints in your script where you want to test it out, use the %run magic to run the script. Check the values of the variables using the %who magic. If you see any errors, debug them with the %debug magic. If you want to clear the session, use %reset.

Instructions

* Create a Python script using nano.
  + Add in whatever code you want, but make sure you add at least one print statement, and at least one variable definition.
* Open the Jupyter console by typing ipython.
* Use the %run magic to run the file you created.
* Use the %who magic to verify that the variable you defined exists.
* Exit Jupyter console by typing exit.

**Autocompletion**

If you hit the *TAB* key while typing a variable name, Jupyter will show you all the possible variables it could be, or auto-complete the name if there's only a single option. If you hit *TAB* after typing a variable name, Jupyter will show you the methods on the variable.

Autocompletion makes it much quicker to write code, and can also enable you to discover new methods on variables.

Instructions

* Open the Jupyter console by typing ipython.
* Create a variable.
* Experiment with autocompletion by typing the variable name.
* Exit Jupyter console by typing exit.

**Accessing the shell**

You can run shell commands in Jupyter console. Just prefix your shell commands with an exclamation point(!). Running !ls in Jupyter will show the contents of the current directory.

This can be useful when you want to quickly inspect a file or check on the contents of a folder.

Instructions

* Open the Jupyter console by typing ipython.
* Make a subdirectory with !mkdir
* Check the contents of the current directory with !ls.
* Exit Jupyter console by typing exit.

In [**5**]: !mkdir leo

In [**6**]: !ls

leo  tester.py

exit

**Pasting code**

You'll often want to paste code into Jupyter console to see if it runs properly. Because of how Python handles indentation, nested for loops, functions, and if statements will fail if you just copy and paste them in.

In order to paste in code with indents, you'll need to use paste magics:

* %cpaste -- opens a special editing area where you can paste in code normally, without whitespace being a problem. You can type -- alone on a line to exit. After you exit, any code you pasted in will be immediately executed.
* %paste -- takes code from your clipboard and runs it in Jupyter. This won't work on a remote system where Jupyter doesn't have access to your clipboard. Since we're using a remote system, you will see ERROR: Getting text from the clipboard on this platform requires Tkinter

Instructions

Copy the code below:

for i in range(10):

if i < 5:

print(i)

else:

print(i \* 2)

* Open the Jupyter console by typing ipython.
* Enter the special pasting area using %cpaste.
* Paste in the above code by pressing Ctrl + A followed by Ctrl + V.
* Type two hypens (-) to exit the pasting area.
* Exit Jupyter console by typing exit.
* n [**2**]: %cpaste
* Pasting code; enter '--' alone on the line to stop or use Ctrl-D.
* :for i in range(10):
* :
* :    if i < 5:
* :
* :        print(i)
* :
* :    else:
* :
* :        print(i \* 2)
* :<EOF>
* 0
* 1
* 2
* 3
* 4
* 10
* 12
* 14
* 16
* 18
* In [**3**]: exit

**Next steps**

 You should now be familiar with how to use Jupyter console to interactively execute Python code. Jupyter console is a great addition to your development workflow, and can help you write and debug code.

We encourage you to keep exploring Jupyter console.

Some specific explorations you can try:

* Explore [more](http://ipython.readthedocs.org/en/stable/interactive/magics.html) of the magics.
* Try using Jupyter to debug exceptions.
* Develop a Python script locally, and see if Jupyter can help with your workflow.

Instructions

Start the Jupyter console, do some exploration, and exit the console.

<https://ipython.readthedocs.io/en/stable/interactive/magics.html>