KDnuggets | Visit KDnuggets.com for more cheatsheets and additional learning resources.

Jupyter Notebook Magic Methods Cheat Sheet

Magic methods are special commands that provide additional functionality beyond standard Python syntax. There are two types of magic methods in Jupyter notebook: line magics and cell magics. Line magics apply to the current line and start with %, while cell magics apply to the entire cell and start with %%.

%lsmagic

Shows a list of magic commands

```
%lsmagic

>>> -> root
        -> line
           automagic. "AutoMagics" ....
         -> cell
           js   "DisplayMagics" ....
```

%quickref

List of common magic commands and their descriptions

```
%quickref

>>> IPython -- An enhanced Interactive Python - Quick
Reference Card ======================================

obj?, obj?? : Get help, or more help for object (also works
as  ?obj, ??obj).
...
```

%history and %recall

%history displays the command history . Use %history -n to display last n-commands with line numbers. %recall <line_no> re-executes command at line_no. You can also specify range of line numbers.

```
%history -n

>>> 1: %lsmagic
    2: %quickref
    3: print("Hello, world!")
       print("This is a test.")
    4: %history -n

%recall 3

>>> print("Hello, world!")
    print("This is a test.")
```

%%time

Measures execution time of the specific block of code

```
%%time
result = 0
for i in range(10000):
   result += i
print(result)

>>> 49995000
CPU times: user 1.58 ms, sys: 737 Ms, total: 2.32 ms
Wall time: 2.69 ms
```

%env

Display a list of all environment variables

```
%env

>>> {'__CFBundleIdentifier': 'com.apple.Terminal',
 'TMPDIR':
'/var/folders/4p/y97mqgts7lv_5m_flbly9s5h0000gn/T/',
 'XPC_FLAGS': '0x0', TERM': 'xterm-color', 'SSH_AUTH_SOCK':
'/private/tmp/com.apple.launchd.JAlJFEonIi/Listeners', ... }
```

%load and %run

%load commands loads the content of external python file into you cell while %run executes an an external Python script

```
%load example.py

>>> # %load example.py

def square(x):
   return x ** 2

print(square(5))

%run example.py

>>> 25
```

%who

Shows list of all variables defined within the current notebook

```
%who

>>> i  result  square
```

%pinfo

Displays important information about the entered variable

```
%pinfo result

>>> Type:      int
    String form: 49995000
    Docstring:
    int([x]) -> integer
    int(x, base=10) -> integer . . .
```

%store

Stores variables in the IPython database

```
# Define a variable
x = 10
%store x
```

Now, x is stored and can be accessed in other notebooks using `%store -r x`

%debug

Activate the interactive debugger

```
def divide_by_zero():
   return 1 / 0
%debug
divide_by_zero()   # Call the function with an error

>>>/var/folders/4p/y97mqgts7lv_5m_flbly9s5h0000gn/T/ipy
kernel_78927/2426985148.py (3) divide_by_zero()
    1 # Example code with an intentional error
    2 def divide_by_zero():
----> 3     return 1 / 0
    4
    5 # # Activate the debugger
ipdb>
```

%%writefile

Writes the contents of a cell to a file

```
%%writefile my_file.py
def greet(name):
   return f"Hello, {name}!"
print(greet('Alice'))

>>> Writing my_file.py
```

%precision

Set the precision of floating point numbers for output

```
%precision 2
num = 3.14159265359
num

>>> 3.14
```

Subscribe to KDnuggets News

Kanwal Mehreen | 2024