

3.IPython

October 27, 2019

1 IPython

Este notebook apresenta os seguintes tópicos:

- Section 1.1 - Mágicas do IPython
- Section 1.1.5 - Como definir Mágicas
- Section 1.2 - Exercício 2
- Section 1.3 - Exercício 3
- Section 1.4 - Exercício 4

1.1 Magicas do IPython

Na parte anterior do minicurso, apresentamos **bang expression** como uma extensão da linguagem Python fornecida pelo kernel IPython para executar comandos no sistema.

Além dessa extensão, o IPython também permite escrever “mágicas”/“magics” que modificam a forma de executar operações. Existem duas principais formas de “magics”:

- line magic: altera o restante da linha
- cell magic: altera a célula inteira

1.1.1 Line magic

A seguir temos um exemplo de line magic que mostra o histórico de células executadas com a numeração de células.

```
[1]: a = 1
```

```
[2]: b = a
```

```
[3]: %history -n
```

```
1: a = 1
2: b = a
3: %history -n
```

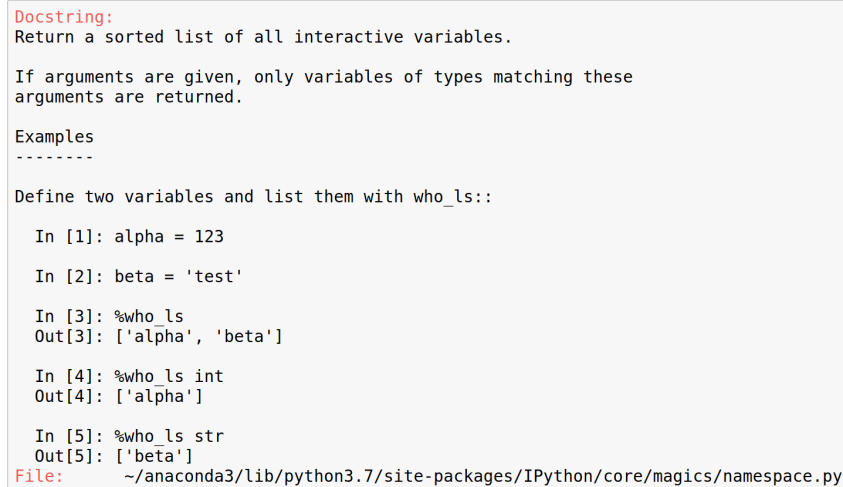
Essa line magic apenas imprimiu o histórico, porém existem outras que podem ser usadas no meio de expressões do Python, como a `%who_ls`, que retorna todas as variáveis de um determinado tipo definidas no notebook.

```
[4]: variaveis = %who_ls int
    for var, _ in zip(variaveis, range(5)):
        print(var, eval(var))
```

```
a 1
b 1
```

Além de estender a sintaxe do Python para adicionar bang expressions e magics, o IPython também permite consultar a documentação de módulos, classes, funções e magics, ao adicionar `?` após o nome.

```
[5]: %who_ls?
```

A screenshot of the IPython documentation for the `%who_ls` magic. The window has a title bar with standard OS icons. The content includes a 'Docstring:' section, a description of the magic's function, examples of its usage, and the source file path.

Docstring:
Return a sorted list of all interactive variables.

If arguments are given, only variables of types matching these arguments are returned.

Examples

Define two variables and list them with `who_ls`::

```
In [1]: alpha = 123
In [2]: beta = 'test'
In [3]: %who_ls
Out[3]: ['alpha', 'beta']
In [4]: %who_ls int
Out[4]: ['alpha']
In [5]: %who_ls str
Out[5]: ['beta']
```

File: ~/anaconda3/lib/python3.7/site-packages/IPython/core/magics/namespace.py

O uso de duas interrogações (`??`) exibe o código fonte.

```
[6]: %who_ls??
```

```

Source:
@skip_doctest
@line_magic
def who_ls(self, parameter_s=''):
    """Return a sorted list of all interactive variables.

    If arguments are given, only variables of types matching these
    arguments are returned.

    Examples
    -----

    Define two variables and list them with who_ls::

    In [1]: alpha = 123

    In [2]: beta = 'test'

    In [3]: %who_ls
    Out[3]: ['alpha', 'beta']

    In [4]: %who_ls int
    Out[4]: ['alpha']

    In [5]: %who_ls str
    Out[5]: ['beta']

```

1.1.2 Cell magic

Cell magics permitem alterar a execução de uma célula por completo. A cell magic a seguir executa código javascript no navegador.

```

[7]: %%javascript

console.log("Teste")

```

<IPython.core.display.Javascript object>

Log do Console do navegador:

Teste main.min.js:2:9

Já a cell magic a seguir calcula o tempo de execução de uma célula Python.

```

[8]: %%time
from time import sleep
sleep(2)

```

CPU times: user 783 µs, sys: 104 µs, total: 887 µs
Wall time: 2 s

1.1.3 Como ocorre a execução

A line magic `%history` apresentada anteriormente pode ser usada para entender o que o IPython está fazendo quando usamos essas magics. Para isso, precisamos ver o histórico traduzido para Python, utilizando a flag `-t`.

```

[9]: %history -t -l 6

```

```

get_ipython().run_line_magic('history', '-n')
variaveis = get_ipython().run_line_magic('who_ls', 'int')
for var, _ in zip(variaveis, range(5)):
    print(var, eval(var))
get_ipython().run_line_magic('pinfo', '%who_ls')
get_ipython().run_line_magic('pinfo2', '%who_ls')
get_ipython().run_cell_magic('javascript', '', '\nconsole.log("Teste")\n')
get_ipython().run_cell_magic('time', '', 'from time import sleep\nsleep(2)\n')

```

Note os seguintes comandos:

```

get_ipython().run_cell_magic('time', '', 'from time import sleep\nsleep(2)\n')
get_ipython().run_line_magic('who_ls', 'int')

```

Eles indicam o que o shell do IPython (resultado de `get_ipython()`) deve executar. A função indica se deve executar cell magic ou line magic. O primeiro parâmetro indica o nome da magic. Por fim, os últimos parâmetros indicam os parâmetros para a função da magic.

Esses comandos podem ser executados diretamente no notebook:

```
[10]: get_ipython().run_line_magic('who_ls', 'int')
```

```
[10]: ['a', 'b']
```

1.1.4 Lista de magics

Podemos usar a magic `%lsmagic` para listar quais são todas as magics do IPython e a magic `%magic` para entender como funciona a parte de magics.

```
[11]: %lsmagic
```

```

[11]: Available line magics:
%alias %alias_magic %autoawait %autocall %automagic %autosave %bookmark
%cat %cd %clear %colors %conda %config %connect_info %cp %debug %dhist
%dirs %doctest_mode %ed %edit %env %gui %hist %history %killbgscripts
%ldir %less %lf %lk %ll %load %load_ext %loadpy %logoff %login
%logstart %logstate %logstop %ls %lsmagic %lx %macro %magic %man
%matplotlib %mkdir %more %mv %notebook %page %pastebin %pdb %pdef %pdoc
%pfile %pinfo %pinfo2 %pip %popd %pprint %precision %prun %psearch
%psource %pushd %pwd %pycat %pylab %qtconsole %quickref %recall %rehashx
%reload_ext %rep %rerun %reset %reset_selective %rm %rmdir %run %save
%sc %set_env %store %sx %system %tb %time %timeit %unalias %unload_ext
%who %who_ls %whos %xdel %xmode

```

Available cell magics:

```

%%! %%HTML %%SVG %%bash %%capture %%debug %%file %%html %%javascript
%%js %%latex %%markdown %%perl %%prun %%pypy %%python %%python2
%%python3 %%ruby %%script %%sh %%svg %%sx %%system %%time %%timeit
%%writefile

```

Automagic is ON, % prefix IS NOT needed for line magics.

Perceba que automagic está ativo, isso significa que podemos usar line magics sem % explícito:

```
[12]: who_ls int
```

```
[12]: ['a', 'b']
```

Para outras magics, veja o arquivo InteratividadeExtra.ipynb

1.1.5 Registrando novas magics

Agora que sabemos como o IPython executa as magics, podemos pensar em criar e registrar novas magics.

```
[13]: from IPython.core.magic import Magics, magic_class, cell_magic

@magic_class
class LenMagic(Magics):
    @cell_magic
    def size(self, line, cell):
        return len(cell)
```

Em seguida registramos a magic:

```
[14]: shell = get_ipython()
      shell.register_magics(LenMagic)
```

Com isso, podemos usar para obter o tamanho de códigos de células:

```
[15]: %%size
      print("a")
```

```
[15]: 11
```

Note que o conteúdo da célula não foi executado. Ao invés disso, ele foi passado para a função size que o processou e retornou 11

Agora vamos para um exemplo mais complicado, com argumentos, criação dinâmica de classes e análise da AST.

```
[16]: import ast
      from IPython.core.magic_arguments import magic_arguments, argument, \
      ↪ parse_argstring

@magic_class
class ASTMagic(Magics):
```

```

@magic_arguments()
@argument(
    "methods",
    default=["visit_Assign", "visit_AugAssign"],
    nargs="*",
    help="method names to be defined on AST Visitor"
)
@cell_magic
def count_ast(self, line, cell):
    args = parse_argstring(self.count_ast, line)
    class CustomVisitor(ast.NodeVisitor):
        def __init__(self):
            self.count = 0

        def _increment_counter(self, node):
            self.count += 1

    for method in args.methods:
        setattr(CustomVisitor, method, CustomVisitor._increment_counter)

    tree = ast.parse(cell)
    visitor = CustomVisitor()
    visitor.visit(tree)
    return visitor.count

shell = get_ipython()
shell.register_magics(ASTMagic)

```

Neste exemplo, definimos argumentos usando decoradores e usamos a função `parse_argstring` para transformá-los em uma estrutura. A definição segue o `argparse` do Python: <https://docs.python.org/3/library/argparse.html>

```

@magic_arguments()
@argument(
    "methods",
    default=["visit_Assign", "visit_AugAssign"],
    nargs="*",
    help="method names to be defined on AST Visitor"
)

```

Além da parte dos argumentos, criamos classes dinamicamente dentro da função e definimos os métodos dela como sendo referências ao método `_increment_counter`.

```

for method in args.methods:
    setattr(CustomVisitor, method, CustomVisitor._increment_counter)

```

Por fim, executamos o visitor e retornamos a contagem.

```

tree = ast.parse(cell)

```

```
visitor = CustomVisitor()
visitor.visit(tree)
return visitor.count
```

```
[17]: %%count_ast
def f():
    pass
a = 1
b = 2
c = 3
```

[17]: 3

```
[18]: %%count_ast visit_FunctionDef
def f():
    pass
a = 10
b = 2
c = 3
```

[18]: 1

1.2 Exercício 2

Modifique a magic `count_ast` para retornar um dicionário ou counter com uma contagem de todos os nós da ast. O nome da magic resultante deve ser `ast_counter`.

Dicas: - Use o método `generic_visit(self, node)` para visitar os nós da AST sem especificar o nome - Obtenha o nome do elemento na AST usando `type(node).__name__` - Visite nós recursivamente

```
[ ]: ...
```

```
[ ]: %%ast_counter
def f():
    pass
a = 1
b = 2
c = 3
```

1.3 Exercício 3

Crie uma magic, `%%radon`, que utilize `radon` para extrair informações de complexidade ciclomática e linhas de código de uma célula.

```
[ ]: from radon.raw import analyze
      from radon.complexity import cc_visit

      template = """
      def __radon_analysis():
          {}
      """

      ...
```

```
[ ]: %%radon
      def f():
          pass
      a = 1
      if a:
          b = 2
          if b:
              c = 3
```

1.4 Exercício 4

Faça uma line magic para clonar repositórios do GitHub recebendo o repositório no formato `Organizacao/Repositorio` e com argumentos para especificar o diretório e o commit.

Exemplo de uso:

```
%clone gems-uff/sapos -d repos/sapos -c a9b0f7b3
```

Dicas:

- Você pode usar **bang expressions** para chamar os comandos `git clone` e `git checkout`.
- Bang expressions aceitam combinar variáveis do Python usando usando `{variavel}`, entre chaves
- A URL de um repositório do tipo `owner/name` no GitHub é `https://github.com/owner/name.git`

```
[ ]: from IPython.core.magic import line_magic

      ...
```

```
[ ]: %clone gems-uff/sapos -d repos/sapos -c a9b0f7b3
```

Continua: [4.Proxy.pdf](#)