

10.Visualizacao.Rica

October 26, 2019

Para entrar no modo apresentação, execute a seguinte célula e pressione -

```
[1]: %reload_ext slide
```

```
<IPython.core.display.Javascript object>
```

1 Visualização Rica

Este notebook apresenta os seguintes tópicos

- Section 1.1 - Visualizações ricas
- Section 1.2 - Exercício 12
- Section 1.3 - Exercício 13

1.1 Visualizacoes ricas

Como dito anteriormente, o Jupyter permite misturar texto formatado com código e com visualizações.

As visualizações até o momento foram feitas usando as bibliotecas `matplotlib` e `pandas` (que também utiliza a `matplotlib`), mas podemos chegar em situações em que queremos criar nossas próprias visualizações para objetos próprios.

Para fazer isso, podemos definir os métodos `repr*(self)` em classes, on `*` pode ser algum dos seguintes formatos suportados pelo Jupyter:

- `svg`
- `png`
- `jpeg`
- `html`
- `javascript`
- `latex`

Por exemplo, podemos definir uma classe `SQRT` que represente uma raiz quadrada de um número usando `Latex`.

```
[2]: class SQRT:
      def __init__(self, number):
```

```

        self.number = number

    def formula(self):
        text = self.number
        if hasattr(self.number, "formula"):
            text = self.number.formula()
        return "\sqrt{%s}" % (text,)

    def _repr_latex_(self):
        return "$$\{ }\$$.format(self.formula())

```

SQRT(25)

[2]:

$$\sqrt{25}$$

Essa classe pode ser usada em conjunto com outra.

[3]: SQRT(SQRT(25))

[3]:

$$\sqrt{\sqrt{25}}$$

Também podemos chamar programas externos para construir imagens. A seguir usaremos GraphViz (dot) para construir tanto uma imagem SVG quanto uma imagem PNG.

```

[4]: import os
from subprocess import Popen, PIPE as P

class Graph:
    def __init__(self, definition):
        self.definition = definition

    def dottext(self):
        result = [
            " {} -> {};" .format(node, other)
            for node, edges in self.definition.items()
            for other in edges
        ]
        return "digraph G {\n ranksep=0.25;\n{}\n}" .format("\n".join(result))

    def dot(self, format="png"): # ToDo: Tratar erro
        kwargs = {} if os.name != 'nt' else {"creationflags": 0x08000000}
        p = Popen(['dot', '-T', format], stdout=P, stdin=P, stderr=P, **kwargs)
        return p.communicate(self.dottext().encode('utf-8'))[0]

    def __repr__(self):

```

```
    return self.dottext()

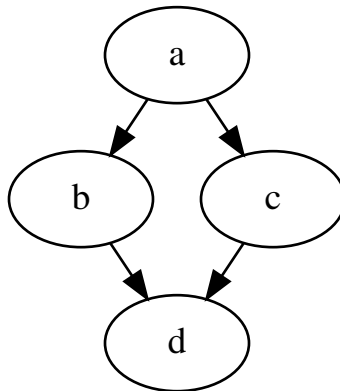
    def _repr_svg_(self):
        return self.dot("svg").decode("utf-8")

    def _repr_png_(self):
        return self.dot("png")

graph = Graph({"a": ["b", "c"], "b": ["d"], "c": ["d"]})
```

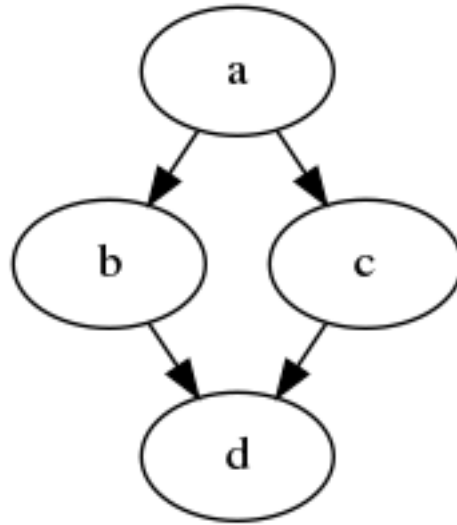
[5]: graph

[5]:



Por padrão, objetos que oferecem tanto visualização png quanto svg são visualizados como SVG no Jupyter. Porém, podemos forçar que sejam visualizados como PNG.

```
[6]: from IPython.display import display_png
      display_png(graph)
```



Ou podemos forçar o uso do `__repr__` do Python.

```
[7]: from IPython.display import display_pretty
display_pretty(graph)
```

```
digraph G {
  ranksep=0.25;
  a -> b;
  a -> c;
  b -> d;
  c -> d;
}
```

Aproximando do assunto da apresentação, podemos usar esse grafo para exibir a AST.

```
[8]: import ast
from collections import defaultdict

class GraphVisitor(ast.NodeVisitor):

    def __init__(self):
        self.parent = []
        self.graph = defaultdict(list)
        self.id = 0

    def generic_visit(self, node):
        old_parent = self.parent
        name = type(node).__name__ + str(self.id)
```

```

self.id += 1
if old_parent:
    self.graph[old_parent].append(name)
self.parent = name
super().generic_visit(node)
self.parent = old_parent

```

Executando para a própria célula.

```

[9]: tree = ast.parse(In[-1])
    visitor = GraphVisitor()
    visitor.visit(tree)

```

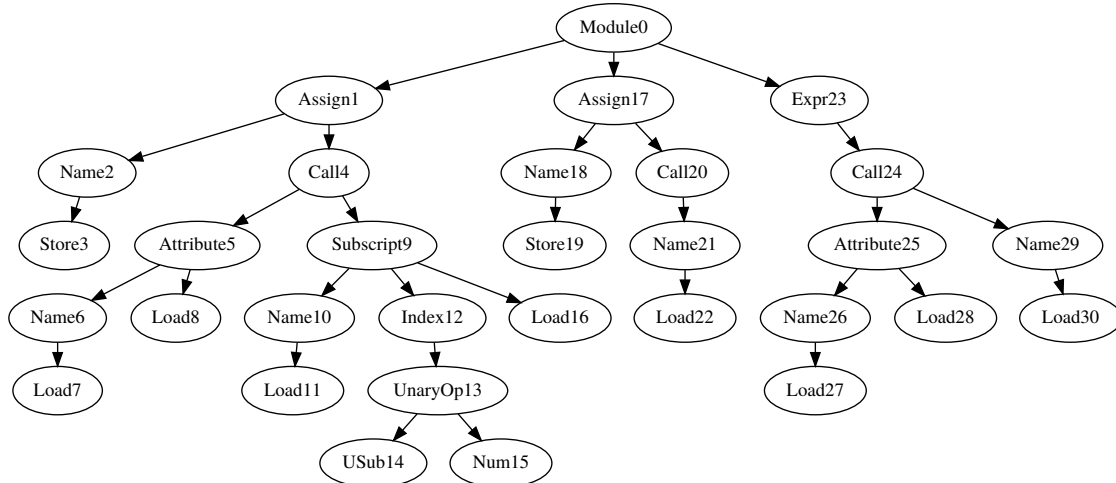
Visualização da AST.

```

[10]: Graph(visitor.graph)

```

[10]:



1.2 Exercício 12

Modifique a classe GraphVisitor para exibir valores nos nós do tipo Name e Num.

```

[11]: ...

```

```

[12]: tree = ast.parse(In[-1])
    visitor = GraphVisitor()
    visitor.visit(tree)

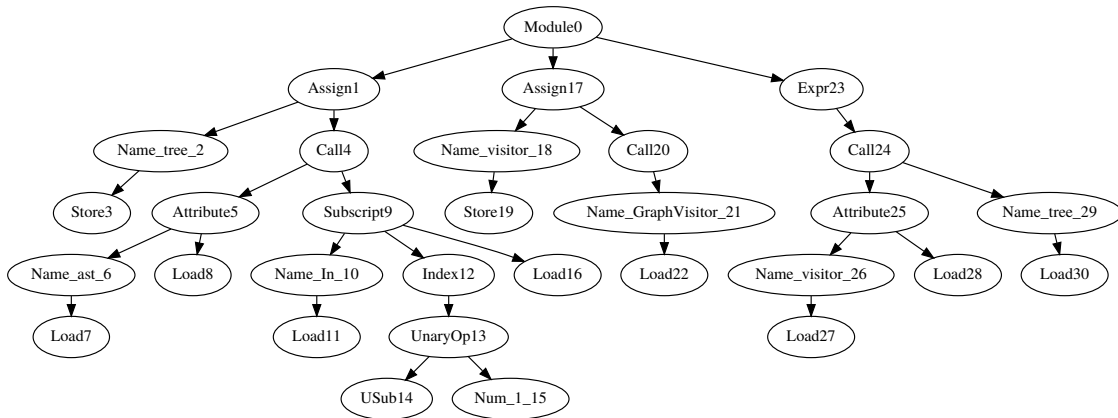
```

```

[13]: Graph(visitor.graph)

```

[13]:



1.3 Exercício 13

Implemente uma cell magic para visualizar a AST.

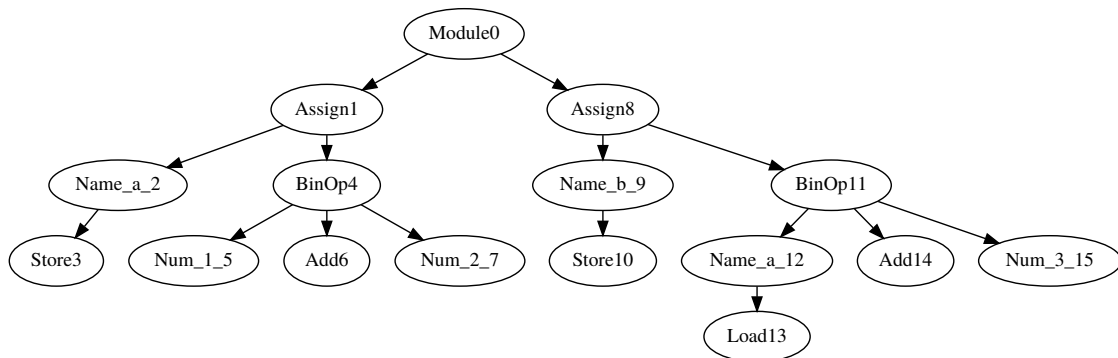
```
[14]: import ast
from IPython.core.magic import Magics, magics_class, cell_magic

@magics_class
class ASTMagic(Magics):
    @cell_magic
    def view_ast(self, line, cell):
        ...

shell = get_ipython()
shell.register_magics(ASTMagic)
```

```
[15]: %%view_ast
a = 1 + 2
b = a + 3
```

[15]:



Continua: [11.Widgets.pdf](#)