A JUPYTER ENHANCEMENT PROPOSALS

Raniere Silva and Tania Sanchez Monroy (apologies)

EuroPython 2018, Edinburgh, 25 July, 2018
RANIERE SILVA

- @rgaiacs
- @rgaiacs
- rgaiacs.com
TANIA SANCHEZ MONROY

- ixek
- trallard
- bitsandchips.me
1. "is a Horizon 2020 European Research Infrastructure project (#676541) that will run from Sept. 2015 to August 2019."
2. "[the goal is] to create and strengthen virtual research environments."
3. "advanced the Jupyter Notebook Ecosystem"

More information at https://opendreamkit.org/.
COMPUTATIONAL MATHEMATICS WITH JUPYTER AT INTERNATIONAL CENTRE FOR MATHEMATICAL SCIENCES (ICMS)

Edinburgh on 16-20 January 2017
Making Choices

Our previous lessons have shown us how to manipulate data, define our own functions, and repeat things. However, the programs we have written so far always do the same things, regardless of what data they're given. We want programs to make choices based on the values they are manipulating. To help us see what decisions they're making, we'll start by looking at how computers manipulate images.

Objectives

- Create a simple "image" made out of colored blocks.
- Explain how the RGB model represents colors.
- Explain the similarities and differences between tuples and lists.
- Write conditional statements including if, elif, and else branches.
- Correctly evaluate expressions containing and and or.
- Correctly write and interpret code containing nested loops and conditionals.
- Explain the advantages of putting frequently-modified code in a function.

Image Grids

Let's start by creating some simple heat maps of our own using a library called iPythonblocks. The first step is to create our own "image":

```python
from iPythonblocks import ImageGrid
```

Unlike the import statements we have seen earlier, this one doesn't load the entire iPythonblocks library. Instead, it just loads ImageGrid from that library, since that's the only thing we need (for now).

Once we have ImageGrid loaded, we can use it to create a very simple grid of colored cells:

```python
grid = ImageGrid(5, 3)
grid.show()
```

![Image Grid Example](image-grid.png)
Our previous lessons have shown us how to manipulate data, define our own functions, and repeat things. However, the programs we have written so far always do the same things, regardless of what data they’re given. We want programs to make choices based on the values they are manipulating. To help us see what decisions they’re making, we’ll start by looking at how computers manipulate images.

Objectives

- Create a simple "image" made out of colored blocks.
- Explain how the RGB model represents colors.
- Explain the similarities and differences between tuples and lists.
- Write conditional statements including if, elif, and else branches.
- Correctly evaluate expressions containing and and or.
- Correctly write and interpret code containing nested loops and conditionals.
- Explain the advantages of putting frequently-modified code in a function.

Image Grids

Let’s start by creating some simple heat maps of our own using a library called `ipythonblocks`. The first step is to create our own “image”:

```
In [6]: from ipythonblocks import ImageGrid
```

Unlike the `import` statements we have seen earlier, this one doesn’t load the entire `ipythonblocks` library. Instead, it just loads `ImageGrid` from that library, since that’s the only thing we need (for now).

Once we have `ImageGrid` loaded, we can use it to create a very simple grid of colored cells:

```
In [7]: grid = ImageGrid(5, 3)
   grid.show()
```

Just like a NumPy array, an `ImageGrid` has some properties that hold information about it:
Making Choices

Our previous lessons have shown us how to manipulate data, define our own functions, and repeat things. However, the programs we have written so far always do the same things, regardless of what data they’re given. We want programs to make choices based on the values they are manipulating. To help us see what decisions they’re making, we’ll start by looking at how computers manipulate images.

Objectives

- Create a simple “image” made out of colored blocks.
- Explain how the RGB model represents colors.
- Explain the similarities and differences between tuples and lists.
- Write conditional statements including `if`, `elif`, and `else` branches.
- Correctly evaluate expressions containing `and` and `or`.
- Correctly write and interpret code containing nested loops and conditionals.
- Explain the advantages of putting frequently-modified code in a function.

Image Grids

Let’s start by creating some simple heat maps of our own using a library called `ipythonblocks`. The first step is to create our own “image”:

```
In [6]: from ipythonblocks import ImageGrid
```

Unlike the `import` statements we have seen earlier, this one doesn’t load the entire `ipythonblocks` library. Instead, it just loads `ImageGrid` from that library, since that’s the only thing we need (for now).

Once we have `ImageGrid` loaded, we can use it to create a very simple grid of colored cells:

```
In [7]: grid = ImageGrid(5, 3)
grid.show()
```
Previous lessons have shown us how to manipulate data, define our own functions, and repeat things. However, the programs we have written so far do the same things, regardless of what data they're given. We want programs to make choices based on the values they are manipulating. To help what decisions they're making, we'll start by looking at how computers manipulate images.

**Images**

Create a simple "image" made out of colored blocks.

Explain how the RGB model represents colors.

Explain the similarities and differences between tuples and lists.

Include if, elif, and else branches.

Include expressions containing and and or.

Exclude code containing nested loops and conditionals.

Create a code snippet containing frequently-modified code in a function.

Let's start by creating some simple heat maps of our own using a library called `ipythonblocks`. The first step is to create our own "image":

```
In [6]: from ipythonblocks import ImageGrid
```

Unlike the `import` statements we have seen earlier, this one doesn't load the entire `ipythonblocks` library. Instead, it just loads `ImageGrid` from that library, since that's the only thing we need (for now).

Once we have `ImageGrid` loaded, we can use it to create a very simple grid of colored cells:

```
In [7]: grid = ImageGrid(5, 3)
grid.show()
```

Just like a NumPy array, an `ImageGrid` has some properties that hold information about it:
Objectives

- Create a simple "image" made out of colored blocks.
- Explain how the RGB model represents colors.
- Explain the similarities and differences between tuples and lists.
- Write conditional statements including if, elif, and else branches.
- Correctly evaluate expressions containing and and or.
- Correctly write and interpret code containing nested loops and conditionals.
- Explain the advantages of putting frequently-modified code in a function.

Image Grids

Let's start by creating some simple heat maps of our own using a library called `ipythonblocks`. The first step is to create our own "image":

```
In [6]: from ipythonblocks import ImageGrid
```

Unlike the `import` statements we have seen earlier, this one doesn't load the entire `ipythonblocks` library. Instead, it just loads `ImageGrid` from that library, since that's the only thing we need (for now).

Once we have `ImageGrid` loaded, we can use it to create a very simple grid of colored cells:

```
In [7]: grid = ImageGrid(5, 3)
grid.show()
```

![Image Grid Example]

Just like a NumPy array, an `ImageGrid` has some properties that hold information about it:

```
In [8]: print 'grid width:', grid.width
print 'grid height:', grid.height
print 'grid lines on:', grid.lines_on
```

grid width: 5
grid height: 3
grid lines on: True

The obvious thing to do with a grid like this is color in its cells, but in order to do that, we need to know how computers represent color. The most common schemes are **RGB**, which is short for "red, green, blue". RGB is an *additive color model*: every shade is some combination of red, green, and blue.
Modules template

Lorem ipsum dolor sit amet understanding yourself in the universe tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

⚠️ Important

Make sure to visit the Getting started section of this page before proceeding to any of the course modules.

Getting started

Information on the course, evaluation, and suggested literature. Please read

Day 1 outline

This is the description for the module

Day 2 outline

This is the description for the module
WHY JUPYTER NOTEBOOKS

- Environment familiar to authors
- Support for over 40 programming languages (by the time of this talk)
PIECES OF THE PIPELINE

- nbconvert
- static site generators, e.g. Sphinx, Hugo, Jekyll, ...

Python package used to convert Jupyter Notebooks into Jekyll ready documents including validation and version control tagging.

- **122 commits**
- **1 branch**
- **2 releases**
- **1 contributor**
- **MIT**

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>nbjekyll</td>
<td>Somehow the list comprehension was killing nb_git.find_notebooks()</td>
<td>6 months ago</td>
</tr>
<tr>
<td>.gitignore</td>
<td>Update .gitignore</td>
<td>7 months ago</td>
</tr>
<tr>
<td>.travis.yml</td>
<td>Add install pytz</td>
<td>6 months ago</td>
</tr>
<tr>
<td>LICENSE</td>
<td>Initial commit</td>
<td>7 months ago</td>
</tr>
<tr>
<td>License.txt</td>
<td>MIT license file</td>
<td>6 months ago</td>
</tr>
<tr>
<td>MANIFEST.in</td>
<td>Moved files for proper packaging</td>
<td>7 months ago</td>
</tr>
<tr>
<td>README.md</td>
<td>Re organise badges</td>
<td>5 months ago</td>
</tr>
<tr>
<td>setup.cfg</td>
<td>Add setup.cfg</td>
<td>6 months ago</td>
</tr>
<tr>
<td>setup.py</td>
<td>Add email</td>
<td>2 months ago</td>
</tr>
<tr>
<td>testenv.yml</td>
<td>Add install pytz</td>
<td>6 months ago</td>
</tr>
</tbody>
</table>
DEMO TIME

jupyter nbconvert --template=custom_templ notebook.ipynb
PEP 0 -- Index of Python Enhancement Proposals (PEPs)

PEP: 0
Title: Index of Python Enhancement Proposals (PEPs)
Last-Modified: 2018-07-23
Author: python-dev@python.org
Status: Active
Type: Informational
Created: 13-Jul-2000

Contents

- Introduction
- Index by Category
  - Meta PEPs (PEPs about PEPs or Processes)
  - Other informational PEPs
  - Provisional PEPs (provisionially accepted; interface may still change)
  - Accepted PEPs (accepted; may not be implemented yet)
  - Open PEPs (under consideration)
  - Finished PEPs (done, with a stable interface)
  - Historical Meta-PEPs and Informational PEPs
  - Deferred PEPs (postponed pending further research or updates)
  - Abandoned, Withdrawn, and Rejected PEPs
- Numerical Index
- Reserved PEP Numbers
- PEP Types Key
- PEP Status Key
- Authors/Owners
- References
Enhancement proposals for the Jupyter Ecosystem

- **38 commits**
- **8 branches**
- **0 releases**
- **9 contributors**
- BSD-3-Clause

<table>
<thead>
<tr>
<th>Branch</th>
<th>Description</th>
<th>Last commit</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>master</td>
<td><a href="https://github.com/jupyter/enhancement-proposals">jupyter-dashboards-deployment-attic</a></td>
<td>2017-09-11</td>
<td>11 months</td>
</tr>
<tr>
<td></td>
<td><a href="https://github.com/jupyter/enhancement-proposals">jupyter-dashboards-extension-incorporation</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><a href="https://github.com/jupyter/enhancement-proposals">jupyter-declarativewidgets-incorporation</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><a href="https://github.com/jupyter/enhancement-proposals">jupyter-enhancement-proposal-guide</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><a href="https://github.com/jupyter/enhancement-proposals">jupyter-kernel-gateway-incorporation</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><a href="https://github.com/jupyter/enhancement-proposals">notebook-diff</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><a href="https://github.com/jupyter/enhancement-proposals">.gitignore</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><a href="https://github.com/jupyter/enhancement-proposals">LICENSE</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><a href="https://github.com/jupyter/enhancement-proposals">README.md</a></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Jupyter Enhancement Proposals**
Add Template as Metatada enhancement proposal #23

rgaiacs commented on 20 Sep 2017

No description provided.

@mpacer was working on machinery to allow custom nbconvert behaviour from the notebook web server, in jupyter/notebook#2413, so I imagine he'll be interested in this.

parente commented on 20 Sep 2017

If I'm reading the proposal properly, I think it's possible to implement some of the proposed use cases via bundler extensions in the notebook server. But bundlers do not necessarily capture format information in notebook documents. They only provide a way for extension writers to plug in arbitrary, server-side actions to take on notebook documents (e.g., running nbconvert on a notebook with a specific template, posting a notebook to some external API).
THANKS!!! AND ASK ME QUESTIONS.

• Raniere: ✈rgaiacs
• Tania: ✈ixek