# Parallel & Distributed Computing: Lecture 11

Alberto Paoluzzi

November 2, 2017

# Student Programming Projects 2017-18

# Project: Add-ons modules to LARLIB.jl

## Student assignements

- boundary: Gabellieri

# Project: Add-ons modules to LARLIB.jl

## Student assignements

- boundary: Gabellieri
- integr: Rampogna - Bartolomeo

# Project: Add-ons modules to LARLIB.jl

## Student assignements

- boundary: Gabellieri
- integr: Rampogna - Bartolomeo
- lar2psm

# Project: Add-ons modules to LARLIB.jl

## Student assignements

- boundary: Gabellieri
- integr: Rampogna - Bartolomeo
- lar2psm
- larcc: Tofoni-Trani

# Project: Add-ons modules to LARLIB.jl

## Student assignements

- boundary: Gabellieri
- integr: Rampogna - Bartolomeo
- lar2psm
- larcc: Tofoni-Trani
- largrid: Antonelli - Graziano

# Project: Add-ons modules to LARLIB.jl

## Student assignements

- boundary: Gabellieri
- integr: Rampogna - Bartolomeo
- lar2psm
- larcc: Tofoni-Trani
- largrid: Antonelli - Graziano
- larstruct: Adiutori - Cossu

# Project: Add-ons modules to LARLIB.jl

## Student assignements

- boundary: Gabellieri
- integr: Rampogna - Bartolomeo
- lar2psm
- larcc: Tofoni-Trani
- largrid: Antonelli - Graziano
- larstruct: Adiutori - Cossu
- mapper: Beco - Sellarione

# Project: Add-ons modules to LARLIB.jl

## Student assignements

- boundary: Gabellieri
- integr: Rampogna - Bartolomeo
- lar2psm
- larcc: Tofoni-Trani
- largrid: Antonelli - Graziano
- larstruct: Adiutori - Cossu
- mapper: Beco - Sellarione
- morph: Camerini - Amerini

# Project: Add-ons modules to LARLIB.jl

## Student assignements

- boundary: Gabellieri
- integr: Rampogna - Bartolomeo
- lar2psm
- larcc: Tofoni-Trani
- largrid: Antonelli - Graziano
- larstruct: Adiutori - Cossu
- mapper: Beco - Sellarione
- morph: Camerini - Amerini
- simplexn: Loprevite-Fatelli

# Project: Add-ons modules to LARLIB.jl

## Student assignements

- boundary: Gabellieri
- integr: Rampogna - Bartolomeo
- lar2psm
- larcc: Tofoni-Trani
- largrid: Antonelli - Graziano
- larstruct: Adiutori - Cossu
- mapper: Beco - Sellarione
- morph: Camerini - Amerini
- simplexn: Loprevite-Fatelli
- splines: Santorsola - valletti

# Project: Add-ons modules to LARLIB.jl

## Student assignements

- boundary: Gabellieri
- integr: Rampogna - Bartolomeo
- lar2psm
- larcc: Tofoni-Trani
- largrid: Antonelli - Graziano
- larstruct: Adiutori - Cossu
- mapper: Beco - Sellarione
- morph: Camerini - Amerini
- simplexn: Loprevite-Fatelli
- splines: Santorsola - valletti
- triangulation: Ciccone - Melillo

# Methodology

# Clone lar-cc on local system

- Install Python-2.7

# Clone lar-cc on local system

- Install Python-2.7
- Install pyplasm

# Clone lar-cc on local system

- Install Python-2.7
- Install pyplasm
- Install larlib

# Clone lar-cc on local system

- Install Python-2.7
- Install pyplasm
- Install larlib
- Clone the python repository on your local machine

```
git clone https://github.com/cvdlab/lar-cc.git
```

# Clone lar-cc on local system

- Install Python-2.7
- Install pyplasm
- Install larlib
- Clone the python repository on your local machine

  ```
  git clone https://github.com/cvdlab/lar-cc.git
  ```

- Test your cloned repository by executing the first tests of `larcc.py`

# Minimal API (Application Programming Interface)

*API is a set of clearly defined methods of communication between various software components. A good API makes it easier to develop a computer program by providing all the building blocks, which are then put together by the programmer (Wikipedia).*

- Start from test files

# Minimal API (Application Programming Interface)

*API is a set of clearly defined methods of communication between various software components. A good API makes it easier to develop a computer program by providing all the building blocks, which are then put together by the programmer (Wikipedia).*

- Start from test files
- Look for meaningful elements of API

# Minimal API (Application Programming Interface)

*API is a set of clearly defined methods of communication between various software components. A good API makes it easier to develop a computer program by providing all the building blocks, which are then put together by the programmer (Wikipedia).*

- Start from test files
- Look for meaningful elements of API
  - execute the test file `test-xy.py`

# Minimal API (Application Programming Interface)

*API is a set of clearly defined methods of communication between various software components. A good API makes it easier to develop a computer program by providing all the building blocks, which are then put together by the programmer (Wikipedia).*

- Start from test files
- Look for meaningful elements of API
  - execute the test file `test-xy.py`
  - look at possible comments and/or figures in `doc/pdf/your-module.pdf`

# Minimal API (Application Programming Interface)

*API is a set of clearly defined methods of communication between various software components. A good API makes it easier to develop a computer program by providing all the building blocks, which are then put together by the programmer (Wikipedia).*

- Start from test files
- Look for meaningful elements of API
  - execute the test file `test-xy.py`
  - look at possible comments and/or figures in `doc/pdf/your-module.pdf`
  - look for external dependencies

# Minimal API (Application Programming Interface)

*API is a set of clearly defined methods of communication between various software components. A good API makes it easier to develop a computer program by providing all the building blocks, which are then put together by the programmer (Wikipedia).*

- Start from test files
- Look for meaningful elements of API
  - execute the test file `test-xy.py`
  - look at possible comments and/or figures in `doc/pdf/your-module.pdf`
  - look for external dependencies
- List the reduced subset of API functions

# Minimal API (Application Programming Interface)

*API is a set of clearly defined methods of communication between various software components. A good API makes it easier to develop a computer program by providing all the building blocks, which are then put together by the programmer (Wikipedia).*

- Start from test files
- Look for meaningful elements of API
  - execute the test file `test-xy.py`
  - look at possible comments and/or figures in `doc/pdf/your-module.pdf`
  - look for external dependencies
- List the reduced subset of API functions
- Identify the local utilities functions

# Local utilities functions

- Understand Name Spaces

# Local utilities functions

- Understand Name Spaces
- Draw the graph of calls (who calls who)

# Local utilities functions

- Understand Name Spaces
- Draw the graph of calls (who calls who)
- Identify functions called outside this module (candidate for external)

# Local utilities functions

- Understand Name Spaces
- Draw the graph of calls (who calls who)
- Identify functions called outside this module (candidate for external)
- Identify functions only used ONLY in this module;

# Local utilities functions

- Understand Name Spaces
- Draw the graph of calls (who calls who)
- Identify functions called outside this module (candidate for external)
- Identify functions only used ONLY in this module;
- Document input, output and meaning of each function;

# Document input, output and meaning of each function 1/2

Major block of information: the macro exporting the module in `src/tex/your-module.tex`

EXAMPLE: `src/tex/larcc.tex`

```
@O larlib/larlib/larcc.py
@{# -*- coding: utf-8 -*-
""" Basic LARCC library """
from larlib import *
import boundary
from boundary import larUnsignedBoundary2,boundary3

@< The MIT Licence @>
@< Importing of modules or packages @>
@< Affine transformations of $d$-points @>
@< From list of triples to scipy.sparse @>
@< Brc to Coo transformation @>
```

# Document input, output and meaning of each function 2/2

## Use ‘type(eval(code_string))‘

### INPUT

```
In [24]: type(eval("EV"))
Out[24]: list [of what?]

In [25]: type(eval("V"))
Out[25]: list [of what?]


In [26]: type(eval("[V,EV,[]]"))
Out[26]: list [of list]
```

### OUTPUT

```
In [21]: type(eval("MKPOL"))
Out[21]: function

In [23]: type(eval("MKPOL([V,EV,[]])"))
Out[23]: pyplasm.xgepy.Hpc
```

# larcc Module

```
\section{Unit tests}
%-------------------------------------------------------------------------
@D Test examples
@{
@< Test example of Brc to Coo transformation @>
@< Test example of Coo to Csr transformation @>
@< Test example of Brc to Csr transformation @>
@< Test examples of Query Matrix shape @>
@< Test examples of Sparse to dense matrix transformation @>
@< Test example of Matrix filtering to produce the boundary matrix @>
@< Test example of Matrix filtering via a generic predicate @>
@< Test examples of From cells and facets to boundary operator @>
@< Test examples of From cells and facets to boundary cells @>
@< Test examples of Computation of cell adjacencies @>
@< Test examples of Extraction of facets of a cell complex @>
@}
%-------------------------------------------------------------------------
```

# Methodology: start your Python Notebook

## AIM: understand and simplify ‘your-module.py‘

1. Name the notebook as `your-module-py.pynb`

# Methodology: start your Python Notebook

## AIM: understand and simplify ‘your-module.py‘

1. Name the notebook as `your-module-py.pynb`
2. Assess the module functionality (MarkDown)

# Methodology: start your Python Notebook

## AIM: understand and simplify ‘your-module.py‘

1. Name the notebook as `your-module-py.pynb`
2. Assess the module functionality (MarkDown)
3. Insert a drawing of dependencies (any tool or by hand)

# Methodology: start your Python Notebook

AIM: understand and simplify ‘your-module.py‘

1. Name the notebook as `your-module-py.pynb`
2. Assess the module functionality (MarkDown)
3. Insert a drawing of dependencies (any tool or by hand)
4. List all the function names (`def name(args): body`)

# Methodology: start your Python Notebook

AIM: understand and simplify ‘your-module.py‘

1. Name the notebook as `your-module-py.pynb`
2. Assess the module functionality (MarkDown)
3. Insert a drawing of dependencies (any tool or by hand)
4. List all the function names (`def name(args): body`)
5. Subdivide the functions between: (both working and non-working)

# Methodology: start your Python Notebook

AIM: understand and simplify ‘your-module.py‘

1. Name the notebook as `your-module-py.pynb`
2. Assess the module functionality (MarkDown)
3. Insert a drawing of dependencies (any tool or by hand)
4. List all the function names (`def name(args): body`)
5. Subdivide the functions between: (both working and non-working)
   - External to the module

# Methodology: start your Python Notebook

AIM: understand and simplify ‘your-module.py‘

1. Name the notebook as `your-module-py.pynb`
2. Assess the module functionality (MarkDown)
3. Insert a drawing of dependencies (any tool or by hand)
4. List all the function names (`def name(args): body`)
5. Subdivide the functions between: (both working and non-working)
   - External to the module
   - API (to be used externally)

# Methodology: start your Python Notebook

AIM: understand and simplify ‘your-module.py‘

1. Name the notebook as `your-module-py.pynb`
2. Assess the module functionality (MarkDown)
3. Insert a drawing of dependencies (any tool or by hand)
4. List all the function names (`def name(args): body`)
5. Subdivide the functions between: (both working and non-working)
   - External to the module
   - API (to be used externally)
   - Local utilities

# Methodology: start your Python Notebook

AIM: understand and simplify ‘your-module.py‘

1. Name the notebook as `your-module-py.pynb`
2. Assess the module functionality (MarkDown)
3. Insert a drawing of dependencies (any tool or by hand)
4. List all the function names (`def name(args): body`)
5. Subdivide the functions between: (both working and non-working)
   - External to the module
   - API (to be used externally)
   - Local utilities
6. Document each API function

# Methodology: start your Python Notebook

AIM: understand and simplify ‘your-module.py‘

1. Name the notebook as `your-module-py.pynb`
2. Assess the module functionality (MarkDown)
3. Insert a drawing of dependencies (any tool or by hand)
4. List all the function names (`def name(args): body`)
5. Subdivide the functions between: (both working and non-working)
   - External to the module
   - API (to be used externally)
   - Local utilities
6. Document each API function
7. Send to me (us) for review

# Methodology: start your Python Notebook

AIM: understand and simplify ‘your-module.py‘

1. Name the notebook as `your-module-py.pynb`
2. Assess the module functionality (MarkDown)
3. Insert a drawing of dependencies (any tool or by hand)
4. List all the function names (`def name(args): body`)
5. Subdivide the functions between: (both working and non-working)
   - External to the module
   - API (to be used externally)
   - Local utilities
6. Document each API function
7. Send to me (us) for review
8. Start your JULIA notebook `your-module-jl.pynb`

# Methodology: start your Julia Notebook

AIM: convert ‘your-module-py.pynb‘

Next time !! ;-)

# References