1 Introduction

First issue:

PermissionError: [Errno 13] Permission denied: '/home/.../plexus/.eggs/pytest.runner-5.3.1-py3.8.egg/pytest.runner-5.3.1.dist-info' '/home/.../plexus/.eggs/pytestrunner-5.3.1-py3.8.egg/EGG-INFO' *ERROR: Command errored out with exit status 1: python setup.py egg.info Check the logs for full command output.*

2 DMPlex

DMPLex is used within PETSC to handle unstructured mesh with useful interfaces for both topology and geometry. A DMPLex object is based on defining a mesh as a Hasse Diagram, where every level of topology is a separate layer in the diagram.

Given that DMPLex objects handles unstructured meshes, every point is defined by its "cone" and "support/closure".

- The support/closure of a point is defined as the set of all the point of the next highest topological dimension that is connected to the current point
- The cone of a point is defined as a set of all the point of the next lowest topological dimension.

Interpolation in DMPLex is defined as a mesh which contains some but not all "intermidate" entities. Nomrmally, a user will have a *fully interpolated mesh.*

Once created, the DMPLex object is then distributed to all available processes. At this point, each process check that its distributed section of the DMPLex object is not NULL, and replace its non-distributed DMPLex object with the distributed ones.

3 Python3

3.1 super

The C3 superclass linearization of a class is the sum of the class plus a unique merge of linearization of its parents and a list of the parent itself. The list of parents as the last argument to the merge process preserves the local precedence order of direct parent classes.

__init__ is an instance method, meaning that it takes as its first argument a reference to an instance. When called directly from the instance, the reference is passed implicitly

```
#try calling init from the class without specifying an instance
A.__init__()# TypeError is raised due to the expected but missing reference
a = A(); a.__init__(); A.__init__(a )
```

Bob Martin "a good architecture allows you to postpone decision-making as long as possible"

4 VSCODE

Remote debugging

With remote debugging only a single Python process is executed on the remote VM then, on client computer, VSCode "attached itself" to this remote process, so you can match the remote code execution with your local files. It is important to keep exactly the same.py files on client and in host so that the debugging process is able to match line by line the two version. The magic lies in a library called ptvsd that makes the bridge for attaching local VSCode to remotely executed process. The remotely executed Python waits until the client debugging agent is attached.