MHPC
Master in High Performance Computing

Iterators in Scientific Computing

N. Cavallini

Provided code

```
g++ -std=c++11 main.cpp
```

A Random Vector

```
template<typename num>

num get_random_number(num s,
num e)

{

    std::random_device rd;

    std::mt19937 gen(rd());


std::uniform_real_distribution
<num> dis(s, e);

    return dis(gen);

};
```

```
template<typename num>

vector<num> get_random_vector()

{

    vector<num> vec;

for (int i=0;
i<get_rdm_number<int>(30,100);
i++)

    {vec.push_back(

get_random_number<num>(0,1));}

    return vec;

}
```

How to loop

```
vector<double> vec = get_random_vector<double>();
    // loop fortran style:
    for (int i ; i<vec.size(); i++)
    cout << "fortran way of looping: " << vec[i] << endl;
    // suppose you don't know the size!?!?
    // suppose you don't have the method size!?!?!
```

How to loop

```
// use iterators

vector<double>::iterator it = vec.begin();

vector<double>::const_iterator eit = vec.end();

for (;it != eit; ++it)

    cout << "looping via iterator: " << *it << endl;

    // why dereferentiating??
```

How to loop

- Just as pointers, are incremented to the next element using operator ++, and decremented to the previous element using operator –.

- One can also jump n elements ahead using the addition operator, it=it+n, and correspondingly to move a number of elements back.

- In addition, and keeping with the tradition of the standard template library, containers provide member functions begin() and end() that provide the first element of a collection and a one-past-the-end iterator, respectively.

How to loop

```
// use iterators

auto it = vec.begin();

auto eit = vec.end();

for (;it != eit; ++it)

    cout << "looping via iterator: " << *it << endl;

    // why dereferentiating??
```

How to loop

```
// Range based iteration

for (auto it : vec)

    cout << "range based iteration: " << it << endl;
```

How to loop

Humans are the future of programming.

```
// Range based iteration
for (auto &it : vec)
    cout << "range based iteration: " << it << endl;
```

Iterators in python

```
for i in [1, 2, 3, 4]:

    print(i)
```

Iterators in python

```
for c in “python”:

    print(c)
```

Iterators in python

```
for line in open("a.txt"):

     print(line)
```

Iterators in python

The built-in function `iter` takes an iterable object and returns an iterator.

```
>>> x = iter([1, 2, 3])
>>> x
<listiterator object at 0x1004ca850>
>>> x.next()
1
>>> x.next()
2
```

Iterators in python

```
class your_range:

    def __init__(self, n):

        self.i = 0

        self.n = n

    def __iter__(self):

        self.i = 0

        return self

   def next(self):

        if self.i < self.n:

            i = self.i

            self.i += 1

            return i

        else:

            raise StopIteration()
```

Iterators in python

```
>>> from your_range import your_range

>>> x = your_range(5)

>>> print x

<your_range.your_range instance at
0x7f8cb43bd248>

>>> x.next()

…

4

>>> x.next()

Traceback (most recent call last):

  File "<stdin>", line 1, in <module>

  File "your_range.py", line 13, in next

    raise StopIteration()

StopIteration
```

Iterators in python

```
>>> from your_range import your_range

>>> x = your_range(5)

>>> print x

<your_range.your_range instance at
0x7f8cb43bd248>

>>> x.next()

…

4

>>> x.next()

Traceback (most recent call last):

  File "<stdin>", line 1, in <module>

  File "your_range.py", line 13, in next

    raise StopIteration()

StopIteration
```

Iterators in python

Many built-in functions accept iterators as arguments.

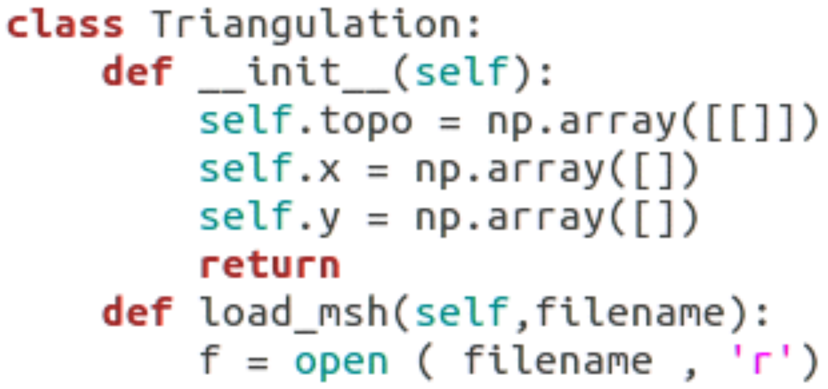
```
>>> print list(your_range(5))

[0, 1, 2, 3, 4]

>>> print sum(your_range(6))

15
```

Iterators in python

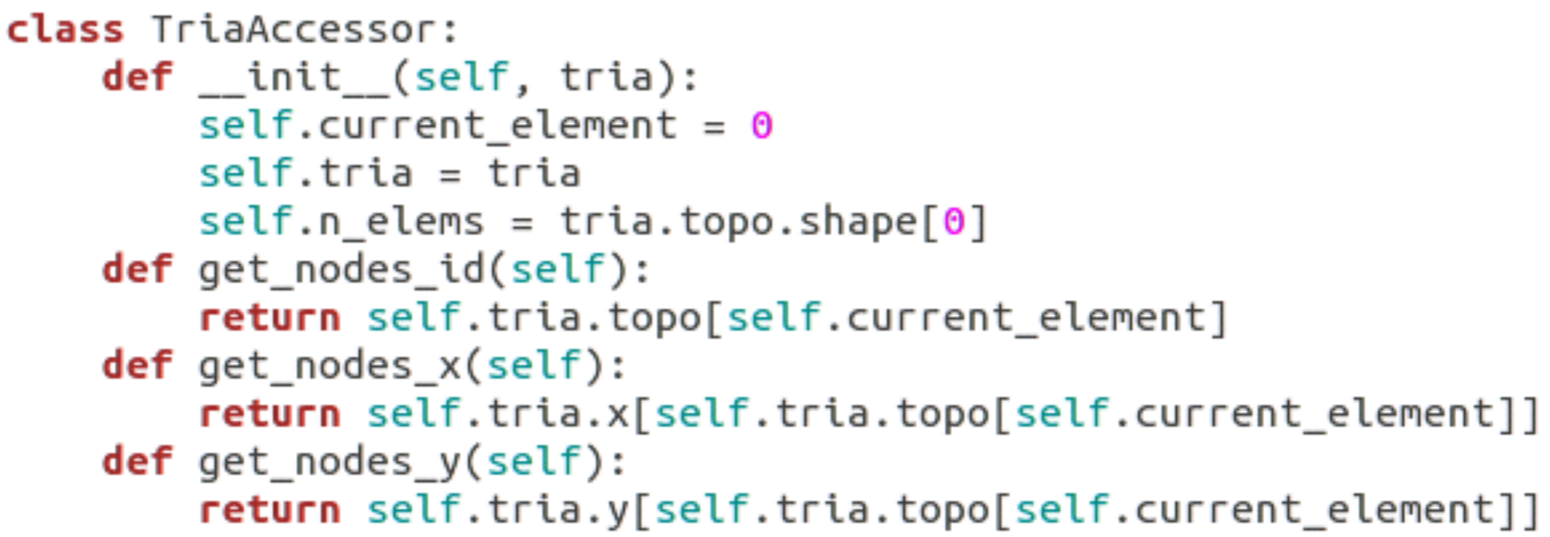
Many built-in functions accept iterators as arguments.

```
>>> print list(your_range(5))

[0, 1, 2, 3, 4]

>>> print sum(your_range(6))

15
```

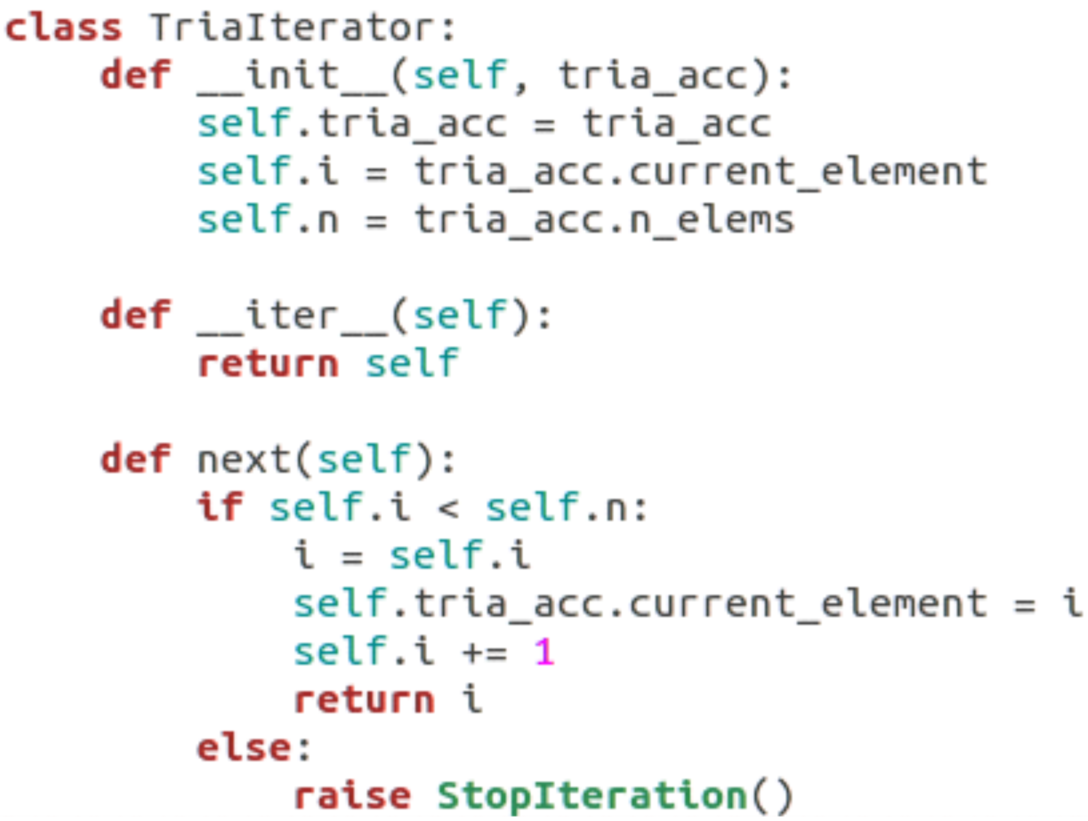
Grid Like-Iterators



```
class Triangulation:
    def __init__(self):
        self.topo = np.array([[]])
        self.x = np.array([])
        self.y = np.array([])
        return
    def load_msh(self,filename):
        f = open ( filename , 'r')
```



```
class TriaAccessor:
    def __init__(self, tria):
        self.current_element = 0
        self.tria = tria
        self.n_elems = tria.topo.shape[0]
    def get_nodes_id(self):
        return self.tria.topo[self.current_element]
    def get_nodes_x(self):
        return self.tria.x[self.tria.topo[self.current_element]]
    def get_nodes_y(self):
        return self.tria.y[self.tria.topo[self.current_element]]
```



```
class TriaIterator:
    def __init__(self, tria_acc):
        self.tria_acc = tria_acc
        self.i = tria_acc.current_element
        self.n = tria_acc.n_elems

    def __iter__(self):
        return self

    def next(self):
        if self.i < self.n:
            i = self.i
            self.tria_acc.current_element = i
            self.i += 1
            return i
        else:
            raise StopIteration()
```