

Programming

Object-oriented programming

Luna Pianesi

Faculty of Technology, Bielefeld University

```
332
333
334     if extrapolate is None:
335         extrapolate = self.extrapolate
336     x = np.asarray(x)
337     x_shape, x_ndim = x.shape, x.ndim
338     x = np.ascontiguousarray(x.ravel(), dtype=np
339
340     # With periodic extrapolation we map x to the
341     # [self.t[k], self.t[n]].
342     if extrapolate == 'periodic':
343         n = self.t.size - self.k - 1
344         x = self.t[self.k] + (x - self.t[self.k]) *
345         extrapolate = False
346
347     out = np.empty((len(x), prod(self.c.shape[1:])),
348                   dtype=self._evaluate(x, nu, extrapolate, out)
349                   .dtype)
350     out = out.reshape(x_shape + self.c.shape[1:])
351     if self.axis != 0:
352         # transpose to move the calculated values to t
353         l = list(range(out.ndim))
354         l = l[x_ndim:x_ndim+self.axis] + l[:x_ndim] + l[x_ndim+self.axis:]
355         out = out.transpose(l)
356     return out
357
358 def _evaluate(self, xp, nu, extrapolate, out):
359     _bspl.evaluate_spline(self.t, self.c.reshape(self.c
360     .shape), xp, nu, extrapolate, out)
361
362 def _ensure_c_contiguous(self):
363     """
364     Ensure that the C-contiguous array self.c and t
365     are C-contiguous.
366     """
367     if not self.c.flags['C_CONTIGUOUS']:
368         self.c = np.ascontiguousarray(self.c)
369     if not self.t.flags['C_CONTIGUOUS']:
370         self.t = np.ascontiguousarray(self.t)
```

Loops

Functions

***Classes,
Modules &
Packages***

***Programming
Errors &
Debugging***

Creating new types

- ❖ A `class` defines a new type
- ❖ It can provide
 - ❖ class variables & functions
 - ❖ instance variables & functions

Classes—example of code reuse

```
1 class Library:
2     description = 'This is a Library'
3
4     def __init__(self, name):
5         # name the library
6         self.name = name
7         # create empty book storage on initialization
8         self.storage = list()
9
10    def addBook(self, book):
11        self.storage.append(book)
12
13    def getAllBooks(self):
14        return tuple(self.storage)
15
16 myLib = Library('Bodleian Library')
17 myLib.addBook('The Art of Computer Programming (D. Knuth)')
```

Modules

- ❖ Every `.py` file is a module
- ❖ Modules can host functions, variables, and classes
- ❖ Imported modules with `import` statement
- ❖ Should not have blocks of code that are immediately executed
- ❖ Explicit reference to module scope: `global`
- ❖ Name of module available as global variable `__name__`

Modules—example of code reuse

mystringutils.py

```
1 #  
2 # A module for all kinds of string utils  
3 #  
4  
5 def findSubstringInStrings(stringCollection,  
6     pattern):  
7     occ = list()  
8     for i, s in enumerate(stringCollection):  
9         j = s.find(pattern)  
10        while j != -1:  
11            occ.append((i, j))  
12            j = s.find(pattern, j+1)  
13    return occ
```

myscript.py

```
1 #!/usr/bin/env python3  
2  
3 import mystringutils  
4  
5 if __name__ == '__main__':  
6     myStringList = ['the_rain_in_spain',  
7         'ain\'t_no_sunshine',  
8         'she_was_greeted_with_disdain']  
9  
10    occOfAin = mystringutils.  
11        findSubstringInStrings(myStringList,  
12            'ain')  
13    print(occOfAin)
```

Modules—example of code reuse

mystringutils.py

```
1 #
2 # A module for all kinds of string utils
3 #
4
5 def findSubstringInStrings(stringCollection,
6                             pattern):
7     occ = list()
8     for i, s in enumerate(stringCollection):
9         j = s.find(pattern)
10        while j != -1:
11            occ.append((i, j))
12            j = s.find(pattern, j+1)
13    return occ
```

myscript.py

```
1 #!/usr/bin/env python3
2
3 import mystringutils as su
4
5 if __name__ == '__main__':
6     myStringList = ['the_rain_in_spain',
7                    'ain\'t_no_sunshine',
8                    'she_was_greeted_with_disdain']
9
10    occOfAin = su.findSubstringInStrings(
11        myStringList, 'ain')
12    print(occOfAin)
```

Modules—example of code reuse

mystringutils.py

```
1 #
2 # A module for all kinds of string utils
3 #
4
5 def findSubstringInStrings(stringCollection,
6     pattern):
7     occ = list()
8     for i, s in enumerate(stringCollection):
9         j = s.find(pattern)
10        while j != -1:
11            occ.append((i, j))
12            j = s.find(pattern, j+1)
13    return occ
```

myscript.py

```
1 #!/usr/bin/env python3
2
3 from mystringutils import
4     findSubstringInStrings
5
6 if __name__ == '__main__':
7     myStringList = ['the_rain_in_spain',
8         'ain\'t_no_sunshine',
9         'she_was_greeted_with_disdain']
10
11     occOfAin = findSubstringInStrings(
12         myStringList, 'ain')
13     print(occOfAin)
```


Packages

- ❖ Way of structuring multiple modules into a directory hierarchy
- ❖ Package directories must contain a `__init__.py` file
- ❖ Can be imported the same way as modules
- ❖ Python itself offers many packages, and even more third-party packages are available through *package managers* such as `conda`

Quiz

- ❖ In Python, a class is _____ for an object.
- a nuisance an instance a blueprint a distraction

- ❖ Consider the following class:

```
1 class Dog:
2     def __init__(self, name, age):
3         self.name = name
4         self.age = age
```

What is the correct statement to instantiate a Dog object?

- ❖ Dog('Rufus', 3)
- ❖ Dog(self, 'Rufus', 3)
- ❖ Dog.__init__('Rufus', 3)

Quiz

- ❖ In Python, a class is _____ for an object.
- a nuisance an instance a blueprint ✓ a distraction

- ❖ Consider the following class:

```
1 class Dog:
2     def __init__(self, name, age):
3         self.name = name
4         self.age = age
```

What is the correct statement to instantiate a Dog object?

- ❖ Dog('Rufus', 3) ✓
- ❖ Dog(self, 'Rufus', 3)
- ❖ Dog.__init__('Rufus', 3)

Recap

Summary

- ❖ Code reuse through
 - ❖ Classes
 - ❖ Modules & Packages

What comes next?

- ❖ Write your first classes and modules
- ❖ Due date for this week's exercises is **Wednesday, Dec 6, 2pm, 2023.**

Next lecture: Input, file processing & text mining ...