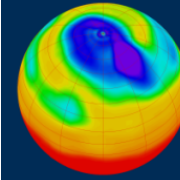




**National Centre for
Atmospheric Science**

NATURAL ENVIRONMENT RESEARCH COUNCIL



**Centre for Environmental
Data Analysis**

SCIENCE AND TECHNOLOGY FACILITIES COUNCIL
NATURAL ENVIRONMENT RESEARCH COUNCIL

Object-Oriented Programming (OOP)

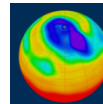
Thanks to all contributors:

Alison Pamment, Sam Pepler, Ag Stephens, Stephen Pascoe,
Kevin Marsh, Anabelle Guillory, Graham Parton, Esther
Conway, Eduardo Damasio Da Costa, Wendy Garland,
Alan Iwi and Matt Pritchard.



**National Centre for
Atmospheric Science**

NATURAL ENVIRONMENT RESEARCH COUNCIL



**Centre for Environmental
Data Analysis**

SCIENCE AND TECHNOLOGY FACILITIES COUNCIL
NATURAL ENVIRONMENT RESEARCH COUNCIL

Let's see how OOP is useful in everyday Python:

```
>>> s = "some silly string"
>>> s.upper()
'SOME SILLY STRING'
>>> s.find("t")
12
>>> s.replace("silly", "sensible").title()
'Some Sensible String'
```

And you can actually interrogate this object s to find out their methods:

```
>>> dir(s)
['__add__', '__class__', '__contains__',
 '__delattr__', ..., '__str__', '__subclasshook__',
 '_formatter_field_name_split', '_formatter_parser',
 'capitalize', 'center', 'count', 'decode', 'encode',
 'endswith', 'expandtabs', 'find', 'format', 'index',
 'isalnum', 'isalpha', 'isdigit', 'islower', 'isspace',
 'istitle', 'isupper', 'join', 'ljust', 'lower',
 'lstrip', 'partition', 'replace', 'rfind', 'rindex',
 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split',
 'splitlines', 'startswith', 'strip', 'swapcase',
 'title', 'translate', 'upper', 'zfill']
```

And you can find out which class *s* is an instance of:

```
>>> type(s)  
<type 'str'>
```

You can build your own **class** for your own domain:

```
class FileAnalyser(object):  
    "A class above the rest"  
  
    def __init__(self, path):  
        items = open(path).read().split()  
        self.data = []  
        for item in items:  
            self.data.append(float(item))  
  
    def max(self):  
        return max(self.data)  
  
    def mean(self):  
        return sum(self.data) / len(self.data)
```

Then create an instance of your class and use it:

```
$ cat some_data.txt      # Inside the data file...  
1000 750 500 250 0
```

```
$ python  
>>> da = FileAnalyser("some_data.txt")  
>>> da.max()  
1000.0  
>>> da.mean()  
500.0
```

You can make use of help() on your own class:

```
>>> help(FileAnalyser)
```

```
Help on class FileAnalyser in module __main__:
```

```
class FileAnalyser(__builtin__.object)
```

```
| A class above the rest
```

```
|
```

```
| Methods defined here:
```

```
|
```

```
| __init__(self, path)
```

```
|
```

```
| max(self)
```

```
|
```

```
| mean(self)
```

```
|
```

```
|
```

```
| -----  
| Data descriptors defined here:
```

Let's look in detail at our class...:

```
class FileAnalyser(object):  
    "A class above the rest"
```

Class Definition:
Defines the class name.

Optionally include a doc
string below.

Let's look in detail at our class...:

```
class FileAnalyser(object):  
    "A class above the rest"
```

```
def __init__(self, path):  
    items = open(path).read().split()  
    self.data = []  
    for item in items:  
        self.data.append(float(item))
```

`__init__` is the
"constructor" method:

- Not necessary
- Very useful
- Always called when class is first created.

"self" means "belonging to this instance/object":

- Needed for all attributes that you want to be visible to every part of the object.

Let's look in detail at our class...:

```
class FileAnalyser(object):  
    "A class above the rest"  
  
    def __init__(self, path):  
        items = open(path).read().split()  
        self.data = []  
        for item in items:  
            self.data.append(float(item))  
  
    def max(self):  
        return max(self.data)
```

Now we add more
methods:

- “self” is always required as first argument.

Let's look in detail at our class...:

```
class FileAnalyser(object):  
    "A class above the rest"  
  
    def __init__(self, path):  
        items = open(path).read().split()  
        self.data = []  
        for item in items:  
            self.data.append(float(item))  
  
    def max(self):  
        return max(self.data)  
  
    def mean(self):  
        return sum(self.data) / len(self.data)
```

More about OOP

Most python packages use OOP extensively.

We'll come across many examples in the next sessions.

E.g.:

```
from netCDF4 import Dataset
# Create HDF5 *format*, classic *model*
dataset = Dataset('data/test.nc', 'w', format='NETCDF4_CLASSIC')
print dataset.file_format
```

OOP Terminology (1)

class

Tell Python to make a new type of thing.

object

Two meanings: the most basic type of thing, and any instance of some thing.

instance

What you get when you tell Python to create a variable of given class.

def

How you define a method of a class.

self

Inside the methods in a class, self is a variable for the instance/object being accessed.

OOP Terminology (2)

inheritance

The concept that one class can inherit traits from another class, much like you and your parents.

attribute

A property that classes have that are from composition and are usually variables.

is-a

A phrase to say that something inherits from another, as in a "salmon" is-a "fish."

Learning to "speak" OOP

See: <http://learnpythonthehardway.org/book/ex41.html>

class

Tell Python to make a new type of thing.

object

Two meanings: the most basic type of thing, and any instance of some thing.

instance

What you get when you tell Python to create a class.

def

How you define a function inside a class.

self

Inside the functions in a class, self is a variable for the instance/object being accessed.

Learning to "speak" OOP

inheritance

The concept that one class can inherit traits from another class, much like you and your parents.

composition

The concept that a class can be composed of other classes as parts, much like how a car has wheels.

attribute

A property classes have that are from composition and are usually variables.

is-a

A phrase to say that something inherits from another, as in a "salmon" is-a "fish."

has-a

A phrase to say that something is composed of other things or has a trait, as in "a salmon has-a mouth."

Learning to "speak" OOP

class X(Y)

"Make a class named X that is-a Y."

class X(object): def __init__(self, J)

"class X has-a __init__ that takes self and J parameters."

class X(object): def M(self, J)

"class X has-a function named M that takes self and J parameters."

foo = X()

"Set foo to an instance of class X."

foo.M(J)

"From foo get the M function, and call it with parameters self, J."

foo.K = Q

"From foo get the K attribute and set it to Q."