

# Introduction to programming for Geoscientists

## Revision Lecture 2

Christian Jacobs

[c.jacobs10@imperial.ac.uk](mailto:c.jacobs10@imperial.ac.uk)

# Classes and Objects

## Definitions

- **Class**: a programming construct used to encapsulate data (attributes) and functions (behaviour) in a single package.
    - They make programs more manageable.
    - They allow information hiding.
  - **Object**: a specific **instance** of a class.
  - **Instantiation**: the process of creating an object from a class.
- 
- A **class** is like a **blueprint/template** from which **objects** can be **created/instantiated**.
  - Cake analogy: A **cake** (object) is created using a **recipe** (class).

# Classes and Objects

## Terminology

- Variables belonging to a class are known as **attributes**.
- Functions belonging to a class are known as **methods**.
- It is good practice to change attributes via **get/set** methods, not directly.

# Classes and Objects

## Example

- General example of a class definition in Python:

```
class ClassName:
    def __init__(self, input1, input2):
        self.a = input1
        self.b = input2
    def method1(self, input1):
        print "Hello %s!" % input1
    def method2(self):
        print "a = %d, b = %d" % (self.a, self.b)
```

A = ClassName(5, 10)

A.method1("world") → "Hello world!"

A.method2() → 5, 10

- self helps to distinguish class attributes from other variables.
- \_\_init\_\_ is a special method used to initialise/setup objects.

# Classes and Objects

## Cake

- The following example describes cake:

```
class Cake:
```

```
    def __init__(self, cake_type):
```

```
        self.type = cake_type
```

```
        self.number_of_slices = 10
```

```
    def eat_slice(self):
```

```
        self.number_of_slices = self.number_of_slices - 1
```

```
        print "%d slices remaining." % self.number_of_slices
```

```
A = Cake("Chocolate")
```

```
B = Cake("Lemon")
```

```
print A.type → "Chocolate"
```

```
print B.type → "Lemon"
```

```
B.eat_slice() → "9 slices remaining."
```

# NumPy Arrays

## Definition

- **Arrays**: data structures which comprise a finite number of **elements**/items/values.
- Similar to lists (or lists of lists), but are of a **fixed size**.
- Arrays are generally **faster than lists** because array elements are stored in **contiguous** areas of memory.

# NumPy Arrays

## linspace and zeros

- Two useful functions for creating arrays:
  - `linspace(start, end, n)`: creates an array of  $n$  uniformly distributed points in the interval  $[start, end]$ .
  - `zeros(n)`: creates an array of  $n$  elements that are all initialised to zero.
- Or...simply define as a list of lists and cast/convert to an array:

$\begin{bmatrix} 0 & 12 & -1 \\ -1 & -1 & -1 \\ 11 & 5 & 5 \end{bmatrix}$	<pre>a = [ [0, 12, -1],       [-1, -1, -1],       [11, 5, 5] ] a = array(a)</pre>
---	---

- Remember to include `from numpy import *` in your program.

# NumPy Arrays

## Referencing/accessing elements

- Referencing/accessing elements in an array is the same as referencing list elements.
- `a[i][j]` accesses the element at row `i` and column `j`.
- Row first, Column second.



# NumPy Arrays

## Vectorised functions

- A **vectorised** function can accept an array as its input...
- ...and for each element of that array, compute the result...
- ...and output all results in a new array.

```
from numpy import *  
a = linspace(0, 1, 10)  
result = sin(a) # Result is an array here.
```

- This is like doing:

```
from numpy import *  
a = linspace(0, 1, 10)  
result = zeros(10)  
for i in range(0, 10):  
    result[i] = sin(a[i])
```

- But with vectorised functions, this for loop is implicit.

# Files

## Reading

- Open a file (for reading) using `f = open("file_name_here.txt", "r")`.
- A file can be thought of as a **list of strings**, with each string being a single line of the file.
- We can select one line at a time using `f.readline()`, ...
- ...or select all the lines in the file using `f.readlines()`.
- It is good practice to **close** the file (once it is no longer needed) with `f.close()`

# Files

## Writing

- Open a file (for writing) using `f = open("file_name_here.txt", "w")`.
- Write a string to the file using `f.write(string_to_write_here)`.
- Once again, remember to close the file after use.

# Dictionaries

## Definition

- **Dictionary**: a data structure whose elements are **key-value pairs**.
- The key does not have to be an integer.
- Example: `d = {"Barcelona":11.0, "Lleida":6.0, "Tarragona":8.0 }`
- `d.keys()` → `["Barcelona", "Lleida", "Tarragona"]`
- `d.values()` → `[11.0, 6.0, 8.0]`
- Items can be added using `b[new_key_here] = value_here`.
- ...or existing items can be accessed using the key: `print b["Barcelona"]`.

# Strings

## Definition

- **String**: a **sequence of characters**, terminated by an **end-of-line marker**.
- Each character  
in a string can be accessed in the same way as elements of a list or array:  
`s = "hello"`  
`print s[0] → "h"`  
`print s[2] → "l"`
- `split` breaks up strings wherever a user-defined **delimiter** is encountered.
- For example, if the delimiter is a comma:  
`s = "hello world, Python is really awesome."`  
`print s.split(",") → ["hello world", " Python is really awesome"]`
- Remember: Strings are **immutable/constant** data structures. They cannot be modified once defined.