C - EXTENSIONS
C - extensions

Some times there are time critical parts of code which would benefit from compiled language

90/10 rule: 90 % of time is spent in 10 % of code
  – only a small part of application benefits from compiled code

It is relatively straightforward to create a Python interface to C-functions
  – data is passed from Python, routine is executed without any Python overheads
C - extensions

- C routines are build into a shared library
- Routines are loaded dynamically with normal import statements

```python
>>> import hello
>>> hello.world()
```

- A library **hello.so** is looked for
- A function **world** (defined in hello.so) is called
Creating C-extension

1) Include Python headers

```c
#include <Python.h>
```

2) Define the C-function

```c
PyObject* world_c(PyObject *self, PyObject *args)
{
    printf("Hello world!\n");
    Py_RETURN_NONE;
}
```

- Type of function is always PyObject
- Function arguments are always the same
  (args is used for passing data from Python to C)
- A macro Py_RETURN_NONE is used for returning "nothing"
Creating C-extension

3) Define the Python interfaces for functions

```c
... static PyMethodDef functions[] = {
    "world", world_c, METH_VARARGS, 0},
    "honey", honey_c, METH_VARARGS, 0},
    /* "Sentinel" notifies the end of definitions */
};
```

- **world** is the function name used in Python code, **world_c** is the actual C-function to be called
- Single extension module can contain several functions (world, honey, ...)
Creating C-extension

4) Define the module initialization function

```c
PyMODINIT_FUNC inithello(void)
{
    (void) Py_InitModule("hello", functions);
}
```

- Extension module should be build into `hello.so`
- Extension is module is imported as `import hello`
- Functions/interfaces defined in functions are called as `hello.world()`, `hello.honey()`, ...
Creating C-extension

5) Compile as shared library

```
$ gcc -shared -o hello.so -I/usr/include/python2.6 -fPIC hello.c
$

$ python
Python 2.4.3 (#1, Jul 16 2009, 06:20:46)
[GCC 4.1.2 20080704 (Red Hat 4.1.2-44)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import hello
>>> hello.world()
Hello world!
```

- The location of Python headers (/usr/include/...) may vary in different systems
- Use exercises/include_paths.py to find out yours!
```c
#include <Python.h>

PyObject* world_c(PyObject *self, PyObject *args) {
    printf("Hello world!\n");
    Py_RETURN_NONE;
}

static PyMethodDef functions[] = {
    {"world", world_c, METH_VARARGS, 0},
    {0, 0, 0, 0}
};

PyMODINIT_FUNC inithello(void) {
    (void) Py_InitModule("hello", functions);
}
```
Passing arguments to C-functions

PyObject* pass_c(PyObject *self, PyObject *args)
{
    int a;
    double b;
    char* str;
    if (!PyArg_ParseTuple(args, "ids", &a, &b, &str))
        return NULL;
    printf("int %i, double %f, string %s\n", a, b, str);
    Py_RETURN_NONE;
}

PyArg_ParseTuple checks that function is called with proper arguments “ids” : integer, double, string and does the conversion from Python to C types
Create and return Python integer from C variable. A “d” would create Python double etc.

Returning tuple:
```
Py_BuildValue("(ids)", a, b, str);
```
Operating with NumPy array

```c
#include <Python.h>
#include <numpy/arrayobject.h>

PyObject* array(PyObject *self, PyObject *args)
{
    PyArrayObject* a;
    if (!PyArg_ParseTuple(args, "O", &a))
        return NULL;

    int size = PyArray_SIZE(a);    /* Total size of array */
    double *data = PyArray_DATA(a); /* Pointer to data */
    for (int i=0; i < size; i++) {
        data[i] = data[i] * data[i];
    }

    Py_RETURN_NONE;
}
```

NumPy provides API also for determining the dimensions of an array etc.
Function `import_array()` should be called in the module initialization function when using NumPy C-API.

```c
PyMODINIT_FUNC inithello(void)
{
    import_array();
    (void) Py_InitModule("hello", functions);
}
```
Tools for easier interfacing

- Cython
- SWIG
- pyrex
- f2py (for Fortran code)
Summary

Python can be extended with C-functions relatively easily
C-extension build as shared library
It is possible to pass data between Python and C code
Extending Python:
http://docs.python.org/extending/
NumPy C-API
http://docs.scipy.org/doc/numpy/reference/c-api.html