Cython Cheat Sheet

Cython (https://cython.org/) is a C extension to the Python programming language that allows for static compilation and strong type declaration. It's aim is to speed up Python significantly.

Compiling Cython

Jupyter notebooks (temporary)

```
%load_ext cython
import Cython
# write code using cython
# run cell
# nb! print functions have to be used
# %%cython has to be written
# to each cell using it
# %%cython -a outputs function documentation
# (which parts are compiled to C
# and which are Python)

%% cython
cdef int multiplyWithOne(int x):
    x*=1;
    return x
print(multiplyWithOne(9))
...
```

pyximport (temporary)

1. write a file.pyx containing the source code
2. import (and autocompile) file.pyx using pyximport:

```
# using pyximport for auto-compilation
import pyximport
pyximport.install()
import file #imports file.pyx
```

setup.py script

1. write a file.pyx containing the source code
2. compile it using a setup.py file containing:

```
from distutils.core import setup
from Cython.Build import cythonize

setup(ext_modules = cythonize(``file.pyx''))
```

3. compile it:

```
$ python setupy.py build_ext --inplace
```

Files

.pyx	Source code containing Cython code
.c	C source code generated from .pyx file
.so	shared object file from compilation
.html	documentation from %%cython -a
./build/	temporary files from compilation

Misc

Calling external C functions

```
cdef extern from ''file.h'':
    double someFunctionsName(double)

from libc.math cimport cos
cdef double functionName(double x)
    return cos(x)
```


Class and function declaration

Classes:

```
cdef class TestClass(object):
    # making variables accessible from Python
    # using public:
    cdef public int a,b
    cdef public double c

    def __init__(self):
        self.a = 0
        self.b = 1
        self.c = 3.14
```

Functions:

```
def functionExample(int x, float y):
    # function is callable from Python and Cython
    return y * x

cdef functionExample(int x, float y):
    # function is callable from Cython only
    # function is optimized
    return y*x

cpdef functionExample(int x, float y):
    # function is callable from Cython (optimized)
    # function is callable from Python (unoptimized)
    return y*x
```

Further, the return type can be added to the function:

```
cdef float functionExample(int x, float y):
    # function is optimized & callable from Cython only
    return y*x
```

Type declaration

Cython offers static typing of the following types:

int A = 1	integer
float A = 1.0	floating point (fp)
double A = 1.0	double precision fp
char *A = '1'	string (pointer (*)!!)
Py_ssize_t	signed integer for NumPy indices

Memory management

Manual memory management can be done by importing memory handling functions from the C standard library:

```
void* malloc
void* realloc
void free
```