C/C++ Python wrappers
Cython

Veronica Arias

Universidad de los Andes

Universidad de los Andes

But first a Python overview...

- High level programming language
- Efficient high level data structures
- Python interpreter
- Very simple language that has a very straightforward syntax.

Python basics

learnpython.org

My first script:

```
print("hello world")
```

```
hello world
```

Indentation:

```
x = 1
if x == 1:
    # indented four spaces
    print("x is 1.")
```

```
x is 1
```

Python basics

Variables and types:

```
myfloat = 7.0
myfloat = float(7)
print(myfloat)
```

7.0

```
one = 1
two = 2
three = one + two
print(three)
```

3

```
hello = "hello"
world = "world"
helloworld = hello + " " + world
print(helloworld)
```

hello world

Python basics

lists:

```
mylist = []
mylist.append(1)
mylist.append(2)
mylist.append(3)
print(mylist[0])
print(mylist[1])
print(mylist[2])
```

```
1
2
3
```

```
for x in mylist:
    print(x)
```

```
1
2
3
```

```
mylist = [1,2,3]
print(mylist[1])
```

```
2
```

Python basics

Aritmetic operators:

```
number = 1 + 2 * 3 / 4.0
print(number)
```

2.5

```
remainder = 11 % 3
print(remainder)
```

2

```
squared = 7 ** 2
Cubed = 2 ** 3
```

```
even_numbers = [2,4,6,8]
odd_numbers = [1,3,5,7]
all_numbers = odd_numbers + even_numbers
print(all_numbers)
```

[1, 3, 5, 7, 2, 4, 6, 8]

Python basics

Example basic string operations:

```
astring = "Hello world!"
print(astring[2:8:2])
```

low

Example conditions:

```
x = 2
print(x == 2) # prints out True
print(x == 3) # prints out False
print(x < 3) # prints out True
```

True
False
True

```
name = "John"
age = 23
if name == "John" and age == 23:
    print("Your name is John, and you are also 23 years
old.")

if name == "John" or name == "Rick":
    print("Your name is either John or Rick.")
```

Python basics

“for” loop:

```
for x in range(3):
    print(x)
```

```
0
1
2
```

```
for x in range(3, 6):
    print(x)
```

```
3
4
5
```

```
for x in range(3, 8, 2):
    print(x)
```

```
3
5
7
```

“while” loop:

```
count = 0
while count < 4:
    print(count)
    count += 1
```

```
0
1
2
3
4
```

Python basics

Functions:

```
def my_function():
    print("Hello From My Function!")

def sum_two_numbers(a, b):
    return a + b


my_function()
```

```
Hello From My Function!
```

```
x = sum_two_numbers(1,2)
print(x)
```

```
3
```

Python basics

Modules and packages: numpy example:

```
import numpy as np

height = [1.87,  1.87, 1.82, 1.91, 1.90, 1.85]
np_height = np.array(height)


print(2*height)          [1.87, 1.87, 1.82, 1.91, 1.90, 1.85, 1.87, 1.87, 1.82, 1.91, 1.90, 1.85]

print(2*np_height)              [3.74  3.74  3.64  3.82  3.8   3.7]


np_height[np_height > 1.87]
```

we can perform element-wise calculations!

Python vs C/C++

Both have advantages and disadvantages that can be summed up into:

C/C++: Faster

Python: Very simple language that has a very straightforward syntax.

Python is a very simple language that has a very straightforward syntax.
Examples:

C++

```
#include <stdio.h>
using namespace std;
int main(){
printf("Hello World!\n");
return 0;
}
```

Python

```
print("Hello World!")
```

C

```c
#include <stdio.h>
using namespace std;
int main(){
FILE *myFile;
myFile = fopen("my_file.txt", "r");
int num;
while(true){
fscanf(myFile, "%d", &num);
if(feof(myFile)) break;
printf("%d ", num);
}
fclose(myFile);
return 0;
}
```

Python

```python
import numpy as np
Data=np.genfromtxt("my_file.txt")
```

C/C++ Python wrappers

- Programming language based on Python
- Extra syntax allowing for optional static type declarations:
- Superset of Python language
- high-level, object oriented, functional, and dynamic programming

How?

“The source code gets translated into optimized C/C++ code and compiled as Python extension modules. This allows for both very fast program execution and tight integration with external C libraries, while keeping up the high programmer productivity for which the Python language is well known.”

Cython tutorial, S.Behnel, R.W.Bradshaw, D. S. Seljebotn, Proceedings of the 8th Python in Science Conference (SciPy 2009)

Building a Cython module using distutils

hello.pyx

```
def say_hello_to(name):
        print("Hello %s!" % name)
```

$cython hello.pyx
$python setup.py build_ext --inplace
$python prueba.py

setup.py

```
from distutils.core import setup
from distutils.extension import Extension
from Cython.Distutils import build_ext
ext_modules = [Extension("hello", ["hello.pyx"])]
setup(
        name = 'Hello world app',
        cmdclass = {'build_ext': build_ext},
        ext_modules = ext_modules
)
```

prueba.py

```
import numpy as np
from hello import say_hello_to

say_hello_to("Veronica")
```

Compiling python code in Cython

ejemplo1.pyx

```
from math import sin

def f(x):
        return sin(x**2)

def integrate_f(a, b, N):
        s = 0
        dx = (b-a)/N

        for i in range(N):
                s += f(a+i*dx)
        return s * dx
```

setup.py

```
from distutils.core import setup
from distutils.extension import Extension
from Cython.Distutils import build_ext
ext_modules = [Extension("ejemplo1", ["ejemplo1.pyx"])]
setup(
        name = 'ejemplo 1 app',
        cmdclass = {'build_ext': build_ext},
        ext_modules = ext_modules
)
```

prueba.py

```
import numpy as np
import ejemplo1

a=0
b=2.0
N=1000
print(ejemplo1.integrate_f(a,b,N))
```

$cython ejemplo1.pyx
$python setup.py build_ext --inplace
$python prueba.py

Faster code by adding data types

ejemplo1.pyx

```
from math import sin

def f(double x):
        return sin(x**2)

def integrate_f(double a, double b, int N):
        cdef int i
        cdef double s, dx
        s = 0
        dx = (b-a)/N
        for i in range(N):
                s += f(a+i*dx)
        return s * dx
```

prueba.py

```
import numpy as np
import ejemplo1

a=0
b=2.0
N=1000
print(ejemplo1.integrate_f(a,b,N))
```

$cython ejemplo1.pyx
$python setup.py build_ext --inplace
$python prueba.py

setup.py

```
from distutils.core import setup
from distutils.extension import Extension
from Cython.Distutils import build_ext
ext_modules = [Extension("ejemplo1", ["ejemplo1.pyx"])]
setup(
        name = 'ejemplo 1 app',
        cmdclass = {'build_ext': build_ext},
        ext_modules = ext_modules
)
```

Calling external C functions

ejemplo1.pyx

```
cdef extern from "math.h":
        double sin(double)

cdef double f(double x):
        return sin(x**2)

def integrate_f(double a, double b, int N):
        cdef int i
        cdef double s, dx
        s = 0
        dx = (b-a)/N
        for i in range(N):
                s += f(a+i*dx)
        return s * dx
```

prueba.py

```
import numpy as np
import ejemplo1

a=0
b=2.0
N=1000
print(ejemplo1.integrate_f(a,b,N))
```

$cython ejemplo1.pyx
$python setup.py build_ext --inplace
$python prueba.py

setup.py

```
from distutils.core import setup
from distutils.extension import Extension
from Cython.Distutils import build_ext
ext_modules = [Extension("ejemplo1", ["ejemplo1.pyx"])]
setup(
        name = 'ejemplo 1 app',
        cmdclass = {'build_ext': build_ext},
        ext_modules = ext_modules
)
```

More examples in the Hands-on session!

References

- http://learnpython.org/
- Cython tutorial, S.Behnel, R.W.Bradshaw, D. S. Seljebotn, Proceedings of the 8th Python in Science Conference (SciPy 2009)
- https://pythonprogramming.net/introduction-and-basics-cython-tutorial/
- http://cython.org/#documentation