# Introduction

## HPC Python

Antonio Gómez-Iglesias
agomez@tacc.utexas.edu

October 30th, 2014

# Why Python

- Easy!
- Nice, readable code
- Great for prototyping
- Many third party libraries

# Data Types

## Dynamic language, but also a strongly typed language

- Objects have a type, which is determined at runtime
- A variable is a value bound to a name: the value has a type, but the variable doesn't
- The interpreter keeps track of all variable types
- You can't do anything that's incompatible with the type of data you're working with:
  - You can do 'string+string' and it will concatenate the strings
  - You can do 'integer+integer'
  - You can't do 'string+integer'

# Data Structures

## Python List

- Dynamic arrays
- Indexed structure
- Items: Python objects
- Items of different types
- Insertion and deletion at random positions

# Data Structures

## Dictionary

- Associative arrays (key - value pairs)
- Indexed by key (string or number)
- Key: unique
- Value: any Python object
- Main operation: store a value with some key and extract the value given the key

# Python in HPC

- You'll hear that Python is slow
- If it's slow, why should you use it?
- If you already have a Python code, what should you do?

# Python in Stampede

- python/2.7.3-epd-7.3.2
- python/2.7.6
- You can install your own modules:
  - `python setup.py install --user`
  - `python setup.py install --home=<dir>`
  - `pip install --user module_name`
- You can use virtualenv

# Before We Begin

## From XSEDE

```
ssh username@login.xsede.org
gsissh -p 2222 stampede.tacc.xsede.org
```

## Local

```
ssh -Y username@stampede.tacc.utexas.edu
```

## Python Exercises

```
cp ~train00/python-hpc.tar.gz .
tar -xzf python-hpc.tar.gz
module load intel/14.0.1.106
module load python/2.7.6
idev -t 2:00:00 -A TACC-HPC-PYTHON
```

# Profiling

`python -m cProfile [-o output_file] [-s sort_order] script.py`

examples/1_intro/profiling.py

```
1  from math import sqrt
2
3  def hello():
4    print "Hello world"
5
6  def sum():
7    for i in range(10000):
8      a = 1
9      b = 1
10     c = a+b
11
12 def vector():
13   a = [ 1., 2., 3., 4., 5.,
           6., 7.]*100000
14   for i in a:
15     t = sqrt(i**2)
16   r = a.reverse()
17   s = a.sort()
18   print reduce(lambda x, y:
           x + y, a)
19
20 if __name__=='__main__':
21   hello()
22   sum()
23   vector()
```

```
Hello world
2800000.0
         1400008 function calls in 0.391 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall
         filename:lineno(function)
        1    0.002    0.002    0.391    0.391 prof.py:1(<module>)
        1    0.105    0.105    0.389    0.389 prof.py:12(vector)
   699999    0.061    0.000    0.061    0.000 prof.py:18(<lambda>)
        1    0.000    0.000    0.000    0.000 prof.py:3(hello)
        1    0.000    0.000    0.001    0.001 prof.py:6(sum)
   700000    0.038    0.000    0.038    0.000 {math.sqrt}
        1    0.089    0.089    0.089    0.089 {method 'sort'}
        1    0.000    0.000    0.000    0.000 {range}
        1    0.095    0.095    0.156    0.156 {reduce}
```

# License