Python

An open source programming language conceived in the late 1980s. It is both compiled and interpreted (compilation step hidden).

Since Python is interpreted it is slower than C/C++ but is fast enough for most applications.
Why Python?

In Python we can do a variety of things

- Work with an interactive interpreter that makes it easy to experiment and try code
- Write object-oriented programming... but you do not need to use classes for everything like in java
- Work with built in modules for text processing and regular expressions
- Automatically convert variable types
- Work with scipy and numpy and do scientific computation like in matlab
But most of all...

Python is easy to read as white space is part of the syntax! Instead of enclosing blocks of code in brackets, we simply indent instead.

Python may be the easiest language to pick up and learn because although there may not be 10 ways (cough Perl) to do something, the way you expect to do it works.
Data types

- **int**: 3
- **float**: 2.5
- **str**: 'abc', "abc"
- **list**: [0,1,2], [0,1,'the']
- **tuple**: (0,1,2), (0,1,'the')
- **dict**: {'a': 1, 'Ohio': 'Columbus', 2: 'b'}

Strings and tuples cannot be changed once they are created.
Let's go play with the interpreter. To start the basic interpreter type python. If you have ipython installed, type ipython to get python with syntax highlighting, word completion and more!
Becareful with ints!

$$\gg \ 1/2$$
0
$$\gg \ 1./2$$
.5

Integer division truncates... :( 
"hello" + "world" "helloworld" # concatenation

"hello" * 3 "hellohellohello" # repetition

"hello" [0] "h" # indexing

"hello" [-1] "o" # (from end)

"hello" [1:4] "ello" # slicing

len("hello") 5 # size

"hello" < "jello" True # comparison

"e" in "hello" True # search
somelist = [1, "abc", "5", 2, [3, 5, "wewt"]]
somelist[0]
1
somelist[2]
'5'
somelist[4][2]
'wewt'
somelist[1:3]
['abc', '5']  
    <---- [a:b] starts at a and goes up to 1 before b
somelist[:2]
[1,'abc']
del(somelist[2])  
    <--- remove an element
list.reverse() - reverses a list
list.append(obj) - appends obj to a list
list.sort() - sort a list
list.index(obj) - finds the first occurrence of a value in a list
list.pop() - pop off last element
help(list) - get documentation
Everything is an object
Working with Dictionaries

- \( d = \{ 'a': 1, 'b': 2 \} \)
- \( d \text{.keys()} \) - returns list of keys
- \( d \text{.values()} \) - returns a list of values
- \( d \text{.items()} \) - returns a list of pairs of keys and values
- \( d \text{.has_key(arg)} \) - is arg a key in d?

\[
d = \{ "duck" : 3, "geese" : "are pretty" \}
d["duck"]
3
d["duck"] = "i like ducks"
\]
On Punctuation

- **parentheses ( ):** defining tuples, calling functions, grouping expressions
  - `t = ('a', 'b', 'c')`
  - `z = func(x, y)`
  - `z = 2.*(x+3) + 4./(y-1.)`

- **square brackets []:** indexing and slicing (lists, dictionaries, arrays)
  - `element = lst[i]`
  - `val = dct['k']`
  - `y = a[i,j]` (numpy array)
  - `sublist = list[i:j]`

- **curly braces {}:** dictionary creation
  - `dct = {'a': 'apple', 'b': 'bear', 'c': 'cat'}`
On variables

- no need to declare
- need to assign
- not strongly typed
- the variable `>` in interactive mode stores the most recent output value (good for arithemtic)
- **everything** is a "variable" (functions, classes, modules)
When we do
\( x = y \)
we are making \( x \) reference the object \( y \) refers to. So

\[
a = [1, 2, 3] \\
b = a \\
a.append(4) \\
print b \\
[1, 2, 3, 4]
\]