Python Language Basics I

Georgia Advanced Computing Resource Center
University of Georgia
Zhuofei Hou, HPC Trainer
zhuofei@uga.edu
Outline

• What is GACRC?
• Hello, Python!
• General Lexical Conventions
• Basic Built-in Data Types
What is GACRC?

Who Are We?
- Georgia Advanced Computing Resource Center
- Collaboration between the Office of Vice President for Research (OVPR) and the Office of the Vice President for Information Technology (OVPIT)
- Guided by a faculty advisory committee (GACRC-AC)

Why Are We Here?
- To provide computing hardware and network infrastructure in support of high-performance computing (HPC) at UGA

Where Are We?
- http://gacrc.uga.edu (Web)
- http://wiki.gacrc.uga.edu (Wiki)
- http://gacrc.uga.edu/help/ (Web Help)
- https://wiki.gacrc.uga.edu/wiki/Getting_Help (Wiki Help)
# GACRC Users September 2015

<table>
<thead>
<tr>
<th>Colleges &amp; Schools</th>
<th>Depts</th>
<th>Pls</th>
<th>Users</th>
</tr>
</thead>
<tbody>
<tr>
<td>Franklin College of Arts and Sciences</td>
<td>14</td>
<td>117</td>
<td>661</td>
</tr>
<tr>
<td>College of Agricultural &amp; Environmental Sciences</td>
<td>9</td>
<td>29</td>
<td>128</td>
</tr>
<tr>
<td>College of Engineering</td>
<td>1</td>
<td>12</td>
<td>33</td>
</tr>
<tr>
<td>School of Forestry &amp; Natural Resources</td>
<td>1</td>
<td>12</td>
<td>31</td>
</tr>
<tr>
<td>College of Veterinary Medicine</td>
<td>4</td>
<td>12</td>
<td>29</td>
</tr>
<tr>
<td>College of Public Health</td>
<td>2</td>
<td>8</td>
<td>28</td>
</tr>
<tr>
<td>College of Education</td>
<td>2</td>
<td>5</td>
<td>20</td>
</tr>
<tr>
<td>Terry College of Business</td>
<td>3</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>School of Ecology</td>
<td>1</td>
<td>8</td>
<td>22</td>
</tr>
<tr>
<td>School of Public and International Affairs</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>College of Pharmacy</td>
<td>2</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td><strong>Centers &amp; Institutes</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>TOTALS:</strong></td>
<td>49</td>
<td>233</td>
<td>1029</td>
</tr>
<tr>
<td>Centers &amp; Institutes</td>
<td>PIs</td>
<td>Users</td>
<td></td>
</tr>
<tr>
<td>----------------------------------------------------</td>
<td>-----</td>
<td>-------</td>
<td></td>
</tr>
<tr>
<td>Center for Applied Isotope Study</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Center for Computational Quantum Chemistry</td>
<td>3</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Complex Carbohydrate Research Center</td>
<td>6</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>Georgia Genomics Facility</td>
<td>1</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Institute of Bioinformatics</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Savannah River Ecology Laboratory</td>
<td>3</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>Skidaway Institute of Oceanography</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Center for Family Research</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Carl Vinson Institute of Government</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>19</td>
<td>59</td>
<td></td>
</tr>
</tbody>
</table>
Hello, Python!

- What is Python
- Where is Python on Clusters
- Run Python Interactively on Clusters
- Scientific Python Modules
- Scientific Python Distributions
What is Python

• Open source general-purpose scripting language (https://www.python.org/)

• Working with *procedural*, *object-oriented*, and *functional* programming

• Glue language with Interfaces to C/C++ (via SWIG), Object-C (via PyObjC), Java (Jython), and Fortran (via F2PY), etc. (https://wiki.python.org/moin/IntegratingPythonWithOtherLanguages)

• Mainstream version is 2.7.x; new version is 3.5.x *(as to March 2016)*
Where is Python on Clusters

Currently GACRC has two clusters *zcluster* and *Sapelo*:

<table>
<thead>
<tr>
<th>Version</th>
<th>Installation Path</th>
<th>Invoke command</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.4.3 (default)</td>
<td>/usr/bin</td>
<td>python</td>
</tr>
<tr>
<td>2.7.2</td>
<td>/usr/local/python/2.7.2</td>
<td>python2.7</td>
</tr>
<tr>
<td>2.7.8</td>
<td>/usr/local/python/2.7.8</td>
<td>/usr/local/python/2.7.8/bin/python</td>
</tr>
<tr>
<td>3.3.0</td>
<td>/usr/local/python/3.3.0</td>
<td>python3</td>
</tr>
<tr>
<td>3.4.0</td>
<td>/usr/local/python/3.4.0</td>
<td>python3.4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Version</th>
<th>Installation Path</th>
<th>Invoke command</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.6.6 (default)</td>
<td>/usr/bin</td>
<td>python</td>
</tr>
<tr>
<td>2.7.8</td>
<td>/usr/local/apps/python/2.7.8</td>
<td>module load python/2.7.8 ; python</td>
</tr>
<tr>
<td>3.4.3</td>
<td>/usr/local/apps/python/3.4.3</td>
<td>module load python/3.4.3 ; python3</td>
</tr>
</tbody>
</table>

[https://wiki.gacrc.uga.edu/wiki/Python](https://wiki.gacrc.uga.edu/wiki/Python) ; [https://wiki.gacrc.uga.edu/wiki/Python-Sapelo](https://wiki.gacrc.uga.edu/wiki/Python-Sapelo)
Run Python Interactively on Clusters

• Run default python interactively on clusters’ interactive nodes (qlogin):

```
zhuofei@compute-14-9:~$ python
Python 2.4.3 (#1, Oct 23 2012, 22:02:41)
[GCC 4.1.2 20080704 (Red Hat 4.1.2-54)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 7
>>> e = 2
>>> a**e
49
>>>   
[zhuofei@n15 ~]$ python
Python 2.6.6 (r266:84292, Jan 22 2014, 09:42:36)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-4)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 7
>>> e = 2
>>> a**e
49
>>>   
```
Run Python Interactively on Clusters

- Run Python script interactively on clusters’ interactive nodes (qlogin):

  ```
  zhuofei@compute-14-9:~$ python myScript.py
  2.4.3 (#1, Oct 23 2012, 22:02:41)
  [GCC 4.1.2 20080704 (Red Hat 4.1.2-54)]
  49
  
  [zhuofei@n15 ~]$ python myScript.py
  2.6.6 (r266:84292, Jan 22 2014, 09:42:36)
  [GCC 4.4.7 20120313 (Red Hat 4.4.7-4)]
  49
  ```

- myScript.py:

  ```python
  import sys
  print sys.version
  a = 7
  e = 2
  print a**e
  ```
Run Python Interactively on Clusters

- Run Python script as an *executable* interactively on clusters’ *interactive nodes*:

```
./myScript.py
```

- myScript.py:

```
#!/usr/local/python/2.7.2/bin/python
import sys
print sys.version
a = 7; e = 2
print a**e
```

```python
#!/usr/bin/env python
import sys
print sys.version
a = 7; e = 2
print a**e
```

```
tell system where the python lives
```

```
the env program will locate the python according to PATH
```
Scientific Python Modules

• Python has a large collection of proven **built-in** modules included in standard distributions:

  https://docs.python.org/2/py-modindex.html

  https://docs.python.org/3/py-modindex.html

• Packages for **scientific** modules:

  - NumPy
  - SciPy
  - Sympy
  - Biopy
  - Matplotlib
Scientific Python Modules

- NumPy: Matlab-ish capabilities, fast N-D array operations, linear algebra, etc. ([http://www.numpy.org/](http://www.numpy.org/))
- matplotlib: High quality plotting ([http://matplotlib.org](http://matplotlib.org))

A scientific Python distribution may include all those packages for you!
Scientific Python Distributions

- Anaconda
  - “A Python distribution including over 195 of the most popular Python packages for science, math, engineering, data analysis”
  - Supports Linux, Mac and Windows (https://www.continuum.io/)

- Python(x,y)
  - Windows only (http://python-xy.github.io/)

- WinPython
  - Windows only (http://winpython.github.io/)
Anaconda with Spyder IDE on my local computer:
Scientific Python Distributions

- **Anaconda** is installed on GACRC zcluster and Sapelo:

<table>
<thead>
<tr>
<th>Version</th>
<th>Installation Path</th>
<th>Python Version</th>
<th>Export (2.3.0 as example)</th>
<th>Invoke Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3.0</td>
<td>/usr/local/anaconda/2.3.0</td>
<td>2.7.11</td>
<td>export PATH=/usr/local/anaconda/2.3.0/bin:$PATH export PYTHONPATH=/usr/local/anaconda/2.3.0/bin:/usr/local/anaconda/2.3.0/lib/python2.7:$PYTHONPATH</td>
<td>python</td>
</tr>
<tr>
<td>3-2.2.0</td>
<td>/usr/local/anaconda/3-2.2.0</td>
<td>3.4.3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Version</th>
<th>Installation Path</th>
<th>Python Version</th>
<th>Module Load (2.2.0 as example)</th>
<th>Invoke Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2.0</td>
<td>/usr/local/apps/anaconda/2.2.0</td>
<td>2.7.11</td>
<td>module load anaconda/2.2.0</td>
<td>python</td>
</tr>
<tr>
<td>2.5.0</td>
<td>/usr/local/apps/anaconda/2.5.0</td>
<td>2.7.11</td>
<td></td>
<td>python</td>
</tr>
<tr>
<td>3-2.2.0</td>
<td>/usr/local/apps/anaconda/3-2.2.0</td>
<td>3.4.3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
General Lexical Conventions

• A code sample:

```
x = 10; y = "Hello!"
z = 3.14

if z == 3.14 or y == "Hello!":
x = x + 1
y = y + " Python!"

print x
print y
```

• Semicolon ; to separate statements on the same line

• Hash # denotes a comment

• Assignment uses =; comparison uses ==

• Logical operators are words: and, or, not

• Consistent indentation within a block (4 spaces)

• For numbers: + - * / % are as expected

• For strings: + means concatenation

• The basic printing statement: print

Output:

```
zhuofei@compute-14-9:~$ python ./myScript_1.py
11
Hello! Python!
```
Basic Built-in Data Types

- “Python is a dynamically typed language where variable names are bound to different values, possibly of varying types, during program execution. Variables names are untyped and can be made to refer to any type of data.”
  — Python Essential Reference, 4th ed.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>10</td>
<td>Integer</td>
</tr>
<tr>
<td>a</td>
<td>3.24</td>
<td>Floating-point number</td>
</tr>
<tr>
<td>a</td>
<td>“Hello!”</td>
<td>String</td>
</tr>
<tr>
<td>a</td>
<td>True</td>
<td>Boolean (True/False)</td>
</tr>
</tbody>
</table>
# Basic Built-in Data Types

<table>
<thead>
<tr>
<th>Type Category</th>
<th>Type Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Numbers</strong></td>
<td>int</td>
<td><code>i = 10;</code> integer</td>
</tr>
<tr>
<td></td>
<td>long</td>
<td><code>l = 73573247851;</code> arbitrary-precision integer <em>(Python 2 only!)</em></td>
</tr>
<tr>
<td></td>
<td>float</td>
<td><code>f = 3.14;</code> floating point</td>
</tr>
<tr>
<td></td>
<td>complex</td>
<td><code>c = 3 + 2j;</code> complex</td>
</tr>
<tr>
<td></td>
<td>bool</td>
<td><code>b = True;</code> Boolean <em>(True or False)</em></td>
</tr>
<tr>
<td><strong>Sequences</strong></td>
<td>str</td>
<td><code>s = “Hello! Python”;</code> character string</td>
</tr>
<tr>
<td></td>
<td>list</td>
<td><code>lst = [1, 2, &quot;abc&quot;, 2.0];</code> list of any typed elements <em>(mutable!)</em></td>
</tr>
<tr>
<td></td>
<td>tuple</td>
<td><code>t = (1, 2, &quot;abc&quot;, 2.0);</code> record of any typed elements <em>(immutable!)</em></td>
</tr>
<tr>
<td><strong>Mapping</strong></td>
<td>dict</td>
<td><code>d = {1:&quot;apple&quot;, 2:&quot;&quot;};</code> mapping dictionary of any typed pairs of key:value</td>
</tr>
</tbody>
</table>
Basic Built-in Data Types

- **List**: A mutable sequence of arbitrary objects of any type

  ```python
  list1 = [1, "David", 3.14, "Mark", "Ann"]
  index : 0 1 2 3 4 ➔ \text{Index}_{\text{max}} = \text{Length} - 1
  ```

  Indexed by integer, starting with zero:

  ```python
  a = list1[1]   # returns the 2nd item "David" ; a = "David"
  list1[0] = "John"  # changes the 1st item 1 to "John" ; list1 = ["John", "David", 3.14, "Mark", "Ann"]
  ```

- **Empty list** is created by:

  ```python
  list2 = []  # an empty list
  list2 = list()  # an empty list
  ```

- Append and insert new items to a list:

  ```python
  list1.append(7)  # appends a new item to the end ; list1 = ["John", "David", 3.14, "Mark", "Ann", 7]
  list1.insert(2, 0)  # inserts a new item into a middle ; list1 = ["John", "David", 0, 3.14, "Mark", "Ann", 7]
  ```
Basic Built-in Data Types

- Extract and reassign a portion of a list by slicing operator \([i, j]\), with an index range of \(i \leq k < j\):

  ```
  a = list1[0:2]  # returns ["John", "David"] ; the 3rd item 0 is NOT extracted!
  b = list1[2:]   # returns [0, 3.14, "Mark", "Ann", 7]
  list1[0:2] = [-3, -2, -1]  # replaces the first two items with the list on the right
  list1 = [-3, -2, -1, 0, 3.14, "Mark", "Ann", 7]
  ```

- Delete items:

  ```
  del list1[0]   # deletes the 1st item ; list1 = [-2, -1, 0, 3.14, "Mark", "Ann", 7]
  del list1[0:4] # delete a slice of the first 4 items ; list1 = ["Mark", "Ann", 7]
  ```

- Concatenate and multiply lists:

  ```
  list2 = [8, 9]  # creates a new list
  list3 = list1 + list2  # list3 = ["Mark", "Ann", 7, 8, 9]
  ```
Basic Built-in Data Types

- Count occurrences of items:
  
  ```python
  list4.count(“Mark”)  # returns 3
  ```

- Remove an item from a list:
  
  ```python
  list1.remove(“Ann”)  # Search for “Ann” and remove it from list1; list1 = [“Mark”, 7]
  ```

- Sort a list in place:
  
  ```python
  list5 = [10, 34, 7, 8, 9]  # creates a new list
  list5.sort()  # list5 = [7, 8, 9, 10, 34]
  ```

- Reverse a list in place:
  
  ```python
  list5.reverse()  # list5 = [34, 10, 9, 8, 7]
  ```

- Copy a list (shallow copy):
  
  ```python
  list6 = list(list5)  # list6 is a shallow copy of list5
  ```
Basic Built-in Data Types

- **Tuple**: A immutable record of arbitrary objects of any type

  \[ t1 = (1, \text{“David”}, 3.14, \text{“Mark”}, \text{“Ann”}) \]

<p>|</p>
<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
</table>

  Indexed by integer, starting with zero:

  ```python
  a = t1[1] # returns the 2nd item “David” ; a = “David”
t1[0] = “John” # Wrong operations! Tuple is immutable!
  ```

  0-tuple (empty tuple) and 1-tuple:

  ```python
  t2 = () # an empty tuple ; same as t2 = tuple()
t3 = ("apple",) # a tuple containing 1 item ; note the trailing comma!
  ```

  Extract a portion of a list by slicing operator \([i, j]\), with an index range of \(i\leq k\leq j\):

  ```python
  a = t1[0:2] # returns (1, “David”) ; the 3rd item 3.14 is NOT extracted!
b = t1[2:] # returns (3.14, “Mark”, “Ann”)
  ```
Basic Built-in Data Types

- **Concatenate and multiply tuples:**
  
  ```
  t4 = t1 + t3  # t4 = (1, “David”, 3.14, “Mark”, “Ann”, “apple”)
  t5 = t3 * 3   # t5 = ("apple", “apple”, “apple”)
  ```

- **Count occurrences of items:**
  
  ```
  t5.count("apple")  # returns 3
  ```

- **Extract values in a tuple without using index:**
  
  ```
  t6 = (1, 2, 3)  # create a new tuple
  a, b, c = t6    # a = 1 ; b = 2 ; c = 3
  person = (“John”, “Smith”, 30)  # another example
  first_name, last_name, age = person  # first_name = “John” ; last_name = “Smith” ; age = 30
  ```
Basic Built-in Data Types

- **String**: A *immutable* sequence of characters
  
  ```python
  s = "HELLO"
  ```

  ```
  \[
  \text{index: }\quad 0 \quad 1 \quad 2 \quad 3 \quad 4
  \]
  ```

  To create a string, enclose characters in single(''), double(""""), or triple("""") quotes:

  ```python
  a = 'Mark'
  b = "Python is good!"
  c = "This function is for calculation of PI"
  d = 'we say "yes!"'
  d = "we say 'yes!'"
  d = """we say 'yes!'""
  ```

  # `''` is usually for short strings
  # """" is usually for string messages to be visible to human
  # """" or """" is usually for Python doc strings; can be used for a string
  # spanning multiple lines

  # same type of quotes used to start a string must be used to terminate it!
Basic Built-in Data Types

Indexed by integer, starting with zero:

```python
b = a[4]          # b = 'o'
```

Extract a portion of a string by slicing operator [i, j], with an index range of i<=k<j:

```python
b = a[0:5]  # b = 'Hello'  
b = a[6:]   # b = 'Python!'  
b = a[4:7]  # b = 'o P'
```

Concatenate and multiply strings:

```python
c = "My name is John."  # a new string  
d = a + ' ' + c  # d = "Hello Python! My name is John."  
d = a * 2  # d = "Hello Python!Hello Python!"
```
Basic Built-in Data Types

Conversion between numbers and strings:

- `a = '77' ; b = '23'`  # two numeric strings
- `c = a + b`  # `c = '7723'`; string concatenation; NO numeric evaluation!
- `c = int(a) + int(b)`  # `c = 100`
- `c = float(a) + int(b)`  # `c = 100.0`

- `i = 77 ; f = 23.0`  # two numbers
- `a = str(i)`  # `a = '77'`
- `b = str(f)`  # `b = '23.0'`

Common string methods:

Next Page!
Basic Built-in Data Types

\[
s = \text{“python is good!”}
\]

<table>
<thead>
<tr>
<th>String Methods</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.capitalize()</td>
<td>Capitalize the 1st character</td>
<td>“Python is good!”</td>
</tr>
<tr>
<td>s.center(w, p)</td>
<td>Centers s in a field of length w, padding with p</td>
<td>(w=30, p=‘-‘) : -------python is good!-------- python is good!--------------------------</td>
</tr>
<tr>
<td>s.ljust(w, p) s.rjust(w, p)</td>
<td>Left-align/Right-align s with w and p</td>
<td></td>
</tr>
<tr>
<td>s.count(substr)</td>
<td>Counts occurrences of substr</td>
<td>s.count(‘o’) returns 3</td>
</tr>
<tr>
<td>s.isalpha() s.isdigit() s.isalnum() s.islower() s.isupper()</td>
<td>True if all characters in s are alphabetic/digits/alphanumeric/lowercase/uppercase</td>
<td>s.isalpha() returns True s.islower() returns True</td>
</tr>
<tr>
<td>s.find(substr)</td>
<td>Finds the 1st occurrence of substr or returns -1</td>
<td>s.find(‘good’) returns 10</td>
</tr>
<tr>
<td>s.index(substr)</td>
<td>Finds the 1st occurrence of substr or raises an error</td>
<td>s.index(‘good’) returns 10</td>
</tr>
<tr>
<td>s.replace(old, new)</td>
<td>Replaces a substring</td>
<td>s.replace(‘good’, ‘bad’) returns “python is bad!”</td>
</tr>
<tr>
<td>s.split(sep)</td>
<td>Splits a string using sep as a delimiter</td>
<td>s.split(‘is’) returns [‘python’, ‘good!’]</td>
</tr>
<tr>
<td>s.partition(sep)</td>
<td>Partitions a string based on sep; returns (head, sep, tail)</td>
<td>s.partition(‘is’) returns (‘python’, ‘is’, ‘good!’)</td>
</tr>
</tbody>
</table>
Basic Built-in Data Types

- Built-in operations common to all sequences: list, tuple, and string

```python
s = "python is good!"
list1 = [0, 1, 2, 3, 4]
```

<table>
<thead>
<tr>
<th>Operations</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>seq[i]</code></td>
<td>Returns the element at index <code>i</code></td>
<td><code>s[0]</code> returns ‘p’</td>
</tr>
<tr>
<td><code>seq[i:j]</code></td>
<td>Returns a slice with an index range of <code>i&lt;=k&lt;j</code></td>
<td><code>s[0:6]</code> returns ‘python’</td>
</tr>
<tr>
<td><code>len(seq)</code></td>
<td>Number of elements in <code>seq</code></td>
<td><code>len(s)</code> returns 15</td>
</tr>
<tr>
<td><code>min(seq)</code></td>
<td>Minimum value in <code>seq</code></td>
<td><code>min(s)</code> returns ‘ ’</td>
</tr>
<tr>
<td><code>max(seq)</code></td>
<td>Maximum value in <code>seq</code></td>
<td><code>max(s)</code> returns ‘y’</td>
</tr>
<tr>
<td><code>sum(seq)</code></td>
<td>Sum of items in <code>seq</code>; ONLY working for numeric list or tuple!</td>
<td><code>sum(list1)</code> returns 10</td>
</tr>
<tr>
<td><code>all(seq)</code></td>
<td>True if all items in <code>seq</code> are True</td>
<td><code>all(list1)</code> returns False</td>
</tr>
<tr>
<td><code>any(seq)</code></td>
<td>True if any item in <code>seq</code> is True</td>
<td><code>any(list1)</code> returns True</td>
</tr>
</tbody>
</table>
Thank You!

Let’s talk about *Python function and class* on next class!

I : Python introduction, running python, Python built-in data types
II : function (procedural and functional programming) and class (OOP)
III: module, package, and practical code sample