Python Language Basics I

Georgia Advanced Computing Resource Center (GACRC)
Enterprise Information Technology Services (EITS)
The University of Georgia
Outline

• GACRC
• Python World
• General Lexical Conventions
• Basic Built-in Data Types
GACRC

- A high-performance-computing (HPC) center at the UGA
- Provide to the UGA research and education community an advanced computing environment:
  - HPC computing and networking infrastructure located at the Boyd Data Center
  - Comprehensive collection of scientific, engineering and business applications
  - Consulting and training services

Wiki: http://wiki.gacrc.uga.edu
Support: https://uga.teamdynamix.com/TDClient/Requests/ServiceCatalog?CategoryID=11593
Web Site: http://gacrc.uga.edu
Python World

- What is Python
- Scientific Python Modules
- Scientific Python Distributions
- Run Python Interactively on Sapelo2
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 other languages, like C/C++ (via SWIG), Object-C (via PyObjC), Java (Jython), and Fortran (via F2PY), etc. (https://wiki.python.org/moin/IntegratingPythonWithOtherLanguages)
• Last Python2 version is 2.7.16; Latest Python3 version is 3.8.2; Current Python3 version on Sapelo2 is 3.7.4
Scientific Python Modules

- Python has a large collection of **built-in** modules included in standard distributions, e.g., io, os, sys, datetime, argparse, etc.:
  
  https://docs.python.org/3/index.html
  
  https://docs.python.org/3/library/index.html

- Packages for **scientific** modules:
  
  - NumPy
  - Biopython
  - SciPy
  - TensorFlow
  - Matplotlib
  - PyTorch
Scientific Python Modules

- NumPy: Matlab-ish capabilities, fast N-D array operations, linear algebra, etc.  
  (http://www.numpy.org/)

- SciPy: Fundamental library for scientific computing (http://www.scipy.org/)

- matplotlib: High quality plotting (http://matplotlib.org/)

- TensorFlow: Open source platform for machine learning  
  (https://www.tensorflow.org/)

- PyTorch: Open source machine learning library (https://pytorch.org/)
Scientific Python Distributions

• Anaconda
  ➢ Comes with 1,500 packages selected from PyPI as well as the conda package and virtual environment manager
  ➢ Supports Linux, Mac and Windows (https://www.anaconda.com/)

• Python(x,y)
  ➢ A scientific-oriented Python Distribution based on Qt and Spyder
  ➢ Windows only (https://python-xy.github.io/)

• WinPython
  ➢ A free open-source portable distribution of the Python
  ➢ Windows only (https://github.com/winpython)
Anaconda with Spyder IDE on my local computer:
Run Python Interactively on Sapelo2

- Run python interactively on interactive node (use qlogin from login node)

zhuofei@sapelo2-sub2 ~> $ qlogin
qsub: waiting for job 2367783.sapelo2 to start
qsub: job 2367783.sapelo2 ready

zhuofei@n204 ~> $ module load Python/3.7.4-GCCcore-8.3.0
zhuofei@n204 ~> $ python
Python 3.7.4 (default, Jan 30 2020, 18:11:14)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information. >>> a = 7
>>> e = 2
>>> a**e
49
>>>
Run Python Interactively on Sapelo2

- **script.py:**
  ```python
  print("Hello, World!")
  a = 7
  e = 2
  print(a**e)
  ```

- Run a Python script on interactive node (use `qlogin` from login node):
  ```bash
  zhuofei@sapelo2-sub2 ~$ qlogin
  qsub: waiting for job 2367783.sapelo2 to start
  qsub: job 2367783.sapelo2 ready

  zhuofei@n204 ~$ module load Python/3.7.4-GCCcore-8.3.0
  zhuofei@n204 ~$ python script.py
  Hello, World!
  49
  ```
General Lexical Conventions

• A Python code clip:

```python
x = 10; y = "Hello!"  # this is a comment
z = 3.14  # z is a floating number

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
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.

```
# a is created to refer to an integer
a = 10

# a is referring to a floating-point number now
a = 3.24

# a is referring to a string now
a = “Hello!”

# a is referring to a boolean (True/False) now
a = True
```
# Basic Built-in Data Types

<table>
<thead>
<tr>
<th>Type Category</th>
<th>Type Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numbers</td>
<td>int</td>
<td>i = 10; integer</td>
</tr>
<tr>
<td></td>
<td>long</td>
<td>l = 73573247851; arbitrary-precision integer (Python 2 only!)</td>
</tr>
<tr>
<td></td>
<td>float</td>
<td>f = 3.14; floating point</td>
</tr>
<tr>
<td></td>
<td>complex</td>
<td>c = 3 + 2j; complex</td>
</tr>
<tr>
<td></td>
<td>bool</td>
<td>b = True; Boolean (True or False)</td>
</tr>
<tr>
<td>Sequences</td>
<td>str</td>
<td>s = “Hello! Python”; character string</td>
</tr>
<tr>
<td></td>
<td>list</td>
<td>lst = [1, 2, ”abc”, 2.0]; list of any typed elements (mutable!)</td>
</tr>
<tr>
<td></td>
<td>tuple</td>
<td>t = (1, 2, “abc”, 2.0); record of any typed elements (immutable!)</td>
</tr>
<tr>
<td>Mapping</td>
<td>dict</td>
<td>d = {1:”apple”, 2:’”}; 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"]
  ```

  Indexes:
  - Indexed by integer, starting with **zero**:
    - `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:
    - `list2 = []`  # an empty list
    - `list2 = list()`  # an empty list
  - Append and insert **new items** to a list:
    - `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]`

  - `Index_{max} = Length − 1`
Basic Built-in Data Types

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

  - ```
    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]
  list5.sort()  # creates a new list
  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
  
  \[ t1 = (1, \text{“David”}, 3.14, \text{“Mark”}, \text{“Ann”}) \]

  index : 0 1 2 3 4

  - Indexed by integer, starting with zero:
    
    | Operation   | Result          |
    |-------------|-----------------|
    | `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:
    
    | Operation   | Result          |
    |-------------|-----------------|
    | `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 \(i \leq k < j\):
    
    | Operation   | Result          |
    |-------------|-----------------|
    | `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:**
  
  ```python
  t4 = t1 + t3  # t4 = (1, “David”, 3.14, “Mark”, “Ann”, “apple”)
  t5 = t3 * 3  # t5 = (“apple”, “apple”, “apple”)
  ```

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

- **Extract values in a tuple without using index:**
  
  ```python
  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"
  index:  0 1 2 3 4
  ```

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

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

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

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

- Concatenate and multiply strings:

  ```
  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 = “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

\[
s = \text{“python is good!”}
\]
\[
\text{list1} = [0, 1, 2, 3, 4]
\]

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