Python Language Basics II

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

• What is GACRC?

• Program Structure: Control Flow and Loop

• Function: Procedural and Functional Programming

• Class: Object-Oriented Programming (OOP)
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](http://gacrc.uga.edu) (Web)
- [http://wiki.gacrc.uga.edu](http://wiki.gacrc.uga.edu) (Wiki)
- [http://gacrc.uga.edu/help/](http://gacrc.uga.edu/help/) (Web Help)
- [https://wiki.gacrc.uga.edu/wiki/Getting_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>PIs</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><strong>40</strong></td>
<td><strong>214</strong></td>
<td><strong>970</strong></td>
</tr>
<tr>
<td><strong>TOTALS:</strong></td>
<td><strong>49</strong></td>
<td><strong>233</strong></td>
<td><strong>1029</strong></td>
</tr>
</tbody>
</table>
# GACRC Users September 2015

<table>
<thead>
<tr>
<th>Centers &amp; Institutes</th>
<th>PIs</th>
<th>Users</th>
</tr>
</thead>
<tbody>
<tr>
<td>Center for Applied Isotope Study</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Center for Computational Quantum Chemistry</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>Complex Carbohydrate Research Center</td>
<td>6</td>
<td>28</td>
</tr>
<tr>
<td>Georgia Genomics Facility</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>Institute of Bioinformatics</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Savannah River Ecology Laboratory</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>Skidaway Institute of Oceanography</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Center for Family Research</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Carl Vinson Institute of Government</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>19</td>
<td>59</td>
</tr>
</tbody>
</table>
Program Structure: Control Flow

If expression:
  statements
Elif expression:
  statements
......
Else:
  statements

If a < 0:
  print “a is negative”
Elif a == 0:
  print “a is zero”
Else:
  print “a is positive”

If a < b:
  smaller = a
Else:
  smaller = b

If name != “John”:
  pass # do nothing
Else:
  print “Hello, John!”

Note: Examples in this material are for Python2
Program Structure: Loop

- **while:**
  ```python
  while expression:
      statements
  ```

- **for:**
  ```python
  for i in seq:
      statements
  ```

**E.g.:**

```python
# s and t are two sequences
i = 0
while i < len(s) and i < len(t):
    x = s[i]
    y = t[i]
    print x + y
    i += 1
```

**E.g.:**

```python
# s and t are two sequences
for x, y in zip(s, t):
    print x + y
```

Hi, this is Not Python style!

s = [1, 2, 3, 4] : a list
t = (5, 6, 7, 8) : a tuple

s = [1, 2, 3, 4] : a list
t = (5, 6, 7, 8) : a tuple

[(1, 5), (2, 6), (3, 7), (4, 8)]
Function: Procedural and Functional Programming

• Function:

```python
def functionName (params):
    statements
```

E.g. 1:

```python
def f(x, y=0):
    return x + y
f(10, 2)  # returns 12
f(10)     # returns 10
```

E.g. 2

```python
def f(x, y=0):
    return (x+y, x−y, x*y, x**y)
```

```python
v1, v2, v3, v4 = f(10, 2)  # v1=12, v2=8, v3=20, v4=100
v1, v2, v3, v4 = f(10)     # v1=10, v2=10, v3=0, v4=1
```
**Function: Procedural Programming**

```python
import sys

def calPrincipal(portfolio):
    """
    Functions: 1. Read 4-column data line by line from a file: Name, Initial_Principal, Interest_Rate, Years
    2. Calculate final principal for each Name
    3. Store 5-column data as a record into a list
    """
    del portfolio[0:]  # clear the storing list
    f = open(sys.argv[1], 'r')  # open a file given as the 1st param. on command line
    for line in f.readlines():  # read lines; return a list; ending \n' is also read
        fields = line.split(',')  # split each line using ',' as a delimiter; return a list
        name = fields[0].strip()  # remove leading and trailing whitespace
        iniPrincipal = float(fields[1].strip())
        principal = iniPrincipal
        rate = float(fields[2].strip())
        years = int(fields[3].strip('
'))  # remove leading and trailing whitespace and \n'
        year = 1
        while year <= years:  # calculate final principal of 5 years for each Name
            principal = principal * (1+rate)
            year += 1
        portfolio.append((name, iniPrincipal, rate, years, principal))  # store 5-column record in list
```

**principal.txt**:

```
Tyler, 2000, 0.05, 5
Mark, 5000, 0.02, 5
Ann, 3000, 0.02, 5
John, 7000, 0.03, 5
```
Function: Procedural Programming

- Calling function (cont.):

```python
portfolio = []  # create a storing list
calPrincipal(portfolio)  # call function
for t in portfolio: print t  # output to screen; yes, you can put them on the same line
```
Function: Procedural Programming

- Run on zcluster’s interactive nodes (qlogin) with default python2.4.3:

  python2.4.3
  Python script
  command line argument

zhuofei@compute-14-9: python principal.py principal.txt
('Tyler', 2000.0, 0.050000000000000003, 5, 2552.5631250000006)
('Mark', 5000.0, 0.02, 5, 5520.4040159999995)
('Ann', 3000.0, 0.02, 5, 3312.2424096000004)
('John', 7000.0, 0.029999999999999999, 5, 8114.9185201)
Function: Functional Programming

• Functional Programming 101 – function Object: function itself is a data!

```python
def square(x):
    return x*x

def g(func):
    return func(10)

result = g(square)
```

• Functional Programming 101 – Decorator: a function wrapper to enhance/alter the behavior of the function object being wrapped

```python
def myDecorator(func):
    print "Hello, I am more human friendly!"
    return func

@myDecorator
def square(x):
    return x*x

print square(10)
```

# a simple regular function
# g is taking a function as a parameter, i.e., function itself is a data!
# result = 100
# special @ symbol means square is going to be wrapped by myDecorator
# here I am calling the wrapped square, and output:
# Hello, I am more human friendly!
# 100
Function: Functional Programming

• Functional Programming 101 – Generator: a function using `yield` keyword to produce a sequence of values for use in iteration

```python
def countdown(n):
    while n > 0:
        yield n
        n -= 1
    return

c = countdown(10)  # define a generator object
v1 = c.next()  # v1 = 10 ; next method of a generator is to produce a value each time it’s called
v2 = c.next()  # v2 = 9
v3 = c.next()  # v3 = 8

for v in c:
    print(v)  # normally we use a generator in a for loop
    # Output:
    # 7
    # 6
    # ……
    # 1
```
Class: Object-Oriented Programming (OOP)

• Python Class Basics
• Inheritance with Class
• Polymorphism and Class
Python Class Basics

- What are *Class Object* and *Instance Object*?
Python Class Basics

• *Class object* is a Python program *blueprint* or *factory* to generate concrete *instance objects*, and support *inheritance* and *polymorphism* of Python OOP

  o Set up a set of *class attributes*: *class variables, methods, etc.*

  o *ClassObject.ClassAttribute* to fully specify a class attribute
Python Class Basics

- *Instance object* is a *real* and *concrete* object the program processes, generated from a *class object*
  - Set up a set of *instance attributes*: per-instance attributes
  - *InstanceObject.InstanceAttribute* to fully specify a instance attribute
  - Class attributes are *shared* by all instance objects created
Python Class Basics

• Let’s try Account!

Class Object:
Class Attributes:
  Account.num_account
  Account.deposit
  Account.withdraw
  Account.inquiry

Instance Object:
Class Attributes:
  Account.num_account=1
  Account.deposit
  Account.withdraw
  Account.inquiry
Instance Attributes:
  self.name="John"
  self.balance=1000

• To create a class object ➔ class statement, e.g., class Account
Python Class Basics

• A simple example - Account class object

```python
class Account(object):
    # class statement is to create a class object with class attributes: num_account, deposit, withdraw, inquiry
    Account.num_account = 0  # class attribute Account.num_account is initialized

    def __init__(self, name, balance):
        # __init__ is used to initialize a instance object, which is referred by self in class definition
        self.name = name  # instance attribute self.name is initialized
        self.balance = balance  # instance attribute self.balance is initialized
        Account.num_account += 1  # class attribute Account.num_account is modified

    def deposit(self, amount):
        # class attribute Account.deposit
        self.balance += amount  # self.balance is modified

    def withdraw(self, amount):
        # class attribute Account.withdraw
        self.balance -= amount  # self.balance is modified

    def inquiry(self):
        # class attribute Account.inquiry
        return self.balance  # self.balance is returned
```

Class Attributes:
- Account.num_account
- Account.deposit
- Account.withdraw
- Account.inquiry

Instance Attributes:
- self.name
- self.balance
Python Class Basics

- To generate a concrete instance object ➔ Calling class object like a function!

```python
ins = Account("John", 1000)  # instance object ins has 2 instance attributes and 4 class attributes initialized!
print ins.name + " has a balance of " + str(ins.inquiry())  # use ins.name and call Account.inquiry
ins.deposit(1500)  # call Account.deposit
ins.withdraw(500)  # call Account.withdraw
print ins.name + " has a balance of " + str(ins.inquiry())  # use ins.name and call Account.inquiry
```

- Output

```
John has a balance of 1000
John has a balance of 2000
```

Call a method with an instance object: `ins.deposit(500)`
Inheritance with Class

• What is Inheritance?
Inheritance with Class

- *Inheritance is a* mechanism for creating a *new class* that *redefines* or *extends* the behavior of existing methods of *base class* ➔ *code reuse*

- Let’s try this!

---

**Base Class Object Account:**
- Class Attributes:
  - Account.num_account
  - Account.deposit
  - Account.withdraw
  - Account.inquiry

**Derived Class Object CheckingAccount:**
- Class Attributes:
  - Account.num_account
  - Account.deposit
  - Account.withdraw
  - Account.inquiry *(to be redefined!)*
Inheritance with Class

• A simple example

```python
class CheckingAccount(Account):
    # CheckingAccount is inherited from Account
    def __init__(self, name, balance):
        # __init__ of the derived class PersonalAccount
        Account.__init__(self, name, balance)
    # initialize base class Account by calling Account.__init__()

def inquiry(self):
    # method from base class is redefined!
    print "Checking Account :" + str(Account.inquiry(self)) + " : " + self.name
    # call Account.inquiry inside

ins = CheckingAccount("Peter", 0)
ins.deposit(500)
ins.inquiry()
```

• Output

Checking Account : 500 : Peter

Call a method with an instance object:
```py
ins.deposit(500)
```
Polymorphism and Class

- **What is Python Polymorphism?**
  - “Capability to get **correct behavior** of an instance **without knowing its type**.”
  - “Code shouldn’t care about **what an object is**, only about **what it does**!”
  - “Code may be written to work with any kind of object whatsoever as long as it has a **certain set of methods**.”

"when I see a bird that walks like a duck and swims like a duck and quacks like a duck, I call that bird a duck"

---

2. Learning Python, 5th ed.
Polymorphism and Class

• A simple example

```python
class DuckThing:
    def quack(self): pass
    def fly(self): pass

class Duck(DuckThing):
    def quack(self):
        print "Quack, quack!"
    def fly(self):
        print "Flap, flap!"

class Person(DuckThing):
    def quack(self):
        print "I'm Quacking!"
    def fly(self):
        print "I'm Flying!"

def testMachine(unknownThing):
    unknownThing.quack()
    unknownThing.fly()

duck = Duck()
Tom = Person()
testList = [duck, Tom]
for t in testList: testMachine(t)
```

Quack, quack!
Flap, flap!
I'm Quacking!
I'm Flying!

Polymorphism is here!
Function testMachine works with any object as long as it has quack and fly methods!
Thank You!

Let’s talk about *Python module and package* 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