

Python Language Basics II

Georgia Advanced Computing Resource Center (GACRC)

Enterprise Information Technology Services(EITS)

The University of Georgia

Outline

- What is GACRC?
- Program Structure and Function
- Module import and File Handling
- An Example: Bank Account
- Class: Object-Oriented Programming (OOP)

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>

Program Structure and Function

```
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 and Function

- while:

```
while expression:
    statements
```

E.g. :

```
# 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
```

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

← Hi, this is Not Python style!

s = [1, 2, 3, 4] : a list

t = (5, 6, 7, 8) : a tuple



[(1, 5), (2, 6), (3, 7), (4, 8)]

- for:

```
for i in seq:
    statements
```

E.g. :

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

Program Structure and Function

- Function:

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

E.g. 1:

```
def f(x, y=0):  
    return x + y  
  
f(10, 2)  
f(10)
```

y has a default value of 0
return a value

returns 12
returns 10

E.g. 2

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

y has default value of 0
return a tuple

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

Module import and File Handling

- How to import a module into your python session?

```
import io
import os, sys
import numpy
import numpy as np
from numpy import random
from numpy import random as npran
```

- A module can be imported multiple times in one python session

Module import and File Handling

- How to open and write a text file?

```
filehandle = open(filename, mode and fileType)
```

For example:

```
file1 = open('myfile.txt', 'rt')
```

```
file2 = open('newfile.txt', 'wt')
```

```
file3 = open('newfile.txt', 'at')
```

Mode: 'r', 'w', 'a' ; fileType: 't', 'b'

- readline, readlines, or write functions can be called via a file handler
- Once a file is read, it needs to be reopened for another reading

An Example: Bank Account

```
import sys                                # load the sys module

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(' \n')) # 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
```

An Example: Bank Account

- Calling function (cont.):

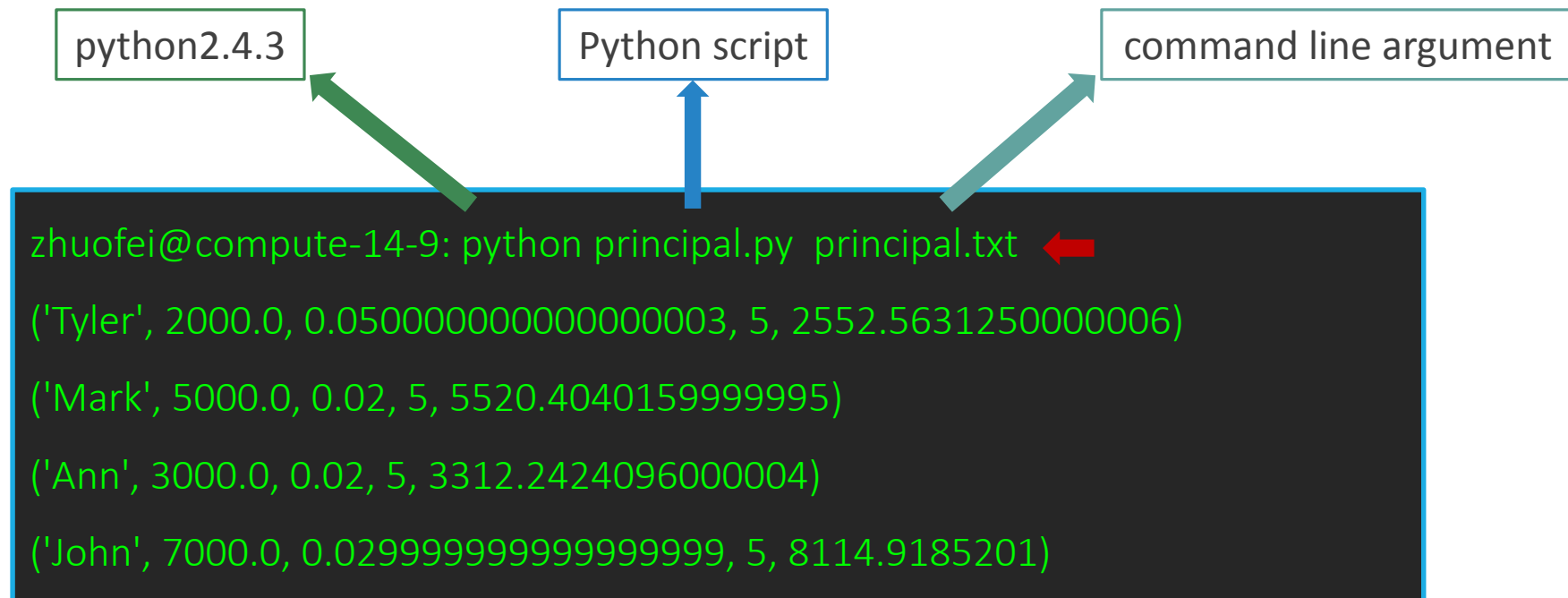
```
portfolio = []                # create a storing list
calPrincipal(portfolio)      # call function
for t in portfolio: print t  # output to screen; yes, you can put them on the same line
```

Next Page to Run!



An Example: Bank Account

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

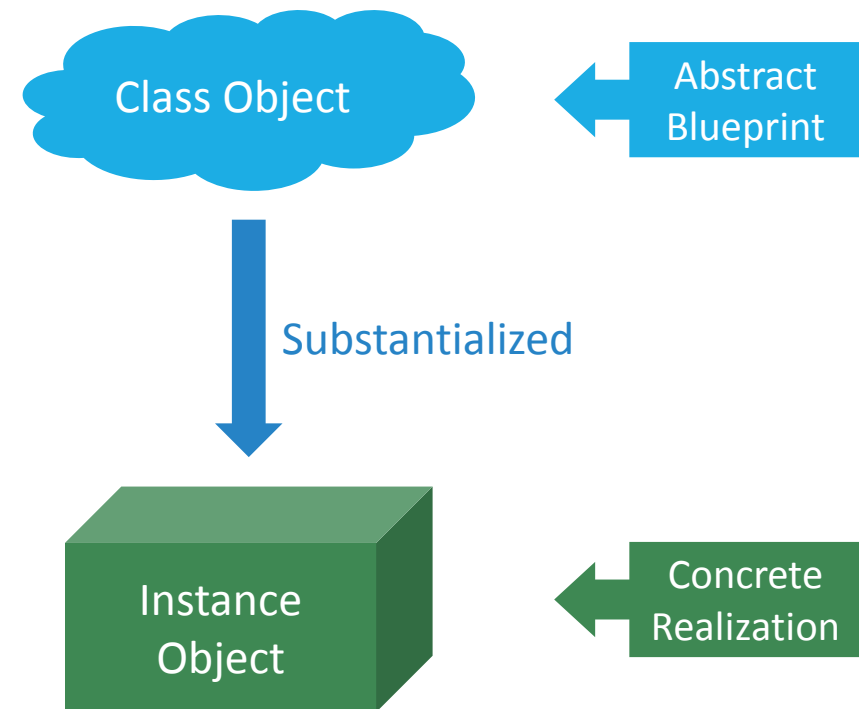
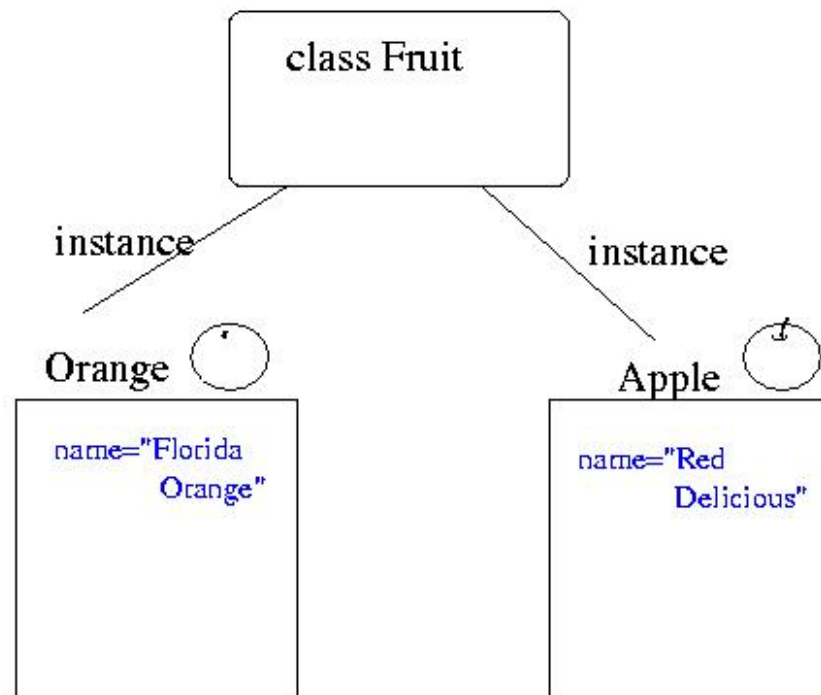


Class: Object-Oriented Programming (OOP)

- Python Class Basics
- Inheritance with Class
- Polymorphism and Class

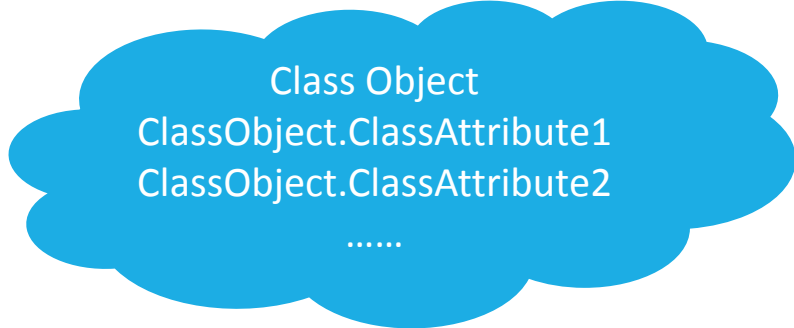
Python Class Basics

- 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
 - Set up a set of **class attributes**: *class variables, methods, etc.*
 - **ClassObject.ClassAttribute** to fully specify a class attribute

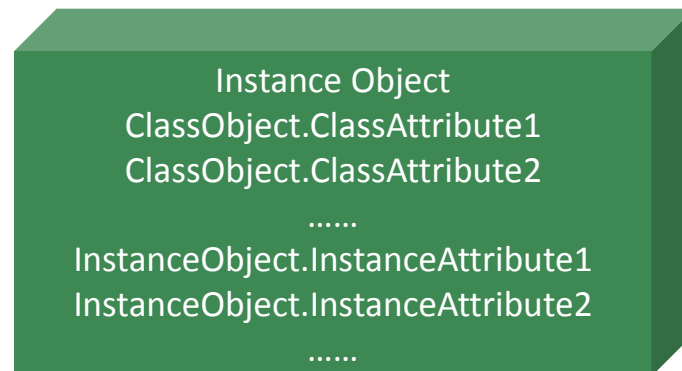


Class Object
ClassObject.ClassAttribute1
ClassObject.ClassAttribute2

.....

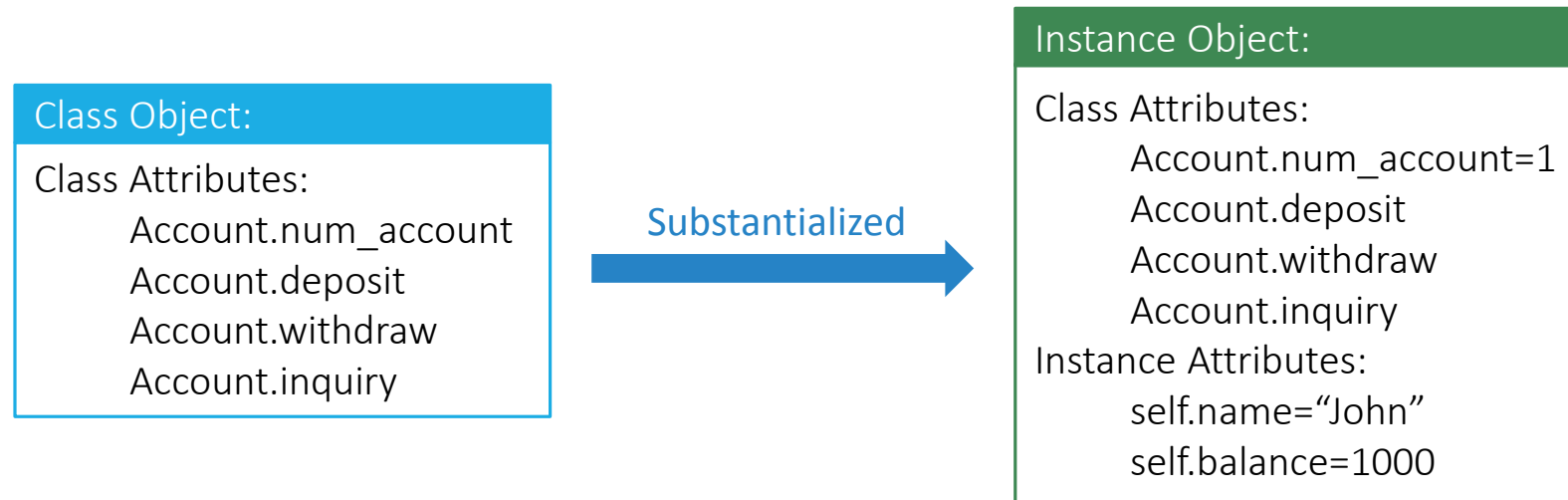
Python Class Basics

- *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!



- To create a class object → `class` statement, e.g., `class Account`

Python Class Basics

- A simple example - Account class object

```
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!**

```
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
```

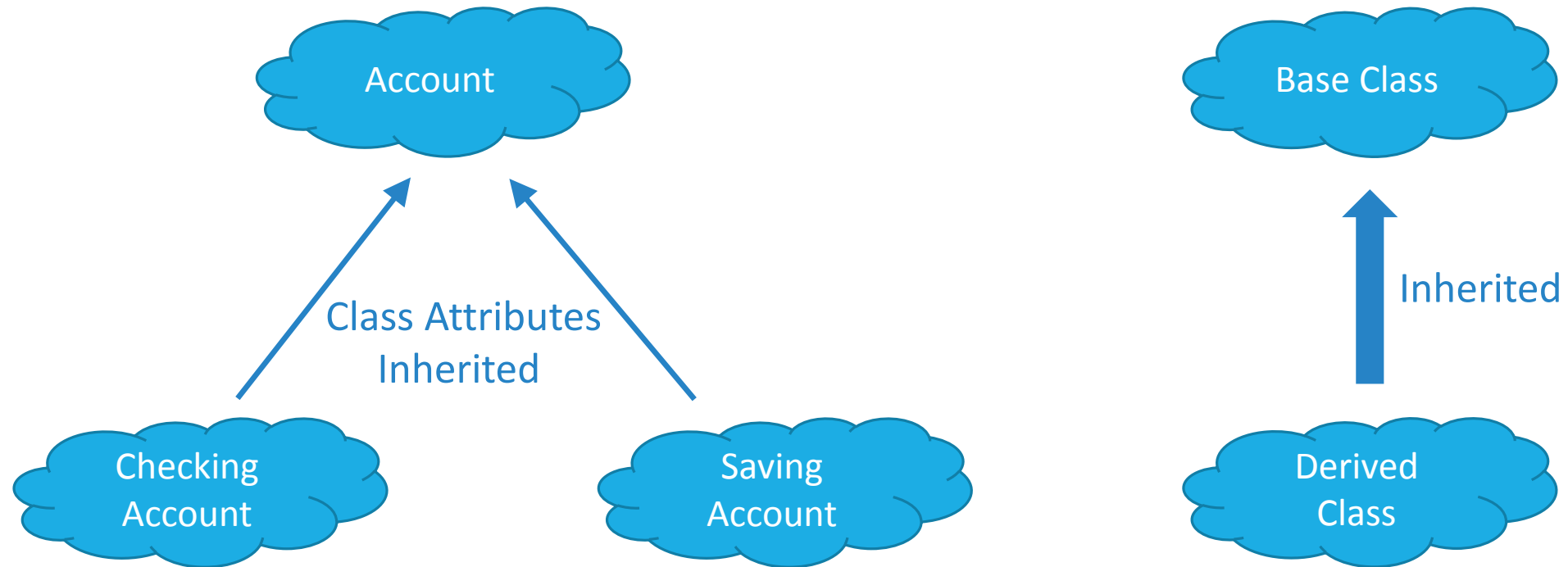
- Output

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

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

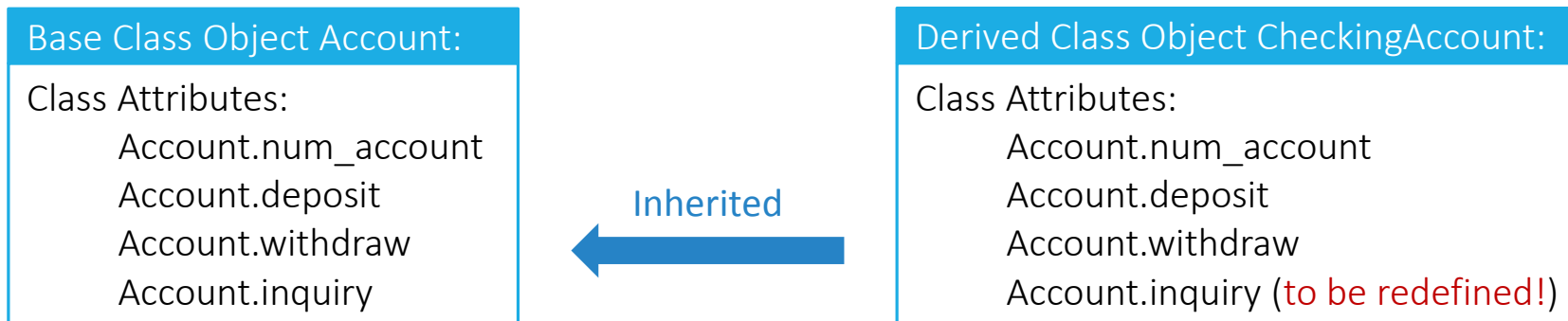
Inheritance with Class

- *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!



Inheritance with Class

- A simple example

```
class CheckingAccount(Account):                                # CheckingAccount is inherited from Account: class DerivedClass(BaseClass)

    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)                             # instance object ins has 2 instance attributes and 4 class attributes initialized!
ins.deposit(500)                                             # will call Account.deposit
ins.inquiry()                                                # call CheckingAccount.inquiry
```

- Output

```
Checking Account : 500 : Peter
```

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

Polymorphism and Class

- *What is Python Polymorphism?*
 - “Capability to get **correct behavior** of an instance **without knowing its type.**” ¹
 - “Code shouldn’t care about *what an object is*, only about ***what it does!***” ²
 - “Code may be written to work with any kind of object whatsoever as long as it has **a certain set of methods.**” ¹



"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"

1. *Python Essential Reference, 4th ed.*
2. *Learning Python, 5th ed.*

Polymorphism and Class

- A simple example

```
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