BIOS6642

Class Inheritance

Fuyong Xing

Department of Biostatistics and Informatics

Colorado School of Public Health

University of Colorado Anschutz Medical Campus

Outline

- **Class inheritance**
- Custom exceptions with class inheritance

Class inheritance

- Python supports class inheritance.
- It allows programmers to define a new class (i.e., subclass) that is a modified version of a previously defined class (i.e., superclass).
  - A subclass can inherit attributes and methods from its superclass.
  - A subclass can add new attributes and methods.
  - A subclass can override methods that are defined in its superclass.
- Class inheritance is useful when there is an existing class in a module or library, we just need to override a few things without having to reimplement everything that is already done.

Class inheritance

- Define a derived class with class inheritance

Name of the derived class (also called subclass or child class)

Name of the base class (also called superclass or parent class)

```
class DerivedClassName(BaseClassName):
    <statement-1>
    .
    .
    .
    <statement-N>
```

Class inheritance

- An simple example

Codes

```
class Animal:
    def __init__(self, age):
        self.age = age
        self.name = None
    def get_age(self):
        return self.age
    def get_name(self):
        return self.name
    def set_age(self, newage):
        self.age = newage
    def set_name(self, newname=""):
        self.name = newname
    def __str__(self):
        return "animal:"+str(self.name)+":"+str(self.age)
```

Codes

```
print("\n---- animal tests ----")
a = Animal(4)
print(a)
print(a.get_age())
a.set_name("fluffy")
print(a)
a.set_name()
print(a)
```

```
#####################################
## Inheritance example
#####################################
class Cat(Animal):
    def speak(self):
        print("meow")
    def __str__(self):
        return "cat:"+str(self.name)+":"+str(self.age)

print("\n---- cat tests ----")
```

Add a new method

override the *__str__* method

Codes

```
print("\n---- cat tests ----")
c = Cat(5)
c.set_name("fluffy")
print(c)
c.speak()
print(c.get_age())
#a.speak() # error because there is no speak method for Animal class
```

Output

```
---- animal tests ----
animal:None:4
4
animal:fluffy:4
animal::4

---- cat tests ----
cat:fluffy:5
meow
5
```


https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-0001-introduction-to-computer-science-and-programming-in-python-fall-2016/index.htm

Class inheritance

- Suppose we have a *Stopwatch* class defined in a *stopwatch.py* file.

```
from time import clock

class Stopwatch:
    """  Provides stopwatch objects that that programmers
         can use to time the execution time of portions of
         a program.  """
    def __init__(self):
        """  Makes a new stopwatch ready for timing.  """
        self.reset()

    def start(self):
        """  Starts the stopwatch, unless it is already running.
             This method does not affect any time that may have
             already accumulated on the stopwatch.  """
        if not self._running:
            self._start_time = clock() - self._elapsed
            self._running = True  # Clock now running
```

```
def stop(self):
    """  Stops the stopwatch, unless it is not running.
         Updates the accumulated elapsed time. """
    if self._running:
        self._elapsed = clock() - self._start_time
        self._running = False   # Clock stopped

def reset(self):
    """  Resets stopwatch to zero.  """
    self._start_time = self._elapsed = 0
    self._running = False

def elapsed(self):
    """  Reveals the stopwatch running time since it
         was last reset.  """
    if not self._running:
        return self._elapsed
    else:
        return clock() - self._start_time
```

Class inheritance

- We would like to define a subclass based on the *Stopwatch* class.
  - Need an object to record the number of times the watch has been started until it is reset.

```
from stopwatch import Stopwatch

class CountingStopwatch(Stopwatch):
    def __init__(self):
        # Allow base class to do its initialization of the
        # inherited instance variables
        super().__init__()
        # Set number of starts to zero
        self._count = 0

    def start(self):
        # Count this start message unless the watch already is running
        if not self._running:
            self._count += 1
        # Let base class do its start code
```

Invoke the *__init__* method in the superclass

Add a new instance variable

Override the Stopwatch's *start* method

```
        super().start()

    def reset(self):
        # Let base class reset the inherited instance variables
        super().reset()
        # Reset new instance variable that the base class method does not know about
        self._count = 0

    def count(self):
        return self._count
```

Override the Stopwatch's *reset* method

Add a new method

- The *CountingStopwatch* class does not explicitly define a *stop* method, but it inherits the *stop* method from the *Stopwatch* class.

Class inheritance

- Suppose the *CountingStopwatch* class is defined in the *countingstopwatch.py* file.
- Some sample code that uses the *CountingStopwatch* class is shown as follows.

Codes

```
from countingstopwatch import CountingStopwatch
from time import sleep

timer = CountingStopwatch()
timer.start()
sleep(10)  # Pause program for 10 seconds
timer.stop()
print("Time:", timer.elapsed(), "   Number:", timer.count())
```

Codes

```
timer.start()
sleep(5)   # Pause program for 5 seconds
timer.stop()
print("Time:", timer.elapsed(), "   Number:", timer.count())

timer.start()
sleep(20)  # Pause program for 20 seconds
timer.stop()
print("Time:", timer.elapsed(), "   Number:", timer.count())
```

Output

```
Time: 10.010378278632945    Number: 1
Time: 15.016618866378108    Number: 2
Time: 35.02881993198008    Number: 3
```

The output might be different if running on Unix or Mac machines.

Class inheritance

- We use the ***isinstance*** function to check whether an object is an instance of a type. It returns *True* if it is, otherwise *False*.

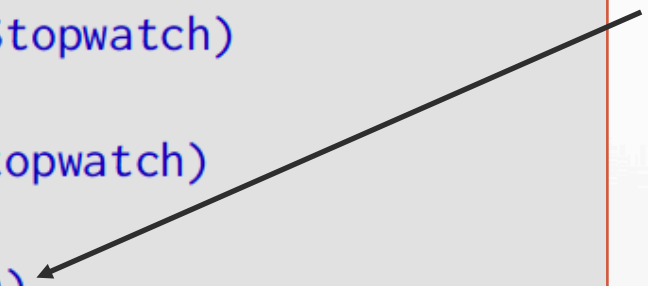
```
isinstance(object, classinfo)
```

- The ***issubclass*** is another built-in function that works with class inheritance. For example, *issubclass(float, int)* is *False* since *float* is not a subclass of *int*.

Class inheritance

- We use the *isinstance* function to check whether an object is an instance of a type. It returns *True* if it is, otherwise *False*.

```
>>> from stopwatch import Stopwatch
>>> from countingstopwatch import CountingStopwatch
>>> sw = Stopwatch()
>>> csw = CountingStopwatch()
>>> type(sw)
<class 'stopwatch.Stopwatch'>
>>> type(csw)
<class 'countingstopwatch.CountingStopwatch'>
>>> isinstance(sw, Stopwatch)
True
>>> isinstance(csw, CountingStopwatch)
True
>>> isinstance(sw, CountingStopwatch)
False
>>> isinstance(csw, Stopwatch)
True
```



An instance of a derived class is also an instance of the base class.

Class inheritance

- Suppose we want to use counting stopwatch objects that limit the number of times the clock can be stopped and restarted.

Codes

```
from countingstopwatch import CountingStopwatch

class RestrictedStopwatch(CountingStopwatch):
    def __init__(self, n):
        """ Restrict the number stopwatch starts to n times. """
        # Allow superclass to do its initialization of the
        # inherited instance variables
        super().__init__()
        self._limit = n

    def start(self):
        """ If the count exceeds the limit, terminate the program's
            execution. """
        if self._count < self._limit:
            super().start() # Let superclass do its start code
        else:
            import sys
            print("Limit exceeded")
            sys.exit(1)     #  Limit exceeded, terminate the program
```

Class inheritance

- Some sample code that uses the *RestrictedStopwatch* class

Codes

```
from restrictedstopwatch import RestrictedStopwatch

sw = RestrictedStopwatch(3)
print("Starting 1")
sw.start()
print("Stopping 1")
sw.stop()
print("Starting 2")
sw.start()
print("Stopping 2")
sw.stop()
print("Starting 3")
sw.start()
print("Stopping 3")
sw.stop()
```

Codes

```
print("Starting 4")
sw.start()
print("Stopping 4")
sw.stop()
print("Done")
```

Output

```
Starting 1
Stopping 1
Starting 2
Stopping 2
Starting 3
Stopping 3
Starting 4
Limit exceeded
```

Class inheritance

- *CountingStopwatch* is a direct base class of *RestrictedStopwatch* and *Stopwatch* is an indirect base class of *RestrictedStopwatch.*

```
>>> from stopwatch import Stopwatch
>>> from countingstopwatch import CountingStopwatch
>>> from restrictedstopwatch import RestrictedStopwatch
>>> rsw = RestrictedStopwatch(3)
>>> isinstance(rsw, RestrictedStopwatch)
True
>>> isinstance(rsw, CountingStopwatch)
True
>>> isinstance(rsw, Stopwatch)
True
```

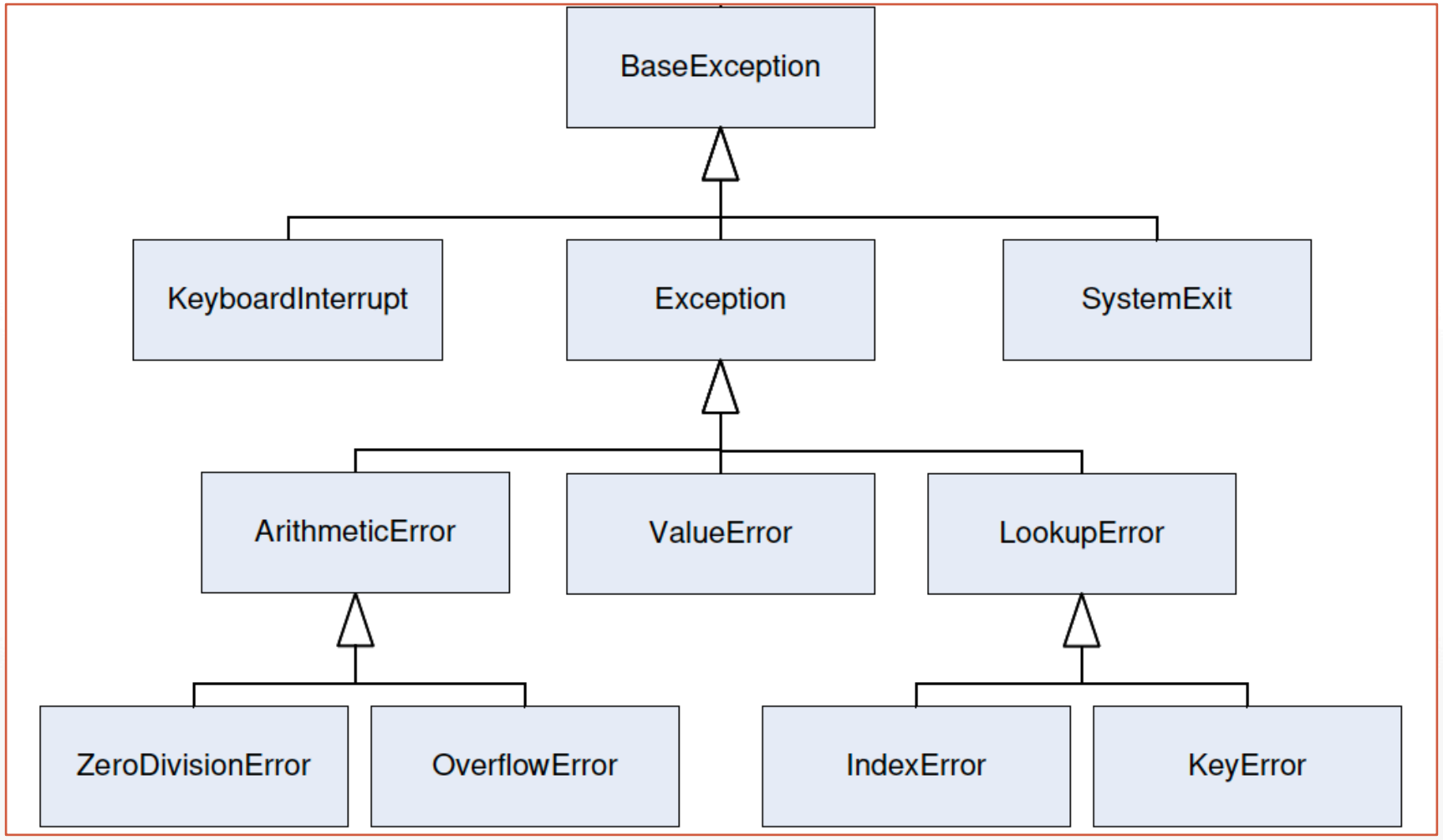
Outline

- Class inheritance
- **Custom exceptions with class inheritance**

Custom exceptions with class inheritance

- Python provides a set of built-in exception classes, and the *BaseException* is the base class for all built-in exceptions.

A small portion of built-in exceptions



Custom exceptions with class inheritance

- *ValueError* is a subclass of *Exception,* which is a subclass of *BaseException.*

```
>>> v = ValueError()
>>> v
ValueError()
>>> isinstance(v, ValueError)
True
>>> isinstance(v, TypeError)
False
>>> isinstance(v, Exception)
True
```

Custom exceptions with class inheritance

- Sometimes we may not find one built-in exception that exactly meets our needs, especially for our own custom types via class definition.
- Take the *Stopwatch* class as an example. Suppose we want to raise an exception if attempting to stop a nonrunning stopwatch. We could reuse a standard exception, e.g., *ValueError*.

```
class Stopwatch:
    # Other details about the class omitted ...
    def stop(self):
        """  Stops the stopwatch, unless it is not running.
             Updates the accumulated elapsed time. """
        if self._running:
            self._elapsed = clock() - self._start_time
            self._running = False   # Clock stopped
        else:
            raise ValueError("Attempt to stop a stopped clock")
```

Custom exceptions with class inheritance

- Is *ValueError* the best solution? Does it well match with an attempt to stop a stopped stopwatch?

```
class Stopwatch:
    # Other details about the class omitted ...
    def stop(self):
        """  Stops the stopwatch, unless it is not running.
             Updates the accumulated elapsed time. """
        if self._running:
            self._elapsed = clock() - self._start_time
            self._running = False   # Clock stopped
        else:
            raise ValueError("Attempt to stop a stopped clock")
```

Custom exceptions with class inheritance

- Is *ValueError* the best solution? Does it well match with an attempt to stop a stopped stopwatch?
- We can use the *help* function to get the information about *ValueError.*

Meaning of *ValueError*

```
>>> help(ValueError)
Help on class ValueError in module builtins:

class ValueError(Exception)
 |  Inappropriate argument value (of correct type).
 |
 |  Method resolution order:
 |      ValueError
 |      Exception
 |      BaseException
 |      object
 |
```

Custom exceptions with class inheritance

- *ValueError* does not well match with an attempt to stop a stopped stopwatch.
- We can define a new exception class, *StopwatchException.*

```
class StopwatchException(Exception):
    pass

class Stopwatch:
    # Other details about the class omitted ...
    def stop(self):
        """  Stops the stopwatch, unless it is not running.
             Updates the accumulated elapsed time. """
        if self._running:
            self._elapsed = clock() - self._start_time
            self._running = False   # Clock stopped
        else:
            raise StopwatchException()
```

Custom exceptions with class inheritance

- Some sample client code using the *StopwatchException* class.

```
try:
    # Some code that may raise a StopwatchException or ValueError
    # exception
except ValueError:
    pass # Add code to process ValueError
except StopwatchException:
    pass # Add code to process StopwatchException
except Exception:
    pass # Add code to process all other normal exceptions
```

Readings

- Think Python, chapter 18

References

- *Think Python: How to Think Like a Computer Scientist*, 2nd edition, 2015, by Allen Downey
- *Python Cookbook*, 3rd edition, by David Beazley and Brian K. Jones, 2013
- *Python for Data Analysis*, 2nd edition, by Wes McKinney, 2018
- *Fundamentals of Python Programming,* by Richard L. Halterman