Object-Oriented Programming in Python

Lecture-12

D.S. Hwang

Department of Software Science
Dankook University

Outline

Overview

Design a simple class

✓ Constructor

✓ Class variables

✓ Object reference

✓ Using variable references

Special methods

Exercise

Summary

Overview I

Basic concept

Class = Functions (Methods) + Data (Variables)

- A class packs together data (a collection of variables) and functions as one single unit
- As a programmer we can create a new class and thereby a new object type (like `float`, `list`, `file`, ...)
- A class is much like a module: a collection of ”global” variables and functions that belong together
- Each object has its own set of data, together with a set of methods that act upon the data.
- Modern programming applies classes to a large extent It will take some time to master the class concept

Overview II

✓

Structured programming:

- Breaking tasks into subtasks
- Writing re-usable methods to handle tasks

Classes and objects:

- To build larger and more complex programs
- To model objects we use in the world

Python classes I

A class describes a set of objects with the same behavior:

- The `str` class describes the behavior of all strings.
- This class specifies how a string stores its characters, which methods can be used with strings, and how the methods are implemented.
- When we have a `str` object, we can invoke the upper method:

$$''\underbrace{\text{Dankook}}_{\text{string object}}''.\qquad\overbrace{\text{upper()}}^{\text{a method of string class}}$$

Python classes II

The `list` class describes the behavior of objects that can be used to store a collection of values:

- The following call would be illegal.

```python
["Dankook", "university"].upper()
```

- The `list` has a `pop()` method, and the following call is legal.

```python
["Dankook", "university"].pop()
```

Python classes III

- The set of all methods provided by a class, together with a description of their behavior, is called the public interface of the class.
- When we work with an object of a class, we do not know how the object stores its data, or how the methods are implemented.
- All we need to know is the public interface, which methods we can apply, and what these methods do.
- The process of providing a public interface, while hiding the implementation details, is called encapsulation.
- Class inheritance makes code usability increase.

Simple class I

Tally counter

Design a class that models a mechanical device that is used to count people. For example, we use it to find out how many people attend a concert or board a bus.

How to use it:

```
tally = Counter() # instantiate an object of Counter class
tally.reset()
tally.click()
tally.click()
result = tally.getValue()  # Result is 2
tally.click()
result = tally.getValue() # Result is 3
```

Simple class II

Instance variables:

- An object stores its data in instance variables.
- An instance of a class is an object of the class.
- Each `Counter` object has a single instance variable named `_value`.
- For example, if `concertCounter` and `boardingCounter` are two objects of the `Counter` class, then each object has its own `_value` variable.
- Instance variables are part of the implementation details that should be hidden from the user of the class.

Simple class III

Class methods:

- The methods provided by the class are defined in the class body.
- The click() method advances the `_value` instance variable by 1.

```python
def click(self) :
    self._value = self._value + 1
```

- A method definition is very similar to a function with these exceptions.
  - A method is defined as part of a class definition.
  - The first parameter variable of a method is called `self`.

Simple class IV

How the `click()` method increments the instance variable `_value`.

- The call to `click()` advances the `_value` variable of the `concertCounter` object.
- No argument was provided when the `click()` method was called even though the definition includes the self parameter variable.
- The `self` parameter variable refers to the object on which the method was invoked concertCounter in this example.

Simple class V

The `getValue()` returns the current `_value` value.

```python
def getValue(self) :
    return self._value
```

- This method is provided so that users of the `Counter` class can find out how many times a particular counter has been clicked.
- A class user should not directly access any instance variables.
- Restricting access to instance variables is an essential part of encapsulation.

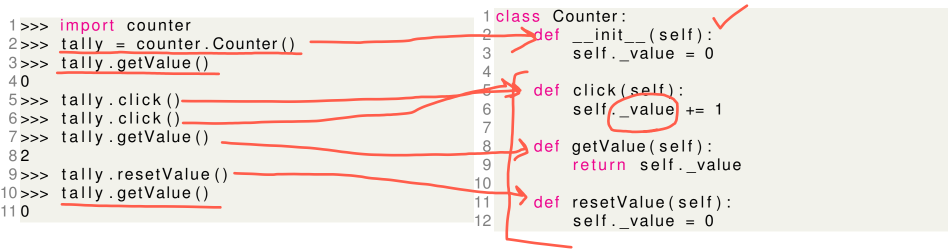
Simple class VI

The `resetValue()` assigns 0 to the `_value` variable.

```python
def resetValue(self) :
    self._value = 0
```

- This method provides a chance to use the preallocated object.

Simple class VII



```
>>> import counter
>>> tally = counter.Counter()
>>> tally.getValue()
0
>>> tally.click()
>>> tally.click()
>>> tally.getValue()
2
>>> tally.resetValue()
>>> tally.getValue()
0
```

counter.py

```
class Counter:
    def __init__(self):
        self._value = 0

    def click(self):
        self._value += 1

    def getValue(self):
        return self._value

    def resetValue(self):
        self._value = 0
```

Simulating cash registers I

A class design for cash register

Design a class to simulate cash registers. A cachier who rings up a sale presses a key to start the sale, then rings up each item. A display shows the amount owed as well as the total number of items purchased.

- Add the price of an item
- Get the total amount owed, and the count of items purchased
- Clear the cash register to start the new sale.

Simulating cash registers II

Outline of the aimed class:

```
class CashRegister:

  def addItem(self, price):

  def getTotal(self):

  def getCount(self):

  def clear(self):
```

Task	Method	Data
✓ Add the price of an item	`addItem()`	`items,total`
Get the total amount owed	`getTotal()`	✓ `total`
Get the count of items purchased	`getCount()`	✓ `items`
lear cash register for a new sale	`clear()`	`items,total`

Constructor

- A constructor is a method that initializes instance variables of an object.
- Python uses the special name `__init__` for the constructor because its purpose is to initialize an instance of the class.
- Only one constructor can be defined per class.
- The first parameter variable of every constructor must be `self`.

```python
def __init__(self):
    self._items = 0
    self._total = 0
```

Designing methods

Implementing the aimed method is very similar to implementing a function except that you access the instance variables of the object in the method body.

```python
def addItem(self, price):
    self._items += 1
    self._total += price

def getTotal(self):
    return self._total

def getCount(self):
    return self._items

def clear(self):
    self._items = 0
    self._total = 0
```

Accessing instance variables I

- Access the variable name through the `self` reference to access an instance variable, such as `_items` or `_total`.
- When one method needs to call another method on the same object, invoke the method on the `self` parameter.

```python
def addItems(self, quantity, price):
    for i in range(quantity):
        self.addItem(price)
```

Accessing instance variables II

```
>>> from cashregister import CashRegister
>>> cr = CashRegister()
>>> dir(cr)
['__doc__', '__init__', '__module__', '_items', '_total', 'addItem', 'addItems', '
    clear', 'getCount', 'getTotal']
>>> cr.getCount()
0
>>> cr.getTotal()
0
>>> cr.addItems(10, 30)
>>> cr.getTotal()
300
>>> cr.getCount()
10
```

cashregister.py

```
class CashRegister:
    def __init__(self):
        self._items = 0
        self._total = 0

    def addItem(self, price):
        self._items += 1
        self._total += price

    def addItems(self, quantity, price):
        for i in range(quantity):
            self.addItem(price)

    def getTotal(self):
        return self._total

```

Accessing instance variables III

```python
    def getCount(self):
        return self._items

    def clear(self):
        self._items = 0
        self._total = 0
```

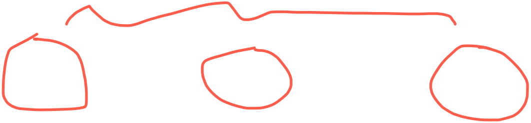
Using class global variables I

- Class variables to a class, not to any object of the class.
- Class variables are often called ”static variables”.
- Class variables are declared at the same level as methods.
- Be cautious that instance variables are created in the constructor.
- We have a constraint that each item

Using class global variables II

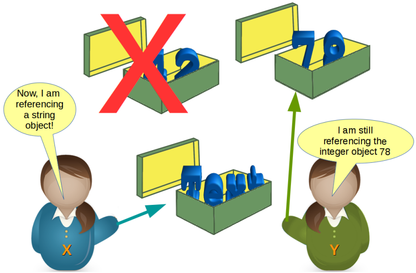
- assign bank account numbers sequentially: the first account is assigned number 1001, the next with number 1002, and so on.
- To solve this problem, we need to have a single value of `_lastAssigned` that is a property of the class, not any object of the class.

```python
class BankAccount:
    _lastAssigned = 1000 # class variable
    def __init__(self):
        self._balance = 0
        BankAccount._lastAssigned = BankAccount._lastAssigned + 1
        self._accountNumber = BankAccount._lastAssigned
```



Using class objects I

- In Python, a variable does not actually hold an object.
- It merely holds the memory location of an object.
- The constructor returns a reference to the new object.
- Multiple object variables may contain references to the same object ('aliases').
- The object itself is stored in another location:


12
78
Now, I am referencing a string object!
I am still referencing the integer object 78
X
Y

Using class objects II

- To check if references are aliases, use the `is` or the `is not` operator:

```python
if var1 is var2:
    print('The variables are aliased!')
if var1 is not var2:
    print('The variables refer to different objects!')
```

- To check if the data contained within objects are equal, use the == operator.

```python
if var1 == var2:
    print("The objects contain the same data.")
```

None reference

- A reference may point to 'no' object.

```python
reg = None
print(reg.getTotal()) # raise RunTime error
```

- To test if a reference is None before using it.

```python
cr = CashRegister()
if cr is None:
    print('Empty instance!')
else:
    print(cr.getTotal())
```

`self` reference

- Every method has a reference to the object on which the method was invoked, stored in the `self` parameter variable.

```python
def addItem(self, price):
    self._items += 1
    self._total += price
```

- It is a reference to the object the method was invoked on.
- It can clarify when instance variables are used:

```python
def addTotal(self, price):
    self._items += 1
    self._total += price
    temp_sum = price
```

Special methods I

- Some special methods are called when an instance of the class is passed to a built-in function.
- For example, suppose you attempt to convert a `Fraction` object to a floating point number using the `int()` function.

```
>>> from cashregister import *
>>> cr = CashRegister()
>>> cr.addItems(100, 3.9)
>>> int(cr)
389
```

int('123')

- Here is a definition of that method.

```
    def __int__(self):
        return int(self._total)
```

Special methods II

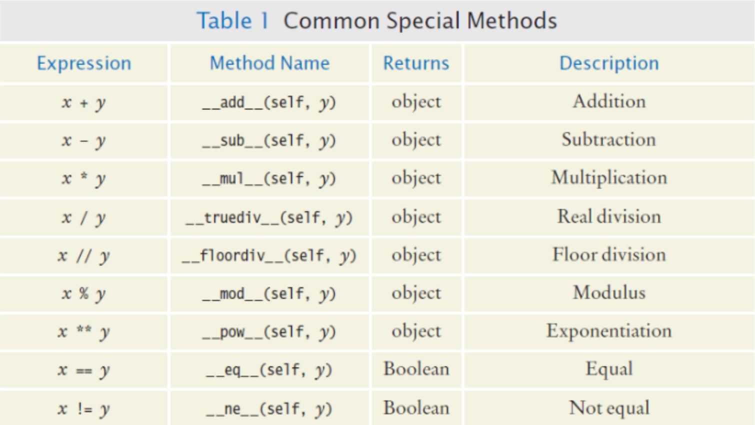


Table 1 Common Special Methods

Expression	Method Name	Returns	Description
x + y	`__add__(self, y)`	object	Addition
x - y	`__sub__(self, y)`	object	Subtraction
x * y	`__mul__(self, y)`	object	Multiplication
x / y	`__truediv__(self, y)`	object	Real division
x // y	`__floordiv__(self, y)`	object	Floor division
x % y	`__mod__(self, y)`	object	Modulus
x ** y	`__pow__(self, y)`	object	Exponentiation
x == y	`__eq__(self, y)`	Boolean	Equal
x != y	`__ne__(self, y)`	Boolean	Not equal

Special methods III

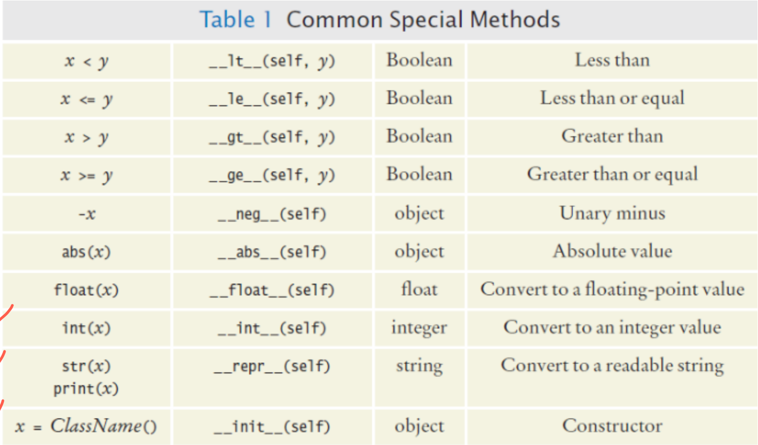


Table 1 Common Special Methods

x < *y*	`__lt__(self, y)`	Boolean	Less than
x <= *y*	`__le__(self, y)`	Boolean	Less than or equal
x > *y*	`__gt__(self, y)`	Boolean	Greater than
x >= *y*	`__ge__(self, y)`	Boolean	Greater than or equal
-*x*	`__neg__(self)`	object	Unary minus
abs(*x*)	`__abs__(self)`	object	Absolute value
float(*x*)	`__float__(self)`	float	Convert to a floating-point value
✓ int(*x*)	`__int__(self)`	integer	Convert to an integer value
✓ str(*x*) print(*x*)	`__repr__(self)`	string	Convert to a readable string
✓ *x* = *ClassName*()	`__init__(self)`	object	Constructor

Exercise I

CreditCard Class

Design a CreditCard class

Implement a CreditCard class based on the design we learnt. The instances defined by the CreditCard class provide a simple model for traditional credit cards. They have identifying information about the customer, bank, account number, credit limit, and current balance. The class restricts charges that would cause a card's balance to go over its spending limit, but it does not charge interest or late payments.

Exercise II

`CreditCard` Class

pep??

```python
class CreditCard:
  """A consumer credit card."""

  def __init__(self, customer, bank, acnt, limit):
    """Create a new credit card instance.

    The initial balance is zero.

    customer  the name of the customer (e.g., 'John Bowman')
    bank      the name of the bank (e.g., 'California Savings')
    acnt      the acount identifier (e.g., '5391 0375 9387 5309')
    limit     credit limit (measured in dollars)
    """
    self._customer = customer
    self._bank = bank
    self._account = acnt
    self._limit = limit
    self._balance = 0

  def get_customer(self):
    """Return name of the customer."""
    return self._customer

  def get_bank(self):
    """Return the bank's name."""
    return self._bank

  def get_account(self):
    """Return the card identifying number (typically stored as a string)."""
    return self._account
```

Exercise III

CreditCard Class

```python

  def get_limit(self):
    """Return current credit limit."""
    return self._limit

  def get_balance(self):
    """Return current balance."""
    return self._balance

  def charge(self, price):
    """Charge given price to the card, assuming sufficient credit limit.

    Return True if charge was processed; False if charge was denied.
    """
    if price + self._balance > self._limit:  # if charge would exceed limit,
      return False                           # cannot accept charge
    else:
      self._balance += price
      return True

  def make_payment(self, amount):
    """Process customer payment that reduces balance."""
    self._balance -= amount

if __name__ == '__main__':
  wallet = []
  wallet.append(CreditCard('John Bowman', 'California Savings',
                           '5391 0375 9387 5309', 2500) )
  wallet.append(CreditCard('John Bowman', 'California Federal',
                           '3485 0399 3395 1954', 3500) )
```

Exercise IV

CreditCard Class

```python
wallet.append(CreditCard('John Bowman', 'California Finance',
                        '5391 0375 9387 5309', 5000) )

for val in range(1, 17):
  wallet[0].charge(val)
  wallet[1].charge(2*val)
  wallet[2].charge(3*val)

for c in range(3):
  print('Customer =', wallet[c].get_customer())
  print('Bank =', wallet[c].get_bank())
  print('Account =', wallet[c].get_account())
  print('Limit =', wallet[c].get_limit())
  print('Balance =', wallet[c].get_balance())
  while wallet[c].get_balance() > 100:
    wallet[c].make_payment(100)
    print('New balance =', wallet[c].get_balance())
  print()
```

Exercise I

Multidimensional `Vector` Class

Multidimensional `Vector` Class

Design a multidimensional `Vector` class by using operator overloading via special methods. A `Vector` class represents the coordinates of a vector in a multidimensional space.

For example, in a 3-d space, we might wish to represent a vector with coordinates $< 5, -2, 3 >$. When working with vectors, if $u =< 5, -2, 3 >$ and $v =< 1, 4, 2 >$, one would expect the expression, $u + v$, return a 3-d vector with coordinates $< 6, 2, 5 >$.

Exercise II

Multidimensional `Vector` Class

```python
import collections

class Vector:
  """Represent a vector in a multidimensional space."""

  def __init__(self, d):
    if isinstance(d, int):
      self._coords = [0] * d
    else:
      try:                                    # we test if param is iterable
        self._coords = [val for val in d]
      except TypeError:
        raise TypeError('invalid parameter type')

  def __len__(self):
    """Return the dimension of the vector."""
    return len(self._coords)

  def __getitem__(self, j):
    """Return jth coordinate of vector."""
    return self._coords[j]

  def __setitem__(self, j, val):
    """Set jth coordinate of vector to given value."""
    self._coords[j] = val

  def __add__(self, other):
    """Return sum of two vectors."""
    if len(self) != len(other):         # relies on __len__ method
      raise ValueError('dimensions must agree')
```

u = Vector(5)
len(u) ==> 5

v[j]=val

v = v + u

Exercise III

Multidimensional `Vector` Class

```python
    result = Vector(len(self))            # start with vector of zeros
    for j in range(len(self)):
      result[j] = self[j] + other[j]
    return result

  def __eq__(self, other):
    """Return True if vector has same coordinates as other."""
    return self._coords == other._coords

  def __ne__(self, other):
    """Return True if vector differs from other."""
    return not self == other              # rely on existing __eq__ definition

  def __str__(self):
    """Produce string representation of vector."""
    return '<' + str(self._coords)[1:-1] + '>'  # adapt list representation

  def __neg__(self):
    """Return copy of vector with all coordinates negated."""
    result = Vector(len(self))            # start with vector of zeros
    for j in range(len(self)):
      result[j] = -self[j]
    return result

  def __lt__(self, other):
    """Compare vectors based on lexicographical order."""
    if len(self) != len(other):
      raise ValueError('dimensions must agree')
    return self._coords < other._coords

```

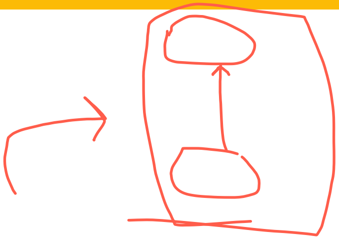
Exercise IV

Multidimensional `Vector` Class

```python
  def __le__(self, other):
    """Compare vectors based on lexicographical order."""
    if len(self) != len(other):
      raise ValueError('dimensions must agree')
    return self._coords <= other._coords

if __name__ == '__main__':
  # the following demonstrates usage of a few methods
  v = Vector(5)              # construct five-dimensional <0, 0, 0, 0, 0>
  v[1] = 23                  # <0, 23, 0, 0, 0> (based on use of __setitem__)
  v[-1] = 45                 # <0, 23, 0, 0, 45> (also via __setitem__)
  print(v[4])                # print 45 (via __getitem__)
  u = v + v                  # <0, 46, 0, 0, 90> (via __add__)
  print(u)                   # print <0, 46, 0, 0, 90>
  total = 0
  for entry in v:            # implicit iteration via __len__ and __getitem__
    total += entry
```

Summary



- ✓ Every thing in Python is a class.
- ✓ Using classes increases the reusability in programming.
- Glance at how to design and code classes in Python.
- Practice designing classes.

Problem 1

Homework

Design and implement a class `Country` that stores the information on countries such as nation name, capital city, population, and area. Then write a program that reads in a set of countries and prints

1. the country with the largest area.
2. the country with the largest population.
3. the country with the largest population density.
4. the country with its capital city.

Problem 2 I

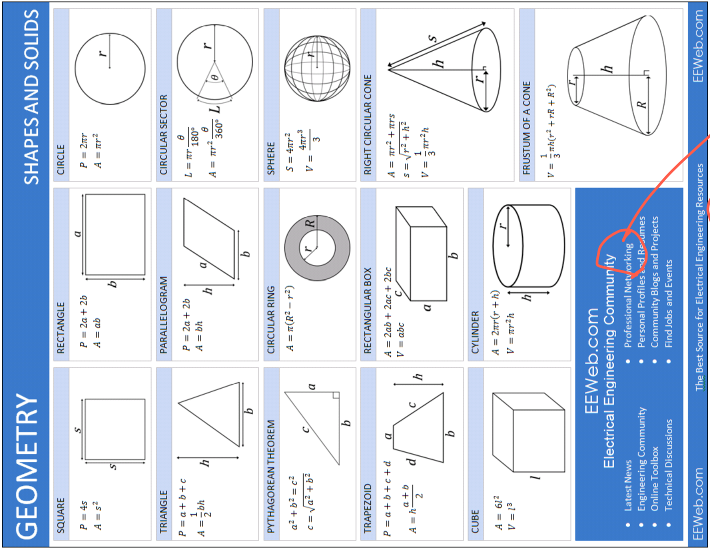
Homework

Based on object-oriented programming, design and implement each class for geometry objects on the next page.

1. Implement and test the class on each object
2. ✓ Place those classes into a `geometry` module. Then write a program that prints a result for the chosen object depending on a user's values.

Problem 2 II

Homework



GEOMETRY

SHAPES AND SOLIDS

SQUARE
$P = 4s$
$A = s^2$

RECTANGLE
$P = 2a + 2b$
$A = ab$

CIRCLE
$P = 2\pi r$
$A = \pi r^2$

TRIANGLE
$P = a + b + c$
$A = \frac{1}{2}bh$

PARALLELOGRAM
$P = 2a + 2b$
$A = bh$

CIRCULAR SECTOR
$L = \pi r \frac{\theta}{180°}$
$A = \pi r^2 \frac{\theta}{360°}$

PYTHAGOREAN THEOREM
$a^2 + b^2 = c^2$
$c = \sqrt{a^2 + b^2}$

CIRCULAR RING
$A = \pi(R^2 - r^2)$

SPHERE
$S = 4\pi r^2$
$V = \frac{4\pi r^3}{3}$

TRAPEZOID
$P = a + b + c + d$
$A = h\frac{a + b}{2}$

RECTANGULAR BOX
$A = 2ab + 2ac + 2bc$
$V = abc$

RIGHT CIRCULAR CONE
$A = \pi r^2 + \pi r s$
$s = \sqrt{r^2 + h^2}$
$V = \frac{1}{3}\pi r^2 h$

CUBE
$A = 6l^2$
$V = l^3$

CYLINDER
$A = 2\pi r(r + h)$
$V = \pi r^2 h$

FRUSTUM OF A CONE
$V = \frac{1}{3}\pi h(r^2 + rR + R^2)$

EEWeb.com
Electrical Engineering Community

- Latest News
- Engineering Community
- Online Toolbox
- Technical Discussions
- Professional Networking
- Personal Profiles and Resumes
- Community Blogs and Projects
- Find Jobs and Events

EEWeb.com The Best Source for Electrical Engineering Resources EEWeb.com

P

Problem 3

Homework

Design a class `Msg` that models an e-mail message. A message has a recipient, a sender, and a message text. Support the following methods:

- A constructor that takes the sender and recipient
- A method `append` that appends a line of text to the message body
- A method `__str__` that returns the whole string like this:

```
        From G. D. Hong
        To: G. I. Dong
        Content: Dear friend, I would like to ....
```