Python 2

Jin Hyun Kim

Fall 2019

Python-Specific Data Structure

List

```
# This is my shopping list
shoplist = ['apple', 'mango', 'carrot', 'banana']

print 'I have', len(shoplist), 'items to purchase.'

print 'These items are:',
for item in shoplist:
    print item,

print '\nI also have to buy rice.'
shoplist.append('rice')
print 'My shopping list is now', shoplist

print 'I will sort my list now'
shoplist.sort()
print 'Sorted shopping list is', shoplist

print 'The first item I will buy is', shoplist[0]
olditem = shoplist[0]
del shoplist[0]
print 'I bought the', olditem
print 'My shopping list is now', shoplist
```

- An array **CLASS (Methods, Data)**

Tuple

```
zoo = ('python', 'elephant', 'penguin')
print 'Number of animals in the zoo is', len(zoo)

new_zoo = 'monkey', 'camel', zoo
print 'Number of cages in the new zoo is', len(new_zoo)
print 'All animals in new zoo are', new_zoo
print 'Animals brought from old zoo are', new_zoo[2]
print 'Last animal brought from old zoo is', new_zoo[2][2]
print 'Number of animals in the new zoo is', \
    len(new_zoo)-1+len(new_zoo[2])
```

- A constant array
  - ***Elements cannot be updated***

Dictionary

```
# 'ab' is short for 'a'ddress'b'ook

ab = {  'Swaroop'   : 'swaroop@swaroopch.com',
        'Larry'     : 'larry@wall.org',
        'Matsumoto' : 'matz@ruby-lang.org',
        'Spammer'   : 'spammer@hotmail.com'
    }

print "Swaroop's address is", ab['Swaroop']

# Deleting a key-value pair
del ab['Spammer']

print '\nThere are {} contacts in the address-book\n'.format(len(ab))

for name, address in ab.items():
    print 'Contact {} at {}'.format(name, address)

# Adding a key-value pair
ab['Guido'] = 'guido@python.org'

if 'Guido' in ab:
    print "\nGuido's address is", ab['Guido']
```

- A set of pairs of key and data

```
d = {key1 : value1, key2 : value2 }
```

Sequence

```
shoplist = ['apple', 'mango', 'carrot', 'banana']
name = 'swaroop'

# Indexing or 'Subscription' operation #
print 'Item 0 is', shoplist[0]
print 'Item 1 is', shoplist[1]
print 'Item 2 is', shoplist[2]
print 'Item 3 is', shoplist[3]
print 'Item -1 is', shoplist[-1]
print 'Item -2 is', shoplist[-2]
print 'Character 0 is', name[0]

# Slicing on a list #
print 'Item 1 to 3 is', shoplist[1:3]
print 'Item 2 to end is', shoplist[2:]
print 'Item 1 to -1 is', shoplist[1:-1]
print 'Item start to end is', shoplist[:]

# Slicing on a string #
print 'characters 1 to 3 is', name[1:3]
print 'characters 2 to end is', name[2:]
print 'characters 1 to -1 is', name[1:-1]
print 'characters start to end is', name[:]
```

- It is for testing membership (in, not in) and indexing operations

Sequence

```
>>> shoplist = ['apple', 'mango', 'carrot', 'banana']
>>> shoplist[::1]
['apple', 'mango', 'carrot', 'banana']
>>> shoplist[::2]
['apple', 'carrot']
>>> shoplist[::3]
['apple', 'banana']
>>> shoplist[::-1]
['banana', 'carrot', 'mango', 'apple']
```

- Step for slicing

Set

```
>>> bri = set(['brazil', 'russia', 'india'])
>>> 'india' in bri
True
>>> 'usa' in bri
False
>>> bric = bri.copy()
>>> bric.add('china')
>>> bric.issuperset(bri)
True
>>> bri.remove('russia')
>>> bri & bric # OR bri.intersection(bric)
{'brazil', 'india'}
```

- Unordered collections of simple objects

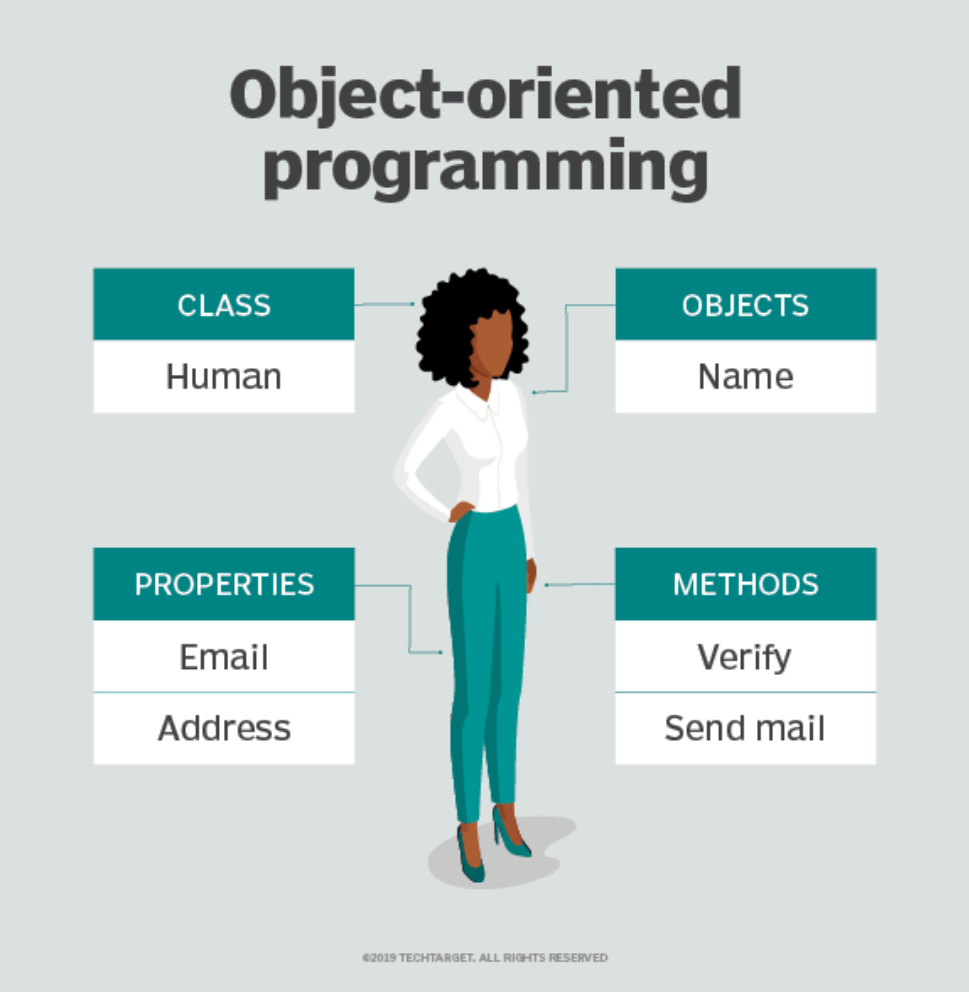
Reference

```
print 'Simple Assignment'
shoplist = ['apple', 'mango', 'carrot', 'banana']
# mylist is just another name pointing to the same object!
mylist = shoplist
# I purchased the first item, so I remove it from the list
del shoplist[0]

print 'shoplist is', shoplist
print 'mylist is', mylist
# Notice that both shoplist and mylist both print
# the same list without the 'apple' confirming that
# they point to the same object

print 'Copy by making a full slice'
# Make a copy by doing a full slice
mylist = shoplist[:]
# Remove first item
del mylist[0]

print 'shoplist is', shoplist
print 'mylist is', mylist
# Notice that now the two lists are different
```

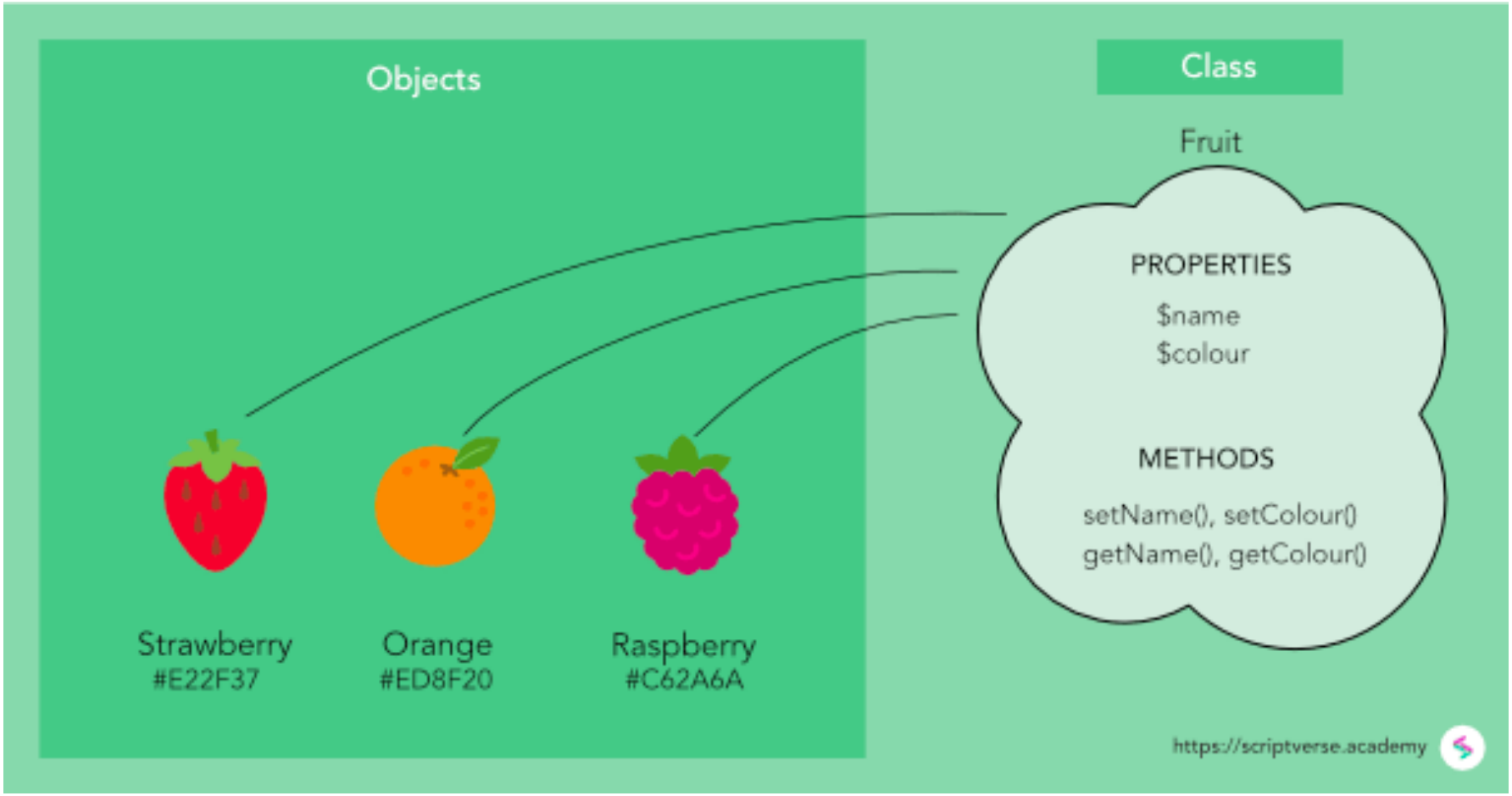
- For collections, “=“ does not copy the source to the destination
  - Notice how to make a copy by using slice

Object-Oriented Programming

OOP


Object-oriented programming
CLASS
Human
OBJECTS
Name
PROPERTIES
Email
Address
METHODS
Verify
Send mail
©2019 TECHTARGET. ALL RIGHTS RESERVED

Class, Object



Python Class

```
class Employee:
    def __init__(self,name):
        self.name = name
    def display (self):
        print('The name of employee is:',self.name)

first = Employee('Rushabh')
second = Employee('Dhaval')

second.display()
first.display()
```

self

- It is a reference to an object of classes

```
class Employee:
    def __init__(self,name):
        self.name = name
    def display (self):
        print('The name of employee is:',self.name)

first = Employee('Rushabh')
second = Employee('Dhaval')

second.display()
first.display()
```

Class and Method

```
class Person:
    def say_hi(self):
        print('Hello, how are you?')

p = Person()
p.say_hi()
# The previous 2 lines can also be written as
# Person().say_hi()
```

- Person
  - say_hi()
- Notice how to instantiate and call an method of an object

init method

```
class Person:
    def __init__(self, name):
        self.name = name
    def say_hi(self):
        print 'Hello, my name is', self.name

p = Person('Swaroop')
p.say_hi()
# The previous 2 lines can also be written as
# Person('Swaroop').say_hi()
```

- init() is automatically called when an object is created from a class

Class Method

```
class Robot:
    """Represents a robot, with a name."""

    # A class variable, counting the number of robots
    population = 0

    def __init__(self, name):
        """Initializes the data."""
        self.name = name
        print "(Initializing {})".format(self.name)

        # When this person is created, the robot
        # adds to the population
        Robot.population += 1

    def die(self):
        """I am dying."""
        print "{} is being destroyed!".format(self.name)

        Robot.population -= 1

        if Robot.population == 0:
            print "{} was the last one.".format(self.name)

        else:
            print "There are still {:d} robots working.".format(
                Robot.population)

    def say_hi(self):
        """Greeting by the robot.

        Yeah, they can do that."""
        print "Greetings, my masters call me {}.".format(self.name)

    @classmethod
    def how_many(cls):
        """Prints the current population."""
        print "We have {:d} robots.".format(cls.population)


droid1 = Robot("R2-D2")
droid1.say_hi()
Robot.how_many()

droid2 = Robot("C-3PO")
droid2.say_hi()
Robot.how_many()

print "\nRobots can do some work here.\n"

print "Robots have finished their work. So let's destroy them."
droid1.die()
droid2.die()

Robot.how_many()
```

Static Method

```
class 클래스이름:
    @staticmethod
    def 메서드(매개변수1, 매개변수2):
        코드
```

class_static_method.py

```
class Calc:
    @staticmethod
    def add(a, b):
        print(a + b)

    @staticmethod
    def mul(a, b):
        print(a * b)

Calc.add(10, 20)    # 클래스에서 바로 메서드 호출
Calc.mul(10, 20)    # 클래스에서 바로 메서드 호출
```

실행 결과

```
30
200
```

- Methods by class ID

Class method vs Static Method

```
class 클래스이름:
    @classmethod
    def 메서드(cls, 매개변수1, 매개변수2):
        코드
```

- A class method is used to access properties and methods within a class

class_class_method.py

```
class Person:
    count = 0    # 클래스 속성

    def __init__(self):
        Person.count += 1    # 인스턴스가 만들어질 때
                             # 클래스 속성 count에 1을 더함

    @classmethod
    def print_count(cls):
        print('{0}명 생성되었습니다.'.format(cls.count))    # cls로 클래스 속성에 접근

james = Person()
maria = Person()

Person.print_count()    # 2명 생성되었습니다.
```

Inheritance

```
class SchoolMember:
    '''Represents any school member.'''
    def __init__(self, name, age):
        self.name = name
        self.age = age
        print '(Initialized SchoolMember: {})'.format(self.name)

    def tell(self):
        '''Tell my details.'''
        print 'Name:"{}" Age:"{}"'.format(self.name, self.age),

class Teacher(SchoolMember):
    '''Represents a teacher.'''
    def __init__(self, name, age, salary):
        SchoolMember.__init__(self, name, age)
        self.salary = salary
        print '(Initialized Teacher: {})'.format(self.name)

    def tell(self):
        SchoolMember.tell(self)
        print 'Salary: "{:d}"'.format(self.salary)
```

```
class Student(SchoolMember):
    '''Represents a student.'''
    def __init__(self, name, age, marks):
        SchoolMember.__init__(self, name, age)
        self.marks = marks
        print '(Initialized Student: {})'.format(self.name)

    def tell(self):
        SchoolMember.tell(self)
        print 'Marks: "{:d}"'.format(self.marks)
```

Exercise

```
>>> cal = Calculator()
>>> print("10 + 20 = %s" %cal.Add(10,20))
10 + 20 = 30
>>> print("10 - 20 = %s" %cal.Min(10,20))
10 - 20 = -10
>>> print("10 * 20 = %s" %cal.Mul(10,20))
10 * 20 = 200
>>> print("10 * 10 = %s" %cal.Mul(10,10))
10 * 10 = 100
>>> cal.ShowCount()
덧셈 : 1
뺄셈 : 1
곱셈 : 2
나눗셈 : 0
```

Exception Handler

Exception

```
>>> s = raw_input('Enter something --> ')
Enter something --> Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
EOFError
```

Exception Handler

```
try:
    text = raw_input('Enter something --> ')
except EOFError:
    print 'Why did you do an EOF on me?'
except KeyboardInterrupt:
    print 'You cancelled the operation.'
else:
    print 'You entered {}'.format(text)
```

```
# Press ctrl + d
$ python exceptions_handle.py
Enter something --> Why did you do an EOF on me?

# Press ctrl + c
$ python exceptions_handle.py
Enter something --> ^CYou cancelled the operation.

$ python exceptions_handle.py
Enter something --> No exceptions
You entered No exceptions
```

Raise Exception

```
class ShortInputException(Exception):
    '''A user-defined exception class.'''
    def __init__(self, length, atleast):
        Exception.__init__(self)
        self.length = length
        self.atleast = atleast

try:
    text = raw_input('Enter something --> ')
    if len(text) < 3:
        raise ShortInputException(len(text), 3)
    # Other work can continue as usual here
except EOFError:
    print 'Why did you do an EOF on me?'
except ShortInputException as ex:
    print ('ShortInputException: The input was ' + \
           '{0} long, expected at least {1}')\
          .format(ex.length, ex.atleast)
else:
    print 'No exception was raised.'
```

finally

```
import sys
import time

f = None
try:
    f = open("poem.txt")
    # Our usual file-reading idiom
    while True:
        line = f.readline()
        if len(line) == 0:
            break
        print line,
        sys.stdout.flush()
        print "Press ctrl+c now"
        # To make sure it runs for a while
        time.sleep(2)
except IOError:
    print "Could not find file poem.txt"
except KeyboardInterrupt:
    print "!! You cancelled the reading from the file."
finally:
    if f:
        f.close()
    print "(Cleaning up: Closed the file)"
```

- While reading a file, an exception is raised.
  - Then, close() should be perform with exception handler