Python 2

Jin Hyun Kim

Fall 2019

This Class

- “A Byte of Python”- Swaroop C H

In this class

- Why “python” in this course?
- Install compiler and IDE (PyCham)
- Lab 00 - “Hello World”

Why

- Simple
- Easy to Learn
- Free and Open Source
- High-level Language
- Portable (이식성)
- Object Oriented Language
- Extensible
- Embeddable
- Extensive Libraries

Lab 1

- print (Hello World)
  - Install python
  - Install “PyCham”
  - Create and configure the project
  - Run “python hello.py”

Basic

Basic

- Comments

```
print 'hello world' # Note that print is a statement
```

or:

```
# Note that print is a statement
print 'hello world'
```

Basic

- Constants
  - Numbers - **5**, **1.23**, **52.3E-4**
  - String - “This is a string”, ‘It’s a string’
  - Multiple string - ‘ ‘ ‘ … ’ ’ ’, “ “ “ … ” ” ”

Format

```
age = 20
name = 'Swaroop'

print '{0} was {1} years old when he wrote this book'.format(name, age)
print 'Why is {0} playing with that python?'.format(name)
```

- A number in “{}” is optional

```
name + ' is ' + str(age) + ' years old'
```

Escape Sequences

- What if a string includes “‘”? `'What's your name?'`

'What\'s your name?'

- ‘\’ -> ‘\\’
- “\n” (newline)

```
'This is the first line\nThis is the second line'
```

- Ignore escape sequence

```
r"Newlines are indicated by \n"
```

Variable Identifiers

- The first character of the identifier must be a letter of the alphabet (uppercase ASCII or lowercase ASCII or Unicode character) or an underscore (_).

- The rest of the identifier name can consist of letters (uppercase ASCII or lowercase ASCII or Unicode character), underscores (_) or digits (0-9).

- Identifier names are case-sensitive. For example, ***myname*** and ***myName*** are not the same. Note the lowercase n in the former and the uppercase in the latter.

Basic

- Variables
- Objects

Lines

Logical vs Physical Lines

Lines

- A line can be broken by "\" in code

```
s = 'This is a string. \
This continues the string.'
print s
```

Output:

```
This is a string. This continues the string.
```

Indentation

- Whitespace at the beginning of the line, called **indentation**, is important

```
i = 5
# Error below! Notice a single space at the start of the line
 print 'Value is ', i
print 'I repeat, the value is ', i
```

- Statements should go together and have the same indentation

```
if True:
    print 'Yes, it is true'
```

Operators

- +, -, x, /
- ** (power): ex) 3 ** 4 = 3 * 3 * 3 * 3
- % (modulo): ex)3 % 2 = 1
- << (bit, shift left): ex) 2 << 2 = 8
- >> (bit, shift right): ex) 11 >> 1 = 5
- & (bit, And): ex) 5 & 3 = 1 (0101 & 0011 = 0001)
- | (bit, Or): ex) 5 | 3 = 7 (0101 & 0011 = 0111)

Operators

- ^ (bit, Xor): ex) 5 ^ 3 = 6 (101 ^ 011 = 110)
- ~ (bit, flip, Return -(x+1)) ex) ~5 = -6
- > (Logical “greater than”)
- < (Logical “less than”)
- <= (Logical “greater than or equal to”)
- >= (Logical “less than or equal to”)
- == (Logical “equal to”)

Operators

- != (Logical “not equal to”)
- not (Logical negation): ex) x = True ; not x
- and (Boolean logical “AND”)
- or (Boolean logical “OR”)

Assignments

- Shortened expression

```
a = 2
a *= 3
```

- Precedence (연산 우선 순위)
  - 2 + 3 * 4

Control Flow

If

```
number = 23
guess = int(raw_input('Enter an integer : '))

if guess == number:
    # New block starts here
    print 'Congratulations, you guessed it.'
    print '(but you do not win any prizes!)'
    # New block ends here
elif guess < number:
    # Another block
    print 'No, it is a little higher than that'
    # You can do whatever you want in a block ...
else:
    print 'No, it is a little lower than that'
    # you must have guessed > number to reach here

print 'Done'
# This last statement is always executed,
# after the if statement is executed.
```

While

```
number = 23
running = True

while running:
    guess = int(raw_input('Enter an integer : '))

    if guess == number:
        print 'Congratulations, you guessed it.'
        # this causes the while loop to stop
        running = False
    elif guess < number:
        print 'No, it is a little higher than that.'
    else:
        print 'No, it is a little lower than that.'
else:
    print 'The while loop is over.'
    # Do anything else you want to do here

print 'Done'
```

for

```
for i in range(1, 5):
    print i
else:
    print 'The for loop is over'
```

break

```
while True:
    s = raw_input('Enter something : ')
    if s == 'quit':
        break
    print 'Length of the string is', len(s)
print 'Done'
```

continue

```
while True:
    s = raw_input('Enter something : ')
    if s == 'quit':
        break
    if len(s) < 3:
        print 'Too small'
        continue
    print 'Input is of sufficient length'
    # Do other kinds of processing here...
```

Functions

Function

```
def say_hello():
    # block belonging to the function
    print 'hello world'
# End of function

say_hello() # call the function
say_hello() # call the function again
```

```
$ python function1.py
hello world
hello world
```

Parameters

```
def print_max(a, b):
    if a > b:
        print a, 'is maximum'
    elif a == b:
        print a, 'is equal to', b
    else:
        print b, 'is maximum'

# directly pass literal values
print_max(3, 4)

x = 5
y = 7

# pass variables as arguments
print_max(x, y)
```

Local Variables

```
x = 50

def func(x):
    print 'x is', x
    x = 2
    print 'Changed local x to', x

func(x)
print 'x is still', x
```

```
$ python function_local.py
x is 50
Changed local x to 2
x is still 50
```

Global Variables

- Local

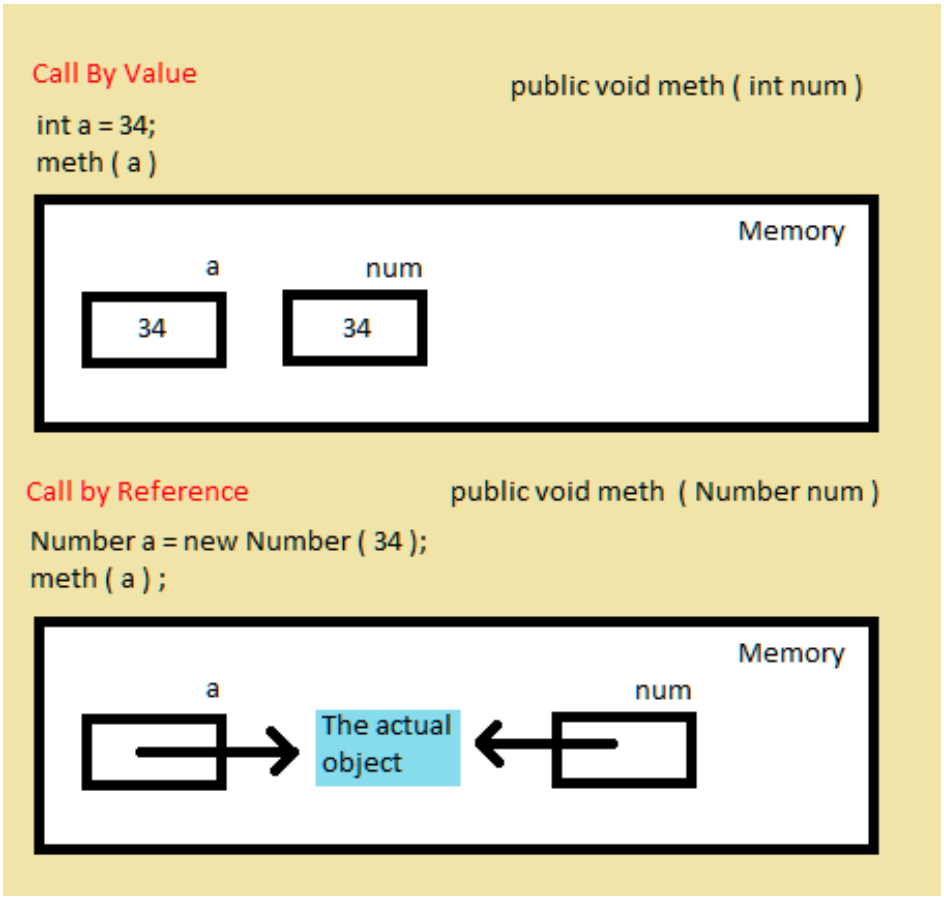
```
x = 50

def func(x):
    print 'x is', x
    x = 2
    print 'Changed local x to', x

func(x)
print 'x is still', x
```

- Global

```
x = 50

def func():
    global x

    print 'x is', x
    x = 2
    print 'Changed global x to', x

func()
print 'Value of x is', x
```

```
$ python function_global.py
x is 50
Changed global x to 2
Value of x is 2
```

Call by Ref. / Val. / …



http://www.javawithus.com/tutorial/call-by-value-and-call-by-reference

Call by … (Python)

- According to variable types
  - Immutable object
    - Call by Value
  - Mutable Object
    - Pass object reference

https://code13.tistory.com/214

```python
def dataCalc1(data):
    data = data + 1
def dataCalc2(data):
    data[0] = data[0] + 1
def main():
    data1 = 1
    data2 = [1]
    # Call by Value
    dataCalc1(data1);
    print 'data1 : ', data1
    # Call by Reference
    dataCalc2(data2);
    print 'data2 : ', data2

if __name__ == '__main__':
    main()
```

결과

data1 : 1

data2 : [2]

Function Parameters

Positional args

Keyword args

```
def say(message, times=1):
    print message * times

say('Hello')
say('World', 5)
```

```
$ python function_default.py
Hello
WorldWorldWorldWorldWorld
```

```
def func(a, b=5, c=10):
    print 'a is', a, 'and b is', b, 'and c is', c

func(3, 7)
func(25, c=24)
func(c=50, a=100)
```

```
$ python function_keyword.py
a is 3 and b is 7 and c is 10
a is 25 and b is 5 and c is 24
a is 100 and b is 5 and c is 50
```

VarArgs
(Variable + Arguments)

- Positional args

```
def save_ranking(*args):
    print(args)
save_ranking('ming', 'alice', 'tom', 'wilson', 'roy')
# ('ming', 'alice', 'tom', 'wilson', 'roy')
```

- Keyword args

```
def save_ranking(**kwargs):
    print(kwargs)
save_ranking(first='ming', second='alice', fourth='wilson', third='tom', fifth='roy')
# {'first': 'ming', 'second': 'alice', 'fourth': 'wilson', 'third': 'tom', 'fifth': 'roy'}
```

- Positional args + Keyword args

```
def save_ranking(*args, **kwargs):
    print(args)
    print(kwargs)
save_ranking('ming', 'alice', 'tom', fourth='wilson', fifth='roy')
# ('ming', 'alice', 'tom')
# {'fourth': 'wilson', 'fifth': 'roy'}
```

https://mingrammer.com/understanding-the-asterisk-of-python/

Return

```
def maximum(x, y):
    if x > y:
        return x
    elif x == y:
        return 'The numbers are equal'
    else:
        return y

print maximum(2, 3)
```

ılt by a function

```
def some_function():
    pass
```

Return

```
1   #!/usr/bin/env python3
2
3   def getPerson():
4       name = "Leona"
5       age = 35
6       country = "UK"
7       return name,age,country
8
9   name,age,country = getPerson()
10  print(name)
11  print(age)
12  print(country)
```

- utputs to be returned

Module

- A module imported by “import” includes methods and

```
import sys

print('The command line arguments are:')
for i in sys.argv:
    print i

print '\n\nThe PYTHONPATH is', sys.path, '\n'
```

achieved as its file name, i.e., a
name

- ***Run …***

from … import

```
from math import sqrt
print "Square root of 16 is", sqrt(16)
```

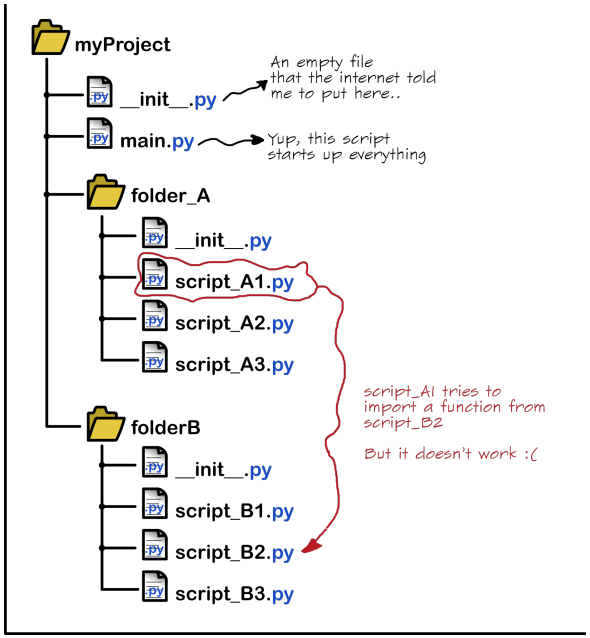
name in Module

```
if __name__ == '__main__':
    print 'This program is being run by itself'
else:
    print 'I am being imported from another module'
```

the name of module

e you are

Module



m_eat.py

name = "홍길동"

```
def cook():
    print("요리하다.")

def eat():
    print("먹다.")
```

Module

```
def say_hi():
    print 'Hi, this is mymodule speaking.'

__version__ = '0.1'
```

```
import mymodule

mymodule.say_hi()
print 'Version', mymodule.__version__
```

```
from mymodule import say_hi, __version__

say_hi()
print 'Version', __version__
```

```
from mymodule import *
```

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

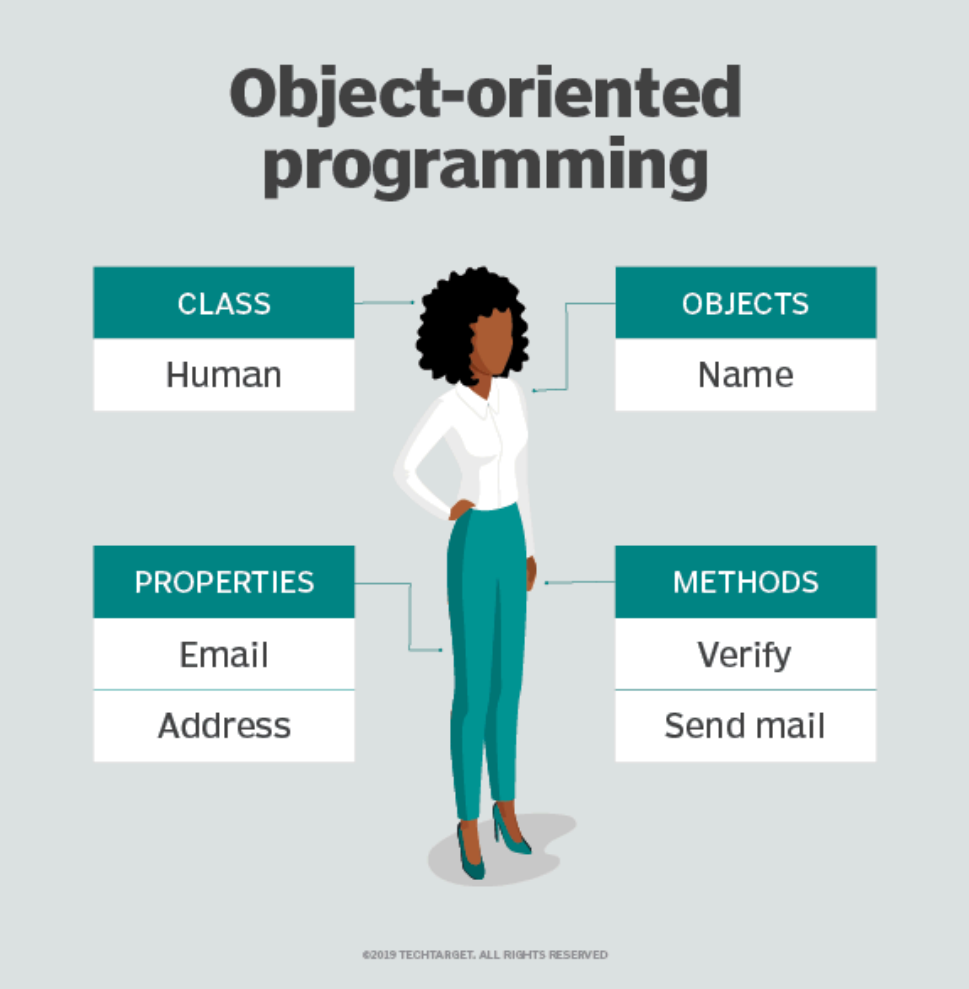
Reference

```
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
```

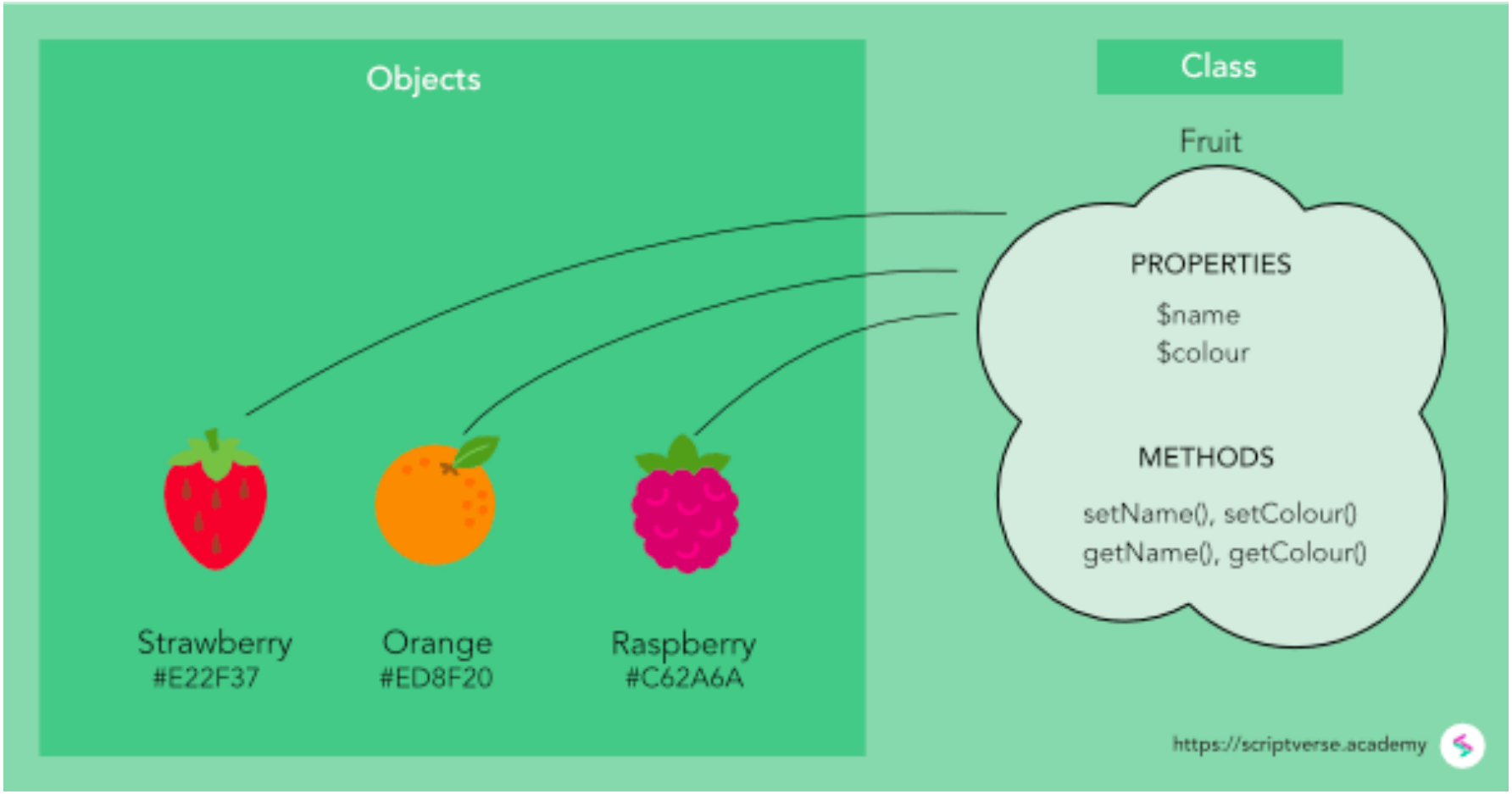
- For collections, “=“ does not copy the source to the destination
  - Notice how to make a copy by using slice

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()
```

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)
```

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