CSE 106

Lecture 4 – Python
Classes and Objects, Dictionaries, Modules, Packages

Acknowledgement: many code examples from learnpython.org, w3schools.com, towardsdatascience.com, and geeksforgeeks.org

Classes and Objects

- Objects encapsulate variables and functions into a single entity
- Objects get their variables and functions from classes
- Classes are essentially a template to create your objects
- Remember: An object is an instance of a class

```
class NameClass:
    name = "Sam"
    def say_name(self):
        print("Hello, my name is %s" %self.name)
```

Classes and Objects

- Create an object from a class
- Access the variables and member functions (methods) from an object
- Methods need “self” as first argument
- Self is a reference to the instance of the class

```
class NameClass:
    _name = "Sam"
    def say_name(self):
        print("Hello, my name is %s" %self._name)

namer = NameClass()
namer.say_name()
# namer._name
```

Classes and Objects

- You can create different objects that are of the same class
- Each object has independent copies of variables defined in the class

```
class NameClass:
    name = "Sam"
    def say_name(self):
        print("Hello, my name is %s" %self.name)

namer1 = NameClass()
namer1.name = "Sally"
namer2 = NameClass()
namer1.say_name()
namer2.say_name()
```

Classes and Objects - Constructors

- Initializes values of class members when an object of the class is created
- The __init__() method is the constructor
- The “self” variable is used here, just like in other methods

```
class Fruit:
    def __init__(self):
        self.color = "red"
        self.numSeeds = 10

    def print_info(self):
        print("Number of seeds: %d" % self.numSeeds)
        print("The color: %s" % self.color)
```

Classes and Objects - Constructors

- Can require values to be passed in when object is created

```
class Fruit:
    def __init__(self, color, numSeeds):
        self.color = color
        self.numSeeds = numSeeds

    def print_info(self):
        print("Number of seeds: %d" % self.numSeeds)
        print("The color: %s" % self.color)

fruit = Fruit("green", 23)
fruit.print_info()
```

Classes and Objects - Inheritance

- Can define a class that inherits all the methods and properties from another class
- Parent Class (Base Class)
  - the class being inherited from
- Child Class (Derived Class)
  - the class that inherits from another class

Classes and Objects - Inheritance

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):  # Student inherits from Person
  pass

student = Student("Mike", "Olsen")
student.printname()
```

Classes and Objects - Inheritance

- When you add the __init__() function, the child class will no longer inherit the parent's __init__() function

```
class Student(Person):
  def __init__(self, fname, lname):
    #add properties etc.
```

Classes and Objects - Inheritance

- To keep the inheritance of the parent's __init__() function, add a call to the parent's __init__() function:
- Alternatively call the Super function

```
class Student(Person):
  def __init__(self, fname, lname):
    Person.__init__(self, fname, lname)

# This is the same as above
class Student(Person):
  def __init__(self, fname, lname):
    super().__init__(fname, lname)
```

Classes and Objects - Inheritance

- Add properties and methods to child classes, like in the example below

```
class Student(Person):
  def __init__(self, fname, lname, year):
    super().__init__(fname, lname)
    self.graduationyear = year

  def welcome(self):
    print("Welcome", self.firstname, self.lastname, "to the class of", self.graduationyear)

mike = Student("Mike", "Olsen", 2019)
mike.welcome()
```

Wake up!

- https://youtu.be/BGLsUxe2pik

Dictionaries

- An unordered collection of key-value pairs (ordered since Python 3.6)
- Very efficient data structure (hash table)
- Keys
  - used to find the corresponding value
  - must be unique and immutable
  - can be strings, numbers or tuples
- Values can be of any type

```
empty_dict = {}
```

Dictionaries

Grades dictionary	
keys	**values**
John	A
Emily	A+
Betty	B
Mike	C
Ashley	A

```
grades = {'John':'A', 'Emily':'A+', 'Betty':'B', 'Mike':'C', 'Ashley':'A'}
```

Dictionaries - Access

- Access a value by its key using [] or get

```
grades = {'John':'A', 'Emily':'A+', 'Betty':'B', 'Mike':'C', 'Ashley':'A'}

print(grades['John'])
print(grades.get('Betty'))
```

Dictionaries - Access

- Get all keys or values… can convert these to a list
- Get key and value with “items”, which is a tuple

```
grades = {'John':'A', 'Emily':'A+', 'Betty':'B', 'Mike':'C', 'Ashley':'A'}

print(grades.keys())
print(list(grades.keys()))
print(grades.values())
print(list(grades.values()))

print(grades.items())
```

Dictionaries – Add and Update

- Dictionaries are mutable so we can update, add or delete items
- Add or update a dictionary with the same syntax

```
grades = {'John':'A', 'Emily':'A+', 'Betty':'B', 'Mike':'C', 'Ashley':'A'}

grades['John'] = 'B'  # edits John’s grade to a B
grades['Edward'] = 'B+'  # Makes a new entry for Edward
print(grades)
```

Dictionaries - Update

- Update a dictionary with a new dictionary

```
grades = {'John':'A', 'Emily':'A+', 'Betty':'B', 'Mike':'C'}
grades_new = {'John':'B', 'Sam':'A', 'Betty':'A'}

grades.update(grades_new)
print(grades)  # {'Betty': 'A', 'Emily': 'A+', 'John': 'B', 'Mike': 'C', 'Sam': 'A'}
```

Dictionaries – Delete Values

- Use the del or pop function to delete an item

```
grades = {'John':'A', 'Emily':'A+', 'Betty':'B', 'Mike':'C', 'Ashley':'A', 'Edward':'C+'}
del(grades['Edward'])
grades.pop('Ashley')  # returns 'A'

print(grades)  # {'Betty': 'B', 'Emily': 'A+', 'John': 'B', 'Mike': 'C'}
```

JSON

- JSON stands for JavaScript Object Notation
- JSON is very common web-based format for sending and receiving messages
- JavaScript and Python have easy ways to create objects and dictionaries from JSON
- Python supports JSON with a built-in package called json

Dictionaries and JSON

- Convert Python dictionary to JSON using:
  - json.dumps() – converts dictionary to a string (hence the s)
  - Json.dump() – writes dictionary to a file

```
import json

grades = {'John':'A', 'Emily':'A+', 'Betty':'B', 'Mike':'C', 'Ashley':'A'}
json_object = json.dumps(grades)
print(json_object)

grades = {'John':'A', 'Emily':'A+', 'Betty':'B', 'Mike':'C', 'Ashley':'A'}
with open("sample.json", "w") as outfile:
    json.dump(grades, outfile)
```

Dictionaries and JSON

- Convert JSON to a Python dictionary using
  - json.loads() – converts JSON string to dictionary (hence the s)
  - json.load() – reads dictionary from a file

```
import json

grades_json = """{"John":"A", "Emily":"A+", "Betty":"B", "Mike":"C", "Ashley":"A"}"""
grades_dict = json.loads(grades_json)
print(grades_dict)


with open('data.json') as json_file:
    data = json.load(json_file)
    print(data)
```

Modules

- Modules in Python are simply Python files with a .py extension
- The name of the module will be the name of the file
- A Python module can have functions, classes or variables defined and implemented

Modules - Import

```
# draw.py
def draw_game():
    #write your function here
def clear_screen(screen):
    ...
```

```
# game.py
import draw

def play_game():
    ...

if __name__ == '__main__’:
    result = play_game()
    draw.draw_game(result)
```

```
mygame/
    game.py
    draw.py
```

Modules – Import to Current Namespace

- Import a function or class into this namespace with **from**
- Notice you don’t need to specify the namespace when you call it

```
# game.py
from draw import draw_game

def play_game():
    ...

if __name__ == '__main__':
    result = play_game()
    draw_game(result)
```

Modules – Import All

- Import all functions and classes from a module with *

```
# draw.py
def draw_game():
    ...
def clear_screen(screen):
    ...


# game.py
from draw import *

def play_game():
    ...

if __name__ == '__main__’:
    result = play_game()
    draw_game(result)
    clear_screen(None)
```

Modules – Import as custom name

- Import module as a custom name

```
# draw.py
def draw_game():
    ...
def clear_screen(screen):
    ...


# game.py
import draw as d

def play_game():
    ...

if __name__ == '__main__’:
    result = play_game()
    d.draw_game(result)
```

Packages

- Namespaces which contain multiple modules and packages
- Imported the same way a module can be imported
- Each package in Python is a directory which **MUST** contain a special file called `__init__.py`
- This file can be empty, and it indicates that the directory it contains is a Python package

```
a_package
    __init__.py
    module_a.py
    a_sub_package
      __init__.py
      module_b.py
```

Packages

```
a_package
   __init__.py
   module_a.py
   a_sub_package
     __init__.py
     module_b.py

import a_package
a_package.a_name  # a_name is defined in __init__.py

# Can’t do a_package.module_a call, must do:
import a_package.module_a
a_package.module_a.func_in_mod_a
```