Python: Object Oriented Programming (OOP), an introduction to classes*

Arash Pourhabib†

1 Classes

- A class packs **data** together with **functions** operating on the data into a single unit.
- Classes enable us to create more efficient and modular codes by grouping data and functions.

Many mathematical operations can be coded without using classes; however, for some cases, such as creating Graphical User Interfaces (GUI), classes help us create more effective programs that are easier to understand.

*References: (1) Langtangen, Hans Petter. A primer on scientific programming with Python, Fourth Edition. Springer, 2014. (2) https://docs.python.org/2/reference/

†School of Industrial Engineering and Management, Oklahoma State University

2 Examples: Bank Accounts[1]

A bank account has some data (the account holder, the account number, current balance), and we can perform operations such as withdrawing money, depositing money, and showing the account balance.

```
class Account:
    def __init__(self, name, acc_number, acc_money):
        self.name = name
        self.no = acc_number
        self.balance = acc_money
```

- The function **__init__** is called a constructor and initializes the data.
- The keyword **self** represents an arbitrary instance of a class.

Example:

```
a1 = Account('John Doe', '5246856', 3000)
a2 = Account('Jane Smith', '9266350', 5000)

print(a1.name)
>>> John Doe #python output
print(a2.balance)
>>> 5000 #python output
```

[1]From Chapter 7.2 of "A primer on scientific programming with Python" by Langtangen.

Next, we add functions to class Account.

```
class Account:
    def __init__(self, name, acc_number, acc_money):
        self.name = name
        self.no = acc_number
        self.balance = acc_money
    def deposit(self,amount):
        self.balance = self.balance + amount
    def withdraw(self, amount):
        self.balance = self.balance - amount
    def show_balance(self):
        print('{:s} has {:.2f} dollars.'.format(self.name, self.balance))
```

Rules about **self**:

- **self** must be the first argument of a class method.
- To access other attributes or methods inside a class, we use prefix **self**, for example **self.XYZ**, where **XYZ** is any attribute or method.
- We drop **self** as an argument when we call class methods.

Example:

```
a1 = Account('John Doe', '5246856', 3000)
a2 = Account('Jane Smith', '9266350', 5000)
a1.deposit(1500)
a2.withdraw(400)
a1.show_balance()
>>> John Doe has 4500.00 dollars.  #python output
a2.show_balance()
>>> Jane Smith has 4600.00 dollars. #python output
```

Terminology:

- Each realization of a class, like **a1** or **a2** above, is called an *instance*.
- Functions inside a class, like **deposit**, are called *methods*.
- The variables initialized by **__init__**, like **name**, are called *attributes*.

3 Inheritance[2]

Class *hierarchy* refers to a family of classes. A class that *inherits* methods and attributes from another class is called a *child class*.

Example: Create a class for savings account as a child class of `Account`.

```
class Savings_Account(Account):
    def __init__(self, name, acc_number, acc_money, interest_rate):
        super().__init__(name, acc_number, acc_money)
        self.ir = interest_rate
    def show_future_balance(self,n):
        fb = self.balance*(1+self.ir)**n
        print('{:s} will have {:.2f} dollars\
        after {:d} years.'.format(self.name, fb, n))
```

Example:

```
c1 = Savings_Account('John Doe', '4258851', 10000, 0.02)
c1.show_balance()
>>> John Doe has 10000.00 dollars.   #python output
c1.show_future_balance(10)
>>> John Doe will have 12189.94 dollars after 10 years.   #python output
```

[2]See Chapter 9 of "A primer on scientific programming with Python" by Langtangen.

4 Exercise

Exercise 1: Write a class that represents circles. Each object of this class has three attributes, x-coordinate, y-coordinate, and the radius. Your class should have two methods, area (which returns the area of the circle) and circumference (which returns the circle's circumference). Create two instances of class circle.

Exercise 2: Create a class called ScrewThread. This class should contain the thread's system SAE or Metric), hand (left or right), diameter, pitch and lead. Create a method that prints all of the thread information, nicely formatted.

Exercise 3: Create a class called Bolt. This class should contain the Bolt's thread (an object of type ScrewThread), grade (5 or 8), headtype (hex or socket) system SAE or Metric) and length. Create a method that returns number of threads on the bolt. Create a method that prints all of the bolt information, nicely formatted.

Exercise 4: Create a class called Person. This class should have the person's name and his/her year of birth as attributes. Create a method that returns the person's age as of today.

Exercise 5: Next, define a child class, Employee, which inherits from Person. This child class (in addition to the parent's attributes) has the following attributes: the employee's phone number and the company that person is working for. Write a method that prints the employee's phone number. Create three

instances of class Employee.

Exercise 6: Next, define a class called Company, This class should contain a list of employees. Write a method that prints the the company phone directory