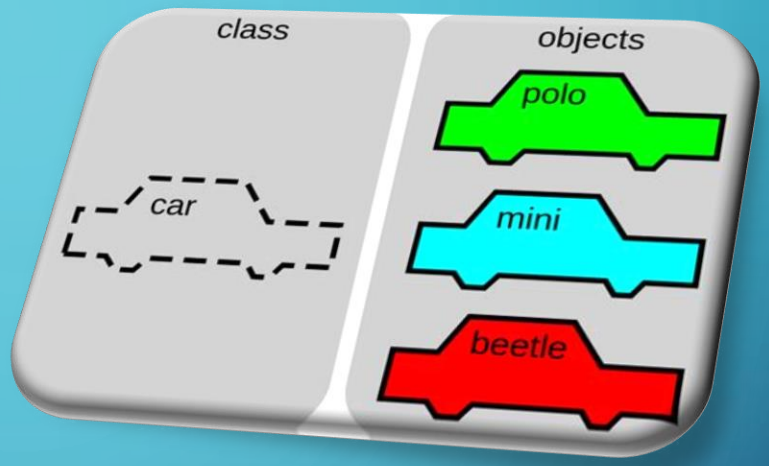


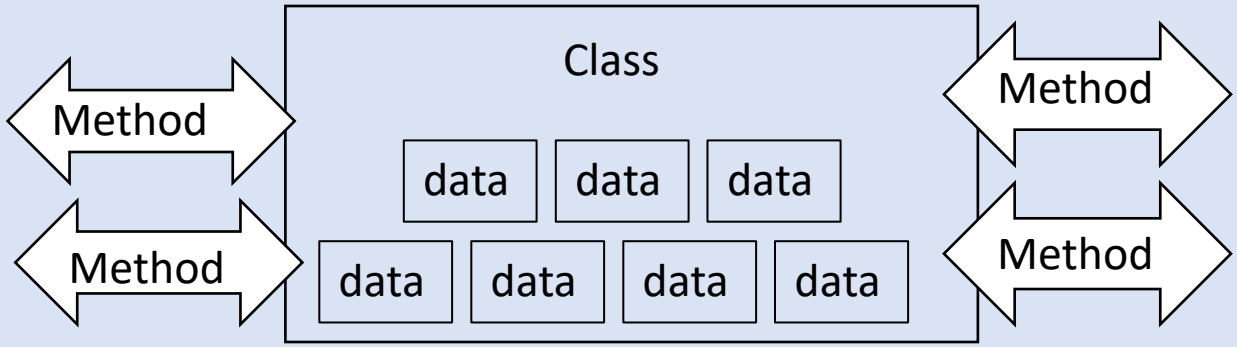
CLASSES

CIS 41A – PROGRAMMING IN PYTHON

BASED ON MATERIALS DEVELOPED BY CLARE NGUYEN

What Is a Class?

- A class is a data type that allows for data storage and a set of methods to access the class data.



- The Python built-in data types: int, str, list, etc. are each a specific type of class.
- Every data type in Python is a class, and Python is called an object oriented language.

Advantage of Classes

- There are 2 main advantages to using classes:
    1. Classes allow us to model real life entities, which makes the design of the software more intuitive.

    2. Classes can be re-used, which makes software development faster because we can take advantage of existing classes.

Modeling with Classes (1 of 2)

- Methods effectively control the behavior of a class.
- The Python List class has a method to sort data, but a Python set class does not have a sort method because data in a set are not in any order.
- User defined classes, like BankAccount, have withdraw() and a deposit() methods, because these are behaviors of a bank account.

Modeling with Classes (2 of 2)

- When writing software for a bank, it is intuitive to think of a Bank class with methods such as set_interest_rate(), advertise(), invest(), etc.
- The Bank class can contain other classes such as CheckingAccount class, SavingsAccount class, SafeDepositBox class, etc.
- Each of these classes will have its own methods or behavior.
- The design of the bank software becomes more “natural” when we have classes that model real life entities of a bank.

Class Re-Use

Some general purpose classes, such as the Python List class, can be used in many applications.

- Someone at Python wrote the List class, and now other programmers can use it without having to code it.
- This makes coding other programs development time effort shorter.
- Python supports class re-use by providing many software modules.

Classes and Objects (1 of 2)

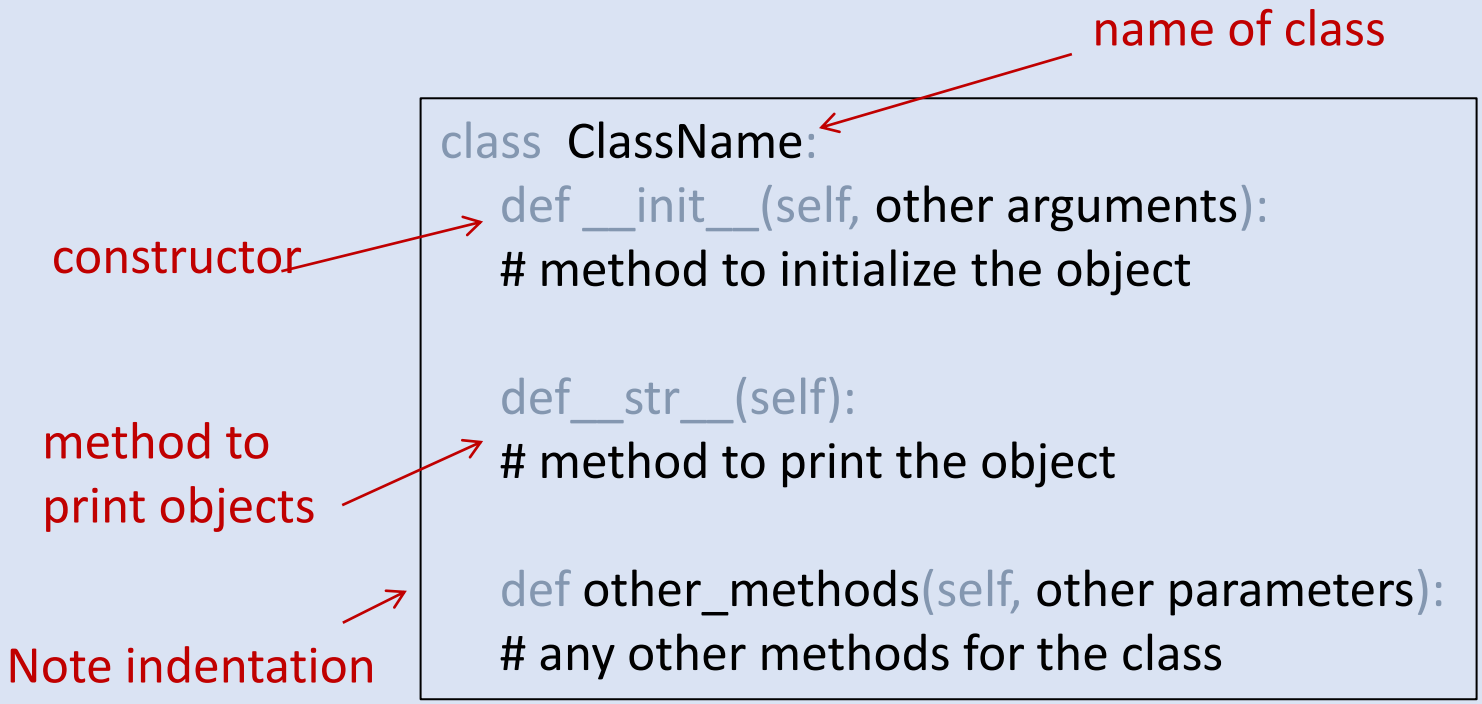
- A class is a data type.
- The BankAccount class is a data type, just like int is a data type.
- When the value 5 is used, memory space is allocated for an integer.
- A Bank Account class object is used, memory space is allocated large enough to contain the Bank Account data.
- The memory space for a class data type is called a *class instance* or *object*.

Classes and Objects (2 of 2)

- A class is a description of the data type and its behavior.
- An object is the actual memory space that stores data. An object is an *instance* of a class.
- A class is like an architect's blue print of a house, a design on paper. The house is the object, it is built from the design.
- Many house "instances" can be built from one blue print.

Defining a Class (1 of 3)

- Format for defining a class:



- Every method has self as the first parameter. **self** specifies the object, and every method works on the object.

Defining a Class (2 of 3)

- The __init__ method:

```
def __init__(self, other arguments):
    # method to initialize the object
```

  - The name starts and ends with 2 underscores (_).
  - When an object created, the __init__ method constructs the object.
  - The data can be a default value or it can be passed in through the arguments.

```
def__str__(self):
 # method to print the object
```

- The __str__ method:
  - The name starts and ends with 2 underscores.
  - When the function print is used to print the object, the __str__ method runs to return a string

- Classes typically have other methods to define its behavior.

Defining a Class (3 of 3)

- Example of a bank account class definition:

```
# Bank account class
class BankAcct():
    # initialize account number and balance
    def __init__(self, num, bal):
        self.num = num
        self.bal = bal

    # return string with object data to be printed
    def __str__(self):
        return "Account: " + str(self.num) + "\nBalance: " + str(self.bal)

    # return account number
    def getNumber(self):
        return self.num

    # return account balance
    def getBalance(self):
        return self.bal

    # allow a deposit
    def deposit(self, amt):
        self.bal = self.bal + amt
```

We must pass in the account number and balance when creating the object

The last 4 methods let us:

- Print the acct info as a string
- Get the acct number
- Get the acct balance
- Make a deposit

- Note that all object data variables start with: self.
  But parameters and temporary variables don't use self.

Working With an Object

- Working with an example object of the BankAcct class:

Code

```
# Bank account class
class BankAcct():
    # initialize account number and
    def __init__(self, num, bal):
        self.num = num
        self.bal = bal

    # return string with object data
    def __str__(self):
        return "Account: " + str(sel

    # return account number
    def getNumber(self):
        return self.num

    # return account balance
    def getBalance(self):
        return self.bal

    # allow a deposit
    def deposit(self, amt):
        self.bal = self.bal + amt
```

```
# instantiate myAcct object
# with acct num 123 and $50
myAcct = BankAcct(123, 50)
# verify acct info
print(myAcct)
# deposit $100
myAcct.deposit(100)
# print acct balance
print("new balance is:", myAcct.getBalance() )
```

Output

```
Account: 123
Balance: 50
new balance is: 150
```