Modules

- The highest level structure of Python
- Each file with the .py suffix is a module
- Each module has its own namespace

import Statement

- In order to use external function
  - Stored in different python file
  - Use the **import** statement
- import module1[, module2[,... moduleN]
  - The interpreter loads and executes each module
  - Multiple imports only load once
- In order to use entities
  - Module name must precede identifier
  - module1.function()

from...import Statement

- In order to avoid the reference to the module namespace (module1.function())
  - Use **from module1 import function**
- from modname import name1[, ... nameN]
  - Imports name1 .. nameN from modname
  - User can use name1 .. nameN directly
- from modname import *
  - Import all items from modname

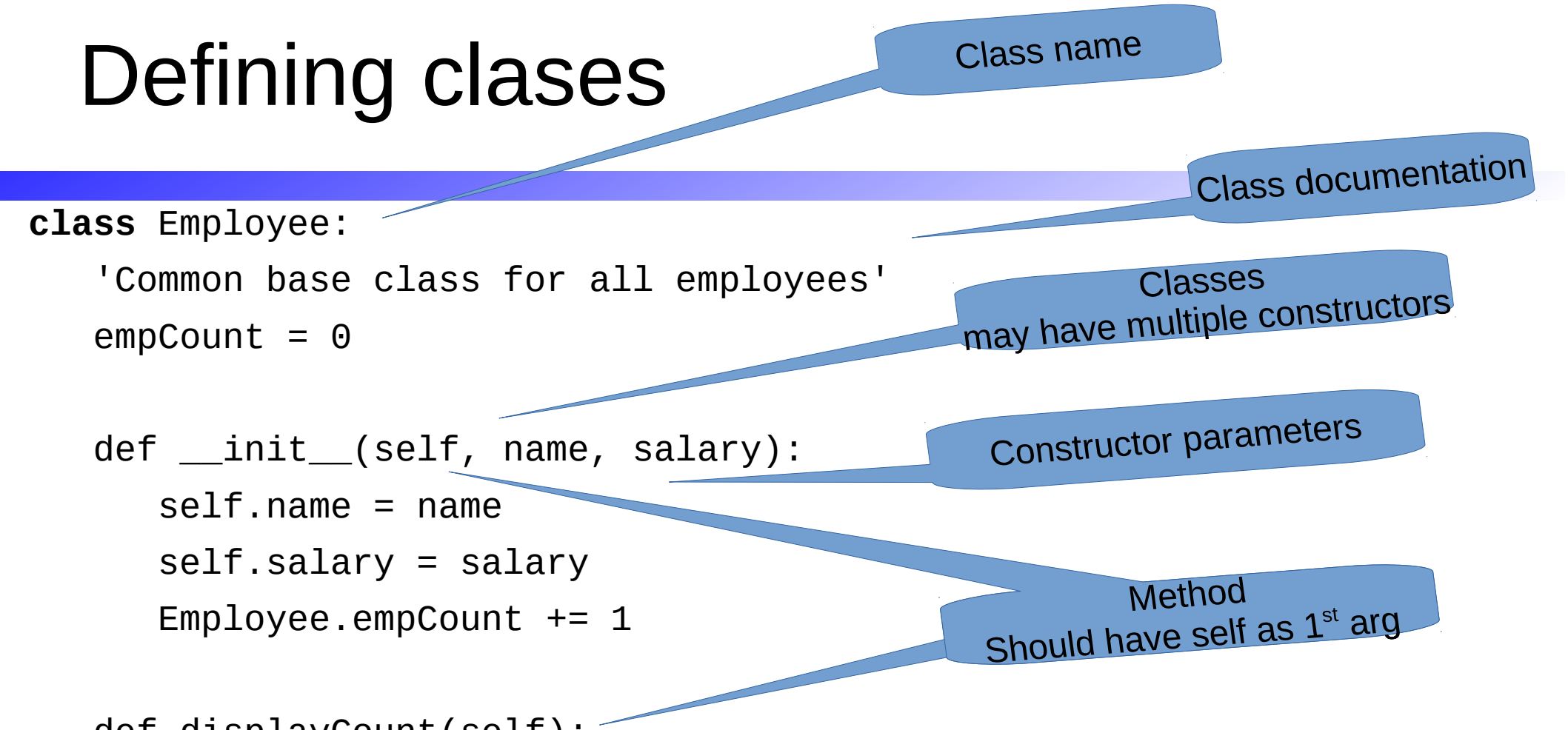
Modules: Imports

import mymodule	Brings all elements of mymodule in, but must refer to as mymodule.<elem>
from mymodule import x	Imports x from mymodule right into this namespace
from mymodule import *	Imports all elements of mymodule into this namespace

Classes and objects

- Class:
  - A user-defined prototype for an object
  - Defines a set of attributes that characterize any object of the class.
  - The attributes are data members (class variables and instance variables) and methods, accessed via dot notation.
- Object
  - A unique instance of a data structure that's defined by its class.
  - An object comprises both data members (class variables and instance variables) and methods.

Defining clases

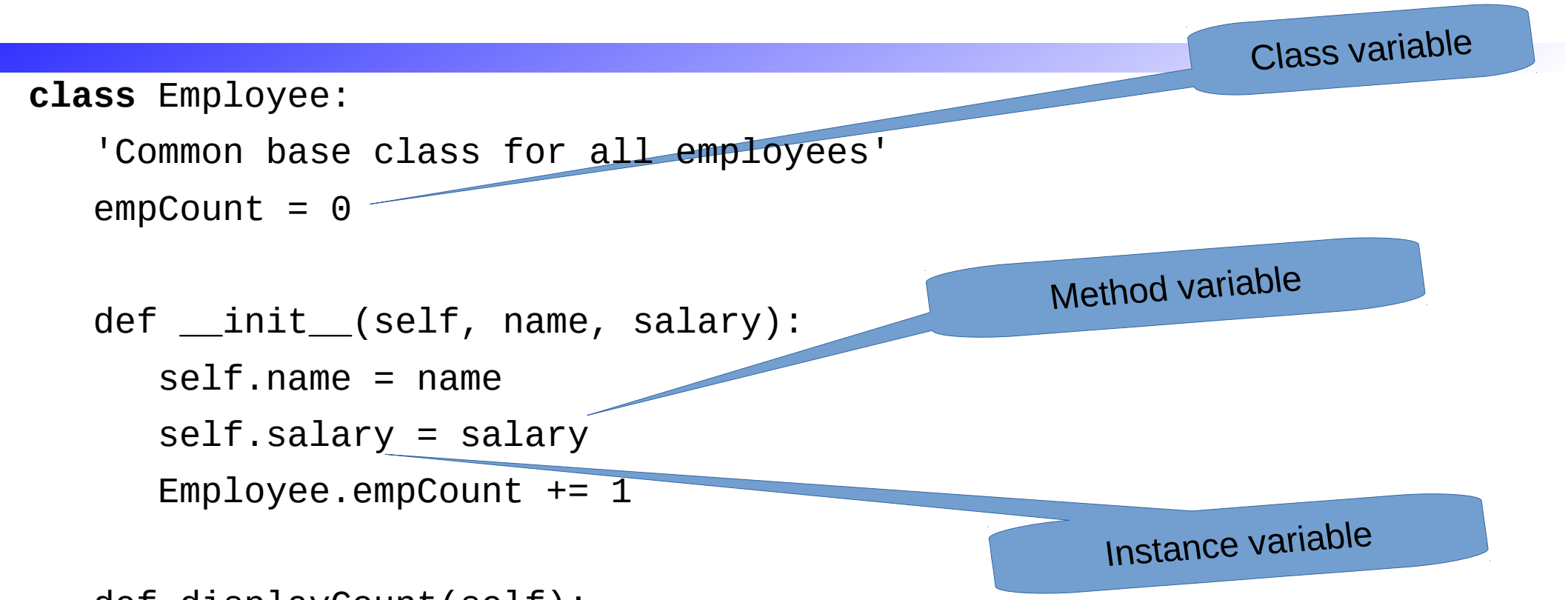



```
class Employee:
   'Common base class for all employees'
   empCount = 0

   def __init__(self, name, salary):
      self.name = name
      self.salary = salary
      Employee.empCount += 1

   def displayCount(self):
     print "Total Employee %d" % Employee.empCount

   def displayEmployee(self):
      print "Name : ", self.name,  ", Salary: ", self.salary
```

Defining clases

- All methods
  - First argument is self (the object running the code)
- Constructors
  - Should be called __init__
- Class variable
  - A variable that is shared by all instances of a class.
  - Class variables are defined within a class but outside any of the class's methods.
  - **`Employee.empCount`**
- Instance variable
  - A variable that is defined inside a method and belongs only to the current instance of a class.
  - **`self.name`**

Defining clases




```
class Employee:
   'Common base class for all employees'
   empCount = 0

   def __init__(self, name, salary):
      self.name = name
      self.salary = salary
      Employee.empCount += 1

   def displayCount(self):
     print "Total Employee %d" % Employee.empCount

   def displayEmployee(self):
      print "Name : ", self.name,  ", Salary: ", self.salary
```

Creating instance objects

- Objref = className(constructor args)

```
"This would create first object of Employee class"
emp1 = Employee("Zara", 2000)
"This would create second object of Employee class"
emp2 = Employee("Manni", 5000)
```

Accessing atributes/methods

- Uses the dot operator
  - ObjectReference.attribute
  - ObjectReference.method(methodArguments)
- Built-in attributes
  - __**dict**__ : Dictionary containing the class's namespace.
  - __**doc**__ : Class documentation string or None if undefined.
  - __**name**__ : Class name.
  - __**module**__ : Module name in which the class is defined. This attribute is "__main__" in interactive mode.
  - __**bases**__ : A possibly empty tuple containing the base classes, in the order of their occurrence in the base class list.

EXERCISE: Classes and objects

- Implement a class (named rpnCalculator)
  - Object attributes
    - Memory: stack of numbers
  - Object methods
    - pushValue:
      - pushes a value to the top of the stack
    - popValue:
      - removes and returns the value on the top of the stack
    - Add / sub:
      - removes two topmost values on the stack, adds them, and pushes the value to the top of the stack
- Test the code on a simple example

Networking

- Python sockets
  - socket module
    - Import socket / from socket import *
  - socket class
    - s = socket()
- https://docs.python.org/2/library/socket.html

Python sockets

- Server
  - s.bind()
    - This method binds address (hostname, port number pair) to socket.
  - s.listen()
    - This method sets up and start TCP listener.
  - s.accept()
    - This passively accept TCP client connection, waiting until connection arrives (blocking)
- Client
  - s.connect()
    - This method actively initiates TCP server connection.
- General
  - s.recv()
    - This method receives TCP message
  - s.send()
    - This method transmits TCP message
  - s.recvfrom()
    - This method receives UDP message
  - s.sendto()
    - This method transmits UDP message
  - s.close()
    - This method closes socket
  - socket.gethostname()
    - Returns the hostname.

```
import socket

s = socket.socket()
host = socket.gethostname()
port = 12345
s.bind((host, port))
s.listen(5)
while True:
   c, addr = s.accept()
   print 'Connection from', addr

   c.send('Hello')
   c.close()
```

```
import socket

s = socket.socket()


host = socket.gethostname()
port = 12345
s.connect((host, port))

print s.recv(1024)
s.close
```

EXERCISE: Remote Calculator

- Implement a request reply protocol
- Server
  - Remote calculator
  - Use the previously defined class as the server processing code
- Client
  - Reads from the keyboard
    - Push, pop, add
  - Send requests to the server
- Additional Information:
  - https://docs.python.org/3/library/socket.html#socket.socket.makefile

Object serialization

- How to store objects?
- How to transfer objects
  - Serialization
- Several options exist
  - XDR, XML, Jsin
- Pickle
  - Proprietary serialization
- https://docs.python.org/2/library/pickle.html

- Pickle module
  - Import pickle
- cPickle
  - import cPickle as pickle

- Serialization
  - pickle.dump(obj, file[, protocol])
    - Stores in file
  - pickle.dumps(obj[, protocol])
    - Returns string

- Deserialization
  - pickle.load(file)
    - Returns a object “from”file
  - pickle.loads(string)¶
    - Returns a object from a file

EXERCISE: Object transfer

- Declare a class (testClass) that:
  - only contains a integer as attribute
  - Contains a constructor that initializes the attribute
- Implement a server that
  - Receives a request containing a number
  - Creates a instance of testClass
    - Initializes it with the received number
  - Send the created object to the client
- Implement a client
  - That sends an integer to the server
  - Receive a testClass object containing such integer