CPSC 231: Classes and Objects

You will learn how to define new types of variables that can have custom attributes and capabilities

slide 1

Some Drawbacks Of Using A List

- Which field contains what type of information? This isn't immediately clear from looking at the program statements.

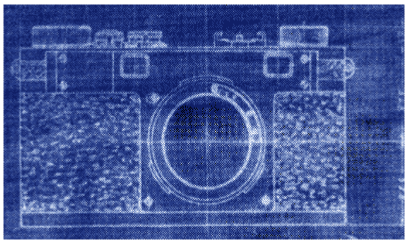
```
client = [“xxxxxxxxxxxxxxx",
          “0000000000",
          “xxxxxxxxx",
          0]
```

The parts of a composite list can be accessed via [index] but they cannot be labeled (what do these fields store?)

- Is there any way to specify rules about the type of information to be stored in a field e.g., a data entry error could allow alphabetic information (e.g., 1-800-BUY-NOWW) to be entered in the phone number field.

Classes

- Can be used to define a generic template for a new non-homogeneous composite type.
- It can label and define more complex entities than a list.
- This template defines what an instance (example) of this new composite type would consist of but it doesn't create an instance.

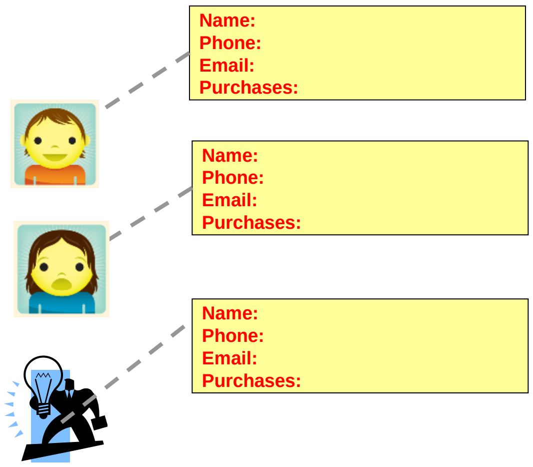


Copyright information unknown

James Tam

Classes Define A Composite Type

- The class definition specifies the type of information (called "**attributes**") that each instance (example) tracks.



James Tam

Defining A Class[1]

Note the convention: The first letter is capitalized.

- **Format:**

```
class <Name of the class>:
     name of first field = <default value>
     name of second field = <default value>
```

- **Example:**

```
class Client:
    name = "default"
    phone = "(123)456-7890"
    email = "foo@bar.com"
    purchases = 0
```

Describes what information that would be tracked by a "Client" but doesn't actually create a client variable

Contrast this with a list definition of a client

```
client = ["xxxxxxxxxxxxxxx",
                "0000000000",
                "xxxxxxxxx",
                0]
```

1 Although this structure isn't commonly used in Python it is common to many other programming languages: Java, C++

Creating An Instance Of A Class

- Creating an actual instance (instance = object) is referred to as *instantiation*
- **Format:**

```
<reference name> = <name of class>()
```

- **Example:**

```
firstClient = Client()
```

Defining A Class Vs. Creating An Instance Of That Class

- Defining a class
  - A template that describes that class: how many fields, what type of information will be stored by each field, what default information will be stored in a field.

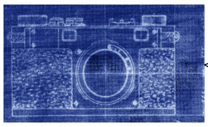


Image copyright unknown

- Creating an object
  - Instances of that class (during instantiation) which can take on different forms.

Accessing And Changing The Attributes

•Format:

```
<reference name>.<field name>                # Accessing value
<reference name>.<field name> = <value>   # Changing value
```

•Example:

```
aClient.name = "James"
```

The Client List Example Implemented Using Classes And Objects

- Name of the online example: `client.py`

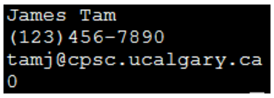
```
class Client:
    name = "default"
    phone = "(123)456-7890"
    email = "foo@bar.com"
    purchases = 0
```

The Client List Example Implemented Using Classes (2)

```
def main():
   firstClient = Client()
   firstClient.name = "James Tam"
   firstClient.email = "tam@ucalgary.ca"
   print(firstClient.name)
   print(firstClient.phone)
   print(firstClient.email)
   print(firstClient.purchases)

main()
```

```
name = "default"
phone = "(123)456-7890"
email = "foo@bar.com"
purchases = 0
```

```
name = "James Tam"
email = "tam@ucalgary.ca"
```



```
James Tam
(123)456-7890
tamj@cpsc.ucalgary.ca
0
```

What Is The Benefit Of Defining A Class?

- It allows new types of variables to be declared.
- The new type can model information about most any arbitrary entity:
  - Car
  - Movie
  - Your pet
  - A bacteria or virus in a medical simulation
  - A ‘critter’ (e.g., monster, computer-controlled player) a video game
  - An ‘object’ (e.g., sword, ray gun, food, treasure) in a video game
  - A member of a website (e.g., a social network user could have attributes to specify the person’s: images, videos, links, comments and other posts associated with the ‘profile’ object).

What Is The Benefit Of Defining A Class (2)

- Unlike creating a composite type by using a list a predetermined number of fields can be specified and those fields can be named.

```
class Client:
    name = "default"
    phone = "(123)456-7890"
    email = "foo@bar.com"
    purchases = 0

firstClient = Client ()
print(firstClient.middleName)  # Error: no such field defined
```

Classes Have **Attributes** But Also **Behaviors**

ATTRIBUTES
Name:
Phone:
Email:
Purchases:

BEHAVIORS
Open account
Buy investments
Sell investments
Close account

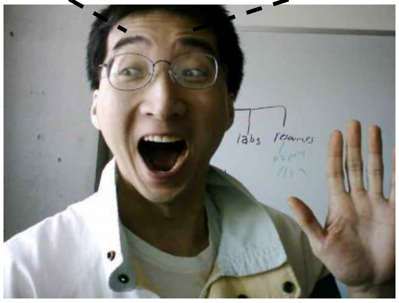


Image of James curtesy of James Tam

Class Methods ("Behaviors")

- **Functions**: not tied to a composite type or object
  - The call is 'stand alone', just name of function
  - E.g.,
  - `print()`, `input()`
- **Methods**: must be called through an instance of a composite[1].
  - E.g.,

String

Method operating on that string

```
filename = "foo.txt"
name, suffix = filename.split('.')
```

- Unlike these pre-created functions, the ones that you associate with classes can be customized to do anything that a regular function can.
- Functions that are associated with classes are referred to as *methods*.

1 Not all composites have methods e.g., arrays in 'C' are a composite but don't have methods

Defining Class Methods

Format:

```
class <classname>:
     def <method name> (self, <other parameters>):
          <method body>
```

Example:

```
class Person:
   name = "I have no name :("
   def sayName (self):
      print ("My name is...", self.name)
```

Unlike functions, every method of a class must have the 'self' parameter (more on this later)

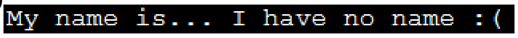
When the attributes are accessed inside the methods of a class they MUST be preceded by the suffix ".self"

James Tam

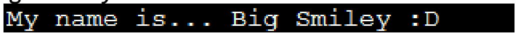
Defining Class Methods: Full Example

- Name of the online example: `person1.py`

```
class Person:
   name = "I have no name :("
   def sayName(self):
      print("My name is...", self.name)

def main():
   aPerson = Person()
   aPerson.sayName()
   aPerson.name = "Big Smiley :D"
   aPerson.sayName()

main()
```



```
My name is... I have no name :(
```



```
My name is... Big Smiley :D
```

James Tam

What Is The ‘Self’ Parameter

- Reminder: When defining/calling methods of a class there is always at least one parameter.
- This parameter is called the ‘self’ reference which allows an object to access attributes inside its methods.
- ‘Self’ needed to distinguish the attributes of different objects of the same class.
- Example:

```
bart = Person()
lisa = Person()
lisa.sayName()
```

```
def sayName():
    print "My name is...", name
```

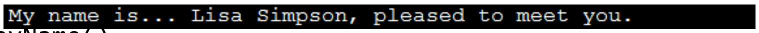
Whose name is this? (This won’t work)

James Tam

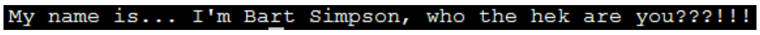
The Self Parameter: A Complete Example

- Name of the online example: person2.py

```
class Person:
   name = "I have no name :("
   def sayName(self):
      print("My name is...", self.name)

def main():
   lisa = Person()
   lisa.name = "Lisa Simpson, pleased to meet you."
   bart = Person()
   bart.name = "I'm Bart Simpson, who the hek are
  you???!!!"

   lisa.sayName()
   bart.sayName()

main()
```



```
My name is... Lisa Simpson, pleased to meet you.
```



```
My name is... I'm Bart Simpson, who the hek are you???!!!
```

“The Simpsons” © Fox

James Tam

Recap: Accessing Attributes & Methods

- **Inside the class definition** (inside the body of the class methods)
  - Preface the attribute or method using the 'self' reference

```
class Person:
    name = "No-name"
    def sayName(self):
        print("My name is...", self.name)
```

- **Outside the class definition**
  - Preface the attribute or method using the name of the reference used when creating the object.

```
def main():
    lisa = Person()
    bart = Person()
    lisa.name = "Lisa Simpson, pleased to meet you."
```

James Tam

Initializing The Attributes Of A Class

- Classes have a special method that can be used to initialize the starting values of a class to some specific values.
- This method is automatically called whenever an object is created.
- **Format**:

No spaces here

```
class <Class name>:
def __init__(self, <other parameters>):
     <body of the method>
```

- **Example**:

```
class Person:
   name = ""
   def __init__(self):
       self.name = "No name"
```

This design approach is consistent with many languages

James Tam

Initializing The Attributes Of A Class

- Because the ‘`init()`’ method is a method it can also be called with parameters which are then used to initialize the attributes.
- **Example**:

```
# Attribute is set to a default in the class definition and then the
# attribute can be set to a non-default value in the init() method.
# (Not standard Python but a common approach with many languages)
class Person
    name = "Default name" # Create attribute here
    def __init___(self, aName):
        self.name = aName
```

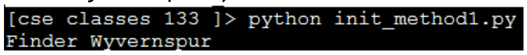
–OR

```
# Create the attribute in the init() method. (Approach often used in
# Python).
class Person
    def __init___(self, aName):
        self.name = aName  # Create attribute here
```

Full Example: Using The “`Init()`” Method

- The name of the online example: `init_method1.py`

```
class Person:
   name = "Nameless bard"

   def __init__(self, aName):
       self.name = aName

def main():
   aPerson = Person("Finder Wyvernspur")
   print(aPerson.name)

main()
```



```
[cse classes 133 ]> python init_method1.py
Finder Wyvernspur
```

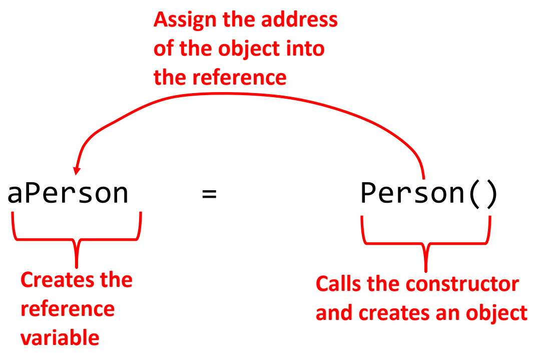
“Nameless bard” & “`Finder Wyvernspur`” © Wizards of the Coast (April 24, 2012)

Constructor: A Special Method

- Constructor method: a special method that is used when defining a class and it is automatically called when an object of that class has been created.
  - E.g., `aPerson = Person()    # This calls the constructor`
- In Python this method is named ‘`init`’.
- Other languages may require a different name for the syntax but it serves the same purpose (initializing the fields of an object as it’s being created).
- This method should never have a return statement that returns a value.
  - Should be (if return is needed) “`return`”
  - Never return a type e.g., `return(12)`

James Tam

Objects Employ References


Assign the address of the object into the reference
aPerson
=
Person()
Creates the reference variable
Calls the constructor and creates an object

James Tam

Objects Employ References (2)

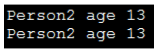
- Similar to lists, objects are accessed through a reference.
- The reference and the object are two separate memory locations.
- Name of the online example: objectReference.py

```
class Person:
    age = 0
    name = "none"
    def __init__(self,newAge,newName):
        self.age = newAge
        self.name = newName

    def displayAge(aPerson):
        print("%s age %d" %(aPerson.name,aPerson.age))
```

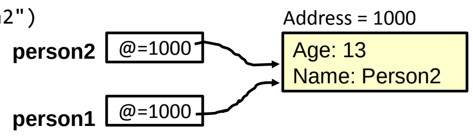
James Tam

Objects Employ References (3)

```
def start():
    person1 = Person(13,"Person2")
    person2 = person1
    displayAge(person1)
    displayAge(person2)
    print()
```



```
Person2 age 13
Person2 age 13
```



```
start()
```

James Tam

Objects Employ References (2)

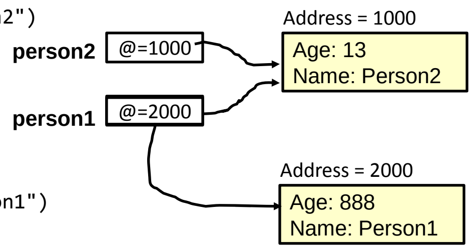
```
def start():
    person1 = Person(13,"Person2")
    person2 = person1
    displayAge(person1)
    displayAge(person2)
    print()
```

```
Person2 age 13
Person2 age 13
```

```
    person1 = Person(888,"Person1")
    displayAge(person1)
    displayAge(person2)
```

```
Person1 age 888
Person2 age 13
```

```
start()
```



James Tam

Default Parameters

- Similar to other methods, '`init`' can be defined so that if parameters aren't passed into them then default values can be assigned.
- **Example**:

```
def __init__ (self, name = "I have no name"):
```

$\underbrace{\texttt{name = "I have no name"}}$

This method can be called either when a personalized name is given or if the name is left out.

- Method calls (to '`init`'), both will work

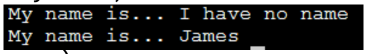
```
smiley = Person()
jt = Person("James")
```

James Tam

Default Parameters: Full Example

- Name of the online example: `init_method2.py`

```
class Person:
   name = ""
   def __init__(self, name = "I have no name"):
       self.name = name

def main():
   smiley = Person()
   print("My name is...", smiley.name)
   jt = Person("James")
   print("My name is...", jt.name)

main()
```



```
My name is... I have no name
My name is... James
```

James Tam

Modules: Dividing Up A Large Program

- Module: In Python a module contains a part of a program in a separate file (module name matches the file name).
- In order to access a part of a program that resides in another file you must 'import' it.[1]
- Example:

File: functions.py

```
def fun ():
    print("I'm fun!")
```

File: driver.py

```
import functions

def main():
  functions.fun()

main()
```

1 Import syntax:

```
From <file name> import <function names>    # Import some functions
From <file name> import *                   # Import all functions
OR
import <file name>                          # Import only module/file
```

Function Modules: Complete Example

- Subdirectory name with all the files for this example: `modules1`
  - Run the program method type: "`python driver.py`"

```
<< In module file1.py >>
def fun1():
  print("I'm fun1!")

def fun2():
  print("I'm fun2!")


<< In module file2.py >>
def fun3():
    print("I'm fun3!")
```

James Tam

Modules: Complete Example (2)

```
<< In file driver.py >>
from file1 import fun1, fun2 #Import file name, function name
import file2                 #Imports only file name

def start():
  fun1()
  fun2()
  file2.fun3()

main ()
```

$\left.\begin{matrix}\texttt{fun1()}\\\texttt{fun2()}\\\texttt{file2.fun3()}\end{matrix}\right\}$ **Note the difference in how fun1 & fun2 vs. fun3 are called**

James Tam

Modules And Classes

- Class definitions are frequently contained in their own module.
- A common convention is to have the module (file) name match the name of the class.

Filename: Person.py

```
class Person:
     def fun1(self):
        print("fun1")

     def fun2 (self):
        print("fun2")
```

- To use the code of class Person from another file module you must include an import:

```
from <filename> import <class name>
from Person import Person
```

James Tam

Modules And Classes: Complete Example

- Subdirectory name with all the files for this example: `modules2`
  - To run the program type: "`python Driver.py`"

<< File Driver.py >>

```
from Greetings import *

def start():
   aGreeting = Greeting()
   aGreeting.sayGreeting()

start()
```

When importing modules containing class definitions the syntax is (star '*' imports everything):

From <*filename*> import <*classes to be used in this module*>

James Tam

Modules And Classes: Complete Example (2)

```
<< File Greetings.py >>
class Greetings:
    def sayGreeting(self):
        print("Hello! Hallo! Sup?! Guten tag/morgen/aben! Buenos! Wei! \
               Konichiwa! Shalom! Bonjour! Salaam alikum! Kamostaka?")
```

James Tam

Calling A Classes' Method Inside Another Method Of The Same Class

- Similar to how attributes must be preceded by the keyword '`self`' before they can be accessed so must the classes' methods:
- **Example**:

```
class Bar:
     x = 1
     def fun1(self):
         print(self.x)  # Accessing attribute 'x'

    def fun2(self):
        self.fun1()     # Calling method 'fun1'
```

James Tam

Naming The **Starting Module**

- Recall: The function that starts a program (first one called) should have a good self-explanatory name e.g., "`start()`" or follow common convention e.g., "`main()`"
- Similarly the file module that contains the '`start()`' or '`main()`' function should be given an appropriate name e.g., "`Driver.py`" (it's the 'driver' of the program or the starting point)

Filename: "Driver.py"

```
def start():
    #Instructions

start()
```

James Tam

Complete Example: **Accessing Attributes** And **Methods**: `Person` Module

- Subdirectory name with all the files for this example: `modules3`
  - To start the program run the '`start`' method (type: "`python Driver.py`" because '`start()`' resides in the '`Driver`' module.

```
<< Person.py >>
class Person:
    name = "Not named yet"
    age = 0

    def __init__(self,newName,newAge):
        self.name = newName
        self.age = newAge
```

James Tam

Complete Example: Accessing **Attributes** And **Methods**: Person Module (2)

```
def haveBirthday(self):
    print("Happy Birthday!")
    self.mature()

def mature(self):
    self.age = self.age + 1
```

James Tam

Complete Example: Accessing **Attributes** And **Methods**: The "Driver" Module

```
<< Driver.py >>
from Person import Person

def main():
    aPerson = Person("Cartman",8)
    print("%s is %d." %(aPerson.name,aPerson.age))
    aPerson.haveBirthday()
    print("%s is %d." %(aPerson.name,aPerson.age))

main()
```

```
def __init__(self,newName,newAge):
    self.name = newName
    self.age = newAge
```

```
Cartman is 8.
```

```
Cartman is 9.
```

```
Happy Birthday!
```

```
def haveBirthday(self)
    print("Happy Birthday!")
    self.mature()
```

```
def mature(self):
    self.age = self.age + 1
```

James Tam

After This Section You Should Now Know

- How to define an arbitrary composite type using a class
- What are the benefits of defining a composite type by using a class definition over using a list
- How to create instances of a class (instantiate)
- How to access and change the attributes (fields) of a class
- How to define methods/call methods of a class
- What is the ‘`self`’ parameter and why is it needed
- What is a constructor (`__init__` in Python), when it is used and why is it used
- How to write a method with default parameters
- How to divide your program into different modules

Copyright Notification