

CPSC 231: Classes and Objects

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

Composites

- What you have seen
 - Lists
 - Strings
 - Tuples
- What if we need to store information about an entity with multiple attributes and those attributes need to be labeled?
 - Example: Client attributes = name, address, phone, email

James Tam

Some Drawbacks Of Using A List

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

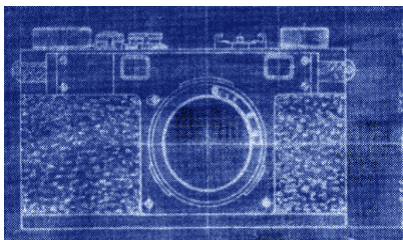
```
client = ["xxxxxxxxxxxxxxxx",
         "0000000000",
         "xxxxxxxx",
         0]
```

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.

New Term: Class

- 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.



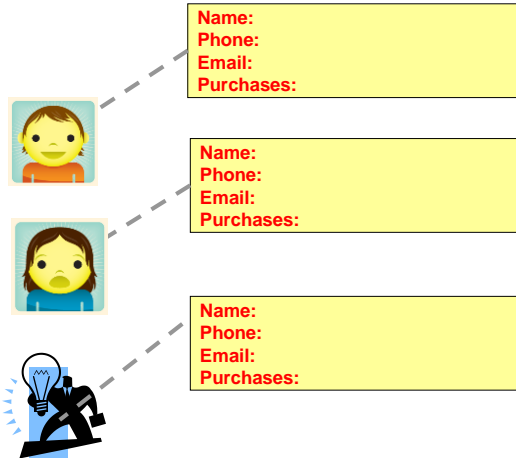
Copyright information unknown

James Tam

New term:
Attribute

Classes Define A Composite Type

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



James Tam

Defining A Class¹

- Format:**

```
class <Name of the class>:
    def __init__(self):
        self.name of first field = <default value>
        self.name of second field = <default value>
```

Note the convention: The first letter is capitalized.

- Example:**

```
class Client:
    def __init__(self):
        self.name = "default"
        self.phone = "(123)456-7890"
```

Describes what information that would be tracked by a “Client” but doesn’t yet create a client variable

Defining a ‘client’ by using a list (yuck!)

```
client = ["xxxxxxxxxxxxxxxx",
          "0000000000",
          "xxxxxxxx",
          0]
```

¹ Although capitalization of the class name isn’t the Python standard it is the standard with many other programming

New terms:

- Instance
- Object

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.

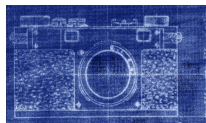


Image copyright unknown

```
Example:
class Client:
    def __init__(self):
        self.name = "default"
        self.phone = "(123)456-7890
```

- **Creating an object**

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



```
Example:
firstClient = Client()
```

Accessing And Changing The Attributes - Outside Class E.g. Start()

- **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: 1client.py

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

No spaces here



The Client List Example Implemented Using Classes (2)

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

start()
```



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

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

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

Important Details

- Accessing attributes **inside** the body of the class

```
class Client:
    def __init__(self):
        self.name = "default"
```

```
self.<attribute name>
```

(More on the 'self' keyword later in this section)

```
<Ref name> = <Class name>()
```

- Accessing attributes **outside** the body of the class (e.g. start() function)

– Need to create a reference to the object `firstClient`

```
firstClient = Client()
```

```
<Ref name>.<attribute name>
```

– Then access the object through that reference

```
firstClient.name = "James Tam"
```

James Tam

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.
 - This provides an error prevention mechanism

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

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

New terms:

- `__init__()`
- Constructor

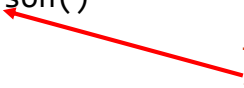
Revisiting A Previous Example: `__init__()`

- `__init__()` is used to initializing the attributes
- Classes have a special function (actually 'method' – more on this later in this section) called a **constructor** 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 e.g. `bob = Person()`

- **Format:**

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

This calls the
`init()`
constructor



- **Example:**

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

James Tam

New term:
initialize

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 can be set starting (initial) values in the init() method.

```
class Person
    def __init__(self, aName, anAge):
        self.name = aName
        self.age = anAge
```

James Tam

Full Example: Using The “Init()” Method

- The name of the online example: 2init_method.py

```
class Person:
    def __init__(self, aName, anAge):
        self.name = aName
        self.age = anAge

def start():
    aPerson = Person("Finder Wyvernspur",1000)
    print(aPerson.name,aPerson.age)

start()
```

`[tamj@csx3 classes]$ python 2init_method.py`
Finder Wyvernspur 1000

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

James Tam

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

ATTRIBUTES

Name:
Phone:
Email:
Purchases:

BEHAVIORS

Open account
Buy investments
Sell investments
Close account



A client

Image of James courtesy of James Tam

New Term: 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 composite¹.
 - E.g.,
 - `aList = []`
 - `aList.append(0)`

List reference → `aList`
Method operating on the list → `.append(0)`
- 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

New term: class method

Defining Class Methods

Format:

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

Example:

```
class Person:
    def __init__(self):
        self.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)

Reminder: 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: 3personV1.py

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

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

start()
```

James Tam

New term: The 'Self' parameter

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.
 - "Self" – inside a method it points to the object *itself* whose method was called.

- Example:

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

```
def sayName(): # No self: problem
    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: 4personV2.py

```
class Person:
    def __init__(self):
        self.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???!!!" My name is... Lisa Simpson, pleased to meet you.

    lisa.sayName()
    bart.sayName()
"The Simpsons" © Fox My name is... I'm Bart Simpson, who the hek are you???!!!
```

Important 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:
    def __init__(self):
        self.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

Leaving Out 'Self': Attributes

- Example program: 5notUsingSelf.py
- Leaving out the keyword accesses a local variable

- **Using functions:**

```
def fun():
    num = 12 # Num is a local
```

- **Using methods:**

```
class Person:
    def __init__(self):
        self.num = 1 # Attribute

    def aMethod(self):
        num = 2 # Local

# Key part of Start function
aPerson = Person();
print(aPerson.num)
aPerson.aMethod()
print(aPerson.num)
```



James Tam

Attributes Vs. Locals

- Locals scope = body of function
- Attributes: one attributes exist *for each* object created

```
class Person:
    def __init__(self):
        self.age = 1 # Attribute

person1 = Person()
person2 = Person()
```

- Each person object has it's own 'age' attribute.
- There's two 'age' attributes for the above example

James Tam

Leaving Out 'Self': Methods

- (This example employs **terrible style** and is only used to show what happens if 'self' is excluded from a method call)

Defining a function with same name as a method, confusing!

```
def method1(): # Function: outside of class
    print("Calling function called method1")
```

```
class Person:
```

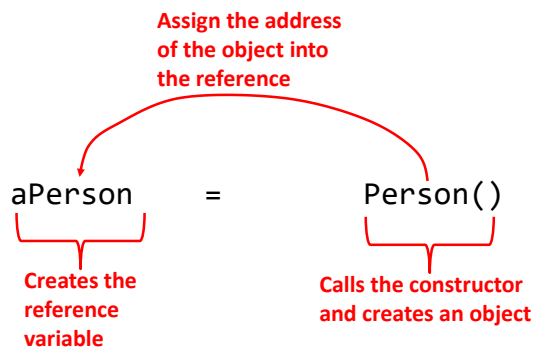
```
    def method1(self): # Method: inside of class
        print("Calling method1")
```

```
# Exclude 'self' calls the function not a method
```

```
def method2(self):
    method1()
```

James Tam

Objects Employ References



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: 6objectReference.py

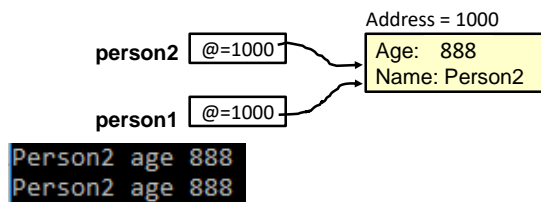
```
class Person:
    def __init__(self, newAge, newName):
        self.age = newAge
        self.name = newName
```

James Tam

Objects Employ References (3)

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

```
def start():
    person1 = Person(13, "Person2")
    person2 = person1
    person2.age = 888
    displayAge(person1)
    displayAge(person2)
    print()
```



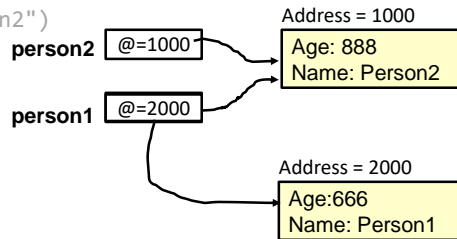
```
start()
```

James Tam

Objects Employ References (2)

```
def start():
```

```
    person1 = Person(13, "Person2")
    person2 = person1
    person2.age = 888
    displayAge(person1)
    displayAge(person2)
    print()
```



```
    person1 = Person(666, "Person1")
    displayAge(person1)
    displayAge(person2)
```

```
start()
```

```
Person1 age 666
Person2 age 888
```

James Tam

Modules: Dividing Up A Large Program

- **New term, 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 start():
    functions.fun()

start()
```

1 Import syntax:

```
From <file name> import <fun1>, <fun2>...
```

```
From <file name> import *
```

OR

```
import <file name>
```

```
# Import some functions
```

```
# Import all functions
```

```
# Import only module/file (not functions)
```


Function Modules: Complete Example

- Subdirectory name with all the files for this example: `modules1` (contains `driver.py`, `file1.py`, `file2.py`)
 - 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 names
import file2                #Imports only file name
```

```
def start():
```

```
    fun1()
```

```
    fun2()
```

```
    file2.fun3()
```

} Note the difference in how
fun1 & fun2 vs. fun3 are called

```
main ()
```

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: **PersonFile.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 PersonFile import Person
```

James Tam

Modules And Classes: Complete Example

- Subdirectory name with all the files for this example: **modules2** (contains **Driver.py** and **Greetings.py**)
 - 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” in the directory where Driver.py resides).

```
<< Person.py >>
class Person:
    def __init__(self,newName,newAge):
        self.name = newName
        self.age = newAge
```

This is a good summary of example of:

- Defining a class with attributes and methods (including `__init__()`)
- Creating an object
- Accessing attributes inside of a class
- Accessing methods inside of a class
- Accessing attributes & methods outside of class
- Splitting a program into modules

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))
```

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

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

```
main()
```

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

```
Cartman is 8.
```

```
Happy Birthday!
```

```
Cartman is 9.
```

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 divide your program into different modules

Copyright Notification

- “Unless otherwise indicated, all images in this presentation are used with permission from Microsoft.”