

Classes and Objects

You will learn how to define new types of variables.

James Tam

Composite Types: Review

- Ones that you should be familiar with now:
 - Strings
 - Lists
 - Tuples
 - Dictionaries
- Lists can be used to track relatively simple information e.g., grades, text-based virtual worlds.
- It is less effective at storing more complex information (e.g., client list) – as you will see.
- Previous example: tracking client information

```
firstClient = ["James Tam"  
              "(403)210-9455",  
              "tamj@cpssc.ucalgary.ca",  
              0]
```

```
secondClient = ["Peter Griffin"  
                "(708)123-4567",  
                "griffinp@familyguy.com",  
                100]
```

James Tam

Composite Types: Review (2)

- If a large number of composite types need to be tracked (e.g., many clients) then you can employ lists of lists.
- (This means that each list element consists of another list).

James Tam

Example: List Of Lists

- You can find the full online example under:
/home/231/examples/classes/list_of_lists.py

```
MAX = 4
```

```
def initialize (myClients):  
    for i in range (0, MAX, 1):  
        temp = [(i+1),  
                "default name",  
                "(111)111-1111",  
                "foo@bar.com",  
                0]  
        myClients.append(temp)
```

James Tam

Example: Lists Of Lists (2)

```
def display (myClients):  
    for i in range (0, MAX, 1):  
        print myClients[i]
```

MAIN

```
def main ():  
    myClients = []  
    initialize (myClients)  
    display(myClients)
```

```
main ()
```

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.

```
temp = [(i+1),
```

```
    "default name",
```

```
    "(111)111-1111",
```

```
    "foo@bar.com",
```

```
    0]
```

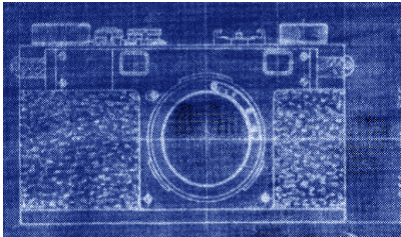
What is this?

- 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 to be entered in the phone number field.

James Tam

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 or example of this new composite type would consist of but it doesn't create an instance.



James Tam

Defining A Class

•Format:

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

Note the convention: The first letter is capitalized.

•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 in memory

James Tam

Creating An Instance Of A Class

- Format:**

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

- Example:**

firstClient = Client ()

James Tam

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.

- Creating a class**

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



James Tam

Accessing And Changing The Fields

- Format:**

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

- Example:**

`aClient.name = "James"`

James Tam

The Client List Example Implemented Using Classes

- You can find the full online example under:
</home/231/examples/classes/client.py>

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

James Tam

The Client List Example Implemented Using Classes (2)

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

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 biological entity in a simulation
 - A 'critter' (e.g., monster, computer-controlled player) a video game
 - An 'object' (e.g., sword, ray gun, food) in a video game
 - Etc.

James Tam

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
```

James Tam

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 } There is no field by  
this name
```

James Tam

Class Methods

- Somewhat similar to the other composite types, classes can have functions associated with them.
 - E.g.,

```
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*.

James Tam

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
```

James Tam

Defining Class Methods: Full Example

- The full online example can be found in UNIX under:
`/home/231/examples/classes/person.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 ()
```

James Tam

What Is The 'Self' Parameter

- When defining/call methods of a class there is always at least one parameter.
- This parameter is called the 'self' reference which allows an object to access it's attributes inside it's methods.
- It's 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

- The full online example can be found under:
</home/231/examples/classes/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"
    bart = Person ()
    bart.name = "I'm Bart Simpson, who the h*ck are you???!!!"

    lisa.sayName ()
    bart.sayName ()

main ()
```

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:**

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

No spaces here

- Example:**

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

James Tam

Full Example: Using The “Init” Method

- The full online example can be found under:
/home/231/examples/classes/init_method.py

```
class Person:
    name = ""
    def __init__(self):
        self.name = "I am the nameless bard"

def main():
    finder = Person()
    print finder.name

main()
```

James Tam

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 objects as it’s being created).
- This method never returns any values.

James Tam

Default Parameters

- Methods such as '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"):
```

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

- The full online example can be found under:
/home/231/examples/classes/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 ()
```

James Tam

Modules: Dividing Up A Large Program

- In Python a module contains a part of a program in a separate file.
- In order to access a part of a program that resides in another file you must 'import' it.
- Example:

File: fun.py

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

File: main.py

```
from fun import *1  
  
def main ():  
    fun ()  
  
main ()
```

¹ Import syntax:

From <filename> import <function names>

James Tam

Modules: Complete Example

- The complete example can be found under:
/home/231/examples/classes/modules1
- Extract both files into the same folder/directory and run the 'main' method (type: "python main.py")

```
<< In file main.py >>  
from fun import fun1, fun2  
  
def main ():  
    fun1 ()  
    fun2 ()  
  
main ()
```

James Tam

Modules: Complete Example (2)

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

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

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

```
def Person:
    pass
```

James Tam

Modules And Classes: Complete Example

- The full online example can be found under:
/home/231/examples/classes/modules2
- Extract both files into the same folder/directory and run the ‘main’ method which is in the file called “driver.py” (type: “python driver.py”)

```
<< File driver.py >>
from Foo import *

def main ():
    aFoo = Foo ()
    aFoo.hello ()

main ()
```

When importing modules containing class definitions the syntax is:

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

James Tam

Modules And Classes: Complete Example (2)

```
<< File foo.py >>
class Foo:
    def hello (self):
        print "Hello! Sup?! Guten tag/morgen/aben! Buenos! Wei! Ohio! \
        Shalom! Bonjour! Salaam alikum! Kamostaka!"
```

James Tam

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 a 'self' parameter and why is it needed
- What is a constructor, 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