Advanced Composite Types

You will learn in this section of notes how to create single and generic instances of non-homogeneous composite types.

What You Know

- How to create composite types (that are composed of other types e.g., integers, real numbers, strings) which are homogeneous.
  - Python implementation of this composite type: List
  - Typical implementation of this composite type in other programming languages (e.g., ‘C’, “C++”, “Pascal”, “Java”): Array

What You Will Learn

- How to create composite types that aren't strictly homogeneous (elements are all the same).

The List Revisited

- This type of list that you have seen before is referred to as an array:
  - Each element stores the same type of information.
  - (Usually) the size of each element is the same.
  - Examples:
    - —percentages = [0.0, 0.0, 0.0, 0.0, 0.0]
    - —letters = [‘A’, ‘A’, ‘A’]
    - —names = [“James Tam”, “Stacey Walls”, “Jamie Smyth”]

The List Revisited (2)

- Problem: What if different types of information needs to be tracked as a composite type?

Example, storing information about a client:

• First name	…series of characters
• Last name	…series of characters
• Phone number	…numerical or character
• Address	…series of characters
• Postal code	…series of characters
• Email address	…series of characters
• Total purchases made	…numerical or character

The List Revisited (3)

- The array type employed by other programming languages won’t work (each element must store the same type of information)
- The list implementation used in Python provides more features that a typical array.
- If just a few clients need to be tracked then a list can be employed:

```
firstClient = ["James",
               "Tam",
               "(403)210-9455",
               "ICT 707, 2500 University Dr NW",
               "T2N-1N4",
               "tamj@cpsc.ucalgary.ca",
               0]
```

The List Revisited (4)

- (Or as a small example)

```
def display (firstClient):
print "DISPLAYING CLIENT INFORMATION"
    print "-----------------------------"
    for i in range (0, 6, 1):
        print firstClient [i]

# MAIN
firstClient = ["James",
              "Tam",
              "(403)210-9455",
              "ICT 707, 2500 University Dr NW",
              "T2N-1N4",
              "tamj@cpsc.ucalgary.ca",
              0]
display (firstClient)
```

The List Revisited (5)

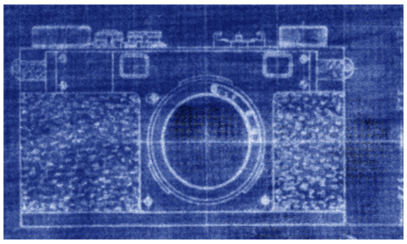
- If only a few instances of the composite type (e.g., “Clients”) need to be created then a list can be employed.

```
firstClient = ["James",
                     "Tam",
                     "(403)210-9455",
                     "ICT 707, 2500 University Dr NW",
                     "T2N-1N4",
                     "tamj@cpsc.ucalgary.ca",
                     0]

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

Classes

- Can be used define a generic template for a new non-homogeneous composite type.
- This template defines what an instance or example of this new composite type would consist of but it doesn't create an instance.



Defining A Class

- **Format:**

  class <*Name of the class*>:

  name of first field = <*default value*>

  name of second field = <*default value*>

- **Example:**

```
class Client:
    firstName = "default"
    lastName = "default"
    phone = "(123)456-7890"
    address = "default address"
    postalCode = "XXX-XXX"
    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

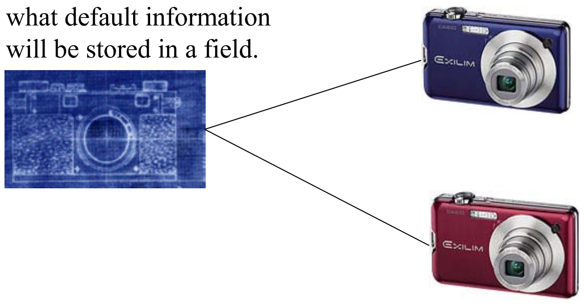
Creating An Instance Of A Class

- **Format:**
  <variable 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.
- Creating a class
  - Examples of (instantiations) of that class which can take on different forms.



Accessing And Changing The Fields

•**Format:**

<variable name>.<field name>

•**Example:**

The full version can be found in UNIX under /home/courses/217/examples/composites/client.py

```
firstClient = Client ()
firstClient.firstName = "James"
firstClient.lastName = "Tam"
firstClient.email = "tamj@cpsc.ucalgary.ca"
print firstClient.firstName
print firstClient.lastName
print firstClient.phone
print firstClient.address
print firstClient.postalCode
print firstClient.email
print firstClient.purchases
```

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’ a video game
  - An ‘object’ in a video game
  - Etc.

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:
    firstName = "default"
    lastName = "default"
    phone = "(123)456-7890"
    address = "default address"
    postalCode = "XXX-XXX"
    email = "foo@bar.com"
    purchases = 0

firstClient = Client ()
print firstClient.middleName
```

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:
    firstName = "default"
    lastName = "default"
    phone = "(123)456-7890"
    address = "default address"
    postalCode = "XXX-XXX"
    email = "foo@bar.com"
    purchases = 0

firstClient = Client ()
```

~~print firstClient.middleName~~ } **There is no field by this name**

You Should Now Know

- How a list can be used to store different types of information (non-homogeneous composite type)
- 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 or fields of a class