

Classes and OOP Concepts

Difference between Object-Oriented and Procedural Oriented Programming

Object-Oriented Programming (OOP)	Procedural-Oriented Programming (Pop)
It is a bottom-up approach	It is a top-down approach
Program is divided into objects	Program is divided into functions
Makes use of Access modifiers 'public', private', protected'	Doesn't use Access modifiers
It is more secure	It is less secure
Object can move freely within member functions	Data can move freely from function to function within programs
It supports inheritance	It does not support inheritance

Some Drawbacks Of Using A List

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

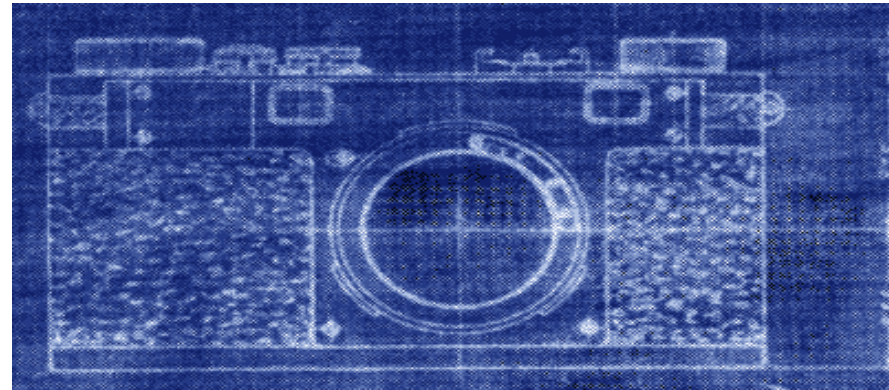
```
client = ["xxxxxxxxxxxxxxxx",  
         "0000000000",  
         "xxxxxxxx",  
         0]
```

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.

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

Classes Define A Composite Type

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



Name:
Phone:
Email:
Purchases:



Name:
Phone:
Email:
Purchases:



Name:
Phone:
Email:
Purchases:

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 variable

Contrast this with a list definition of a client

```
client = ["xxxxxxxxxxxxxxxx",  
          "000000000000",  
          "xxxxxxxx",  
          0]
```

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

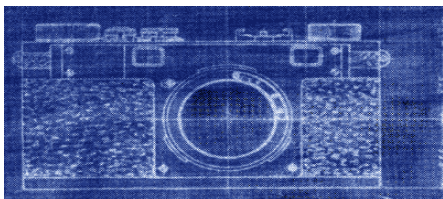


Image copyright unknown



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
 - A bacteria or virus in a medical simulation
 - 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:

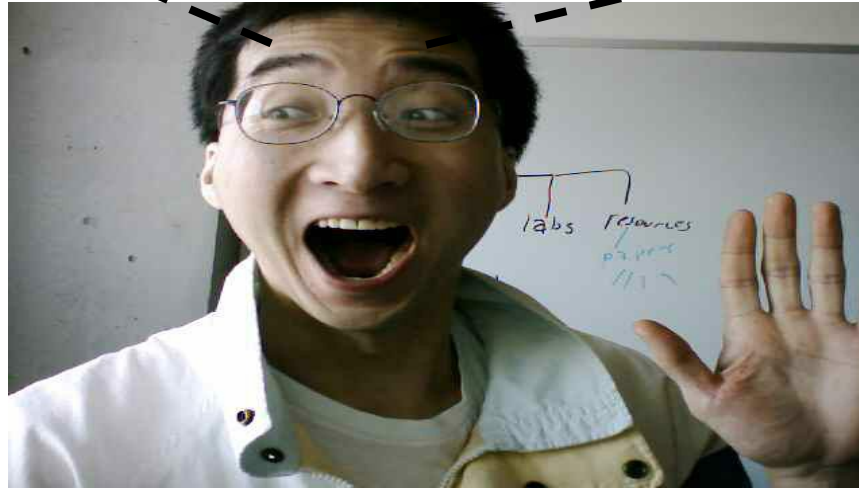
BEHAVIORS

Open account

Buy investments

Sell investments

Close account

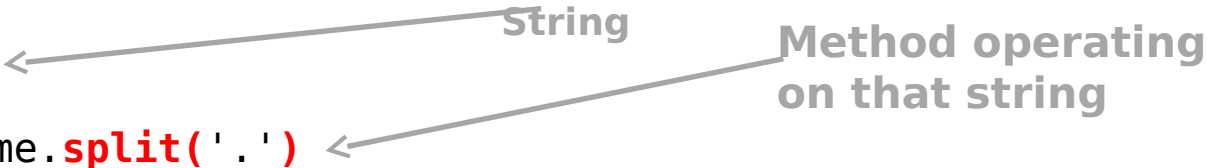


Class Methods (“Behaviors”)

- **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.,

```
filename = "foo.txt"
```

```
name, suffix = filename.split('.)
```



String

Method operating on that string
- 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*.

Defining Class Methods

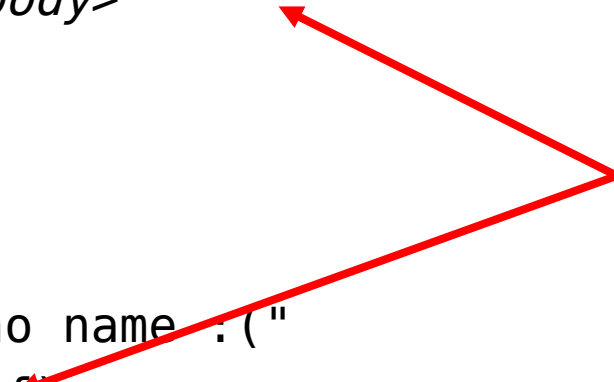
Format:

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

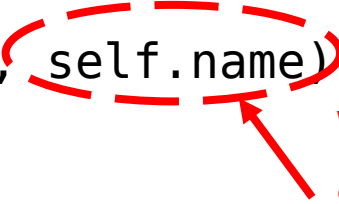
Example:

```
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"



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

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)

The Self Parameter Example

```
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???!!!
```

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."
```

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"
```

This design approach is consistent with many languages

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)
```

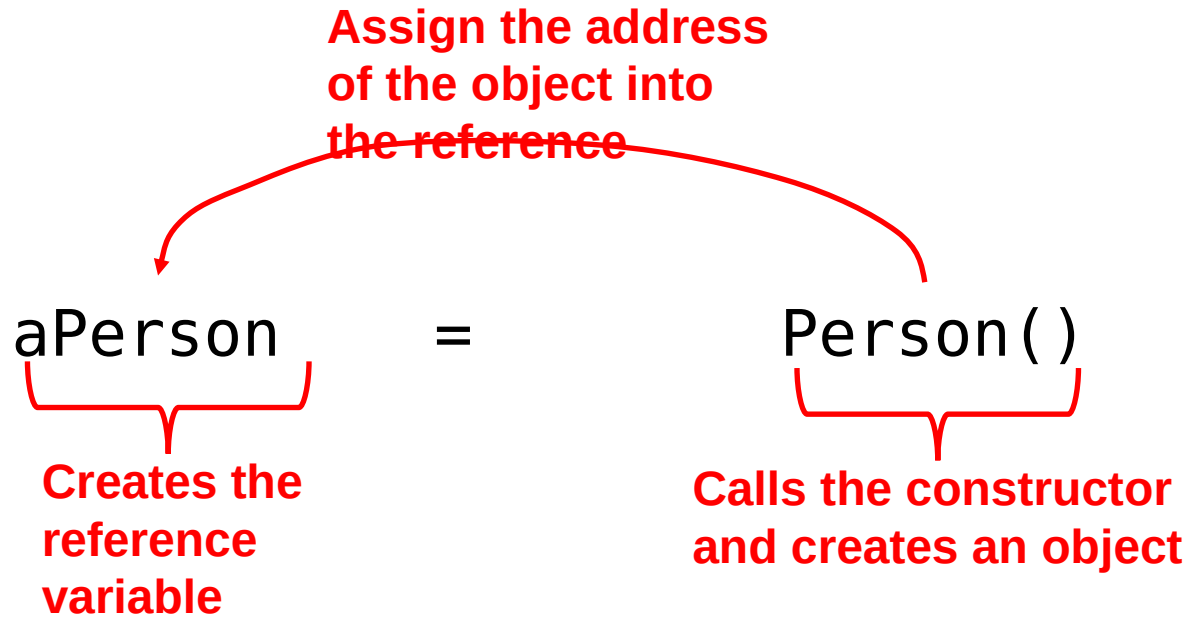
```
[cse classes 133 ]> python init_method1.py  
Finder Wyvernspur
```

```
main()
```

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)`

Objects Employ References



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))
```

Objects Employ References (3)

```
def start():
```

```
    person1 = Person(13,"Person2")
```

```
    person2 = person1
```

```
    displayAge(person1)
```

```
    displayAge(person2)
```

```
    print()
```

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

person2

@=1000

person1

@=1000

Address = 1000

Age: 13

Name: Person2

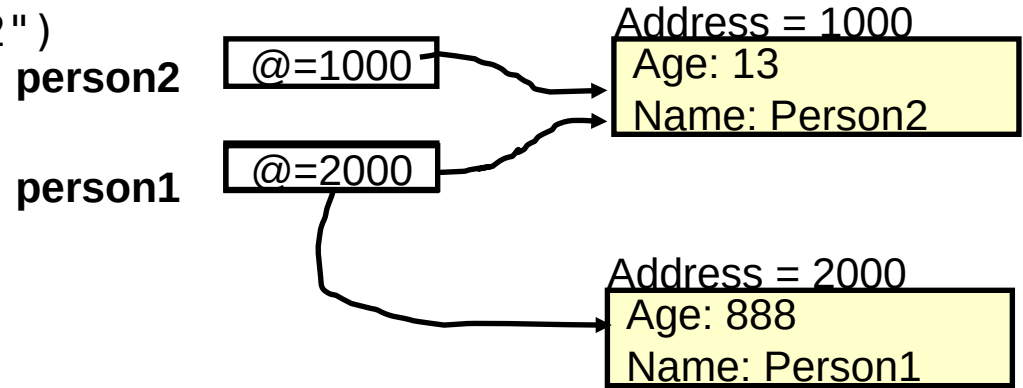
start()

Objects Employ References (2)

```
def start():  
    person1 = Person(13, "Person2")  
    person2 = person1  
    displayAge(person1)  
    displayAge(person2)  
    print()  
  
    person1 = Person(888, "Person1")  
    displayAge(person1)  
    displayAge(person2)
```

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

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



```
start()
```

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"):
```



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")
```

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
```