كلية معتمدة من الهيئة القومية لضمان جودة التعليم والاعتماد

Assiut University

Course Title: Data Structures and Algorithms
Course Code: CS211

Prof. Dr. Khaled F. Hussain

Reference

- Michael T. Goodrich, Roberto Tamassia, and Michael H. Goldwasser, “Data Structures and Algorithms in Python”

Object-Oriented Design Goals

- Software implementations should achieve ***robustness***, ***adaptability***, and ***reusability***.
  - **Robustness**: a program produces the right output for all the anticipated inputs in the program's application. In addition, it is capable of handling unexpected inputs that are not explicitly defined for its application.
  - **Adaptability**: Software, therefore, needs to be able to evolve over time in response to changing conditions in its environment.
  - **Reusability**: same code should be usable as a component of different systems in various applications.

Class Definitions

- A class serves as a blueprint for its instances.
- **self** serves to identify the particular instance upon which a member is invoked.

```
class CreditCard:
    def __init__(self, customer, bank, acnt, limit):
        self._customer = customer
        self._bank = bank
        self._account = acnt
        self._limit = limit
        self._balance = 0

    def get_customer(self):
        return self._customer
....
```

Class Definitions (Cont.)

```
def charge(self, price):
    if price + self._balance > self. limit: # if charge would exceed limit,
        return False # cannot accept charge
    else:
        self._balance += price
        return True

def make_payment(self, amount):
    self._balance −= amount
```

Class Definitions (Cont.)

- The Constructor

cc=CreditCard(“John Doe”, “1st Bank” , “5391 0375 9387 5309” , 1000)

- Names beginning with a single underscore are conventionally akin to protected, while names beginning with a double underscore (other than special methods) are akin to private.

- If a user were to make a call such as visa.charge(“test”), the code would presumably crash when attempting to add that parameter to the current balance. If this class were to be widely used in a library, we might use more rigorous techniques to raise a TypeError when facing such misuse.

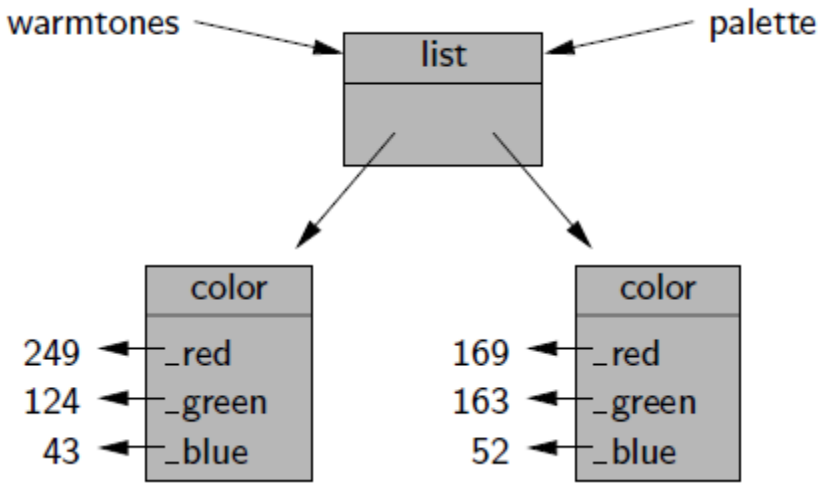
Extending the CreditCard Class

```
class PredatoryCreditCard(CreditCard):
    def __init__(self, customer, bank, acnt, limit, apr):
        super().__init__(customer, bank, acnt, limit)      # call super constructor
        self._apr = apr

    def charge(self, price):
        success = super( ).charge(price)
        if not success:
            self._balance += 5
        return success
```

Shallow and Deep Copying

- We let identifier warmtones denote an existing list of such colors (e.g., oranges, browns).
- An assignment statement palette = warmtones makes the name palette an ***alias*** for the object identified as warmtones.
- If we subsequently add or remove colors from "palette," we modify the list identified as warmtones.

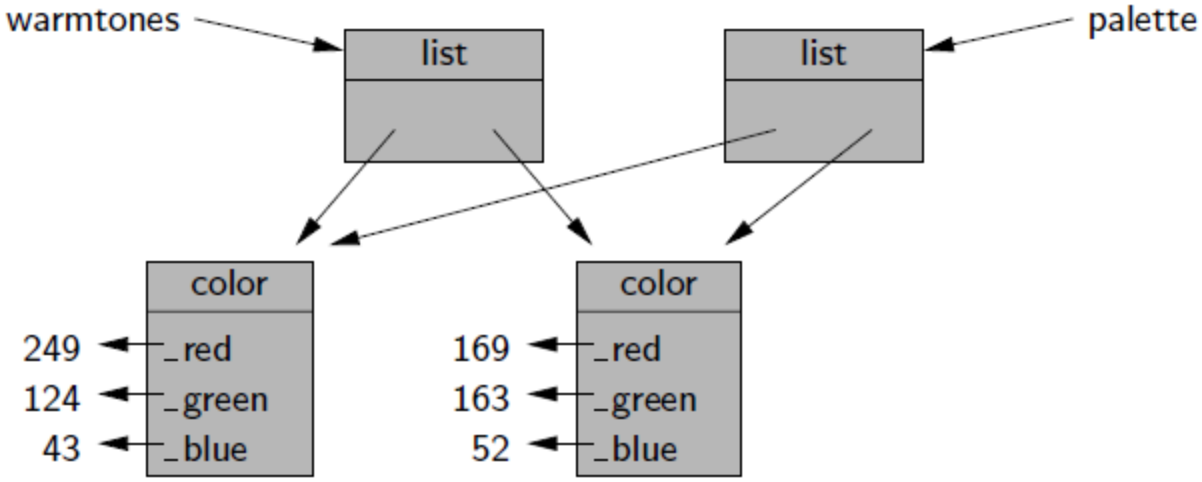


Shallow and Deep Copying (Cont.)

- We can instead create a new instance of the list class by using the syntax:

  palette = list(warmtones)

- In this case, we explicitly call the list constructor, sending the first list as a parameter. This causes a new list to be created.
- It is what is known as a ***shallow copy***.
- palette[0] and warmtones[0] as aliases for the same color instance.
- We can add or remove elements from palette without affecting warmtones.
- However, if we edit a color instance from the palette list, we effectively change the contents of warmtones.



Shallow and Deep Copying (Cont.)

- Python provides a very convenient module, named copy, that can produce both shallow copies and deep copies of arbitrary objects.

  palette = copy.deepcopy(warmtones)