Classes 4

Thomas Schwarz, SJ
Marquette University

Doc Strings

- Classes are reusable
  - No need to reinvent a working name class
  - But need to provide documentation
  - In Python:
    - This is done primarily with the so-called doc string
      - Right after the definition of a class or function
      - Included between triple quotes

Doc Strings

- The contents are made available to the help function

Example

- A simple checking account class

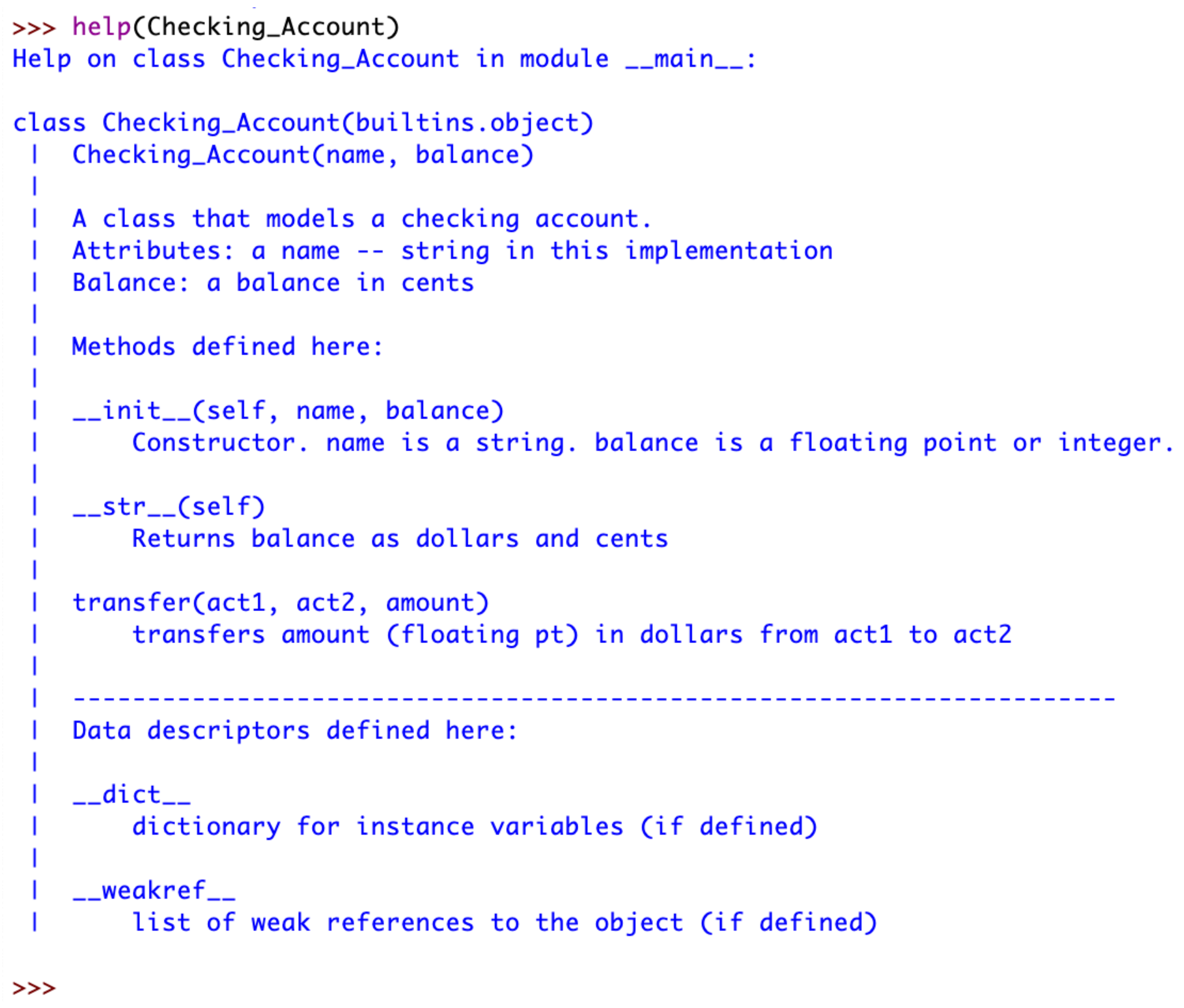
```
class Checking_Account:
    """A class that models a checking account.
        Attributes: a name -- string in this implementation
        Balance: a balance in cents
    """
    def __init__(self, name, balance):
        """Constructor. name is a string. balance is a floating point or integer."""
        self.name = name
        self.balance = round(balance*100)
    def __str__(self):
        """Returns balance as dollars and cents"""
        return "Account for {} with balance US${:d}.{:02d}".format(
            self.name,
            self.balance//100,
            self.balance%100)
    def transfer(act1, act2, amount):
        """transfers amount (floating pt) in dollars from act1 to act2"""
        amount = round(amount*100)
        act1.balance -= amount
        act2.balance += amount
```

Example

```
if __name__ == "__main__":
    a1 = Checking_Account("Thomas Schwarz", 1543.285)
    a2 = Checking_Account("Joseph Cuelho", 1009)
    print(a1)
    print(a2)
    print("Transferring")
    Checking_Account.transfer(a1, a2, 500.01)
    print(a1)
    print(a2)
```

Example

- This is the result of typing help(Checking_Account)



```
>>> help(Checking_Account)
Help on class Checking_Account in module __main__:

class Checking_Account(builtins.object)
 |  Checking_Account(name, balance)
 |
 |  A class that models a checking account.
 |  Attributes: a name -- string in this implementation
 |  Balance: a balance in cents
 |
 |  Methods defined here:
 |
 |  __init__(self, name, balance)
 |      Constructor. name is a string. balance is a floating point or integer.
 |
 |  __str__(self)
 |      Returns balance as dollars and cents
 |
 |  transfer(act1, act2, amount)
 |      transfers amount (floating pt) in dollars from act1 to act2
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)

>>>
```

Example

- As you can see, Python has automatically created a help file from the information you provided.

Tricks with Currency Amounts

- Currency is usually a decimal number with exactly two digits precision.
    - Could use the decimal - class
    - Could use third party classes
    - We build our own
- Idea: Present currency as multiples of cents.

```
class Checking_Account:
    """A class that models a checking account.
        Attributes: a name -- string in this implementation
        Balance: a balance in cents
    """
    def __init__(self, name, balance):
        """Constructor. name is a string. balance is a
           floating point or integer.
        “""
        self.name = name
        self.balance = round(balance*100)
```

Tricks with Currency Accounts

- To print out currencies, we break the cents apart into the dollars (displayed normally) and the cents amount proper.
  - The format mini-language allow us to print out amounts with leading 0.
  - Just stick a 0 in front of the width field

```
def __str__(self):
    """Returns balance as dollars and cents"""
    return "Account for {} with balance US${:d}.{:02d}".format(
        self.name,
        self.balance//100,
        self.balance%100)
```

Specify leading zero in the format mini-language

Self Test

- Modify the __str__ function so that a negative amount is written in the form
    - `-US$103.05`

Solution

- Just make a case distinction, but make sure that you do not change the field

```
    def __str__(self):
        """Returns balance as dollars and cents"""
        if self.balance >= 0:
            return "Account for {} with balance US${:d}.{:02d}".format(
                self.name,
                self.balance//100,
                self.balance%100)
        else:
            balance = -self.balance
            return "Account for {} with balance -US${:d}.{:02d}".format(
                self.name,
                balance//100,
                balance%100)
```