Chapter17_Exercises

June 6, 2017

1 Chapter 17 Exercises

```
In [2]: def print_attributes(obj):
            for attr in vars(obj):
                print(attr, getattr(obj, attr))
```

1.1 Exercise 17.1

Download the code from this chapter from http: // thinkpython2. com/ code/ Time2. py . Change the attributes of Time to be a single integer representing seconds since midnight. Then modify the methods (and the function int_to_time) to work with the new implementation. You should not have to modify the test code in main. When you are done, the output should be the same as before.

```
In [22]: # Time2.py
         """This module contains a code example related to

         Think Python, 2nd Edition
         by Allen Downey
         http://thinkpython2.com

         Copyright 2015 Allen Downey

         License: http://creativecommons.org/licenses/by/4.0/
         """

         from __future__ import print_function, division


         class Time:
             """Represents the time of day.

             attribute: second
             """
             def __init__(self, hours=0, minutes=0, second=0):
                 """Initializes a time object.
```

```
        second: int or float
        """
        self.seconds = second + minutes*60 + hours*3600

    def __str__(self):
        """Returns a string representation of the time."""
        hours = self.seconds // 3600
        minutes = (self.seconds - hours*3600) // 60
        seconds = self.seconds - hours*3600 - minutes*60
        return '%.2d:%.2d:%.2d' % (hours, minutes, seconds)

    def print_time(self):
        """Prints a string representation of the time."""
        print(str(self))

    def time_to_int(self):
        """Computes the number of seconds since midnight."""
        return self.seconds

    def is_after(self, other):
        """Returns True if t1 is after t2; false otherwise."""
        return self.time_to_int() > other.time_to_int()

    def __add__(self, other):
        """Adds two Time objects or a Time object and a number.

        other: Time object or number of seconds
        """
        if isinstance(other, Time):
            return self.add_time(other)
        else:
            return self.increment(other)

    def __radd__(self, other):
        """Adds two Time objects or a Time object and a number."""
        return self.__add__(other)

    def add_time(self, other):
        """Adds two time objects."""
        assert self.is_valid() and other.is_valid()
        seconds = self.time_to_int() + other.time_to_int()
        return int_to_time(seconds)

    def increment(self, seconds):
        """Returns a new Time that is the sum of this time and seconds.""
        seconds += self.time_to_int()
        return int_to_time(seconds)
    def is_valid(self):
        """Checks whether a Time object satisfies the invariants."""
        if self.seconds < 0:
            return False
        return True

def int_to_time(seconds):
    """Makes a new Time object.

    seconds: int seconds since midnight.
    """
    return Time(0, 0, seconds)


def main():
    start = Time(9, 45, 00)
    start.print_time()

    end = start.increment(1337)
    #end = start.increment(1337, 460)
    end.print_time()

    print('Is end after start?')
    print(end.is_after(start))

    print('Using __str__')
    print(start, end)

    start = Time(9, 45)
    duration = Time(1, 35)
    print(start + duration)
    print(start + 1337)
    print(1337 + start)

    print('Example of polymorphism')
    t1 = Time(7, 43)
    t2 = Time(7, 41)
    t3 = Time(7, 37)
    total = sum([t1, t2, t3])
    print(total)


if __name__ == '__main__':
    main()
```

```
09:45:00
10:07:17
```

```
Is end after start?
True
Using __str__
09:45:00 10:07:17
11:20:00
10:07:17
10:07:17
Example of polymorphism
23:01:00
```

1.1.1 Expected output

09:45:00 10:07:17 Is end after start? True Using **str** 09:45:00 10:07:17 11:20:00 10:07:17 10:07:17 Example of polymorphism 23:01:00

1.2 Exercise 17.2

This exercise is a cautionary tale about one of the most common, and difficult to find, errors in Python. Write a definition for a class named Kangaroo with the following methods: 1. An **init** method that initializes an attribute named pouch_contents to an empty list. 2. A method named put_in_pouch that takes an object of any type and adds it to pouch_contents. 3. A **str** method that returns a string representation of the Kangaroo object and the contents of the pouch. Test your code by creating two Kangaroo objects, assigning them to variables named kanga and roo, and then adding roo to the contents of kanga's pouch.

```
In [37]: class Kangaroo:

             def __init__(self, pouch_contents=[]):
                 self.pouch_contents = pouch_contents

             def put_in_pouch(self, obj):
                 self.pouch_contents.append(obj)

             def __str__(self):
                 return str(self.pouch_contents)

         if __name__ == '__main__':
             kanga = Kangaroo()
             roo = Kangaroo()

             kanga.put_in_pouch(roo)
             kanga.put_in_pouch('hi')

             print(kanga)
```

```
[<__main__.Kangaroo object at 0x000001695ED3FE48>, 'hi']
```

Download http: // thinkpython2. com/ code/ BadKangaroo. py . It contains a solution to the previous problem with one big, nasty bug. Find and fix the bug.

```
In [8]: """This module contains a code example related to

        Think Python, 2nd Edition
        by Allen Downey
        http://thinkpython2.com

        Copyright 2015 Allen Downey

        License: http://creativecommons.org/licenses/by/4.0/
        """

        from __future__ import print_function, division

        """

        WARNING: this program contains a NASTY bug.  I put
        it there on purpose as a debugging exercise, but
        you DO NOT want to emulate this example!

        """

        class Kangaroo:
            """A Kangaroo is a marsupial."""

            def __init__(self, name, contents=None):
                """Initialize the pouch contents.

                name: string
                contents: initial pouch contents.
                """
                self.name = name
                if contents == None:
                    contents = []
                self.pouch_contents = contents

            def __str__(self):
                """Return a string representaion of this Kangaroo.
                """
                t = [ self.name + ' has pouch contents:' ]
                for obj in self.pouch_contents:
                    if isinstance(obj, Kangaroo):
                        s = '    ' + object.__str__(obj.name)
                    else:
                        s = '    ' + object.__str__(obj)
                    t.append(s)
                return '\n'.join(t)

            def put_in_pouch(self, item):
                """Adds a new item to the pouch contents.

                item: object to be added
                """
                self.pouch_contents.append(item)


        kanga = Kangaroo('Kanga')
        roo = Kangaroo('Roo')
        kanga.put_in_pouch('wallet')
        kanga.put_in_pouch('car keys')
        kanga.put_in_pouch(roo)

        print(kanga)
        print(roo)

        # If you run this program as is, it seems to work.
        # To see the problem, trying printing roo.

        # Hint: to find the problem try running pylint.
```

```
Kanga has pouch contents:
    'wallet'
    'car keys'
    'Roo'
Roo has pouch contents:
```

1.2.1 Original output:

Kanga has pouch contents: 'wallet' 'car keys' 'Roo' Roo has pouch contents: 'wallet' 'car keys' 'Roo'

Mutable default values in initialization means all instances refer to the same object