Think Python

Exercise 17.1.

Download the code from this chapter from https: // thinkpython. 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.

```
class Time:
    """Represents the time of day in seconds since midnight."""

    def __init__(self, hour=0, minute=0, second=0):
        """Initialises a time object."""
        self.seconds = hour * 3600 + minute * 60 + second

    def __str__(self):
        """Returns a string representation of the time."""
        minutes, second = divmod(self.seconds, 60)
        hour, minute = divmod(minutes, 60)
        return '%.2d:%.2d:%.2d' % (hour, minute, second)

    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.seconds > other.seconds

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

    def increment(self, seconds):
        """Returns a new Time that is the sum of this time and seconds."""
        seconds += self.seconds
        return int_to_time(seconds)

    def is_valid(self):
        """Checks whether a Time object satisfies the invariants."""
        return self.seconds >= 0 and self.seconds < 24 * 60 * 60
```

```
def int_to_time(seconds):
    """Makes a new Time object."""
    return Time(0, 0, seconds)


def main():
    start = Time(9, 45, 0)
    start.print_time()

    end = start.increment(1337)
    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()
```

Exercise 17.1 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
```

Consolidated into a single seconds attribute representing the total seconds since midnight.

Accepts hour, minute, and second but calculates total seconds from these values and stores it in the seconds attribute.

Modified __add__, __radd__, add_time, increment to operate on the single seconds attribute

Test data remains the same.

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.

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

```
class Kangaroo:
  def __init__(self):
    self.pouch_contents = []

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

  def __str__(self):
    contents = ", ".join(str(item) for item in self.pouch_contents)
    return f"Kangaroo with pouch contents: [{contents}]"


if __name__ == "__main__":
  # Create two Kangaroo objects
  kanga = Kangaroo()
  roo = Kangaroo()

  # Add roo to kanga's pouch
  kanga.put_in_pouch(roo)

  # Print both objects
  print(kanga)
  print(roo)
```

Exercise 17.2 Output.

```
Kangaroo with pouch contents: [Kangaroo with pouch contents: []]

Kangaroo with pouch contents: []
```

The Problem

The problem arises when one instance (roo) is added to the pouch_contents of another instance (kanga). A reference to the roo object is being added, not a separate copy. This means kanga.pouch_contents holds a reference to roo. When printing kanga, it accesses roo's __str__ method to describe its contents.

Exercise 17.2.

Modify the __str__ method to handle nested Kangaroo objects.

```
class Kangaroo:
   def __init__(self):
        self.pouch_contents = []

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

   def __str__(self):
        # Handle nested Kangaroo objects
        contents = ", ".join(
              f"Kangaroo with pouch contents: [{len(item.pouch_contents)} items]"
              if isinstance(item, Kangaroo) else str(item)
              for item in self.pouch_contents
        )
        return f"Kangaroo with pouch contents: [{contents}]"


if __name__ == "__main__":
   # Create two Kangaroo objects
   kanga = Kangaroo()
   roo = Kangaroo()

   # Add roo to kanga's pouch
   kanga.put_in_pouch(roo)

   # Print both objects
   print(kanga)
   print(roo)
```

Exercise 17.2 Output.

```
Kangaroo with pouch contents: [Kangaroo with pouch contents: [0 items]]

Kangaroo with pouch contents: []
```

Exercise 17.2.

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

Original	Modified

`def __init__(self, name, contents=[]):`
`  """Initialize the pouch contents.`
 	
`  name: string`
`  contents: initial pouch contents.`
`  """`
`  self.name = name`
`  self.pouch_contents = contents`
	`def __init__(self, name, contents=None):`
	`  """Initialize the pouch contents.`
	
	`  name: string`
	`  contents: initial pouch contents.`
	`  """`
	`  if contents is None:`
	`  contents = []`
	`  self.name = name`
	`  self.pouch_contents = contents`

The Problem

When instantiating a new kagaroo with no arguments the default is used and this is used the next time producing the same bug as before.

If no contents argument is passed use contents=None in the –init-- used instead of a mutable default list.

When a new instance is created check if contents is None, if it is, create a new empty list contents = [].