Chapter 15

Solutions for Hands-On Activities

15.1 Solution for activity 1.1: Variable stars

```
# Here we define some general needed functions
def _average(data):
    return sum(data) / len(data)


def _var(data):
    prom = _average(data)
    suma = 0
    for i in data:
        suma += (i - prom) ** 2
    return suma / len(data)


class Star:
    def __init__(self, star_class, RA, DEC, id, observations=None):
        self.star_class = star_class
        self.RA = RA
        self.DEC = DEC
        self.id = id
        if observations is None:
            observations = []
        self.observations = observations

    def get_magnitudes(self):
        return [i.magnitude for i in self.observations]

    def averageBrightness(self):
        magnitudes = self.get_magnitudes()
        return _average(magnitudes)

    def varBrightness(self):
        magnitudes = self.get_magnitudes()
        return _var(magnitudes)

```

```
35      def addObservation(self, magnitude, tiempo, error):
36          self.observations.append(Observation(magnitude, tiempo, error))
37
38
39  class Observation(object):
40      def __init__(self, magnitude, tiempo, error):
41          self.magnitude = magnitude
42          self.tiempo = tiempo
43          self.error = error
44
45
46  class Field:
47      def __init__(self, stars=None):
48          if stars is None:
49              stars = []
50          self.stars = stars
51
52      def addStar(self, star):
53          self.stars.append(star)
54
55
56  class Sky:
57      def __init__(self, fields=None):
58          if fields is None:
59              fields = []
60          self.fields = fields
61
62      def addField(self, field):
63          self.fields.append(field)
64
65
66  if __name__ == '__main__':
67      sky = Sky()
68
69      e0 = Star('RRLyrae', 0, 0, 0, [Observation(2, 1000, (1, 3)),
```

```
                                    Observation(3, 1000, (2, 3)),
                                    Observation(4, 1000, (3, 5))])

    e1 = Star('Eclipsing Binaries', 20, 30, 1, [Observation(2, 1000, (1, 3)),
                                                Observation(5, 1000, (2, 7)),
                                                Observation(6, 1000, (6, 9))])

    e2 = Star('Mira', 15, 50, 2, [Observation(7, 1000, (1, 10)),
                                  Observation(8, 1000, (1, 30)),
                                  Observation(9, 1000, (7, 10))])

    field0 = Field()

    e3 = Star('Cepheids', 50, 15, 3)
    e4 = Star('Cepheids', 120, 120, 4)
    e5 = Star('Eclipsing Binaries', 0, 90, 5)

    e3.addObservation(21, 1000, (15, 30))
    e3.addObservation(22, 1000, (15, 30))
    e3.addObservation(23, 1000, (15, 30))
    e4.addObservation(24, 1000, (15, 30))
    e4.addObservation(25, 1000, (15, 30))
    e4.addObservation(26, 1000, (15, 30))
    e5.addObservation(27, 1000, (15, 30))
    e5.addObservation(28, 1000, (15, 30))
    e5.addObservation(29, 1000, (15, 30))

    field0.addStar(e3)
    field0.addStar(e4)
    field0.addStar(e5)

    sky.addField(field0)
    sky.addField(Field([e0, e1, e2]))

    print(sky.fields[0].stars[0].get_magnitudes())
```

15.2 Solution for activity 1.2: Geometric Shapes

```
from abc import ABCMeta, abstractmethod, abstractproperty
from math import pi, sqrt, cos, sin, asin


def calculate_points(trasl, radius, angles):
    points = []
    for k in angles:
        """
        Here we are assuming that aux_list has the same dimension
        than the argument trasl.
        """
        aux_list = [radius * cos(k), radius * sin(k)]
        for i in range(len(aux_list)):
            aux_list[i] += trasl[i]
        points.append(aux_list)
    return points


class Figure(metaclass=ABCMeta):
    def __init__(self, center):
        self._center = center

    @property
    def center(self):
        return self._center

    @center.setter
    def center(self, value):
        self._center = value

    @abstractproperty
    def perimeter(self):
        pass

```

```
35      @abstractproperty
36      def area(self):
37          pass
38
39      @abstractmethod
40      def grow_area(self, times):
41          pass
42
43      @abstractmethod
44      def grow_perimeter(self, amount):
45          pass
46
47      def translate(self, vector):
48          """
49          A better implementation to this method is as follows:
50          self.center = tuple(map(lambda x, y: x + y, self.center,
51                                  vector))
52
53          We refer the reader to Chapter 3 - Functional Programming
54          for more details about map and lambda functions.
55          """
56          if len(vector) == len(self.center):
57              for i in range(len(vector)):
58                  self.center[i] += vector[i]
59          else:
60              print('Wrong vector size')
61              return
62
63      def __repr__(self):
64          return '{} - Perimeter: {:.2f}, Area: {:.2f}, Center: {}' \
65              .format(type(self).__name__,
66                      self.perimeter, self.area, self.center)
67
68      # Properties useful to implement the property vertices
69      # This is one possible solution
```

```
70      @abstractproperty
71      def dist_center_vertex(self):
72          pass
73
74      @abstractproperty
75      def angles(self):
76          pass
77
78      @property
79      def vertices(self):
80          return calculate_points(
81              self.center, self.dist_center_vertex, self.angles)
82
83
84  class Rectangle(Figure):
85      def __init__(self, a, b, center):
86          super().__init__(center)
87          self._a = a
88          self._b = b
89
90      @property
91      def a(self):
92          return self._a
93
94      @a.setter
95      def a(self, value):
96          self._a = value
97
98      @property
99      def b(self):
100         return self._b
101
102     @b.setter
103     def b(self, value):
104         self._b = value
```

```
105
106     @property
107     def perimeter(self):
108         return 2 * (self._a + self._b)
109
110     @property
111     def area(self):
112         return self._a * self._b
113
114     def grow_area(self, times):
115         self._a *= sqrt(times)
116         self._b *= sqrt(times)
117
118     def grow_perimeter(self, amount):
119         self._a += self._a * amount / (2 * (self._a + self._b))
120         self._b += self._b * amount / (2 * (self._a + self._b))
121
122     # Properties useful for the vertices property
123     @property
124     def dist_center_vertex(self):
125         return sqrt(self.a ** 2 + self.b ** 2) / 2
126
127     @property
128     def angles(self):
129         angles = list()
130         angles.append(2 * asin(self.b /
131                                (2 * self.dist_center_vertex)))
132         angles.append(pi)
133         angles.append(pi + 2 * asin(self.b /
134                                     (2 * self.dist_center_vertex)))
135         angles.append(0)
136         return angles
137
138
139 class EquilateralTriangle(Figure):
```

```
140     def __init__(self, l, center):
141         super().__init__(center)
142         self._l = l
143 
144     @property
145     def l(self):
146         return self._l
147 
148     @l.setter
149     def l(self, value):
150         self._l = value
151 
152     @property
153     def perimeter(self):
154         return 3 * self._l
155 
156     @property
157     def area(self):
158         return (self._l ** 2) * sqrt(3) / 4
159 
160     def grow_area(self, times):
161         self._l *= sqrt(times)
162 
163     def grow_perimeter(self, amount):
164         self._l *= amount / 3
165 
166     # Properties useful for implementing the vertices property
167     @property
168     def dist_center_vertex(self):
169         return self.l / sqrt(3)
170 
171     @property
172     def angles(self):
173         angles = list()
174         angles.append(2 * pi / 3)
```

```
175         angles.append(4 * pi / 3)
176         angles.append(0)
177         return angles
178
179
180 # Testing the solution
181
182 if __name__ == '__main__':
183     figures = list()
184     figures.append(EquilateralTriangle(5, [0, 0]))
185     figures.append(Rectangle(6, 8, [0, 0]))
186
187     print(*figures, sep="\n")
188     print("*" * 20)
189
190     for i in figures:
191         i.grow_perimeter(0)
192
193     print(*figures, sep="\n")
194     print("*" * 20)
195
196     for i in figures:
197         i.grow_area(1)
198
199     print(*figures, sep="\n")
200     print("*" * 20)
201
202     print("Before translating")
203     for i in figures:
204         print(i.vertices)
205     print("*" * 20)
206
207     print("After translating")
208     for i in figures:
209         i.translate((2, -1))
```

```
210             print(i.vertices)
211         print("*" * 20)
212
213         print(*figures, sep="\n")
214         print("*" * 20)
```

15.3 Solution for activity 2.1: Production line of bottles

```
from collections import deque
from package import Package
from bottle import Bottle


class Machine:
    def process(self, incoming_production_line):
        print("----------------------")
        print("Machine {} started working.".format(
            self.__class__.__name__))


class BottleModulator(Machine):
    def __init__(self):
        self.bottles_to_produce = 0

    def process(self, incoming_production_line=None):
        super().process(incoming_production_line)
        production_line = deque()
        while len(production_line) != self.bottles_to_produce:
            if len(production_line) == 0:
                production_line.append(Bottle())
            elif len(production_line) % 5 == 0:
                previous_bottle = production_line[-1]
                production_line.append(
                    Bottle(liters=previous_bottle.liters * 3))
            elif len(production_line) % 6 == 0:
                previous_bottle = production_line[-1]
                bottle_before_previous = production_line[-2]
                production_line.append(
                    Bottle(liters=(previous_bottle.liters / 2 +
                                   bottle_before_previous.liters * 4)))
            else:
                production_line.append(Bottle())
```

```
35          return production_line
36
37
38  class LowFAT32(Machine):
39      def __init__(self):
40          self.discarded_bottles = []
41
42      def discard_bottle(self, bottle):
43          self.discarded_bottles.append(bottle)
44
45      def print_discarded_bottles(self):
46          print("{} bottles were discarded".format(
47              len(self.discarded_bottles)))
48
49      def process(self, incoming_production_line):
50          super().process(incoming_production_line)
51          production_line = deque()
52          while len(incoming_production_line) != 0:
53              bottle = incoming_production_line.popleft()
54              if len(production_line) == 0:
55                  production_line.append(bottle)
56              elif production_line[-1].liters <= bottle.liters:
57                  production_line.append(bottle)
58              elif production_line[0].liters >= bottle.liters:
59                  production_line.appendleft(bottle)
60              else:
61                  self.discard_bottle(bottle)
62          self.print_discarded_bottles()
63          return production_line
64
65
66  class HashSoda9001(Machine):
67      def process(self, incoming_production_line):
68          super().process(incoming_production_line)
69          stacks = dict()
```

```
70          while len(incoming_production_line) != 0:
71              first = incoming_production_line.popleft()
72              if first.liters not in stacks:
73                  stacks[first.liters] = []
74
75              stack = stacks[first.liters]
76              stack.append(first)
77          return stacks
78
79
80  class PackageManager(Machine):
81      def process(self, incoming_production_line):
82          packages = deque()
83          for stack in incoming_production_line.values():
84              package = Package()
85              package.add_bottles(stack)
86              packages.append(package)
87          return packages
88
89
90  class Factory:
91      def __init__(self):
92          self.bottlemodulator = BottleModulator()
93          self.lowFAT32 = LowFAT32()
94          self.hashSoda9001 = HashSoda9001()
95          self.packageManager = PackageManager()
96
97      def producir(self, num_bottles):
98          self.bottlemodulator.bottles_to_produce = num_bottles
99          product = None
100         for machine in [self.bottlemodulator,
101                         self.lowFAT32,
102                         self.hashSoda9001,
103                         self.packageManager]:
104             product = machine.process(product)
```

```
105             return product
106
107
108 if __name__ == "__main__":
109
110     num_bottles = 423
111
112     factory = Factory()
113     output = factory.producir(num_bottles)
114     print("----------------------")
115     print("{} bottles produced {} packages".format(
116         num_bottles, len(output)))
117     for package in output:
118         package.see_content()
119     print("----------------------")
```

15.4 Solution for activity 2.2: Subway Map

```
from subway_stations import Direction, Map, Station

# Returns True if there is a path from the origin station to the destination
# station, otherwise False.
# You only can control the variables related to the origin_station, and ready
# station.

def path(origin_station, destination_station):
    global exist_path, route
    exist_path = False
    if search_rec(origin_station, destination_station):
        print("Route: ", end=" ")
        for station in route:
            print(station, end=" ")
        return True
    return False


def search_rec(origin_station, destination_station, past_stations=[]):
    global exist_path, route
    for d in Direction:
        if not exist_path:
            actual_station = origin_station.directions[d]
            past_stations_temp = list(past_stations)
            if actual_station == destination_station:
                past_stations_temp.append(actual_station)
                route = list(past_stations_temp)
                exist_path = True
            elif actual_station not in past_stations_temp and actual_station:
                past_stations_temp.append(actual_station)
                search_rec(actual_station,
                           destination_station,
                           past_stations_temp)
    return exist_path
```

```
35
36
37 if __name__ == "__main__":
38     map = Map.example_map()
39     print(path(map.first_station, map.stations[10]))
40     print(path(map.stations[1], map.first_station))
41     print(path(map.stations[9], map.stations[14]))
42     print(path(map.first_station, map.first_station))
43     print(path(map.first_station, map.last_station))
```

15.5 Solution for activity 3.1: Patients in a Hospital

```
class Patient:

    def __init__(self, year, month, day, color, hour, release_reason):
        self.id = next(Patient.id)
        self.year = year
        self.month = month
        self.day = day
        self.color = color
        self.hour = hour
        self.release_reason = release_reason

    def id_patient():
        id = 0
        while True:
            yield id
            id+=1
    id = id_patient()

    def __str__(self):
        return '{}\t{}\t{}\t{}\t{}\t{}\t{}'.format(self.id, self.year,
                                                  self.month, self.day,
                                                  self.color, self.hour,
                                                  self.release_reason)

class Report:

    def __init__(self):
        self.patients = []

    def __iter__(self):
        return iter(self.patients)

    def patients_by_color(self, color):
        return [p for p in self.patients if p.color == color]
```

```
35
36 class Reader:
37     def file():
38         with open("Report.txt","r") as file:
39             for line in file:
40                 yield line
41
42     line = file()
43
44 if __name__ == '__main__':
45
46     reporte = Report()
47
48     lines_read = 0
49     while True:
50         try:
51             datos = next(Reader.line).split("\t")
52
53         except StopIteration:
54             print("End of file")
55             break
56
57         valores = ["year","month","day","color","hour","release_reason"]
58         args = dict(zip(valores, datos))
59
60         reporte.patients.append(Patient(**args))
61         lines_read+=1
62
63     print('{} lines read'.format(lines_read))
64
65     for patient in reporte:
66         print(patient)
```

15.6 Solution for activity 3.2: Soccer Team

```
from datetime import date
from functools import reduce

path = "players.txt"


def read_file():
    def splitter(line):
        return tuple(line.split(";"))

    def transform_to_int(foo):
        return tuple(map(int, foo))

    splitted = list(map(splitter, [line for line in open(path)]))
    tuplas = map(lambda foo: foo[0:5] + transform_to_int(foo[5:11]),
                 splitted)
    return list(tuplas)


def has_the_name(_list):
    mi_nombre = ("Jaime", "Castro", "Retamal")

    def any_match(tupl):
        return any(map(lambda foo1, foo2: foo1 == foo2, tupl, mi_nombre))

    return list(filter(any_match, _list))


def chilean_lefties(_list):
    tupl = ("Chile", "izquierdo")
    return list(filter(lambda foo: foo[3:5] == tupl, _list))


def get_ages(_list):
```

```
35      year = date.today().year
36      return list(map(lambda foo: foo[0:2] + (year - foo[7],), _list))
37
38
39  def sub_17(_list):
40      age_tuples = get_ages(_list)
41      return list(filter(lambda foo: foo[2] <= 17, age_tuples))
42
43
44  def top_scorer(_list):
45      def better_scorer(foo1, foo2):
46          return foo1 if foo1[8] > foo2[8] else foo2
47
48      return reduce(better_scorer, _list)
49
50
51  def highest_obesity_risk(_list):
52      def bmi(tupl):
53          return (tupl[3] / (tupl[2] / 100) ** 2,)
54
55      def max_bmi(foo1, foo2):
56          return foo1 if foo1[4] > foo2[4] else foo2
57
58      # filter for chileans
59      team = list(filter(lambda foo: foo[3] == 'Chile', _list))
60      # map only the useful fields
61      mapped = list(map(lambda foo: foo[0:2] + foo[9:11], team))
62      # we add the BMIs
63      bmi_tuples = list(map(lambda foo: foo + bmi(foo), mapped))
64
65      return reduce(max_bmi, bmi_tuples)
66
67
68  def print_results(func, _list):
69      result = func(_list)
```

```
70
71      print(func.__name__.title())
72      print("-" * len(func.__name__))
73
74      if type(result) is list:
75          print(*result, sep='\n')
76      else:
77          print(result)
78
79      print("")
80
81
82  if __name__ == '__main__':
83      tupls = read_file()
84      print_results(has_the_name, tupls)
85      print_results(chilean_lefties, tupls)
86      print_results(get_ages, tupls)
87      print_results(sub_17, tupls)
88      print_results(top_scorer, tupls)
89      print_results(highest_obesity_risk, tupls)
```

15.7 Solution for activity 3.3: Hamburger Store

```
def compare_by(attr):
    """
    This is a class decorator generator that takes an attribute as
    an argument. Adds the ability to compare class objects by the
    given attribute. The attribute must be a comparable type.
    """

    def decorator(cls):
        def less_than(a, b):
            return getattr(a, attr) < getattr(b, attr)

        def less_equal(a, b):
            return getattr(a, attr) <= getattr(b, attr)

        def equal(a, b):
            return getattr(a, attr) == getattr(b, attr)

        def greater_equal(a, b):
            return getattr(a, attr) >= getattr(b, attr)

        def greater_than(a, b):
            return getattr(a, attr) > getattr(b, attr)

        # we now add the new methods to the class
        setattr(cls, '__lt__', less_than)
        setattr(cls, '__le__', less_equal)
        setattr(cls, '__eq__', equal)
        setattr(cls, '__ge__', greater_equal)
        setattr(cls, '__gt__', greater_than)

        # we return the modified class
        return cls

    return decorator
```

```
35
36
37  def save_instances(cls):
38      """
39      This decorator modifies a class's behaviour such that the
40      class now posseses a static list attribute that contains all
41      the instances that have been created. This list may be accessed
42      through Class.instances.
43      """
44
45      # a reference to the original __init__ is saved
46      prev_init = getattr(cls, '__init__')
47
48      def new_init(self, *args, **kwargs):
49          # The new __init__ shall call the previous and add the created
50          # object to the class instances' list.
51
52          prev_init(self, *args, **kwargs)
53          cls.instances.append(self)
54
55      # Class attributes are added/modified
56      setattr(cls, 'instances', list())
57      setattr(cls, '__init__', new_init)
58
59      # The original class is returned after the modifications
60      return cls
61
62
63  def change_tax(func):
64      """This decorator modifies price calculating functions in order
65      to consider a change in sales tax in the prices"""
66
67      def inner(num):
68          """
69          We subtract the 100 spent in transport and then multiply
```

```
70          by 1.23/1.19 in order to account for the change in sales tax.
71          The 100 transport fee is then added back.
72          """
73          return (func(num) - 100) * 1.23 / 1.19 + 100
74
75      return inner
76
77
78  @compare_by("meat_quantity")
79  @save_instances
80  class Hamburger:
81      def __init__(self, high, diameter, meat_quantity):
82          self.high = high
83          self.diameter = diameter
84          self.meat_quantity = meat_quantity
85
86      def __repr__(self):
87          return ('Hamburger {0} cms high, '
88                  '{1} cm of diameter and '
89                  '{2} meat quantity').format(self.high, self.diameter,
90                                              self.meat_quantity)
91
92
93  @change_tax
94  def price_after_tax(price_before_tax):
95      return (price_before_tax * 1.19 + 100)
96
97
98  if __name__ == "__main__":
99      hamburger1 = Hamburger(10, 15, 2)
100     hamburger2 = Hamburger(7, 10, 3)
101     hamburger3 = Hamburger(10, 9, 2)
102
103     print(hamburger2 > hamburger1)
104     print(hamburger2 == hamburger3)
```

```
105     print(hamburger1 < hamburger3)
106
107     print(Hamburger.instances)
108     hamburger4 = Hamburger(12, 20, 4)
109     print(Hamburger.instances)
110
111     print(price_after_tax(2000))
```

15.8 Solution for activity 4.1: MetaRobot

```
class MetaRobot(type):
    def __new__(meta, name, base_classes, dictionary):
        if (name != 'Robot'):
            raise NameError('A class other than Robot is attempting to '
                            'be created')

        creator = 'my_user_name'
        start_ip = '190.102.62.283'

        def check_creator(self):
            if self.creator in self.creators:
                print('The creator of the robot is in the list of '
                      'programmers!')
                return True
            print('Danger!! The creator of the robot is trying to blame'
                  'someone else! Stop him!!')
            return False

        def change_node(self, node):
            print('Moving from conection {} to conection {}!'
                  .format(self.actual.ide, node.ide))
            self.actual = node

        def disconnect(self):

            if self.Verify():
                print('Congratulations! You have found a hacker and ',
                      end='')
                print('you kicked him out of the network!')

                # disconnect the hacker from the actual port
                self.actual.hacker = 0
                return True

```

```
35              print('Hey! There is no hacker here!')
36
37          dictionary['creator'] = creator
38          dictionary['start_ip'] = start_ip
39          dictionary['check_creator'] = check_creator
40          dictionary['change_node'] = change_node
41          dictionary['disconnect'] = disconnect
42          return super().__new__(meta, name, base_classes, dictionary)
```

15.9 Solution for activity 5.1: Calculator

```
import enum
from math import e, pi


class Operations(enum.Enum):
    sum = ('+', 'plus')
    substraction = ('-', 'minus')
    multiplication = ('*', 'times')
    division = ('/', 'divided')
    module = ('%', 'module')

    def find_operation(operacion_str):
        for operacion in Operations:
            if operacion.value[0] == operacion_str:
                return operacion

    def is_operation(value):
        for operacion in Operations:
            if operacion.value[0] == value:
                return True
        return False


def index_generator(n):
    for i in range(n):
        yield i


class Calculator:
    def __init__(self, letras_especiales):
        self.special_letters = letras_especiales

    def find_operations(self, statement):
        operations = []
```

```
35          operands = []
36          last = 0
37          for i in range(len(statement)):
38              if Operations.is_operation(statement[i]):
39                  operands.append(statement[last:i])
40                  operations.append(statement[i])
41                  last = i + 1
42          if last != len(statement):
43              operands.append(statement[last:])
44          return operations, operands
45
46      def read_operations(self, statement):
47          statement = statement.replace(' ', '')
48          operations, operand = self.find_operations(statement)
49          operands_index = index_generator(len(operand))
50          operator1 = self.digit_value(
51              operand[next(operands_index)], statement)
52          for current_operation in operations:
53              try:
54                  operator2 = self.digit_value(
55                      operand[next(operands_index)], statement)
56              except StopIteration:
57                  print('[ERROR] {0}'.format(StopIteration.__name__))
58                  print('There\'s one missing operator in {0}'.format(statement))
59              else:
60                  operation = Operations.find_operation(current_operation)
61                  result = self.do_operation(
62                      operator1, operator2, operation, statement)
63                  operator1 = result
64
65      def do_operation(self, operator1, operator2, operation, statement):
66          try:
67              if operation is Operations.sum:
68                  result = operator1 + operator2
69              elif operation is Operations.substraction:
```

```
70                  result = operator1 - operator2
71              elif operation is Operations.multiplication:
72                  result = operator1 * operator2
73              elif operation is Operations.division:
74                  try:
75                      result = operator1 / operator2
76                  except ZeroDivisionError:
77                      print('[ERROR] {0}'.format(ZeroDivisionError.__name__))
78                      result = 'infinite'
79              else:
80                  result = operator1 % operator2
81              print('{0} {1} {2} is equal to {3}'.format(
82                  operator1, operation.value[1], operator2, result))
83              return result
84          except:
85              print('The operation wasn\'t exceuted {0}'.format(statement))
86
87      def digit_value(self, number_letter, statement):
88
89          if number_letter.isdigit():
90              return int(number_letter)
91
92          elif number_letter.isalpha():
93              try:
94                  v = self.special_letters[number_letter]
95              except KeyError:
96                  print('[ERROR] {0}'.format(KeyError.__name__))
97                  print(('\'{0}\' doesn\'t any assigned values .'
98                         'You must added before using it.').format(
99                      number_letter))
100             else:
101                 return v
102
103         elif Calculator.isfloat(number_letter):
104             return float(number_letter)
```

```
105
106         else:
107             print(('[ERROR] The sintaxis \'{0}\' is incorrect. '
108                    'Read the manual for more information.').format(statement))
109
110     def isfloat(s):
111         try:
112             float(s)
113             return True
114         except ValueError:
115             print('[ERROR] {0}'.format(ValueError.__name__))
116             print('\'{0}\' cannot be parse to float'.format(s))
117             return False
118
119     def add_letter(self, letter, value):
120         try:
121             v = self.special_letters[letter]
122         except KeyError:
123             self.special_letters[letter] = value
124         else:
125             print('[ERROR] {0}'.format(KeyError.__name__))
126             print(('Letter \'{0}\' won\'t be agregated. '
127                    'It already exist in memory.').format(letter))
128
129
130 # DO NOT MODIFY THIS.
131 special_letters = {'g': 9.81, 'e': e, 'pi': pi}
132 calculator = Calculator(special_letters)
133 tested_operations = ['3+5', 'g*3', 'pi*4', '76 /2 + 35 / 5']
134 calculator.add_letter('g', 5757)
135 print('')
136 statements_for_testing = ['1/0', 'a+2', 'g+0', '88/2+0+', '1=2', '8.953 + 1']
137 for operations in statements_for_testing:
138     calculator.read_operations(operations)
139     print('')
```

15.10 Solution for activity 6.1: Testing the encryptor

```
1  import string
2  import encryptor as enc
3  import pytest
4
5
6  def check_bijection(rot):
7      dom = range(26)
8      rec = []
9
10     for x in dom:
11         y = rot.get(x)
12         assert y is not None
13         rec.append(y)
14
15     assert set(dom) == set(rec)
16
17
18 def setup_module(module):
19     enc.create_alphabet((list(string.ascii_lowercase)))
20
21
22 class TestRotor:
23     def setup_class(cls):
24         cls.rotors = []
25         for i in range(1, 4):
26             cls.rotors.append(enc.Rotor('files/rotor{0}.txt'.format(i)))
27
28     def test_function(self):
29         for rotor in self.rotors:
30             check_bijection(rotor)
31
32
33 class TestReflector:
34     def setup_class(cls):
```

```
        cls.reflector = enc.Reflector('files/reflector.txt')

    def test_function(self):
        dom = range(26)
        check_bijection(self.reflector)
        for x in dom:
            y = self.reflector.get(x)
            x2 = self.reflector.get(y)
            assert x2 is not None and x2 == x


class TestEncoding:
    def setup_class(cls):
        rots = ['files/rotor1.txt', 'files/rotor2.txt', 'files/rotor3.txt']
        refl = 'files/reflector.txt'
        cls.list_ = ['thequickbrownfoxjumpsoverthelazydog',
                     'python', 'bang', 'dragonfly', 'csharp']
        cls.encoder = enc.Encoder(rots, refl)

    def test_encoding(self):
        for text in self.list_:
            a = self.encoder.encrypt(text)
            assert a != text
            b = self.encoder.encrypt(a)
            assert text == b

    def test_exception(self):
        with pytest.raises(ValueError):
            self.encoder.encrypt('I AM SURE THIS WILL PASS')
```

```
import encryptor


# To verify that the function is biyective you can use:
def check_bijection(rot):
    dom = 26
```

```
7       rec = []
8
9       for x in dom:
10          y = rot.get(x)
11          assert y is not None
12          rec.append(y)
13
14      assert set(dom) == set(rec)
15
16
17  def setup_module(module):
18      pass
19
20
21  class TestRotor:
22      def setup_class(cls):
23          cls.rotors = []
24          for i in range(1, 4):
25              cls.rotors.append(
26                  encryptor.Rotor('files/rotor{0}.txt'.format(i))
27              )
28
29      def test_function(self):
30          # You can use the rotors by calling: self.rotors
31          pass
32
33
34  class TestReflector:
35      def setup_class(cls):
36          cls.reflector = encryptor.Reflector('files/reflector.txt')
37
38      def test_function(self):
39          # You can use the reflector by calling: self.reflector
40          pass
41
```

```

class TestEncoder:
    def setup_class(cls):
        pass

    def test_encoding(self):
        pass

    def test_exception(self):
        pass
```

```
import string


def create_alphabet(alphabet):
    global ALPHABET, LETTER_DICT, SIZE
    ALPHABET = alphabet
    SIZE = len(alphabet)
    LETTER_DICT = {ALPHABET[i]: i for i in range(SIZE)}


class Rotor:
    def __init__(self, path_disk):
        self.disk = []
        with open(path_disk) as disk_file:
            for line in disk_file:
                self.disk.append(int(line.strip()))
        self.disk_ini = list(self.disk)

    def reset(self):
        self.disk = list(self.disk_ini)

    def rotate_disk(self):
        newdisk = [self.disk[(i - 1) % SIZE] for i in range(SIZE)]
        self.disk = newdisk

```

```
26      def get(self, number):
27          if number > SIZE - 1 or number < 0:
28              return None
29          return self.disk[number]
30
31      def get_out(self, number):
32          return self.disk.index(number)
33
34
35  class Reflector:
36      def __init__(self, path_disk):
37          self.refl = {}
38          with open(path_disk) as disk_file:
39              num1 = 0
40              for line in disk_file:
41                  num2 = int(line.strip())
42                  self.refl[num1] = num2
43                  self.refl[num2] = num1
44                  num1 += 1
45
46      def get(self, number):
47          return self.refl.get(number)
48
49
50  class Encoder:
51      def __init__(self, path_rotors, path_reflectors):
52          self.pos = 0
53          self.rotors = []
54          self.size = len(path_rotors)
55          for path in path_rotors:
56              self.rotors.append(Rotor(path))
57          self.refl = Reflector(path_reflectors)
58
59      def reset(self):
60          self.pos = 0
```

```
61          for rot in self.rotors:
62              rot.reset()
63
64      def transform_letter(self, num):
65          aux = num
66          for rot in self.rotors:
67              aux = rot.get(aux)
68          aux = self.refl.get(aux)
69          for rot in reversed(self.rotors):
70              aux = rot.get_out(aux)
71          return aux
72
73      def rotate_disks(self):
74          self.pos += 1
75          self.rotors[0].rotate_disk()
76          for i in range(1, self.size):
77              if self.pos % (SIZE ** i) == 0:
78                  self.rotors[i].rotate_disk()
79          self.pos = self.pos % (SIZE ** (self.size - 1))
80
81      def encrypt(self, text):
82          out = []
83          self.reset()
84
85          for let1 in text:
86              num1 = LETTER_DICT.get(let1)
87              if num1 is None:
88                  raise ValueError(
89                      'The character %s is not in the alphabets' % let1)
90              num2 = self.transform_letter(num1)
91              let2 = ALPHABET[num2]
92              self.rotate_disks()
93              out.append(let2)
94
95          return ''.join(out)
```

```
96
97      def write_text(self, path_in, path_out):
98          with open(path_out, 'w') as out:
99              with open(path_in) as inp:
100                 for line in inp:
101                     encrypted = self.encrypt(line.strip())
102                     out.write(encrypted + '\n')
103
104
105 if __name__ == '__main__':
106     rots = ['files/rotor1.txt', 'files/rotor2.txt', 'files/rotor3.txt']
107     refl = 'files/reflector.txt'
108
109     # entrega el alfabeto
110     create_alphabet((list(string.ascii_lowercase)))
111
112     enc = Encoder(rots, refl)
113
114     # encriptando con la funcion directamente
115     a = enc.encrypt('hello')
116     b = enc.encrypt(a)
117     print(a, b)
118
119     # leyendo desde el archivo
120     enc.write_text('files/input.txt', 'files/output.txt')
```

15.11 Solution for activity 6.2: Testing ATMs

```
import unittest
from bank import Bank, ATM


class Test_ATM(unittest.TestCase):
    def setUp(self):
        self.bank = Bank("Seguritas")
        self._id1 = "18.375.852-2"
        self.name1 = "John Dupre"
        self.password1 = 2345
        self._id2 = "13.432.113-k"
        self.name2 = "Emma Cashter"
        self.password2 = 5912
        self.bank.add_user(self._id1, self.name1, self.password1)
        self.bank.add_user(self._id2, self.name2, self.password2)
        self.atm = ATM(self.bank)

    def test_credentials(self):
        # first case: _id y password right
        self.atm.login(self._id1, self.password1)
        _idingresado = self.bank.actual_user._id
        self.assertEqual(self._id1, _idingresado)
        # second case: _id right but password incorrect
        self.atm.login(self._id1, 1234)
        self.assertIsNone(self.bank.actual_user)
        # tercer case: _id no está en la bank database
        self.atm.login("10.000.000-1", 1234)
        self.assertIsNone(self.bank.actual_user)

    def test_balance(self):
        self.atm.withdraw_money(self._id1, self.password1, 20000)
        balance = self.bank.actual_user.balance
        # the user must have balance 0, ya que nunca ha depositado
        self.assertEqual(0, balance)
```

```
35          # the test fails, you can see that the balance results in
36          # -20.000 when it should be 0
37
38      def test_amount_updated(self):
39          self.atm.login(self._id1, self.password1)
40          # deposit of 10.000
41          self.bank.deposit(self.bank.actual_user, 10000)
42          # withdrawal of 5.000
43          self.atm.withdraw_money(self._id1, self.password1, 5000)
44          balance = self.bank.actual_user.balance
45          # balance must end up in 5000
46          self.assertEqual(5000, balance)
47
48      def test_account_tercero(self):
49          # Will try to transfer to an account that does not exist
50          self.atm.login(self._id1, self.password1)
51          self.bank.deposit(self.bank.actual_user, 10000)
52          self.atm.transfer_money(
53              self._id1, self.password1, "1.000.000-3", 5000)
54          self.assertIsNone(self.bank.third_person)
55          # Indeed the destination user is not created and it is not found
56
57      def test_amounts_updated(self):
58          self.atm.login(self._id1, self.password1)
59          # account 1 receives 15.000
60          self.bank.deposit(self.bank.actual_user, 15000)
61          # 5.000 transfered from account 1 to account 2
62          self.atm.transfer_money(self._id1, self.password1, self._id2,
63                                  3000)
64          # we should prove that account 1 balance = 12.000 and account
65          # 2 balance = 3.000
66          amountUser = self.bank.actual_user.balance
67          amountThird = self.bank.third_person.balance
68          self.assertEqual(amountUser, 12000)
69          self.assertEqual(amountThird, 3000)
```

```
70          # Here the test fails
71
72      def test_verify_error(self):
73          # what if the third user does not exist
74          self.atm.login(self._id1, self.password1)
75          # account 1 receives a 10.0000 deposit
76          self.bank.deposit(self.bank.actual_user, 10000)
77          # lets transfer to a non existing account
78          self.atm.transfer_money(
79              self._id1, self.password1, "1.000.000-3", 5000)
80          # lets verify that the transference is not performed
81          amountUser = self.bank.actual_user.balance
82          self.assertEqual(amountUser, 10000)
83          # we can see that anyway the 5.000 is substracted despite the
84          # error the test fails
85
86
87  if __name__ == "__main__":
88      unittest.main()
```

15.12 Solution for activity 7.1: Godzilla

```
import threading
import time
import random


class Godzilla(threading.Thread):

    def __init__(self, hp):
        super().__init__()
        self.hp = hp
        self.alive = True

    def run(self):
        while self.hp > 0 and self.alive is True:
            time.sleep(8)
            if self.alive is True:  # Godzilla can die while he is waiting
                self.attack()
        print('The simulation has finished')


    def attacked(self, soldier):
        self.hp -= soldier.attack
        if self.hp <= 0:
            self.alive = False
            print('Godzilla has died!!')
        else:
            print(
                'Godzilla has been attacked! The soldier has caused damage '
                + str(soldier.attack) + '. HP Godzilla ' + str(self.hp))
            soldier.attacked(int(soldier.attack / 4))

    def attack(self):
        for i in soldiers_list:
            if i.alive is True:
```

```
                i.attacked(3)


class Soldier(threading.Thread):

    def __init__(self, Godzilla, velocity, hp, attack):
        super().__init__()
        self.alive = True
        self.Godzilla = Godzilla
        self.velocity = velocity
        self.hp = hp
        self.ID = next(Soldier.get_i)
        self.attack = attack


    def run(self):
        while self.hp > 0 and Godzilla.alive is True:
            time.sleep(self.velocity)
            if Godzilla.alive is True:
                # Godzilla can die while he is waiting
                Godzilla.attacked(self)


    def attacked(self, attack):
        self.hp -= attack
        print('The soldier' + str(self.ID) + ' has been damaged!! HP '
            + str(self.hp))
        if self.hp <= 0:
            self.alive = False
            print('The soldier' + str(self.ID) + ' has died :( !!!')


    def id_():
        i = 0
        while True:
```

```
70              yield i
71              i += 1
72
73      get_i = id_()
74
75  if __name__ == '__main__':
76      print('Starting simulation!')
77
78      num_soldiers = 20  # int(input('How many soldiers do you want?'))
79      soldiers_list = []
80      Godzilla = Godzilla(1000)
81      Godzilla.start()
82      for i in range(num_soldiers):
83          soldier = Soldier(Godzilla, random.randint(4, 20), 60, 30)
84          soldier.setDaemon(True)
85          soldier.start()
86          soldiers_list.append(soldier)
```

15.13 Solution for activity 7.2: Mega Godzilla

```
import threading
import time
import random


class MegaGodzilla(threading.Thread):

    lockgod = threading.Lock()

    def __init__(self, hp):
        super().__init__()
        self.hp = hp
        self.scream = False

    @property
    def alive(self):
        if self.hp > 0:
            return True
        return False

    def run(self):
        while self.hp > 0 and self.alive:
            attack_time = random.randint(3, 6)
            time.sleep(attack_time)
            if self.alive:
                attack_type = random.randint(0, 1)
                if attack_type == 0 or self.scream:
                    # Godzilla can die while he is waiting
                    self.attack(3)
                elif attack_type == 1 and not self.scream:
                    self.scream = True
                    print('MEGA-GODZILLA: 1 , 2 ,3 frozen!!!')
                    self.changestatus()
                    self.attack(6)
```

```
35
36          print('The simulation has finished')
37
38
39      def attacked(self, soldier):
40          self.hp -= soldier.attack
41          if not self.alive:
42              print('MegaGodzilla has died!!')
43          else:
44              print('Mega-Godzilla has been attacked! The soldier has'
45                    'caused damage' + str(soldier.attack) +
46                    '. HP Godzilla ' + str(self.hp))
47              soldier.attacked(int(soldier.attack / 4))
48
49      def changestatus(self, status=False):
50          for i in soldiers_list:
51              if i.alive:
52                  i.status = status
53          if status:
54              self.scream = False
55
56      def attack(self, damage):
57          for i in soldiers_list:
58              if i.alive:
59                  i.attacked(damage)
60
61
62  class Soldier(threading.Thread):
63
64      locksold = threading.Lock()
65      lockattack = threading.Lock()
66
67      def __init__(self, MegaGodzilla, velocity, hp, attack):
68          super().__init__()
69          self.MegaGodzilla = MegaGodzilla
```

```
70          self.velocity = velocity
71          self.hp = hp
72          self.status = True
73          self.ID = next(Soldier.get_i)
74          self.attack = attack
75          self.attacktime = random.randint(1, 3)
76
77      @property
78      def alive(self):
79          if self.hp > 0:
80              return True
81          return False
82
83      def run(self):
84          while self.hp > 0 and MegaGodzilla.alive:
85              time.sleep(self.velocity)
86              Soldier.locksold.acquire()
87              if not self.status:
88                  time.sleep(10)
89                  print('WE CAN CONTINUE!!')
90                  MegaGodzilla.changestatus(status=True)
91              if MegaGodzilla.alive:
92                  Soldier.lockattack.acquire()
93                  MegaGodzilla.attacked(self)
94                  time.sleep(self.attacktime)
95                  Soldier.lockattack.release()
96              Soldier.locksold.release()
97
98
99      def attacked(self, attack):
100         self.hp -= attack
101         print('The soldier' + str(self.ID) +
102               ' has been damaged!!  HP ' + str(self.hp))
103         if not self.alive:
104             print('The soldier' + str(self.ID) + ' has died :( !!!')
```

```
105
106
107         def id_():
108             i = 0
109             while True:
110                 yield i
111                 i += 1
112
113         get_i = id_()
114
115
116 if __name__ == '__main__':
117         print('Starting simulation!')
118
119         num_soldiers = 20  # int(input('How many soldiers do you want?'))
120         soldiers_list = []
121         MegaGodzilla = MegaGodzilla(1000)
122         MegaGodzilla.start()
123         for i in range(num_soldiers):
124             soldier = Soldier(MegaGodzilla, random.randint(4, 20), 60, 30)
125             soldier.setDaemon(True)
126             soldier.start()
127             soldiers_list.append(soldier)
```

15.14 Solution for activity 8.1: Client queues

```
from collections import deque
from random import uniform

"""
Events:
1. Client arrives to the system
2. Client goes out from teller 1
3. Client goes out from teller 2
Relevant state variables:
1. Watch simulation (current simulation time)
2. Length of queue1
3. Length of queue2
4. Next time instance of each event
"""


class Client:
    pass


class Teller:
    def __init__(self, t):
        self.queue = deque()
        self.end_time = 2 * t
        """
        Initially the ending time of serve must be out of  range
        because no client can leave a teller if there is not a
        client being attended. With 2*t we ensure that someone always
        'arrives' before the event 'goes out of teller' is activated.
        """


class Bank:
    def __init__(self, t):
```

```
35          self.tellers = [Teller(t), Teller(t)]
36
37
38  class Simulation:
39      def __init__(self, max_time, arrival, departure):
40
41          self.ending_time_sim = max_time
42          self.simulation_time = 0
43          self.next_client_arrival = uniform(1, arrival)
44          self.bank = Bank(max_time)
45          self.departure = departure
46          self.arrival = arrival
47
48      def departure_first(self, queue_yes, queue_no):
49
50          queue = self.bank.tellers[queue_yes].queue
51          print('[start] departure queue', queue_yes,
52                len(self.bank.tellers[queue_yes].queue),
53                len(self.bank.tellers[queue_no].queue))
54
55          if len(self.bank.tellers[queue_yes].queue) > 0:
56              queue.popleft()
57
58          if len(self.bank.tellers[queue_no].queue) > \
59                          len(self.bank.tellers[queue_yes].queue) + 1 >= 0:
60              print('client changes queue')
61              self.bank.tellers[queue_yes].queue.append(
62                  self.bank.tellers[queue_no].queue.popleft())
63
64          if len(self.bank.tellers[queue_yes].queue) == 0:
65              self.bank.tellers[queue_yes].end_time = \
66                  2 * self.ending_time_sim
67          else:
68              self.bank.tellers[
69                  queue_yes].end_time = self.simulation_time + \
```

```
70                                      uniform(1, self.departure)
71
72          print('[end] departure queue', queue_yes,
73                len(self.bank.tellers[queue_yes].queue),
74                len(self.bank.tellers[queue_no].queue))
75
76      def arrival_first(self, first):
77
78          print('client arrives', first)
79          self.next_client_arrival = self.simulation_time + \
80                                     uniform(1, self.arrival)
81
82          if len(self.bank.tellers[0].queue) <= \
83                  len(self.bank.tellers[1].queue):
84              if len(self.bank.tellers[0].queue) == 0:
85                  self.bank.tellers[0].end_time = \
86                      self.simulation_time + \
87                      uniform(1, self.departure)
88              self.bank.tellers[0].queue.append(Client())
89          else:
90              if len(self.bank.tellers[1].queue) == 0:
91                  self.bank.tellers[1].end_time = \
92                      self.simulation_time + \
93                      uniform(1, self.departure)
94              self.bank.tellers[1].queue.append(Client())
95
96          print('queues', len(self.bank.tellers[0].queue),
97                len(self.bank.tellers[1].queue))
98
99      def run(self):
100
101         while True:
102             first = min(self.bank.tellers[0].end_time,
103                         self.bank.tellers[1].end_time,
104                         self.next_client_arrival)
```

```
105                 if first >= self.ending_time_sim:
106                     break
107
108                 self.simulation_time = first
109
110                 if self.next_client_arrival == first:
111                     self.arrival_first(first)
112                 else:
113                     if self.bank.tellers[0].end_time == first:
114                         self.departure_first(0, 1)
115                     else:
116                         self.departure_first(1, 0)
117
118
119 if __name__ == '__main__':
120     s = Simulation(80, 3, 10)
121     s.run()
```

15.15 Solution for activity 8.2: GoodZoo

File AC08_1_classes.py

```
from abc import ABCMeta
from random import expovariate, uniform, choice


class Animal(metaclass=ABCMeta):
    def __init__(self, time):
        # This is just a simple way to give an id to each object
        if not hasattr(self.__class__, 'current_ident'):
            setattr(self.__class__, 'current_ident', 0)
        if time == 0:
            setattr(self.__class__, 'statistics', {
                'extinction_time': -1,
                'new_animal': 0,
                'death_eaten': 0,
                'death_no_energy': 0,
                'death_old': 0,
                'current_num_animals': 0,
                'time_lived': 0,
                'time_waited_for_food': 0,
                'waited_for_food': 0
            })
        self.ident = self.__class__.current_ident
        self.__class__.current_ident += 1
        self.energy = 50
        self.time_since_hungry = -1
        self._dead = False
        self.moment_of_dead = self.metadata['life_expectancy'] + time
        self.next_eating_time = time
        self.next_new_animal_time = time
        self.set_next_eating_time()
        self.set_next_new_animal_time()
        self.current_time = time
```

```
33          self.birth_date = time
34
35      @property
36      def dead(self):
37          return self._dead
38
39      @dead.setter
40      def dead(self, value):
41          self._dead = value
42          self.__class__.statistics['current_num_animals'] -= 1
43          if self.__class__.statistics['current_num_animals'] == 0:
44              self.__class__.statistics['extinction_time'] = self.current_time
45          self.__class__.statistics['time_lived'] += self.current_time - \
46                                                     self.birth_date
47
48      def next_action_time(self):
49          return min(self.next_new_animal_time,
50                     self.moment_of_dead,
51                     self.next_eating_time)
52
53      def set_next_eating_time(self, food=True):
54          if food:
55              param = self.metadata['time_for_food']
56              self.next_eating_time += int(uniform(param[0], param[1]))
57          else:
58              self.next_eating_time += 1
59
60      def set_next_new_animal_time(self):
61          self.next_new_animal_time += \
62              int(expovariate(self.metadata['new_animal']))
63
64      def eat(self, food, time):
65          selected = choice(food)
66
67          # Can eat
```

```
68          if selected.__class__.__name__ in self.metadata['food']:
69              print('{0}{2} I am eating a {1}{3}'.format(
70                  self.__class__.__name__,
71                  selected.__class__.__name__,
72                  self.ident, selected.ident))
73
74              if self.time_since_hungry > 0:
75                  self.__class__.statistics['time_waited_for_food'] += \
76                      self.time_since_hungry
77                  self.__class__.statistics['waited_for_food'] += 1
78
79              self.time_since_hungry = 0
80              self.energy += self.metadata['food_energy']
81              self.set_next_eating_time()
82              selected.die('eaten', time)
83          else:
84              self.time_since_hungry += 1
85              self.set_next_eating_time(False)
86              self.energy -= self.metadata['food_energy'] / 2
87              if self.energy < 0:
88                  self.die('energy', time)
89              else:
90                  print('{0}{2} hungry. Found {1}'.format(
91                      self.__class__.__name__,
92                      selected.__class__.__name__,
93                      self.ident))
94
95      def new_animal(self, time):
96          self.energy -= self.metadata['new_animal_energy']
97          self.set_next_new_animal_time()
98          self.__class__.statistics['new_animal'] += 1
99          self.__class__.statistics['current_num_animals'] += 1
100         print('Born a new {0} from {1}'.format(
101             self.__class__.__name__,
102             self.ident))
```

```
103
104             if self.energy == 0:
105                 self.die('energy', time)
106
107         def die(self, reason, time):
108             self.dead = True
109             self.current_time = time
110             if reason == 'old':
111                 self.__class__.statistics['death_old'] += 1
112             elif reason == 'energy':
113                 self.__class__.statistics['death_no_energy'] += 1
114             elif reason == 'eaten':
115                 self.__class__.statistics['death_eaten'] += 1
116
117             print('{0}{2} has died because of {1}'.format(
118                 self.__class__.__name__,
119                 reason, self.ident))
120
121         def __repr__(self):
122             return '{0}{1}'.format(self.__class__.__name__, self.ident)
123
124
125     class Tiger(Animal):
126         metadata = {
127             'food': ['Elephant', 'Jaguar', 'Penguin'],
128             'new_animal': 1 / 75,
129             'new_animal_energy': 15,
130             'life_expectancy': 300,
131             'time_for_food': [20, 40],
132             'food_energy': 30
133         }
134
135         def __init__(self, time):
136             super().__init__(time=time)
137
```

```
138
139 class Jaguar(Animal):
140     metadata = {
141         'food': ['Elephant', 'Tiger', 'Penguin'],
142         'new_animal': 1 / 80,
143         'new_animal_energy': 10,
144         'life_expectancy': 350,
145         'time_for_food': [35, 55],
146         'food_energy': 20
147     }
148
149     def __init__(self, time):
150         super().__init__(time=time)
151
152
153 class Elephant(Animal):
154     metadata = {
155         'food': ['Grass'],
156         'new_animal': 1 / 200,
157         'new_animal_energy': 7,
158         'life_expectancy': 500,
159         'time_for_food': [8, 15],
160         'food_energy': 4
161     }
162
163     def __init__(self, time):
164         super().__init__(time=time)
165
166
167 class Penguin(Animal):
168     metadata = {
169         'food': ['Cephalopod'],
170         'new_animal': 1 / 80,
171         'new_animal_energy': 10,
172         'life_expectancy': 90,
```

```
173             'time_for_food': [4, 15],
174             'food_energy': 5
175         }
176
177         def __init__(self, time):
178             super().__init__(time=time)
179
180
181     class Cephalopod:
182         ident = ''
183
184         def die(self, reason, time):
185             pass
186
187
188     class Grass:
189         ident = ''
190
191         def die(self, reason, time):
192             pass
```

File AC08_1_sim.py

```
from AC08_1_classes import Tiger, Elephant, Penguin, Jaguar, \
Grass, Cephalopod

MAX_SIMUL_TIME = 100
MAX_EXPERIMENTS = 10.0


def run_simulation(time, animals, unlimited_food):
    while time < MAX_SIMUL_TIME and len(animals) != 0:
        time = min(animals.values())
        newborns = []
        deaths = []
        animals_with_actions = list(filter(lambda k:
                                           animals[k] == time,
                                           animals.keys()))

        for a in animals_with_actions:

            if a.moment_of_dead == time:  # die
                a.die('old', time)

            if not a.dead and a.next_eating_time == time:  # eat
                a.eat(list(filter(lambda k: not k.dead, animals
                                  .keys()))
                      + unlimited_food, time)

            # born
            if not a.dead and a.next_new_animal_time == time:
                newborn = a.__class__(time)
                a.new_animal(time)
                newborns.append(newborn)

            if a.dead:
                deaths.append(a)
```

```
35              else:
36                  animals.update({a: a.next_action_time()})
37
38          if len(deaths) != 0:
39              for dead in deaths:
40                  del animals[dead]
41
42          if len(newborns) != 0:
43              for nb in newborns:
44                  animals.update({nb: nb.next_action_time()})
45
46
47  def run(ecosystem, statistics):
48      simul_time = []
49      for i in range(int(MAX_EXPERIMENTS)):
50
51          print(i)
52          animals = {}
53          time = 0
54          unlimited_food = [Grass()] * ecosystem[Elephant] * 3 + \
55                           [Cephalopod()] * ecosystem[Penguin] * 5
56          for t, number in ecosystem.items():
57              for _ in range(number):
58                  instance = t(time)
59                  animals.update({instance: instance.next_action_time()})
60              t.statistics['current_num_animals'] = number
61
62          simul_time.append(run_simulation(time, animals, unlimited_food))
63
64          for t in ecosystem.keys():
65              for stat_name, value in t.statistics.items():
66                  if stat_name != 'extinction_time':
67                      statistics[t][stat_name] += value
68
69                  else:
```

```
70                      if value != -1:
71                          statistics[t][stat_name] += value
72                          statistics[t]['extinction'] += 1
73
74      return statistics, simul_time
75
76
77  if __name__ == '__main__':
78      ecosystem = {
79          Tiger: 10,
80          Elephant: 10,
81          Penguin: 10,
82          Jaguar: 10
83      }
84      statistics = {_type: {'extinction': 0,
85                            'extinction_time': 0,
86                            'new_animal': 0,
87                            'death_eaten': 0,
88                            'death_no_energy': 0,
89                            'death_old': 0,
90                            'current_num_animals': 0,
91                            'time_lived': 0,
92                            'time_waited_for_food': 0,
93                            'waited_for_food': 0
94                            } for _type in ecosystem.keys()}
95      # Comienza
96      statistics, simul_time = run(ecosystem, statistics)
97
98      print('------------\nRESULTS\n------------')
99      for _type, stats in statistics.items():
100         print(_type.__name__.upper())
101         print('Times extincted = {0}'.format(stats['extinction']))
102         if stats['extinction'] > 0:
103             print('Average time of specie survival '
104                   'when extincted = {0}'.format(
```

```
105                 stats['extinction_time'] / stats['extinction']))
106         print('Average number of newborns = {0}'.format(
107             stats['new_animal'] / MAX_EXPERIMENTS))
108         print('Average number of {0} eaten by other animals = {1}'
109             .format(_type.__name__, stats['death_eaten']
110                     / MAX_EXPERIMENTS))
111         print('Average number of deaths caused by lack of energy = {0}'
112                 .format(stats['death_no_energy']
113                         / MAX_EXPERIMENTS))
114         print('Average number of deaths caused by aging = {0}'.
115             format(stats['death_old'] / MAX_EXPERIMENTS))
116 
117         if stats['death_eaten'] + stats['death_no_energy'] + \
118                 stats['death_old'] > 0:
119             print('Average age of death = {0}'.format(
120                 stats['time_lived'] /
121                 (stats['death_eaten'] +
122                  stats['death_no_energy'] +
123                  stats['death_old'])))
124 
125         print('Average number of {0} by the end of '
126               'simulation time = {1}'.format(
127             _type.__name__, stats['current_num_animals']
128                             / MAX_EXPERIMENTS))
129 
130         print('Average number of {0} that waited for food = {1}'.format(
131             _type.__name__, stats['waited_for_food']
132                             / MAX_EXPERIMENTS))
133 
134         if stats['waited_for_food'] != 0:
135             print('Average time waited for food = {0}'.format(
136                 stats['time_waited_for_food']
137                 / stats['waited_for_food']))
```

15.16 Solution for activity 9.1: Fixing data

```
import csv


class Student:
    def __init__(self, name, middle_name, last_name):
        self.name = name
        self.middle_name = middle_name
        self.last_name = last_name


class RescueERP:
    def __init__(self, file_name='students.csv'):
        self.students = [student for student in self.reader(file_name)]

    def reader(self, file_name='students.csv'):
        with open(file_name) as file:
            reader = csv.DictReader(file)
            self.headers = reader.fieldnames
            for row in reader:
                name = self.prepare_string(row['name'])
                middle_name = self.prepare_string(row['middle_name'])
                last_name = self.prepare_string(row['last_name'])
                yield (Student(name, middle_name, last_name))

    @classmethod
    def prepare_string(cls, string):
        result = cls.to_upper_case(string)
        result = cls.correct_number_of_ns(result)
        result = cls.remove_number_if_present(result)
        return result

    @classmethod
    def remove_number_if_present(cls, string):
        aux_name = string.split(' ')
```

```
35          if aux_name[0].isnumeric():
36              aux_name = ' '.join(aux_name[1:])
37          else:
38              aux_name = ' '.join(aux_name[:])
39          return aux_name
40
41      @classmethod
42      def correct_number_of_ns(cls, string):
43          aux_text = string.replace('rrr', '##')
44          aux_text = aux_text.replace('rr', 'r')
45          aux_text = aux_text.replace('##', 'rr')
46          return aux_text
47
48      @classmethod
49      def to_upper_case(cls, string):
50          return string.upper()
51
52      def to_latex(self, file_name='students.tex'):
53          out = open(file_name, 'w')
54          # Header file
55          out_t = '\\begin{table}[h]\n\\begin{tabular}{|l|l|l|}\n\\hline\n'
56
57          for h in self.headers:
58              if h == 'name':
59                  out_t += h + '\\\\ \\hline\n'
60              else:
61                  out_t += h + ' & '
62
63          out.write(out_t)
64
65          for register in self.students:
66              out_t = '{0} & {1} & {2} \\\\ \\hline\n' \
67                  .format(register.middle_name, register.last_name,
68                          register.name)
69              out.write(out_t)
```

```
70
71          out_t = '\end{tabular}\n \end{table}\n'
72          out.write(out_t)
73          out.close()
74
75      def to_html(self, file_name='students.html'):
76          out = open(file_name, 'w')
77          # Header archivo
78          out_t = '<table>\n<tr>'
79
80          for h in self.headers:
81              out_t += '<th>{0}</th>'.format(h)
82          out_t += '</tr>'
83          out.write(out_t)
84          for register in self.students:
85              out_t = '<tr>\n<td>{0}</td>\n<td>{1}</td>\n<td>{2}</td>\n</tr>\n' \
86                  .format(register.middle_name, register.last_name, register.name)
87              out.write(out_t)
88
89          out_t = '</table>'
90          out.write(out_t)
91          out.close()
92
93      def to_markdown(self, file_name='students.md'):
94          out = open(file_name, 'w')
95          # Header archivo
96          out_t = '|'
97
98          for h in self.headers:
99              out_t += h + '|'
100         out_t += '\n|------------|--------|---------|\n'
101         out.write(out_t)
102         out_t = ''
103         for register in self.students:
104             out_t = '|{0}|{1}|{2}|\n' \
```

```
105                     .format(register.middle_name, register.last_name, register.name)
106                 out.write(out_t)
107             out.close()
108
109
110 if __name__ == '__main__':
111     rescue_siding = RescueERP()
112     rescue_siding.to_latex()
113     rescue_siding.to_html()
114     rescue_siding.to_markdown()
```

15.17 Solution for activity 9.2: Audio files

```
def create_output_file(name_ouput, sound, byte_array, _help):
    length_bytes = len(byte_array) // 2
    length_in_bytes = length_bytes.to_bytes(4, byteorder='little')
    length_in_bytes_with_header = (length_bytes + 36) \
        .to_bytes(4, byteorder='little')

    with open(name_ouput, 'wb') as split_file:
        split_file.write(_help[0:4])
        split_file.write(length_in_bytes_with_header)
        split_file.write(_help[8:40])
        split_file.write(length_in_bytes)

        for i in range(len(byte_array) // 2):
            byte_audio = byte_array[
                2 * i + int(sound)].to_bytes(1, byteorder='little')
            split_file.write(byte_audio)


with open('music.wav', 'rb') as weird_audio:
    file_header = bytearray(weird_audio.read(44))
    content_bytes = bytearray(weird_audio.read())

create_output_file('song1.wav', True, content_bytes, file_header)
create_output_file('song2.wav', False, content_bytes, file_header)
```

15.18 Solution for activity 11.1: Cashiers' data

```
from PyQt4 import QtCore, QtGui
import datetime
import pickle
import os


class Client:
    def __init__(self, name, id, spent):
        self.name = name
        self.ID = id
        self.accumulated_spent = spent

    def __getstate__(self):
        new = self.__dict__.copy()
        new.update({'last_purchase': str(datetime.datetime)})
        return new

    def __setstate__(self, state):
        self.__dict__ = state

    def update_spent(self, spent):
        self.accumulated_spent += spent


class Cashier(QtGui.QDialog):
    def __init__(self, parent=None, username=''):
        super(Cashier, self).__init__(parent)
        self.setWindowTitle('User {}'.format(username))
        self.button_box = QtGui.QDialogButtonBox(self)
        self.button_box.setOrientation(QtCore.Qt.Horizontal)
        self.button_box.setStandardButtons(
            QtGui.QDialogButtonBox.Cancel | QtGui.QDialogButtonBox.Ok)

        self.client_label = QtGui.QLabel('client name', self)
```

```
35          self.client_text = QtGui.QLineEdit(self)
36          self.id_label = QtGui.QLabel('ID', self)
37          self.id_text = QtGui.QLineEdit(self)
38          self.spent_label = QtGui.QLabel('spent', self)
39          self.spent_text = QtGui.QLineEdit(self)
40
41          self.vertical_layout = QtGui.QVBoxLayout(self)
42          self.vertical_layout.addWidget(self.client_label)
43          self.vertical_layout.addWidget(self.client_text)
44          self.vertical_layout.addWidget(self.id_label)
45          self.vertical_layout.addWidget(self.id_text)
46          self.vertical_layout.addWidget(self.spent_label)
47          self.vertical_layout.addWidget(self.spent_text)
48          self.vertical_layout.addWidget(self.button_box)
49
50          self.button_box.accepted.connect(self.serialize_client)
51          self.button_box.rejected.connect(self.close)
52
53      def serialize_client(self):
54
55          ID_client = self.id_text.text()
56          client_file = str(ID_client) + '.walkcart'
57
58          # We verify if the client exist in the DB
59          if client_file in os.listdir('ClientsDB'):
60
61              # If he exists, we open, change and close
62
63              with open('ClientsDB/' + client_file, 'rb') as file:
64                  existent_client = pickle.load(file)
65
66              existent_client.update_spent(int(self.spent_text.text()))
67
68              with open('ClientsDB/' + client_file, 'wb') as file:
69                  pickle.dump(existent_client, file)
```

```
70
71          else:
72              # He doesn't exist, we create and close
73              new_client = Client(
74                  self.client_text.text(), int(self.id_text.text()), \
75                  int(self.spent_text.text()))
76
77              new_file = 'ClientsDB/' + str(new_client.ID) + '.walkcart'
78
79              with open(new_file, 'wb') as file:
80                  pickle.dump(new_client, file)
81
82          self.client_text.setText('')
83          self.id_text.setText('')
84          self.spent_text.setText('')
85
86
87  class Admin(QtGui.QDialog):
88      def __init__(self, parent=None):
89          super(Admin, self).__init__(parent)
90
91          self.file_button = QtGui.QPushButton('TOP')
92          self.file_button.clicked.connect(self.create_file)
93
94          self.cancel_button = QtGui.QPushButton('Cancel')
95          self.cancel_button.clicked.connect(self.close)
96
97          self.horizontal_layout = QtGui.QVBoxLayout(self)
98          self.horizontal_layout.addWidget(self.file_button)
99          self.horizontal_layout.addWidget(self.cancel_button)
100
101     def create_file(self):
102
103         client_TOP = False
104
```

```
105         for file_ in os.listdir('ClientsDB'):
106             if file_.endswith('.walkcart'):
107                 with open('ClientsDB/' + file_, 'rb') as file:
108                     new_client = pickle.load(file)
109                     if not client_TOP:
110                         client_TOP = new_client
111                     else:
112                         if client_TOP.accumulated_spent <= \
113                                 new_client.accumulated_spent:
114                             client_TOP = new_client
115 
116         top = client_TOP
117         import json
118 
119         with open('TOP.walkcart', 'w') as file_:
120             json.dump(top.__dict__, file_)
121 
122 
123 class Input(QtGui.QWidget):
124     def __init__(self, parent=None):
125         super(Input, self).__init__(parent)
126 
127         self.user_name_text = QtGui.QLineEdit(self)
128 
129         self.push_button_window = QtGui.QPushButton(self)
130         self.push_button_window.setText('Log in')
131         self.push_button_window.clicked.connect(
132             self.on_push_button_clicked)
133 
134         self.layout = QtGui.QHBoxLayout(self)
135         self.layout.addWidget(self.user_name_text)
136         self.layout.addWidget(self.push_button_window)
137 
138     @QtCore.pyqtSlot()
139     def on_push_button_clicked(self):
```

```
140         #####
141
142         # Obtenemos el input dado
143         usuario = self.user_name_text.text()
144
145         # Tomamos la lista de cajeros
146         with open('Files21/cashiers.walkcart', 'rb') as file:
147             authorized_cashiers = pickle.load(file)
148
149         # Identificación de usuario
150         if usuario == 'WalkcartUnlimited':
151             # Interfaz para admin
152             self.user_window = Admin(self)
153             self.hide()
154             self.user_window.show()
155         elif usuario in authorized_cashiers:
156             # Interfaz para cajero
157             self.user_window = Cashier(
158                 self, username=self.user_name_text.text())
159             self.hide()
160             self.user_window.show()
161
162
163 if __name__ == '__main__':
164     import sys
165
166     app = QtGui.QApplication(sys.argv)
167     app.setApplicationName('Log-in WM')
168
169     main = Input()
170     main.show()
171
172     sys.exit(app.exec_())
```