Practice Sheet

Problem P.1 (3 points)

Write a program that prints to the screen all even numbers and also the numbers that are divisible by 3 for all numbers between 1 and 200 (inclusive). The program should then print on the screen the following:

```
2
3
4
6
8
9
10
...
```

Solution:

```python
# Problem P.1

for i in range(1, 201):
  if i % 2 == 0 or i % 3 == 0:
    print(i)
```

Problem P.2 (5 points)

Write a program which reads in 6 integers from the keyboard and then prints these numbers and their square in the opposite order of their input to a file named `squares.txt`.
So if you enter

```
1
2
3
4
5
6
```

your output should look like this then:

```
6 36
5 25
4 16
3 9
2 4
1 1
```

You will lose points if you are using separate variables for each input.

Solution:

```python
# Problem P.2

f = open('squares.txt', 'w')

input_lst = [int(input()) for _ in range(6)]
input_lst.reverse()

for i in input_lst:
  f.write('{} {}\n'.format(i, i**2))

f.close()
```

Problem P.3 (3 points)

Please write a program where you first ask for an integer `n`. The program then prints `n` rows with the pattern below using nested loops.
So if you enter 6, 6 rows will be printed:

```
A
AB
ABC
ABCD
ABCDE
ABCDEF
```

Solution:

```python
# Problem P.3

n = int(input())

for i in range(n+1):
  for j in range(i):
    print(chr(ord('A')+j), end='')
  if i > 0:
    print()
```

Problem P.4 (4 points)

Please write a program that converts feet and inches to meters.
First write the definition of a function `to_meters(foot, inch)` that converts feet and inches to one single length meters.
Then write a short test program that repeatedly reads feet and inches from the keyboard and converts the lengths using the function
`to_meters(foot, inch)` and prints the result from outside the function. If **both** inputs are `0` then the program quits without printing anything.
Keep in mind that 1 foot = 30.5 *cm and* 1 *inch* = 2.54 *cm.*

Solution:

```python
# Problem P.4

def to_meters(foot, inch):
  """Converts feet and inches to meters"""
  return foot*30.5 + inch*2.54

# Driver code
while True:
  foot, inch = input('Insert feet and inches (separated by a space): ').split()
  foot, inch = float(foot), float(inch)
  if foot == 0 and inch == 0:
    break
  else:
    print(to_meters(foot, inch))
```

Problem P.5 (5 points)

It is suggested that passwords mix letters and numbers, so passwords are not that easy to guess.
Here, a good password must have at least 8 characters and needs to contain at least three numbers.
Please write a password checker where you can **repeatedly** enter passwords. If the string is empty, the program quits. After each password entered, your program determines whether the password is good or not, and either prints "PASSWORD IS GOOD" or "PASSWORD IS BAD" to the screen.

Solution:

```
# Problem P.5

import re

while True:
  passwd = input('Enter your password: ')

  # find all the digits
  nums = re.findall('\d', passwd)

  # quit if nothing is entered
  if len(passwd) == 0:
    break
  elif len(passwd) >= 8 and len(nums) >= 3:
    print('PASSWORD IS GOOD')
  else:
    print('PASSWORD IS BAD')
```

Problem P.6 (4 points)

Write the definition of the following function

`substitute_vowels(str, ch)`

A vowel is one of the letters `a, e, i, o, u` (it is sufficient to just substitute lowercase characters). The function takes a string and replaces all vowels with the given character `ch`. The function returns a string with the replacements done.
*You may not **use** the `replace` method to solve this assignment.*
Thus the following (incomplete) piece of code

```
s = "This is a sentence"
print(s)
n = substitute_vowels(s, 'o')
print(n)
```

will print

```
Thos os o sontonco
```

Solution:

```
# Problem P.6

def substitute_vowels(str, ch):
  """Substitutes all vowels in str by the argument ch"""
  vowels = 'aeiou'
  out = ''
  for char in str:
    if char in vowels:
      out += ch
    else:
      out += char
  return out

# Driver code
s = "This is a sentence"
print(s)
n = substitute_vowels(s, 'o')
print(n)
```

Problem P.7 (3 points)

A store sells pens. Each pen costs 45 cents. However, for more than 50 pens, the price of all pens drops to 38 cent per item, and for more than 100 pens you will just need to pay 30 cent per pen.
Please write a program where you can repeatedly enter the number of pens you would like to buy. If you enter a negative number the program should stop. Your program should compute the price that is to be paid and display this price in Euros and Cent on the screen.

Solution:

```
# Problem P.7

while True:
  num_pens = int(input())

  if num_pens < 0:
    break
  elif 0 <= num_pens <= 50:
    print('Price: €{:.2f}'.format(num_pens*45/100))
  elif 51 <= num_pens <= 100:
    print('Price: €{:.2f}'.format(num_pens*38/100))
  elif num_pens > 100:
    print('Price: €{:.2f}'.format(num_pens*30/100))
```

Problem P.8 (3 points)

Write a function that returns the smallest value of a list of integers. You have free choice for the return type and the parameters, but do not use global values in your function.
You may not use any `sort` *method in this problem.*

Solution:

```
# Problem P.8

def find_smallest_int(lst):
  smallest = 0
  for elem in lst:
    if elem < smallest:
      smallest = elem
  return smallest

# Driver code
lst = [54, 23, 764, 2, -12, 466, 0, 23464, 43]
print(find_smallest_int(lst))
```

Problem P.9 *Formatting table* (1 points)

Write a conversion program that converts miles to meter. The result should be printed in a well formatted table. Three values will be read from standard input (the keyboard): first the start length, then the end length and the step size. The lengths should be printed with a decimal precision of three. The alignment of the columns should be to the right.

```
  mile          m
  10.0  16093.440
  20.0  32186.880
  30.0  48280.320
  40.0  64373.760
```

Note that 1 *mile* = 1609.344 *m*

Solution:

```
# Problem P.9

start, end, step = input('Enter start, end and step size: ').split()
start, end, step = float(start), float(end), float(step)

# calculate the total number of steps
steps = int((end-start)/step + 1)

# print the header
print('{:>6} {:>11}'.format('mile', 'm'))

# print the table
for i in range(steps):
  x = start + step*i
  print('{:>6.1f} {:>11.3f}'.format(x, x*1609.344))
```

Problem P.10 *Formatting table with files* (2 points)

Change the program in such a way that you ask for a filename as fourth input. If the input is empty, output is written to standard output, otherwise the output is written to the given file.

Solution:

```
# Problem P.10

# read the input
args = input('Enter start, end, step, filename: ').split()
start, end, step = float(args[0]), float(args[1]), float(args[2])

if len(args) == 4:
  filename = args[3]
else:
  filename = None

# calculate the total number of steps
steps = int((end-start)/step + 1)

# write the header
out = '{:>6} {:>11}\n'.format('mile', 'm')

# write the table
for i in range(steps):
  x = start + step*i
  out += '{:>6.1f} {:>11.3f}\n'.format(x, x*1609.344)

# write to stdout or file
if not filename:
  print(out)
else:
  f = open(filename, 'w')
  f.write(out)
  f.close()
```

Problem P.11 *Files and exceptions* (2 points)

Write a program which ask for a filename (which contains two integers separated by space) from the keyboard and print into another file called "`division.txt`" the result of dividing the first integer by the second one. Handle the exception cases to inform the user if the opening of one of the files fails or a division by zero occurs.

Solution:

```
# Problem P.11

filename = input('Enter filename: ')

try:
  # open files
  in_file = open(filename, 'r')
  out_file = open('division.txt', 'w')

  # extract data
  in_str = in_file.read().split()
  int1, int2 = int(in_str[0]), int(in_str[1])

  # compute quotient and write to file
  result = str(int1/int2)
  out_file.write(result)

except OSError:
  print('Unable to open one of the files')

except ZeroDivisionError:
  msg = 'Division by zero is not allowed'
  out_file.write(msg)
  print(msg)

else:
  in_file.close()
  out_file.close()
```

Problem P.12 *A stack* (3 points)

Write two alternative implementations of a stack using a) list and b) deque. A stack is a LIFO (Last-In, First-Out) data structure. You need to implement the methods `push(...)`, `pop(...)` and `is_empty(...)` for adding an element to the top of the stack, removing and returning the element from the the top of the stack, and checking if the stack is empty or not.

Input structure

There are several commands that manipulate the stack which are send to standard input. These commands manipulate the stack:

- `s <number>` pushes a number onto the stack
- `p` pops a number off the stack and prints it on the screen
- `e` empties the stack by popping one element after the other
- `q` quits the program

Solution:

```
# Problem P.12

from collections import deque

class Stack(object):
  """Implementation of a stack using a list or deque"""

  def __init__(self, lst=[], type='list'):
    if type == 'list':
      self.container = lst
    else:
      self.container = deque(lst)

  def push(self, num):
    """Push an element into the stack"""
    out = self.container.append(num)

  def pop(self):
    """Pop an element from the stack"""
    out = self.container.pop()
    return out

  def is_empty(self):
    """Return true if the stack is empty"""
    if len(self.container) == 0:
      return True
    else:
      return False


# Driver code

# create a list based stack
# for a deque based stack, use type='deque'
stack = Stack(type='list')

while True:
  command = input('> ')
  if command[0] == 's':
    stack.push(float(command[1:]))
  elif command == 'p':
    if not stack.is_empty():
      print(stack.pop())
  elif command == 'e':
    while not stack.is_empty():
      print(stack.pop())
  elif command == 'q':
    break
```

Problem P.13 *A queue* (3 points)

Write two alternative implementations of a queue using a) list and b) deque. While a stack is a LIFO (Last-In, First-Out) data structure, a queue is a FIFO (First-In, First-Out). You need to support `push(...)`, `pop(...)` and `is_empty(...)` for adding an element to the end of the queue, removing and returning the element from the beginning of the queue, and checking if the queue is empty or not. The input structure is the same as above as in the previous problem.

Solution:

```
# Problem P.13

from collections import deque

class Queue(object):
  """Implementation of a queue using a list or deque"""

  def __init__(self, lst=[], type='list'):
    self.type = type
    if self.type == 'list':
      self.container = lst
    else:
      self.container = deque(lst)

  def push(self, num):
    """Push an element into the stack"""

    out = self.container.append(num)

  def pop(self):
    """Pop an element from the stack"""

    if self.type == 'list':
      out = self.container.pop(0)
    else:
      out = self.container.popleft()
    return out

  def is_empty(self):
    """Return true if the stack is empty"""

    if len(self.container) == 0:
      return True
    else:
      return False


# Driver code

# create a list based queue
# for a deque based queue, use type='deque'
queue = Queue(type='list')

while True:
  command = input('> ')
  if command[0] == 's':
    queue.push(float(command[1:]))
  elif command == 'p':
    if not queue.is_empty():
      print(queue.pop())
  elif command == 'e':
    while not queue.is_empty():
      print(queue.pop())
  elif command == 'q':
    break
```

Problem P.14 *Class constructor* (2 points)

Write a class `Person` for modeling a person entity. You should be able to set the first name, last name and the profession of the person.

Solution:

```
# Problem P.14

class Person(object):

  def __init__(self, fname, lname, prof):
    self._fname = fname
    self._lname = lname
    self._prof = prof

  def set_first_name(self, fname):
    self._fname = fname

  def set_last_name(self, lname):
    self._lname= lname

  def set_profession(self, prof):
    self._lname= prof

# Driver code
person = Person('Steve', 'Jobs', 'CEO')
person.set_first_name('Tim')
person.set_last_name('Cook')
```

Problem P.15 *Class operators* (2 points)

Extend the previous program by adding methods to the previous `Person` class, such that two person can be compared based on their names according to alphabetical order. One method should test for equality, further methods need to support the remaining relations (alphabetically < and >). If two persons have the same lastname then the alphabetical order of their firstname should count. Write a simple test program to check the correctness of your methods.

Solution:

```
# Problem P.15

class Person(object):

  def __init__(self, fname, lname, prof):
    self.fname = fname
    self.lname = lname
    self.prof = prof

  def set_first_name(self, fname):
    self.fname = fname

  def set_last_name(self, lname):
    self.lname= lname

  def set_profession(self, prof):
    self.lname= prof

  def __eq__(self, other):
    if self is other:
      return True
    elif type(self) != type(other):
      return False
    else:
      return self.fname == other.fname and self.lname == other.lname

  def __lt__(self, other):
    if self.lname == other.lname:
      return self.fname < other.fname
    else:
      return self.lname < other.lname

  def __gt__(self, other):
    if self.lname == other.lname:
      return self.fname > other.fname
    else:
      return self.lname > other.lname

  def __str__(self):
    return self.fname + ' ' + self.lname


# Driver code

def compare(first, second):
  if first == second:
    print('{} and {} are ordered equally\n'.format(first, second))
  elif first < second:
    print('Names in alphabetical order')
    print('1. {}'.format(first))
    print('2. {}\n'.format(second))
  elif first > second:
    print('Names in alphabetical order')
    print('1. {}'.format(second))
    print('2. {}\n'.format(first))

# case of comparing only last names
steve = Person('Steve', 'Jobs', 'former CEO')
jobs = Person('Tim', 'Cook', 'CEO')
# case of comparing first names
alice = Person('Alice', 'Doe', 'former CEO')
bob = Person('Bob', 'Doe', 'CEO')
# case of equivalent ordering
john = Person('John', 'Doe', 'former CEO')
jane = Person('Jane', 'Doe', 'CEO')

compare(steve, jobs)
compare(alice, bob)
compare(john, jane)
```

Problem P.16 *List of objects* (2 points)

Extend the previous program such that it that creates 10 person objects and adds them to a list. Then the list should be shuffled and printed. Then use the **sort** method to reorder the persons alphabetically based on their names.

Solution:

```
# Problem P.16

import random

class Person(object):

  def __init__(self, fname, lname, prof):
    self.fname = fname
    self.lname = lname
    self.prof = prof

  def set_first_name(self, fname):
    self.fname = fname

  def set_last_name(self, lname):
    self.lname= lname

  def set_profession(self, prof):
    self.lname= prof

  def __eq__(self, other):
    if self is other:
      return True
    elif type(self) != type(other):
      return False
    else:
      return self.fname == other.fname and self.lname == other.lname

  def __lt__(self, other):
    if self.lname == other.lname:
      return self.fname < other.fname
    else:
      return self.lname < other.lname

  def __gt__(self, other):
    if self.lname == other.lname:
      return self.fname > other.fname
    else:
      return self.lname > other.lname

  def __str__(self):
    return self.fname + ' ' + self.lname


# Driver code
students = []
first_names = ['Aaliyah', 'Benjamin', 'Camilla', 'Damien', 'Edith',
     'Fernando', 'Grace', 'Hector', 'Irene', 'Julius']
last_names = ['Eberhardt', 'Eberhardt', 'O\'Brian', 'O\'Brian', 'Raczkowski',
        'Raczkowski', 'Verghese', 'Verghese', 'Zimmer', 'Zimmer']

for i in range(10):
  students.append(Person(first_names[i], last_names[i], 'Student'))

print('Shuffled Students')
random.shuffle(students)
for idx, student in enumerate(students):
  print(idx+1, student)

print('\nSorted Students')
students.sort()
for idx, student in enumerate(students):
  print(idx+1, student)
```

Problem P.17 *Save objects* (2 points)

Write a program that saves the list of persons created in the previous program into the output file "`persons.dat`". Use the pickle module to "pickle" objects before writing to the file.

Solution:

```
# Problem P.17

import pickle

class Person(object):

  def __init__(self, fname, lname, prof):
    self.fname = fname
    self.lname = lname
    self.prof = prof

  def set_first_name(self, fname):
    self.fname = fname

  def set_last_name(self, lname):
    self.lname= lname

  def set_profession(self, prof):
    self.lname= prof

  def __eq__(self, other):
    if self is other:
      return True
    elif type(self) != type(other):
      return False
    else:
      return self.fname == other.fname and self.lname == other.lname

  def __lt__(self, other):
    if self.lname == other.lname:
      return self.fname < other.fname
    else:
      return self.lname < other.lname

  def __gt__(self, other):
    if self.lname == other.lname:
      return self.fname > other.fname
    else:
      return self.lname > other.lname

  def __str__(self):
    return self.fname + ' ' + self.lname


# Driver code
students = []
first_names = ['Aaliyah', 'Benjamin', 'Camilla', 'Damien', 'Edith',
     'Fernando', 'Grace', 'Hector', 'Irene', 'Julius']
last_names = ['Eberhardt', 'Eberhardt', 'O\'Brian', 'O\'Brian', 'Raczkowski',
        'Raczkowski', 'Verghese', 'Verghese', 'Zimmer', 'Zimmer']

for i in range(10):
  students.append(Person(first_names[i], last_names[i], 'Student'))

f = open('persons.dat', 'wb')

for student in students:
  pickle.dump(student, f)

f.close()
```

Problem P.18 *Load objects* (2 points)

Write a program that loads a list of persons from the input file "`persons.dat`". Use the pickle module to "unpickle" the data.

Solution:

```
# Problem P.18

import pickle

class Person(object):

  def __init__(self, fname, lname, prof):
    self.fname = fname
    self.lname = lname
    self.prof = prof

  def set_first_name(self, fname):
    self.fname = fname

  def set_last_name(self, lname):
    self.lname= lname

  def set_profession(self, prof):
    self.lname= prof

  def __eq__(self, other):
    if self is other:
      return True
    elif type(self) != type(other):
      return False
    else:
      return self.fname == other.fname and self.lname == other.lname

  def __lt__(self, other):
    if self.lname == other.lname:
      return self.fname < other.fname
    else:
      return self.lname < other.lname

  def __gt__(self, other):
    if self.lname == other.lname:
      return self.fname > other.fname
    else:
      return self.lname > other.lname

  def __str__(self):
    return self.fname + ' ' + self.lname


# Driver code
students = []

f = open('persons.dat', 'rb')

while True:
  try:
    student = pickle.load(f)
    students.append(student)
  except EOFError:
    f.close()
    break

print('Students loaded from file:\n')
for student in students:
  print(student)

f.close()
```

Problem P.19 *Add and subtract objects* (3 points)

Write a class called `Rectangle` which has two properties `length` and `width`. Write a constructor for the class and overload the + operator and the − operator such that you can add and subtract two rectangle objects by adding and subtracting their corresponding areas. Create a few rectangle objects, add them, subtract them and print the results of the addition and subtraction on the screen. Make sure that on bad input exceptions are raised and/or caught (i.e., strings as input or negative values as input for the length and width). In the case of an exception the program should exit its execution.

Solution:

```
# Problem P.19

class Rectangle(object):

  def __init__(self, length, width):
    types = [int, float]

    if type(length) in types:
      self.length = length
    else:
      raise TypeError('Expecting a float or int for \'length\' '
              'but got a {}'.format(type(length).__name__))

    if type(width) in types:
      self.width = width
    else:
      raise TypeError('Expecting a float or int for \'width\' '
              'but got a {}'.format(type(width).__name__))

  def __add__(self, other):
    """Adds up the areas of two triangles"""
    return (self.length*self.width) + (other.length*other.width)

  def __sub__(self, other):
    """Subtracts the area of the other triangle from the self triangle"""
    return (self.length*self.width) - (other.length*other.width)

# Driver code

try:
  A = Rectangle(5, 6)
  B = Rectangle(10, 8)

  print('Area of B + A is {}'.format(B + A))
  print('Area of B - A is {}'.format(B - A))

  C = Rectangle('8', '9')

  print('Area of C - A is {}'.format(C - A))

except TypeError as e:
  print(e)
  exit()
```

Problem P.20 *Draw a house* (2 points)

Write a program which draws a simple house consisting of at least two windows, one door and a roof using different colors.

Solution:

```
# Problem P.20

from graphics import *

def main():
    win = GraphWin()
    window1 = Rectangle(Point(50, 120), Point(80,150))
    window2 = window1.clone()
    window2.move(85, 0)
    pane_x1 = Line(Point(50, 135), Point(80, 135))
    pane_y1 = Line(Point(65, 120), Point(65, 150))
    pane_x2 = pane_x1.clone()
    pane_y2 = pane_y1.clone()
    pane_x2.move(85, 0)
    pane_y2.move(85, 0)
    door = Rectangle(Point(95, 130), Point(120, 180))
    base = Rectangle(Point(40, 65), Point(175, 180))
    roof = Polygon(Point(40,65), Point(107.5, 30) ,Point(175, 65))

    window1.setOutline("black")
    window2.setOutline("black")
    door.setOutline("black")
    base.setOutline("black")
    roof.setOutline("black")
    pane_x1.setOutline("black")
    pane_y1.setOutline("black")
    pane_x2.setOutline("black")
    pane_y2.setOutline("black")

    window1.setFill("white")
    window2.setFill("white")
    door.setFill("maroon")
    base.setFill("red")
    roof.setFill("brown")
    pane_x1.setFill("black")
    pane_y1.setOutline("black")
    pane_x2.setFill("black")
    pane_y2.setOutline("black")

    base.draw(win)
    roof.draw(win)
    window1.draw(win)
    window2.draw(win)
    pane_x1.draw(win)
    pane_y1.draw(win)
    pane_x2.draw(win)
    pane_y2.draw(win)
    door.draw(win)

    win.getMouse()
    win.close()

main()
```

Problem P.21 *Draw a house by clicks* (2 points)

Change your previous program such that every component of the figure (i.e., polygon, rectangle, lines) appears on the screen one by one at a mouse click.

Solution:

```
# Problem P.21

from graphics import *

def main():
  win = GraphWin()
  window1 = Rectangle(Point(50, 120), Point(80,150))
  window2 = window1.clone()
  window2.move(85, 0)
  pane_x1 = Line(Point(50, 135), Point(80, 135))
  pane_y1 = Line(Point(65, 120), Point(65, 150))
  pane_x2 = pane_x1.clone()
  pane_y2 = pane_y1.clone()
  pane_x2.move(85, 0)
  pane_y2.move(85, 0)
  door = Rectangle(Point(95, 130), Point(120, 180))
  base = Rectangle(Point(40, 65), Point(175, 180))
  roof = Polygon(Point(40,65), Point(107.5, 30) ,Point(175, 65))

  window1.setOutline("black")
  window2.setOutline("black")
  door.setOutline("black")
  base.setOutline("black")
  roof.setOutline("black")
  pane_x1.setOutline("black")
  pane_y1.setOutline("black")
  pane_x2.setOutline("black")
  pane_y2.setOutline("black")

  window1.setFill("white")
  window2.setFill("white")
  door.setFill("maroon")
  base.setFill("red")
  roof.setFill("brown")
  pane_x1.setFill("black")
  pane_y1.setOutline("black")
  pane_x2.setFill("black")
  pane_y2.setOutline("black")

  obj_lst = [base, roof, window1, window2,
        pane_x1, pane_y1, pane_x2, pane_y2, door]

  for obj in obj_lst:
    win.getMouse()
    obj.draw(win)

  win.getMouse()
  win.close()

main()
```

Problem P.22 *Interactive graphics* (3 points)

Write a program which creates a window with the size of your choice. Then the program should add an entry field in one of the corners of the window and an exit button (rectangle with the text "`exit`" inside) in another corner. The program should run as follows: the user enters a numerical value into the entry field, then the user clicks somewhere in the window. This click point is taken as center for a drawn circle with the radius taken from the entry field. Finally, if the user clicks on the exit button the program should exit (i.e., terminate execution).

Solution:

```
# Problem P.22

from graphics import *

def main():
    win = GraphWin("Draw Circle", 300, 300)
    win.setCoords(0.0, 0.0, 4.0, 4.0)

    # Draw the background
    background = Rectangle(Point(-0.1,-0.1), Point(4.1, 4.1))
    background.setFill('white')
    background.setOutline('white')
    background.draw(win)

    # Draw the interface
    input = Entry(Point(1, 0.25), 5)
    input.setText("0.0")
    input.draw(win)

    while True:
        radius_label = Text(Point(0.35, 0.25), "Radius: ")
        button_label = Text(Point(3.65, 0.25), "exit")
        button = Rectangle(Point(3.4, 0.4), Point(3.9, 0.125))
        button.setFill('gray')
        button.setOutline('black')
        button.draw(win)
        radius_label.draw(win)
        button_label.draw(win)

        # wait for click and then quit
        mouse_click = win.getMouse()

        if 3.4 < mouse_click.getX() < 3.9 and 0.125 < mouse_click.getY() < 0.4:
            win.close()
            break
        else:
            circle = Circle(mouse_click, eval(input.getText()))
            circle.setFill('Yellow')
            circle.draw(win)

main()
```

Problem P.23 *CSV data generation* (2 points)

Write a program which generates a CSV file with random values related to article data with the following data and order: ID, price, count, storage ID, order count. Consider `int` and `float` data when generating the data. Generate 1000 entries formatted as a `csv` file.

Solution:

```
# Problem P.23

import random
import csv

f = open('data.csv', 'w')

writer = csv.writer(f)
writer.writerow(['ID', 'price', 'count', 'storage ID', 'order count'])

for i in range(1000):
  ID = random.randint(100000,999999)
  price = float('{:.2f}'.format(random.uniform(0.1, 1000)))
  count = random.randint(1,100)
  storage_ID = random.randint(50000,60000)
  order_count = random.randint(0,100)
  writer.writerow([ID, price, count, storage_ID, order_count])

f.close()
```

Problem P.24 *CSV data extraction* (3 points)

Use the previously generated `csv` file and use the `csv` module to extract and write into a file called "`result.csv`" all the entries which are cheaper than `10.0` (unit of money, e.g., Euros).

Solution:

```
# Problem P.24

import random
import csv

in_file = open('data.csv', 'r')
out_file = open('result.csv', 'w')

reader = csv.reader(in_file)
writer = csv.writer(out_file)

# write the header
writer.writerow(reader.__next__())

for entry in reader:
  if float(entry[1]) < 10.0:
    writer.writerow(entry)

in_file.close()
out_file.close()
```

Problem P.25 *Problem variations* (2 points)

Consider any of the previous problems in the following format: some incomplete code is given and you have to extend it to work or to reach some specified goal.

Solution:

Consider **Problem P.5**, but this time you are only allowed to check that a password contains 8 characters and has at least three numbers by using the `re.search(pattern, string)` method as a condition in an `if` statement. The only other condition you are allowed to check for is that a password has been entered (and quit in case nothing has been entered).

To see the implementation, go to the next page

```python
# Problem P.25

import re

while True:
  passwd = input('Enter your password: ')

  # quit if nothing is entered
  if len(passwd) == 0:
    break
  elif re.search('(?=(?:.*\d){3,})^.{8,}', passwd):
    print('PASSWORD IS GOOD')
  else:
    print('PASSWORD IS BAD')
```

Explanation:
The regular expression `(?=(?:.*\d){3,})^.{8,}$` can be broken down into the following components:

- `(?=(?:.*\d){3,})` Positive look-ahead that matches at least 3 digits
- `^.{8,}$` Match any character that appears at least 8 times from the start to the end of the string