**1. string\_formatting\_examples.py**

name = **"Gibson L-5 CES"**year = 1922  
cost = 16035.4  
  
*# The ‘old’ manual way to format text with string concatenation:*print(**"My guitar: "** + name + **", first made in "** + str(year))  
  
*# A better way - using str.format():*print(**"My guitar: {}, first made in {}"**.format(name, year))  
print(**"My guitar: {0}, first made in {1}"**.format(name, year))  
print(**"My {0} was first made in {1} (that's right, {1}!)"**.format(name, year))  
  
*# Formatting currency (grouping with comma, 2 decimal places):*print(**"My {} would cost ${:,.2f}"**.format(name, cost))  
  
*# Aligning columns:*numbers = [1, 19, 123, 456, -25]  
for number in numbers:  
 print(**"Number is {:>5}"**.format(number))  
  
*# A version of the above loop using the enumerate function, useful when you want the index and value*for i, number in enumerate(numbers):  
 print(**"Number {0} is {1:>5}"**.format(i + 1, number))  
  
*# TODO: Use string formatting to produce the output:  
# 1922 Gibson L-5 CES for about $16,035!*print(**"**\n**{} {} for about ${:,.0f}!**\n**"**.format(year, name, cost))  
  
*# TODO: Using a for loop with the range function and string formatting,  
# produce the following right-aligned output (do not use a list):  
# 0  
# 50  
# 100  
# 150*for num in range(0, 200, 50):  
 print(**"{:>3}"**.format(num))

**2. randoms.py**

import random  
  
print(random.randint(5, 20)) *# line 1*print(random.randrange(3, 10, 2)) *# line 2*print(random.uniform(2.5, 5.5)) *# line 3  
# On line 1, I saw a number between 5 and 20. The smallest number I could have seen is 5, and the largest number is 20  
  
# On line 2, I saw either 3, 5, 7, or 9. Line 2 cannot produces a 4 because the step is 2 from 3 onwards.  
  
# On line 3, I saw decimal number between 2.5 and 5.5. The smallest possible number is 2.5, and the largest is 5.5.  
  
# Produce random number between 1 and 100 inclusive*print(random.randint(1, 100))

**3. capitalist\_conrad.py**

*"""  
CP1404/CP5632 - Practical  
Capitalist Conrad wants a stock price simulator for a volatile stock.  
The price starts off at $10.00, and, at the end of every day there is  
a 50% chance it increases by 0 to 10%, and  
a 50% chance that it decreases by 0 to 5%.  
If the price rises above $1000, or falls below $0.01, the program should end.  
The price should be displayed to the nearest cent (e.g. $33.59, not $33.5918232901)  
"""*import random  
  
MAX\_INCREASE = 0.175 *# 17.5%*MAX\_DECREASE = 0.05 *# 5%*MIN\_PRICE = 1.0  
MAX\_PRICE = 100.0  
INITIAL\_PRICE = 10.0  
  
out\_file = open(**"OUTPUT\_FILE.txt"**, **"w"**) *# write the output into a file called OUTPUT\_FILE*price = INITIAL\_PRICE  
print(**"${:,.2f}"**.format(price), file=out\_file)  
day = 0 *# for day counter. Day start with 0 at the beginning*while price >= MIN\_PRICE and price <= MAX\_PRICE:  
 price\_change = 0  
 *# generate a random integer of 1 or 2  
 # if it's 1, the price increases, otherwise it decreases* if random.randint(1, 2) == 1:  
 *# generate a random floating-point number  
 # between 0 and MAX\_INCREASE* price\_change = random.uniform(0, MAX\_INCREASE)  
 day = day + 1 *# day counter* else:  
 *# generate a random floating-point number  
 # between negative MAX\_DECREASE and 0* price\_change = random.uniform(-MAX\_DECREASE, 0)  
 day = day + 1 *# day counter* price \*= (1 + price\_change)  
 print(**"On day {} price is: ${:,.2f}"**.format(day, price), file=out\_file)  
  
out\_file.close() *# close file*

4**. exceptions\_demo.py**

try:  
 numerator = int(input(**"Enter the numerator: "**))  
 denominator = int(input(**"Enter the denominator: "**))  
 *# While loop to validate denominator. User cannot enter 0 for the denominator* while denominator == 0:  
 print(**"Denominator cannot be 0! Please enter again."**)  
 denominator = int(input(**"Enter the denominator: "**))  
  
 fraction = numerator / denominator  
 print(fraction)  
except ValueError:  
 print(**"Numerator and denominator must be valid numbers!"**)  
except ZeroDivisionError:  
 print(**"Cannot divide by zero!"**)  
print(**"Finished."**)  
  
*# 1. When will a ValueError occur?  
# The ValueError happens when the user enter decimal numbers or letters  
  
# 2. When will a ZeroDivisionError occur?  
# This error happens when the user enter 0 for the denominator input  
  
# 3. Could you change the code to avoid the possibility of a ZeroDivisionError?  
# Putting a while loop that will not break until the user enter a valid denominator  
# Changes to the code has been made*

**5. exceptions\_to\_complete.py**

finished = False  
result = 0  
while not finished:  
 try:  
 result = int(input(**"Enter a number: "**))  
 finished = True  
 except ValueError:   
 print(**"Please enter a valid integer."**)  
print(**"Valid result is:"**, result)

6**. files.py**

*# Question 1:*name\_file = open(**"name.txt"**, **"w"**) *# create a .txt and write on it*user\_name = input(**"Enter your name: "**) *# get user's input*print(user\_name, file=name\_file) *# write the input into the .txt file*name\_file.close() *# close the .txt file so that it save our progress  
  
# Question 2:*name = open(**"name.txt"**, **"r"**) *# open the .txt file, read only*sentence = **"Your name is "** + name.read() *# create the statement with the file's content*print(sentence) *# print the statement  
  
# Question 3:*number = open(**"numbers.txt"**, **"r"**) *# open the text file, read only*first\_num = number.readline() *# read the first line*second\_num = number.readline() *# read the second line  
# add up the first two line*two\_line\_total = int(first\_num) + int(second\_num) *# Using int() because readline() returns a string  
# print out the sum*print(**"The sum of the first two line is:"**, two\_line\_total)  
  
*# Question 4:*total = 0  
numbers\_file = open(**"numbers.txt"**, **"r"**) *# open the text file, read only*for each\_line in numbers\_file: *# read one line at a time  
 # add up all the lines* total = total + int(each\_line) *# Using int() because readline() returns a string  
# print out the total of all lines*print(**"**\n**The total of all numbers in the text file is:"**, total)

7**. password\_checker.py**

MIN\_LENGTH = 5  
MAX\_LENGTH = 15  
SPECIAL\_CHARS\_REQUIRED = False  
SPECIAL\_CHARACTERS = **"!@#$%^&\*()\_-=+`~,./'[]<>?{}|**\\**"**def main():  
 *"""Program to get and check a user's password."""* print(**"Please enter a valid password"**)  
 print(**"Your password must be between"**, MIN\_LENGTH, **"and"**, MAX\_LENGTH,  
 **"characters, and contain:"**)  
 print(**"**\t**1 or more uppercase characters"**)  
 print(**"**\t**1 or more lowercase characters"**)  
 print(**"**\t**1 or more numbers"**)  
 if SPECIAL\_CHARS\_REQUIRED:  
 print(**"**\t**and 1 or more special characters: "**, SPECIAL\_CHARACTERS)  
 password = input(**"> "**)  
 while not is\_valid\_password(password):  
 print(**"Invalid password!"**)  
 password = input(**"> "**)  
 print(**"Your {}-character password is valid: {}"**.format(len(password),  
 password))  
  
  
def is\_valid\_password(password):  
 *"""Determine if the provided password is valid."""  
 # return False if the password has the wrong length* if len(password) < MIN\_LENGTH or len(password) > MAX\_LENGTH:  
 return False  
  
 count\_lower = 0  
 count\_upper = 0  
 count\_digit = 0  
 count\_special = 0  
 *# check every character in password* for char in password:  
 *# count lower case characters* if char.islower():  
 count\_lower = count\_lower + 1  
 *# count upper case characters* elif char.isupper():  
 count\_upper = count\_upper + 1  
 *# count numbers* else:  
 count\_digit = count\_digit + 1  
  
 *# return False if the password has no number, lower and upper case character* if count\_lower == 0 or count\_upper == 0 or count\_digit == 0:  
 return False  
  
 *# check if special character are in password or not, and count special characters* for char in password:  
 if char in SPECIAL\_CHARACTERS:  
 count\_special = count\_special + 1  
 else:  
 count\_special = count\_special  
  
 *# return False if the password has no special character* if count\_special == 0:  
 return False  
  
 *# if we get here (without returning False), then the password must be valid* return True  
  
  
main()