Computer Science 1-Honors	**Lab 13B** **Practice/Perform Major Python Assignment**
The “Mean, Median and Mode” Program	**80, 90, 100 & 110 Point Versions**

Assignment Purpose:

This program requires knowledge of creating and manipulating arrays. Additionally, it requires an understanding of the three types of "averages."

Mathematics recognizes three levels of "centrality" or averages. They are called the *mean*, the *median* and the *mode*. The *average* most people are familiar with is the *mean*.

The *mean* is computed by adding up the sum of all the numbers and then dividing by the count of the numbers. This type of average is normally used to find things like your homework average, your quiz average, your test average, etc.

The *median* is the middle number in a list of numbers. This type of average is frequently used to find the average home value in a city or neighborhood. Using the *mean* can distort the average. It is possible that an extremely expensive home is included. With the mean computation the expensive home will raise the average to an unrealistic average.

The *mode* is the number that occurs the most frequently in a list of numbers.

Lab 13B Student Version	**Do not copy this file, which is provided.**

```
# Lab13Bst.py
# "Mean, Median and Mode"
# This is the student, starting version of Lab 13B.
# NOTE: This lab is meant for students in CS1-HONORS.
#       Students in REGULAR CS1 will do Lab 13A.


from random import *


sum = "shadowed"
# prevents the use of this built-in function


def heading():
   print()
   print("**********************************")
   print("Lab 13B, Mean, Median and Mode")
   print("80 Point Version")
   print("By: JOHN SMITH") # Substitute your own name here.
   print("**********************************")
   print("\n")

```

```python

def buildArray():
   # Specifying a seed creates "pseudo random" numbers
   # that are the same with every program execution.
   seed(100)
   for k in range(amount):
      numbers.append(randint(10,99))


def displayArray():
   print()
   print(numbers)
   print()


def computeMean():
   return 0


def computeMedian():
   return 0


# precondition: The <numbers> array has exactly 1 mode.
def computeMode():
   return 0



##########
#  MAIN  #
##########

heading()
numbers = []
amount = eval(input("How many numbers do you want to generate?  -->  "))
buildArray()
displayArray()
print("Mean:  ",computeMean())
print("Median:",computeMedian())
print("Mode:  ",computeMode())
```

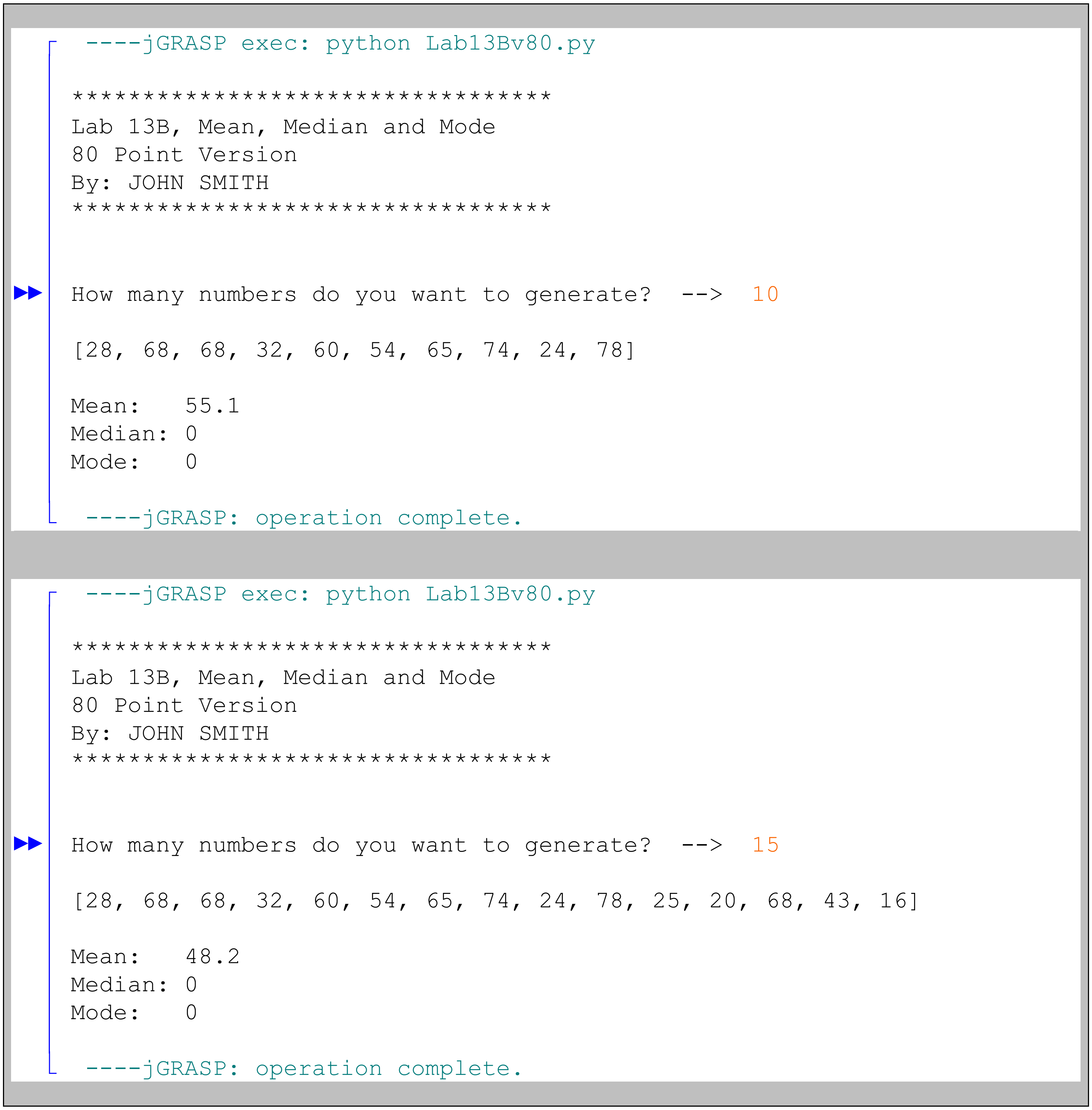
80-Point Version Specifics

For the 80-point version your program will only compute the **mean**. This means that you need to complete the **computeMean** function. Right now, this function along with **computeMedian** and **computeMode** simply return **0** which allows the program to execute. You need to make the **computeMean** function "compute the mean" and then return it.

The “**MAIN”** section of the program will call all 3 functions to display the *mean*, *median* and the *mode* of all of the numbers in the **numbers** array. For the 80-point version, a value of **0** will be displayed for the *median* and the *mode*.

80-Point Version Output

Run the program twice, first with **10** numbers, and then with **15**.



```
 ----jGRASP exec: python Lab13Bv80.py

**********************************
Lab 13B, Mean, Median and Mode
80 Point Version
By: JOHN SMITH
**********************************



How many numbers do you want to generate?  -->  10

[28, 68, 68, 32, 60, 54, 65, 74, 24, 78]

Mean:    55.1
Median:  0
Mode:    0

 ----jGRASP: operation complete.
```

```
 ----jGRASP exec: python Lab13Bv80.py

**********************************
Lab 13B, Mean, Median and Mode
80 Point Version
By: JOHN SMITH
**********************************



How many numbers do you want to generate?  -->  15

[28, 68, 68, 32, 60, 54, 65, 74, 24, 78, 25, 20, 68, 43, 16]

Mean:    48.2
Median:  0
Mode:    0

 ----jGRASP: operation complete.
```

90 & 100-Point Version Specifics

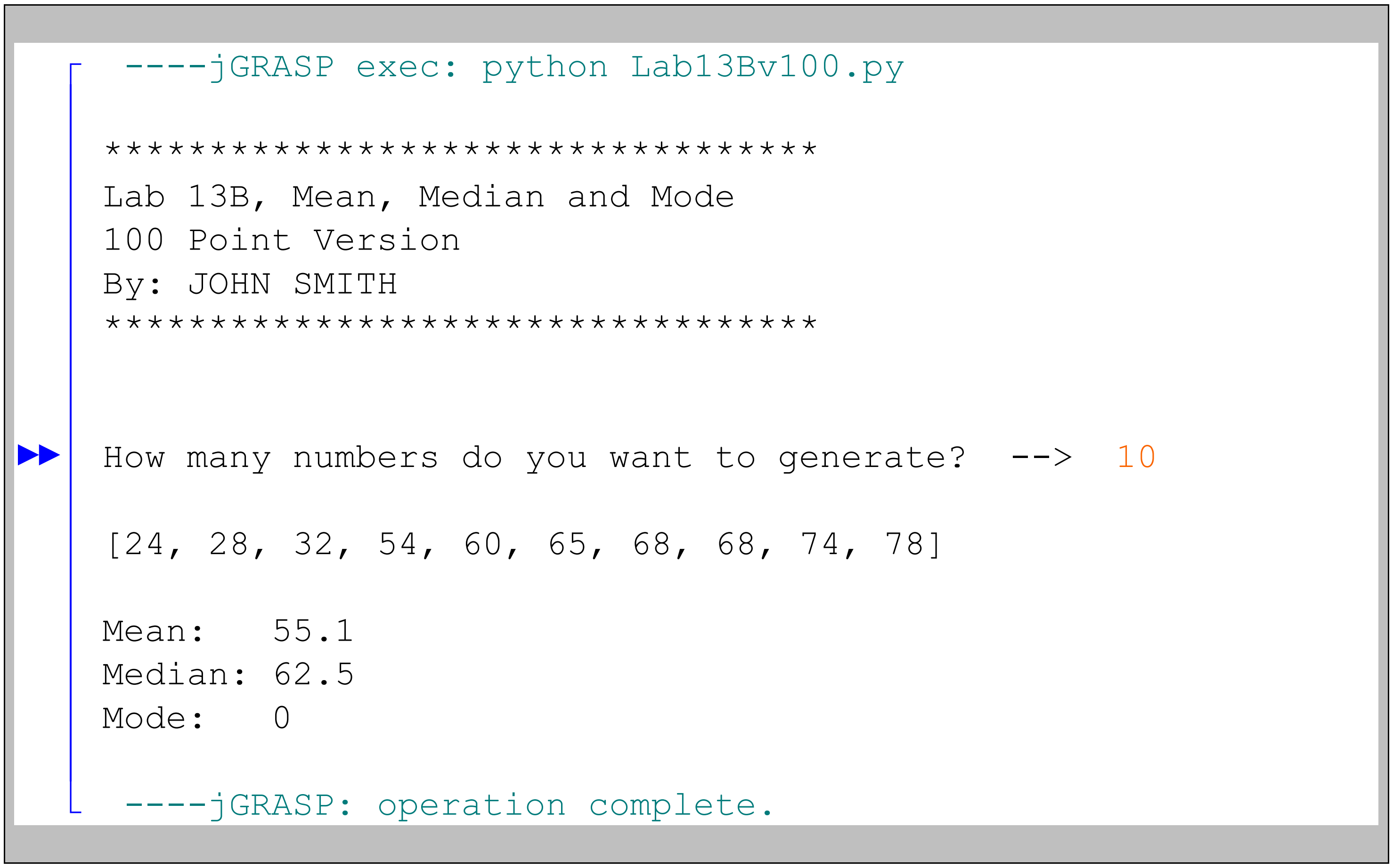
The 100-point version displays the **mean** and the **median**. This means that in addition to the **computeMean** function from the 80-point version, you also need to complete **computeMedian**. The median is the middle number in a sorted list. The list is currently random, so it will need to be sorted before you can do anything else. After the list is sorted, the computer needs to determine if this is an *odd* list or an *even* list. This is necessary because computing the **median** is actually done differently in each situation. If the number of integers in the list is *odd*, the **median** is simply the integer in the exact middle. If the number of integers in the list is *even*, computing the **median** becomes more challenging. An *even* list has 2 integers in the middle. To find the **median** of an *even* list, you need to find the **mean** of these 2 middle numbers.

NOTE: To earn a grade of **100** your program must work for both of the outputs below.
If your program only works for one of the outputs, you will be assigned a grade of **90**.
This means your program works for *even* lists only or *odd* lists only, but not both.

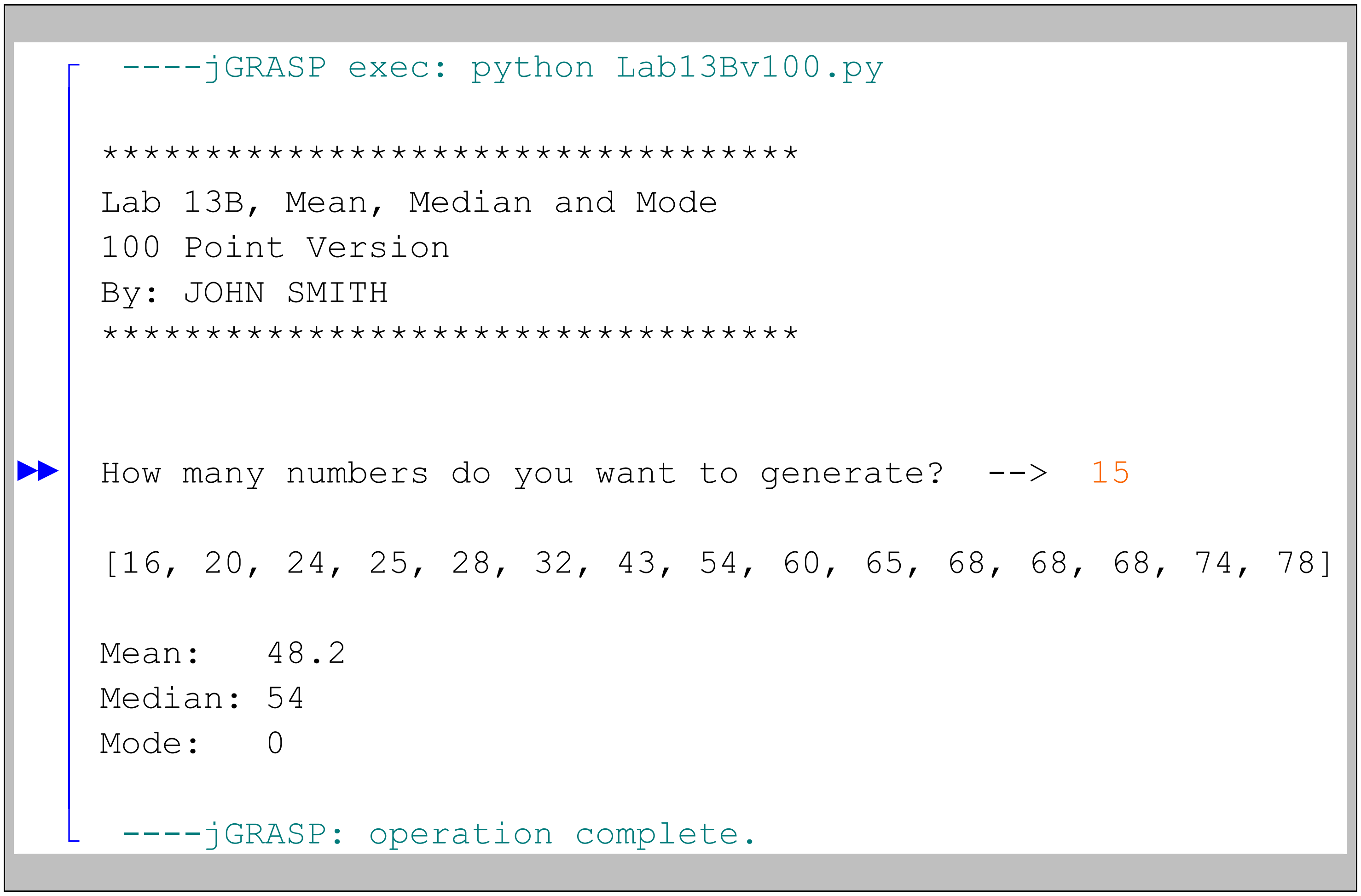
ALSO: The **numbers** array should be sorted when it is displayed. For this to work the sorting needs to take place after the list is *built*, and before it is *displayed*.

90 & 100-Point Version Output

Run the program twice, first with **10** numbers, and then with **15**.



```
 ----jGRASP exec: python Lab13Bv100.py

**********************************
Lab 13B, Mean, Median and Mode
100 Point Version
By: JOHN SMITH
**********************************



How many numbers do you want to generate?  -->  10

[24, 28, 32, 54, 60, 65, 68, 68, 74, 78]

Mean:    55.1
Median: 62.5
Mode:    0

 ----jGRASP: operation complete.
```



```
 ----jGRASP exec: python Lab13Bv100.py

**********************************
Lab 13B, Mean, Median and Mode
100 Point Version
By: JOHN SMITH
**********************************



How many numbers do you want to generate?  -->  15

[16, 20, 24, 25, 28, 32, 43, 54, 60, 65, 68, 68, 68, 74, 78]

Mean:   48.2
Median: 54
Mode:   0

 ----jGRASP: operation complete.
```

<u>110-Point Version Specifics</u>

For the 110-point version, you need to complete the **computeMode** function. This process is easily stated. Find the number that occurs most frequently in the **numbers** array and return it. This is definitely not a trivial task. Computing the mode requires an AP level understanding of array processing. This is why it is for the 110-point version.

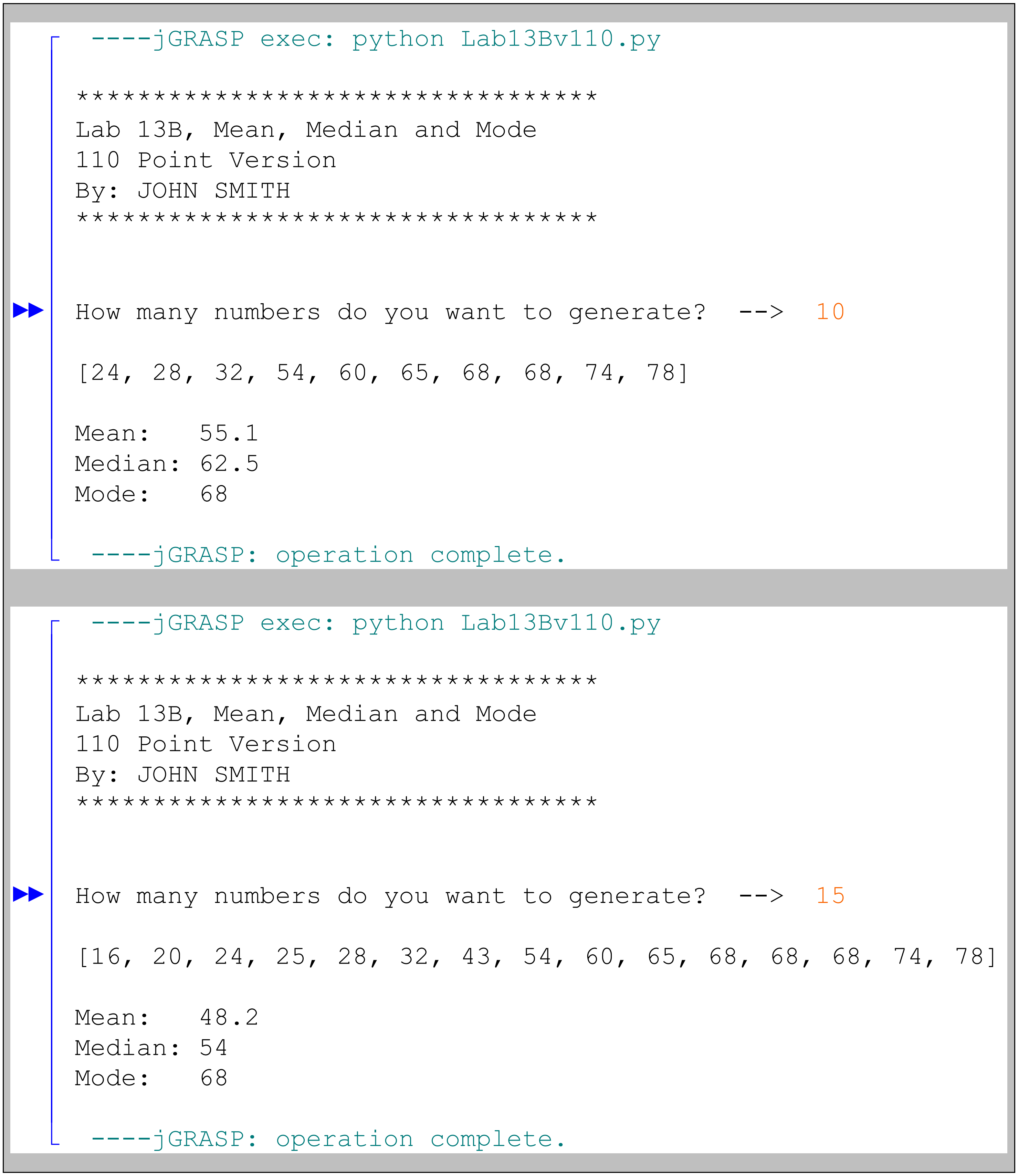
NOTE: It is technically possible for a list of numbers to have "multiple modes" or "no mode".
For this assignment you are not concerned with either of these cases.
The precondition (which is like a disclaimer) for the function states that the **numbers** array "has exactly 1 mode".

ALSO: You can take advantage of the fact that the list is now sorted.
All like numbers are grouped together.
This is the only hint that you will get for the 110-point version.

FINALLY: You may not use *Dictionaries* to compute the mode. You will learn about *dictionaries* in Chapter 15, but they may not be used in this lab.

110-Point Version Output

Run the program twice, first with **10** numbers, and then with **15**.



```
 ----jGRASP exec: python Lab13Bv110.py

**********************************
Lab 13B, Mean, Median and Mode
110 Point Version
By: JOHN SMITH
**********************************



How many numbers do you want to generate?  -->  10

[24, 28, 32, 54, 60, 65, 68, 68, 74, 78]

Mean:   55.1
Median: 62.5
Mode:   68

 ----jGRASP: operation complete.
```

```
 ----jGRASP exec: python Lab13Bv110.py

**********************************
Lab 13B, Mean, Median and Mode
110 Point Version
By: JOHN SMITH
**********************************



How many numbers do you want to generate?  -->  15

[16, 20, 24, 25, 28, 32, 43, 54, 60, 65, 68, 68, 68, 74, 78]

Mean:   48.2
Median: 54
Mode:   68

 ----jGRASP: operation complete.
```