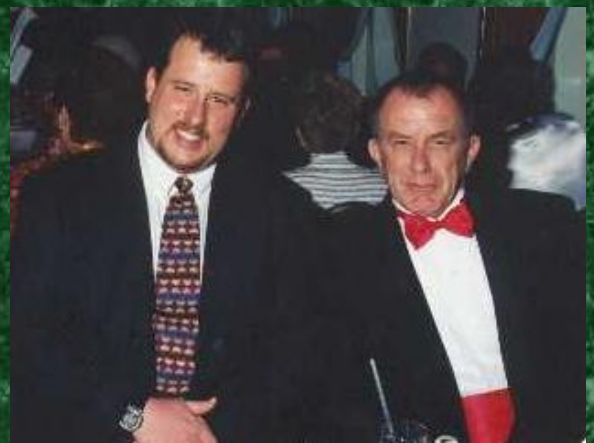
Exposure CS 2021 for CS1

Chapter 13 Section 3-5 Slides

Array Commands, Random Arrays, and The for..each Loop

PowerPoint Presentation
created by:
Mr. John L. M. Schram
and Mr. Leon Schram
Authors of Exposure
Computer Science



Section 13.3

Array Commands

```python
# ArrayCommands01.py
# This program reviews the <len> function which was
# the first array command shown in this chapter.


names = ["John","Greg","Maria","Heidi","Diana","David"]

print()
print("There are",len(names),"names in the array.")
print()

for k in range(len(names)):
    print(names[k], end = " ")
print()
```

```
 ----jGRASP exec: python ArrayCommands01.py

There are 6 names in the array.

John Greg Maria Heidi Diana David

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

```
1  # ArrayCommands02.py
2  # This program demonstrates the <append> command
3  # which will add a new item to the end of an array.
4
5
6  names = ["John","Greg","Maria","Heidi","Diana","David"]
7
8  print()
9  print(names)
10
11 names.append("Mike")
12
13 print()
14 print(names)
15
16 names.append("Kisha")
17
18 print()
19 print(names)
```

```
['John', 'Greg', 'Maria', 'Heidi', 'Diana', 'David']

['John', 'Greg', 'Maria', 'Heidi', 'Diana', 'David', 'Mike']

['John', 'Greg', 'Maria', 'Heidi', 'Diana', 'David', 'Mike', 'Kisha']
```

```
 6 names = ["John","Greg","Maria","Heidi","Diana","David"]
 7
 8 print()
 9 print(names)
10
11 names.append("Mike")
12
13 print()
14 print(names)
15
16 names.append("Kisha")
17
18 print()
19 print(names)
```

```
 1 # ArrayCommands03.py
 2 # This program uses the <append> command to build an
 3 # array from scratch.  The empty brackets on line 9
 4 # indicate that <numbers> starts out as an empty array,
 5 # which is displayed.  Then, in the <for> loop, several
 6 # numbers are "appended" and the array is displayed again.
 7
 8
 9 numbers = []
10
11 print()
12 print(numbers)
13
14 for k in range(10):
15     numbers.append(100 + k)
16
17 print()
18 print(numbers)
19
```

```
----jGRASP exec: python ArrayCommands03.py

[]

[100, 101, 102, 103, 104, 105, 106, 107, 108, 109]

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

```
 9 numbers = []
10
11 print()
12 print(numbers)
13
14 for k in range(10):
15     numbers.append(100 + k)
16
17 print()
18 print(numbers)
19
```

```
1  # ArrayCommands04.py
2  # This program demonstrates the <insert> command which will
3  # "insert" a new item in the array at a particular index.
4
5
6  names = ["John","Greg","Maria","Heidi","Diana","David"]
7
8  print()
9  print(names)
10
11 names.insert(4,"Mike")
12
13 print()
14 print(names)
15
16 names.insert(2,"Kisha")
17
18 print()
19 print(names)
```

```
['John', 'Greg', 'Maria', 'Heidi', 'Diana', 'David']

['John', 'Greg', 'Maria', 'Heidi', 'Mike', 'Diana', 'David']

['John', 'Greg', 'Kisha', 'Maria', 'Heidi', 'Mike', 'Diana', 'David']
```

```
6  names = ["John","Greg","Maria","Heidi","Diana","David"]
7
8  print()
9  print(names)
10
11 names.insert(4,"Mike")
12
13 print()
14 print(names)
15
16 names.insert(2,"Kisha")
17
18 print()
19 print(names)
```

```
1 # ArrayCommands05.py
2 # This program demonstrates the <index> function
3 # which will return the "index" of a particular
4 # item in the array.
5
6
7 names = ["John","Greg","Maria","Heidi","Diana","Kisha"]
8
9 print()
10 print(names)
11
12 print("\nDiana is located at index",names.index("Diana"))
13
14 print("\nJohn is located at index",names.index("John"))
15
```

```
['John', 'Greg', 'Maria', 'Heidi', 'Diana', 'Kisha']

Diana is located at index 4

John is located at index 0
```

```python
# ArrayCommands06.py
# This program demonstrated the <remove> command
# which will remove a specific item from the array.


names = ["John","Greg","Kisha","Maria","Heidi","Mike","Diana","David"]

print()
print(names)

names.remove("Mike")

print()
print(names)

names.remove("John")

print()
print(names)
```

```
['John', 'Greg', 'Kisha', 'Maria', 'Heidi', 'Mike', 'Diana', 'David']

['John', 'Greg', 'Kisha', 'Maria', 'Heidi', 'Diana', 'David']

['Greg', 'Kisha', 'Maria', 'Heidi', 'Diana', 'David']
```

```python
# ArrayCommands07.py
# This program demonstrates that if the same item is in the
# array more than once, the <index> command will only find
# the "index" of the first occurrence and the <remove> command
# will only "remove" the first occurrence from the array.


names = ["John","Greg","Maria","Heidi","David","Diana","David"]

print()
print(names)

print("\nDavid is located at index",names.index("David"))
names.remove("David")

print()
print(names)

print("\nDavid is located at index",names.index("David"))
```

```
['John', 'Greg', 'Maria', 'Heidi', 'David', 'Diana', 'David']

David is located at index 4

['John', 'Greg', 'Maria', 'Heidi', 'Diana', 'David']

David is located at index 5
```

```
 1 # ArrayCommands08.py
 2 # This program demonstrates that attempting to
 3 # "remove" an item that is not in the array will
 4 # cause the program to crash.
 5
 6
 7 names = ["John","Greg","Maria","Heidi","Diana","David"]
 8
 9 print()
10 print(names)
11
12 names.remove("Mike")
13
14 print()
15 print(names)
```

```
['John', 'Greg', 'Maria', 'Heidi', 'Diana', 'David']
Traceback (most recent call last):
  File "ArrayCommands08.py", line 12, in <module>
    names.remove("Mike")
ValueError: list.remove(x): x not in list
```

```
 1 # ArrayCommands09.py
 2 # This program demonstrates that attempting to find
 3 # the "index" of an item that is not in the array will
 4 # cause the program to crash.
 5
 6
 7 names = ["John","Greg","Maria","Heidi","Diana","David"]
 8
 9 print()
10 print(names)
11
12 print("\nMike is located at index",names.index("Mike"))
13
```

```
['John', 'Greg', 'Maria', 'Heidi', 'Diana', 'David']
Traceback (most recent call last):
  File "ArrayCommands09.py", line 12, in <module>
    print("\nMike is located at index",names.index("Mike"))
ValueError: 'Mike' is not in list
```

```
1  # ArrayCommands10.py
2  # The program uses the <in> operator to check if a particular item
3  # is "in" the array before attempting to find its "index".
4  # This prevents the program from crashing.
5
6
7  names = ["John","Greg","Maria","Heidi","Diana","David"]
8
9  print()
10 print(names)
11 print()
12
13 if "Heidi" in names:
14    print("'Heidi' is in the list at index",names.index("Heidi"))
15 else:
16    print("'Heidi' is not in the list.")
17 print()
18
19 if "Leon" in names:
20    print("'Leon' is in the list at index",names.index("Leon"))
21 else:
22    print("'Leon' is not in the list.")
```

```
['John', 'Greg', 'Maria', 'Heidi', 'Diana', 'David']

'Heidi' is in the list at index 3

'Leon' is not in the list.
```

```
 7 names = ["John","Greg","Maria","Heidi","Diana","David"]
 8
 9 print()
10 print(names)
11 print()
12
13 if "Heidi" in names:
14    print("'Heidi' is in the list at index",names.index("Heidi"))
15 else:
16    print("'Heidi' is not in the list.")
17 print()
18
19 if "Leon" in names:
20    print("'Leon' is in the list at index",names.index("Leon"))
21 else:
22    print("'Leon' is not in the list.")
```

```
1  # ArrayCommands11.py
2  # The program uses the <in> operator to check if a particular item
3  # is "in" the array before attempting to "remove" it.
4  # This prevents the program from crashing.
5
6
7  names = ["John","Greg","Maria","Heidi","Diana","David"]
8  print()
9  print(names)
10 print()
11
12 name = input("Who do you wish to remove from the list?  -->  ")
13 print()
14
15 if name in names:
16     names.remove(name)
17     print(name,"has been removed from the list.")
18     print()
19     print(names)
20 else:
21     print(name,"was not in the list.")
```

```
----jGRASP exec: python ArrayCommands11.py
['John', 'Greg', 'Maria', 'Heidi', 'Diana', 'David']
Who do you wish to remove from the list?  -->  Heidi
Heidi has been removed from the list.
['John', 'Greg', 'Maria', 'Diana', 'David']
----jGRASP: operation complete.
```

```
----jGRASP exec: python ArrayCommands11.py
['John', 'Greg', 'Maria', 'Heidi', 'Diana', 'David']
Who do you wish to remove from the list?  -->  Leon
Leon was not in the list.
----jGRASP: operation complete.
```

```
 1 # ArrayCommands12.py
 2 # This program was originally program InputProtection04.py from
 3 # Chapter 11.  The second <while> loop shows that an array can
 4 # be used to make a long compound condition less complicated.
 5 # Without it, the command would read:
 6 # genderOK = gender == 'M' or gender == 'm' or gender == 'F' or gender == 'f'
 7
 8
 9 sat = 0
10 satOK = False
11 while not satOK:
12     sat = eval(input("\nEnter SAT  {400..1600}  -->  "))
13     satOK = 400 <= sat <= 1600
14     if not satOK:
15         print("\nError! Please enter a number between 400 & 1600.")
16
17 gender = ''
18 genderOK = False
19 while not genderOK:
20     gender = input("\nEnter your gender {M/F} -->  ")
21     genderOK = gender in ['M','m','F','f'] # This is an array.
22     if not genderOK:
23         print("\nError! Please enter either an 'M' or an 'F'.")
24
```

```
1  # ArrayCommands13.py
2  # This program demonstrates how to delete the item
3  # at a specific index.
4
5
6  names = ["John","Greg","Maria","Heidi","Mike","Diana","David"]
7  print()
8  print(names)
9
10 del names[0]
11 print()
12 print(names)
13
14 del names[3]
15 print()
16 print(names)
17
```

Index	0	1	2	3	4	5	6

```
----jGRASP exec: python ArrayCommands13.py

['John', 'Greg', 'Maria', 'Heidi', 'Mike', 'Diana', 'David']

['Greg', 'Maria', 'Heidi', 'Mike', 'Diana', 'David']

['Greg', 'Maria', 'Heidi', 'Diana', 'David']

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

```
1  # ArrayCommands14.py
2  # This program demonstrates another way to delete the item
3  # at a specific index.
4
5
6  names = ["John","Greg","Maria","Heidi","Mike","Diana","David"]
7  print()
8  print(names)
9
10 names.pop(0)
11 print()
12 print(names)
13
14 names.pop(3)
15 print()
16 print(names)
17
```

Index	0	1	2	3	4	5	6

```
----jGRASP exec: python ArrayCommands14.py

['John', 'Greg', 'Maria', 'Heidi', 'Mike', 'Diana', 'David']

['Greg', 'Maria', 'Heidi', 'Mike', 'Diana', 'David']

['Greg', 'Maria', 'Heidi', 'Diana', 'David']

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

```
1  # ArrayCommands15.py
2  # This program demonstrates that <pop> actually
3  # returns the removed item.  <del> does not do this.
4
5  names = ["John","Greg","Maria","Heidi","Mike","Diana","David"]
6  print()
7  print(names)
8
9  print()
10 print(names.pop(0))
11 print(names)
12
13 print()
14 print(names.pop(3))
15 print(names)
16
17 print()
18 print(names.pop())  # Removes the last element
19 print(names)
```

```
----jGRASP exec: python ArrayCommands15.py

['John', 'Greg', 'Maria', 'Heidi', 'Mike', 'Diana', 'David']

John
['Greg', 'Maria', 'Heidi', 'Mike', 'Diana', 'David']

Mike
['Greg', 'Maria', 'Heidi', 'Diana', 'David']

David
['Greg', 'Maria', 'Heidi', 'Diana']

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

```
1  # ArrayCommands16.py
2  # This program shows that <del> allows you to delete
3  # an entire "slice" from an array.  Remember, you need
4  # to specify the starting and stopping index.  In this
5  # program, the [2:5] specifies deleting the names from
6  # indexes 2-4 (not 5).  This cannot be done with <pop>.
7
```

Index	0	1	2	3	4	5	6	7

```
9  names = ["John","Greg","Maria","Heidi","Mike","Kisha","Diana","David"]
10
11 print()
12 print(names)
13
14 del names[2:5]
15
16 print()
17 print(names)
18
```

```
----jGRASP exec: python ArrayCommands16.py

['John', 'Greg', 'Maria', 'Heidi', 'Mike', 'Kisha', 'Diana', 'David']

['John', 'Greg', 'Kisha', 'Diana', 'David']

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

Index	0	1	2	3	4	5	6	7

```
7
8
9  names = ["John","Greg","Maria","Heidi","Mike","Kisha","Diana","David"]
10
11 print()
12 print(names)
13
14 del names[2:5]
15
16 print()
17 print(names)
18
```

```
 1 # ArrayCommands17.py
 2 # This program is very similar to ArraySyntax12.py
 3 # It shows that the <extend> command does the same
 4 # thing as the += operator.
 5
 6
 7 list = [11,22,33,44,55]
 8
 9 print()
10 print(list)
11
12 list.extend([66,77,88,99])
13
14 print()
15 print(list)
16
```

```
1  # ArrayComm
2  # This prog
3  # It shows
4  # thing as
5
6
7  list = [11,22,33,44,55]
8
9  print()
10 print(list)
11
12 list.extend([66,77,88,99])
13
14 print()
15 print(list)
16
```

```
----jGRASP exec: python ArrayCommands17.py

[11, 22, 33, 44, 55]

[11, 22, 33, 44, 55, 66, 77, 88, 99]

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

This does the same exact thing as:

`list += [66,77,88,99]`

```python
# ArrayCommands18.py
# This program demonstrates the <sort> command which
# will "sort" an array of strings alphabetically and
# "sort" an array of numbers in ascending order.


names = ["John","Greg","Maria","Heidi","Diana","David","Mike","Kisha"]
ages  = [23,76,13,45,54,18,3,103,37,89,63]
gpas  = [3.9,2.75,2.1,1.65,4.0,3.25,2.45,0.95,3.88,3.75]

print()
print(names)
print()
print(ages)
print()
print(gpas)
print()

names.sort()
ages.sort()
gpas.sort()

print()
print(names)
print()
print(ages)
print()
print(gpas)
```

```
----jGRASP exec: python ArrayCommands18.py

['John', 'Greg', 'Maria', 'Heidi', 'Diana', 'David', 'Mike', 'Kisha']

[23, 76, 13, 45, 54, 18, 3, 103, 37, 89, 63]

[3.9, 2.75, 2.1, 1.65, 4.0, 3.25, 2.45, 0.95, 3.88, 3.75]


['David', 'Diana', 'Greg', 'Heidi', 'John', 'Kisha', 'Maria', 'Mike']

[3, 13, 18, 23, 37, 45, 54, 63, 76, 89, 103]

[0.95, 1.65, 2.1, 2.45, 2.75, 3.25, 3.75, 3.88, 3.9, 4.0]

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

```
1  # ArrayCommands19.py
2  # This program demonstrates how to sort an array of
3  # numbers in descending order.  It is done in 2 steps.
4  # First you <sort> the list; then you <reverse> it.
5
6
7  gpas = [3.9,2.75,2.1,1.65,4.0,3.25,2.45,0.95,3.88,3.75]
8
9  print()
10 print(gpas)
11
12 gpas.sort()
13
14 print()
15 print(gpas)
16
17 gpas.reverse()
18
19 print()
20 print(gpas)
```

```
----jGRASP exec: python ArrayCommands19.py

[3.9, 2.75, 2.1, 1.65, 4.0, 3.25, 2.45, 0.95, 3.88, 3.75]

[0.95, 1.65, 2.1, 2.45, 2.75, 3.25, 3.75, 3.88, 3.9, 4.0]

[4.0, 3.9, 3.88, 3.75, 3.25, 2.75, 2.45, 2.1, 1.65, 0.95]

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

```
1  # ArrayCommands20.py
2  # This program demonstrates the <sorted> function.
3  # While similar to the <sort> command, there are 2
4  # important differences:
5  # 1. <sorted> does not actually sort the array.
6  #    Instead, it returns a sorted copy of the array.
7  # 2. <sorted> has an option to sort in reverse order.
8
9
10 gpas1 = [3.9,2.75,2.1,1.65,4.0,3.25,2.45,0.95,3.88,3.75]
11
12 gpas2 = sorted(gpas1)
13
14 gpas3 = sorted(gpas1, reverse=True)
15
16 print()
17 print(gpas3)
18 print(gpas2)
19 print(gpas1)
```

```
----jGRASP exec: python ArrayCommands20.py

[4.0, 3.9, 3.88, 3.75, 3.25, 2.75, 2.45, 2.1, 1.65, 0.95]
[0.95, 1.65, 2.1, 2.45, 2.75, 3.25, 3.75, 3.88, 3.9, 4.0]
[3.9, 2.75, 2.1, 1.65, 4.0, 3.25, 2.45, 0.95, 3.88, 3.75]

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

```
 9
10 gpas1 = [3.9,2.75,2.1,1.65,4.0,3.25,2.45,0.95,3.88,3.75]
11
12 gpas2 = sorted(gpas1)
13
14 gpas3 = sorted(gpas1, reverse=True)
15
16 print()
17 print(gpas3)
18 print(gpas2)
19 print(gpas1)
```

```
1  # ArrayCommands21.py
2  # This program demonstrates a problem that happens when you try
3  # to sort strings that contain both CAPITAL and lowercase letters.
4  # Capital letters have numeric code values from 65-90.
5  # Lowercase letters have numeric code values from 97-122.
6  # If lowercase letters are sorted together with capital letters,
7  # the capitals will come first because of the smaller code values.
8
9
10 animals = ["hippo","monkey","ZEBRA","TOUCAN","lion",
               "GIRAFFE","cheetah","ELEPHANT"]
11
12 print()
13 print(sorted(animals))
14 animals.sort()
15 print(animals)
16
```

```
----jGRASP exec: python ArrayCommands21.py

['ELEPHANT', 'GIRAFFE', 'TOUCAN', 'ZEBRA', 'cheetah', 'hippo', 'lion', 'monkey']
['ELEPHANT', 'GIRAFFE', 'TOUCAN', 'ZEBRA', 'cheetah', 'hippo', 'lion', 'monkey']

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

Section 13.4

Random Arrays

```python
# RandomArrays01.py
# This program fills an array with random
# 3-digit integers and then displays the
# entire contents of the array.


from random import randint


list = []
for k in range(20):
    list.append(randint(100,999))

print()
for k in range(20):
    print("list["+str(k)+"] =",list[k])

```

```
----jGRASP exe

list[0] = 215
list[1] = 337
list[2] = 648
list[3] = 679
list[4] = 224
list[5] = 750
list[6] = 507
list[7] = 399
list[8] = 262
list[9] = 125
list[10] = 551
list[11] = 677
list[12] = 952
list[13] = 327
list[14] = 521
list[15] = 258
list[16] = 335
list[17] = 527
list[18] = 804
list[19] = 415

----jGRASP: op
```

```python
# RandomArrays02.py
# This program will display 15 random sentences.
# With 7 different ranks, 7 different people,
# 7 different actions and 7 different locations,
# there are more than 2400 different sentences possible.


from random import randint


ranks = ["Private","Corporal","Sargent","Lieutenant","Captain","Major","General"]

names = ["Smith","Gonzales","Brown","Jackson","Powers","Jones","Nguyen"]

actions = ["drive the tank","drive the jeep","take the troops",
           "bring all supplies","escort the visitor","prepare to relocate",
           "bring the Admiral"]

locations = ["over the next hill", "to the top of the mountain",
             "outside the barracks", "30 miles into the desert",
             "to the middle of the forest", "to my present location",
             "to anywhere but here"]


for k in range(15):
   randomRank     = randint(0,len(ranks)-1)
   randomPerson   = randint(0,len(names)-1)
   randomAction   = randint(0,len(actions)-1)
   randomLocation = randint(0,len(locations)-1)

   sentence = ranks[randomRank] + " " + names[randomPerson] + ", " + \
              actions[randomAction] + " " + locations[randomLocation] + "."

   print("\n" + sentence)
```

```
 ----jGRASP exec: python RandomArrays02.py
Private Gonzales, escort the visitor to my present location.
Lieutenant Nguyen, prepare to relocate over the next hill.
Captain Brown, bring the Admiral to my present location.
Captain Jackson, take the troops to the top of the mountain.
Major Jones, take the troops to the top of the mountain.
Lieutenant Jackson, drive the tank to anywhere but here.
Private Gonzales, drive the tank outside the barracks.
General Nguyen, prepare to relocate to the middle of the forest.
Sargent Jones, drive the jeep to my present location.
Captain Brown, prepare to relocate outside the barracks.
Corporal Smith, drive the tank over the next hill.
General Brown, drive the tank outside the barracks.
Lieutenant Nguyen, take the troops to anywhere but here.
Corporal Powers, bring all supplies 30 miles into the desert.
Private Brown, escort the visitor 30 miles into the desert.
 ----jGRASP: operation complete.
```

Section 13.5

The for..each Loop

```
 1 # forEach01.py
 2 # This program compares 2 "flexible" ways to
 3 # "traverse" an array.  This first reviews the
 4 # used of the <len> function.  The second uses
 5 # the <for..each> loop which some people prefer.
 6 # For displaying the contents of an array, both
 7 # ways work just fine.
 8
 9
10 listA = [100,101,102,103,104,105,106,107,108,109]
11 print()
12 for k in range(len(listA)):
13     print(listA[k], end = " ")
14 print()
15
16
17 listB = [100,101,102,103,104,105,106,107,108,109]
18 print()
19 for number in listB:
20     print(number, end = " ")
21 print()
```

```
  ----jGRASP exec: python forEach01.py

100 101 102 103 104 105 106 107 108 109

100 101 102 103 104 105 106 107 108 109

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

```
10 listA = [100,101,102,103,104,105,106,107,108,109]
11 print()
12 for k in range(len(listA)):
13     print(listA[k], end = " ")
14 print()
15
16
17 listB = [100,101,102,103,104,105,106,107,108,109]
18 print()
19 for number in listB:
20     print(number, end = " ")
21 print()
```

```
----jGRASP exec: python forEach01.py

100 101 102 103 104 105 106 107 108 109

100 101 102 103 104 105 106 107 108 109

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

```
10 listA = [100,101,102,103,104,105,106,107,108,109]
11 print()
12 for k in range(len(listA)):
13     print(listA[k], end = " ")
14 print()
15
16
17 listB = [100,101,102,103,104,1
18 print()
19 for number in listB:
20     print(number, end = " ")
21 print()
```

NOTE: This is called the "for..each loop" because the item variable, number in this case, will take on the value of each item in the array.

```
 1 # forEach02.py
 2 # This program does another comparison between
 3 # the <for> loop and the <for..each> loop.
 4 # This time, the mission is to change all
 5 # of the numbers in both arrays to 500.
 6 # The output shows that the <for..each> loop is
 7 # not able to change the contents of an array.
 8
 9
10 listA = [100,101,102,103,104,105,106,107,108,109]
11 for k in range(len(listA)):
12     listA[k] = 500
13
14 listB = [100,101,102,103,104,105,106,107,108,109]
15 for number in listB:
16     number = 500
17
18 print()
19 print(listA)
20 print()
21 print(listB)
```

```
----jGRASP exec: python forEach02.py

[500, 500, 500, 500, 500, 500, 500, 500, 500, 500]

[100, 101, 102, 103, 104, 105, 106, 107, 108, 109]

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

```
10 listA = [100,101,102,103,104,105,106,107,108,109]
11 for k in range(len(listA)):
12     listA[k] = 500
13
14 listB = [100,101,102,103,104,105,106,107,108,109]
15 for number in listB:
16     number = 500
17
18 print()
19 print(listA)
20 print()
21 print(listB)
```

```
----jGRASP exec: python forEach02.py

[500, 500, 500, 500, 500, 500, 500, 500, 500, 500]

[100, 101, 102, 103, 104, 105, 106, 107, 108, 109]

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

```
10 listA = [100,101,102,103,104,105,106,107,108,109]
11 for k in range(len(listA)):
12     listA[k] = 500
13
14 listB = [100,101,102,103,104,105,106,107,108,109]
15 for number in listB:
16     number = 500
17
18 print()
19 print(listA)
20 print()
21 print(listB)
```

NOTE: The item variable, number, gets a COPY of each item in the array. Changing this copy has no effect on the contents of the array.