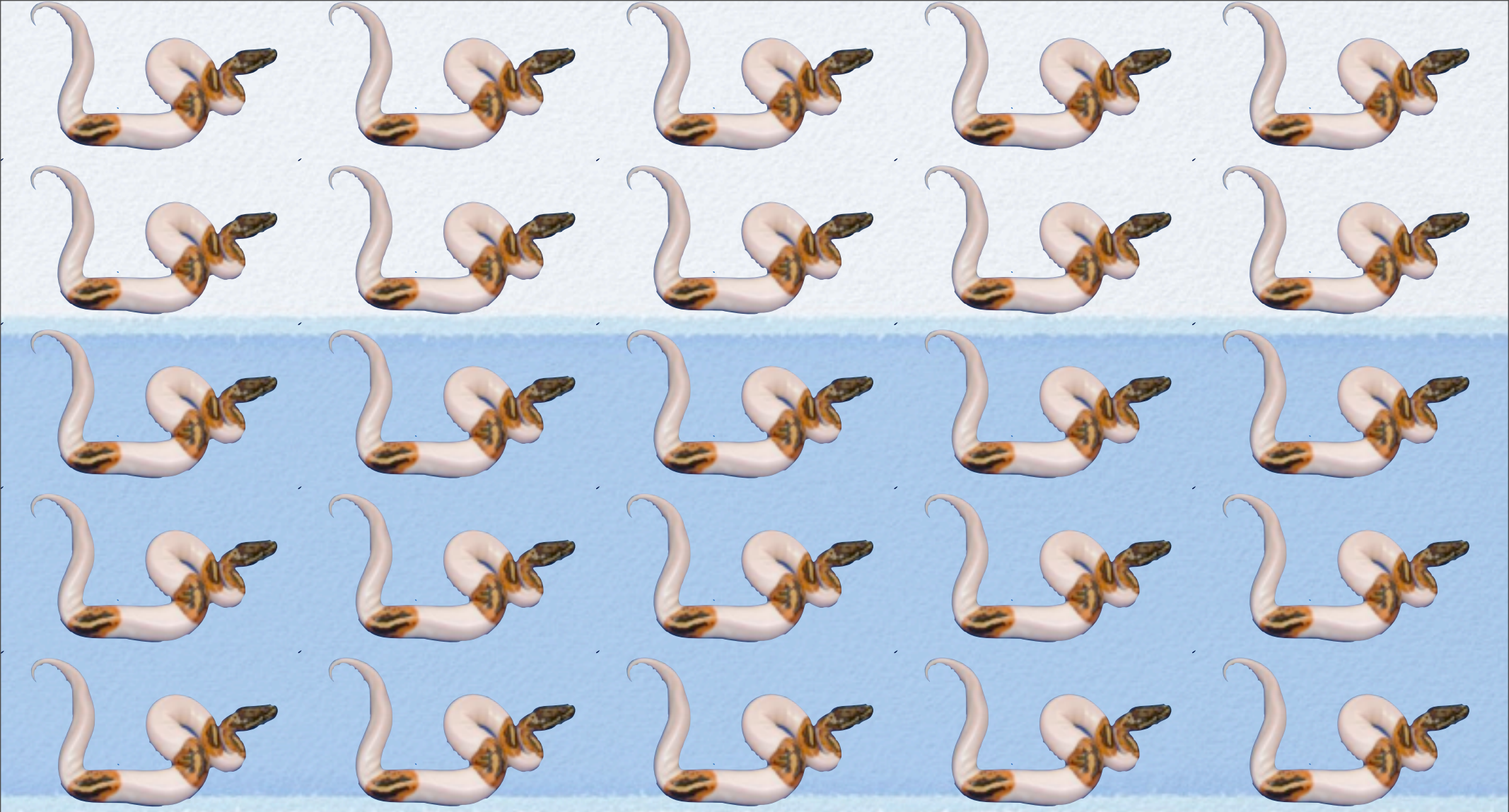


DATA COLLECTIONS
CHAPTER 11

Chapter Learning Outcomes

- To understand the use of lists (arrays) to represent a collection of related data.
- To be familiar with the functions and methods available for manipulating Python lists.
- To be able to write programs that use lists to manage a collection of information.
- To understand the use of Python dictionaries for storing nonsequential collections.

PROBLEM

- Calculate the mean and standard deviation for the COMP 201 Quiz 1 scores.
- Formula for Mean:

$$\bar{x} = \frac{1}{n} \cdot \sum_{i=1}^{n} x_i$$

- Formula for Standard Deviation:

$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (x_i - \overline{x})^2},$$

Calculating the Mean

- The mean can be easily calculated using the now-standard practice of an accumulator and a counter:

```
def mean():
    sum = 0
    count = 0
    xStr = input("Number? (<enter> to quit)")
    while xStr != "":
        x = eval(xStr)
        sum = sum + x
        count = count + 1
        xStr = input("Number? (<enter> to quit)")
    mean = sum/count
```

Calculating the Standard Deviation

- The standard deviation poses a problem for us.

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \overline{x})^2},$$

- Note how we need to first calculate the mean, and then do a second round of calculations with each of the individual values.
- We need to somehow remember all the values that were entered...
- ...but we don't know in advance how many values there will be!

Lists, Revisited

- We have worked with something called lists back in Chapters 3 and 4

```
>>> "Hong Kong Polytechnic University".split()
['Hong', 'Kong', 'Polytechnic', 'University']
>>> a = [1, 3, 5, 7, 9]
>>> print(a)
[1, 3, 5, 7, 9]
>>> print(a[0])
1
```

- Each of these collections of values enclosed by square brackets is called a list.

Lists and sequences

- Lists are derived from the mathematical notion of **sequences**.
- A sequence in maths looks like this:

$$S = s_1, s_2, s_3, \ldots, s_n$$

- The values in the sequence are collectively called *S*.
- When we want to talk about a specific value, then we use the **subscript**, for example, s_7.

Lists and sequences

- Computer lists operate in a similar way as mathematical sequences.
- A single variable is used to represent an entire sequence of values.
- The individual items are accessed using **indexing**.

```
>>> a = [1, 3, 5, 7, 9]
>>> a[0]
1
>>> a[3]
7
```

- Note that the first item in a list has index 0.
- In some other programming languages, lists are commonly known as **arrays**.

Lists

- To summarize, a list or an array is a sequence of items where:
  - The entire sequence of referred to by a single name (`grade` in this example);
  - Individual items can be selected by indexing (e.g. `grade[0]`, `grade[6]`, etc).

First element of an array has index 0

`grade[0]`	33
`grade[1]`	25
`grade[2]`	79
`grade[3]`	12
`grade[4]`	100
`grade[5]`	70
`grade[6]`	92
`grade[7]`	40
`grade[8]`	68
`grade[9]`	91

Last element of an array has index n-1, where n is the number of items.

Making a List

- Create the list explicitly by listing out all the items:

```
a = [1, 3, 5, 7, 9]  # List with 5 items: 1, 3, 5, 7, 9
```

- Create an empty list and add items to it one by one:

```
a = []  # Create an empty list
a.append(1)   # list a is now [1]
a.append(10)  # list a is now [1, 10]
```

- Create a list with a single element and repeat it:

```
a = [0]  # a = [0]
a = a*5   # list a is now [0, 0, 0, 0, 0]
```

- The method we choose will depend on our application.

List Operations

- The list operations are the same as those that we used before on strings (except for membership).

Operator	Meaning
+	Concatenation
*	Repetition
`<list>[]`	Indexing
`<list>[:]`	Slicing
`len(<list>)`	Length
`for <var> in <list>`	Iteration through list items
`<expr> in <list>`	Membership check

EXAMPLES

- Unlike strings, lists are mutable -- we can change the values of any of the items in the list, and we can even change entire subsequences of the list.

```
>>> a = [1, 3, 5, 7]
>>> a[2]
5
>>> 3 in a
True
>>> 4 in a
False
>>> a[2] = "Hi"
>>> a
[1, 3, 'Hi', 7]
>>> len(a)
4
```

```
>>> b = [2, 4, 6]
>>> a + b
[1, 3, 'Hi', 7, 2, 4, 6]
>>> a[1:3] = ["Poly", "U"]
>>> a
[1, 'Poly', 'U', 7]
>>> a = []
>>> a
[]
>>> a = [1]*3
>>> a
[1, 1, 1]
```

More List Operations

- These operations may only be used on lists (the previous were common to lists and strings):

Operator	Meaning
`<list>.append(x)`	Add element x to the end of list.
`<list>.sort()`	Sort the list.
`<list>.reverse()`	Reverse the list
`<list>.index(x)`	Return the index of the first occurence of x
`<list>.insert(i, x)`	Insert element x into the list at position i
`<list>.count(x)`	Returns the number of occurences of x.
`<list>.remove(x)`	Deletes the first occurrence of x in list
`<list>.pop(i)`	Deletes the ith element of the list and returns its value

EXAMPLES

```
>>> a = [1, 3, 5, 7]
>>> a.append[2]
>>> a
[1, 3, 5, 7, 2]
>>> a.sort()
>>> a
[1, 2, 3, 5, 7]
>>> a.reverse()
>>> a
[7, 5, 3, 2, 1]
>>> a.index(3)
2
>>> a.insert(4, "Hi")
>>> a
[7, 5, 3, 2, 'Hi', 1]
```

```
>>> a.count(2)
1
>>> a.append(2)
>>> a.count(2)
2
>>> a
[7, 5, 3, 2, 'Hi', 1, 2]
>>> a.remove(2)
>>> a
[7, 5, 3, 'Hi', 1, 2]
>>> a.pop(3)
'Hi'
>>> a
[7, 5, 3, 1, 2]
```

Modifying Lists

- Most of the list-specific methods do not return a value; instead, they alter the list in some way.
- We can also delete items from a list using the `del()` function:

```
>>> a = [1, 3, 5, 7]
>>> del(a[1])
>>> a
[1, 5, 7]
>>> del(a[1:len(a)])
>>> a
[1]
```

Lists basic principles

- Python lists are sequences of items stored as a single object (i.e. with a single name).
- Items in a list can be accessed by indexing, and sublists can be accessed by slicing.
- Lists are mutable; individual items or entire slices can be replaced through assignment statements.
- Lists support a number of convenient and frequently used methods, such as appending, sorting and deleting.
- Lists can grow and shrink as needed.

From other programming languages...

- For those of you with programming backgrounds: most programming languages offer support for arrays, but they are not usually as flexible as Python lists.
- Usually fixed in size and homogeneous (stores elements of only one data type)
- The rest of you: you'll see what I mean when you go on to COMP 304/305 and C/C++.

EXERCISES

- Suppose that you are writing a program that will require a list called `myList` to store 50 numbers. How do you build that list?

- Write a program statement to set element number 37 of `myList` to 59.

EXERCISES

- Write a statement to make a list called `series` with the following data values (in order): 5, 9, -1, 4, 0.

- What would be printed by the following program statements?

```
list = [2, 1, 2, 1, 1, 2, 3, 2, 1, 2]
print(list[2])
print(list[list[2]])
print(list[list[2]+list[3])
```

EXERCISES

- Assume that you already have the following list:

```
list = [2, 1, 2, 1, 1, 2, 3, 2, 1, 2]
```

- Write program statements to add the values of the list into the variable `sum`.

```

```

SOLVING OUR PROBLEM

- Now that we know lists, we can solve our average and standard deviation problem.
- The following function stores student scores into a list that grows as needed.
- The number of student scores can be determined using the `len()` function to return the size of the list.

```
def inputScores():
    scores = []  # empty list to start with.
    xStr = input("Number? (<enter> to quit)")
    while xStr != "":
        x = eval(xStr)
        scores.append(x)
        xStr = input("Number? (<enter> to quit)")
    return scores
```

Calculating Statistics

- The following function will calculate the mean of the student scores that were entered:

```
def calculateMean(scores):
    count = len(scores)  # number of scores
    sum = 0
    for score in scores:
        sum = sum + score
    return sum/count
```

Standard Deviation

- We can use the mean to calculate the standard deviation:

```
def calculateStdev(scores, mean):
    count = len(scores)  # number of scores
    sumOfDiffs = 0
    for score in scores:
        difference = score - mean
        sumOfDiffs = sumOfDiffs + difference*difference
    stdev = sqrt(sumOfDiffs/(count-1))
    return stdev
```

Putting it all together

- We can now put everything together by adding a `main()` function:

```
def main():
    scores = inputScores()  # number of scores
    mean = calculateMean(scores)
    stdev = calculateStdev(scores, mean)
    print("The mean is", mean)
    print("The standard deviation is", stdev)
    print("There were", len(scores), "students")
```

EXERCISES

- Write a program that will do the following:
  - Ask the user for the numerical scores of 5 students, and store them into a list, `scores`.
  - Convert the numerical scores of the students into letter grades (conversion table is below), and store the letter grades into a list called grades.
  - Print out the numerical score and the corresponding letter grade for each student.

≥85	≥75	≥65	≥55	<55
A	B	C	D	F

EXERCISES

- Complete the program below so that the output of the program is:

[1, 9, 25, 49, 81]

- Hint: use `range()` together with `len()` and indexing.

```
def squareAllValues(                    ):



def main():
    list = [1, 3, 5, 7, 9]
```

EXERCISE

- There are two ways of iterating over a list:

```
for item in list:
    # Do something
```

```
for i in range(len(list)):
    # Do sth with list[i]
```

- When do we use which?

Two-Dimensional Lists

- Under some circumstances, we need to have lists that resemble a grid or a table, with both rows and columns.
- What we need is a 2-dimensional list.

```
>>> a = [[1, 2, 3, 4], [5, 6, 7, 8]]
>>> a[0][1]
2
>>> a[1][2]
6
>>> for i in range(2):
        for j in range(4):
            print(a[i][j], end=" ")
        print()
1 2 3 4
5 6 7 8
>>>
```

	col 0	col 1	col 2	col 3
row 0	1	2	3	4
row 1	5	6	7	8

Making a 2D list

- Making a 2D list is more complex than making a 1D list.
- You can create the list explicitly:

```
a = [[1, 2, 3, 4], [5, 6, 7, 8]]
```

- You can create an empty list and add in the items one by one:

```
a = []  # Create an empty list
a.append([])   # list a is now [[]]
a[0].append(10)  # a is now [[10]]
a.append([])    # a is now [[10][]]
a[1].append(49)  # a is now [[10],[49]]
```

Making a 2-D List

- What if we want to make an list of 4 rows and 3 columns, but we don't know what the items are yet?
- We need to do this one row at a time:

```
a = []  # Create an empty list
for i in range(0, 4):
  a.append([0]*3)  # add in 1 row of 3 columns
```

Using a 2-D List

Terminal — Python — 69×17

```
>>> a  = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]
>>> a
[[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]
>>> len(a)
3
>>> len(a[0])
4
>>> len(a[1])
4
>>> a[1][2] = -1
>>> a
[[1, 2, 3, 4], [5, 6, -1, 8], [9, 10, 11, 12]]
>>> for i in range(len(a)):
...     for j in range(len(a[i])):
...             print(a[i][j], end = " ")
...
>>>
```

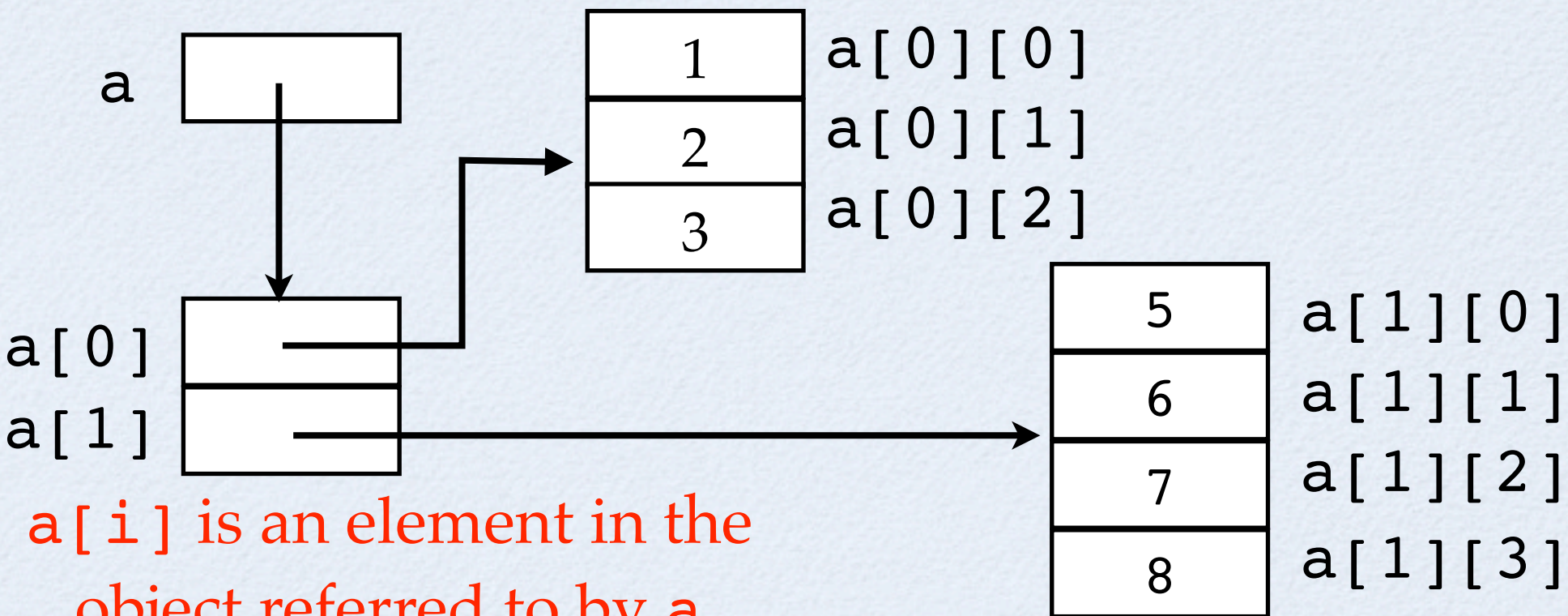
What exactly is a 2D list?

- Consider the following two program statements:
  - `len(a)` -- gets the number of rows in the list.
  - `len(a[0])` -- number of columns in row 0.
- Since both `a` and `a[0]` can have `len()` applied to them, this implies that `a` and `a[0]` are both the same kind of thing!
  - Recall: What is `a`?

2D Lists: A 1D List of References to 1D Lists

```
a = [[1, 2, 3], [5, 6, 7, 8]]
```

`a[i][j]` is an element in the object referred to by `a[i]`



`a[i]` is an element in the object referred to by `a`

2D LISTS

- A 2D list in Python is actually a 1D list of references to 1D lists.
- Therefore, this allows us to create a 2D list one row at a time:

```
a = []  # Create an empty list
a.append([0]*3)  # add in 1 row of 3 columns
a.append([0]*5)  # append another row of 5 columns
```

- Since the rows are independent, they can have different lengths. In other words, Python lists can have "irregular shapes".
- After the list has been created, we access each element using *listname*[*row*][*column*].
- We can use *listname*[*row*] to get us the length (number of columns) in that row.

EXERCISE

- Write programming statements to declare a 2-d list named `table` with 4 rows and 5 columns.

- Write a statement to assign 97 into the element in Row 2 and Column 3 of the list that we just created.

EXERCISES

- Suppose we already have a 2D list called `matrix`. Write programming statements to calculate the total of all the numbers stored in `matrix`. The value should be stored into the variable `total`.

EXERCISES

- Write a program to store the multiplication table into a 2D list of size 10x10. The program should then print out the table.

```
1    2    3    4    5    6    7    8    9   10
2    4    6    8   10   12   14   16   18   20
...
```

Lists and Functions

- We know what happens when a list is passed between functions.
- A reference to the list is passed, and the called function can make modifications to the list.
- These changes will be visible to the calling function when the called function terminates.

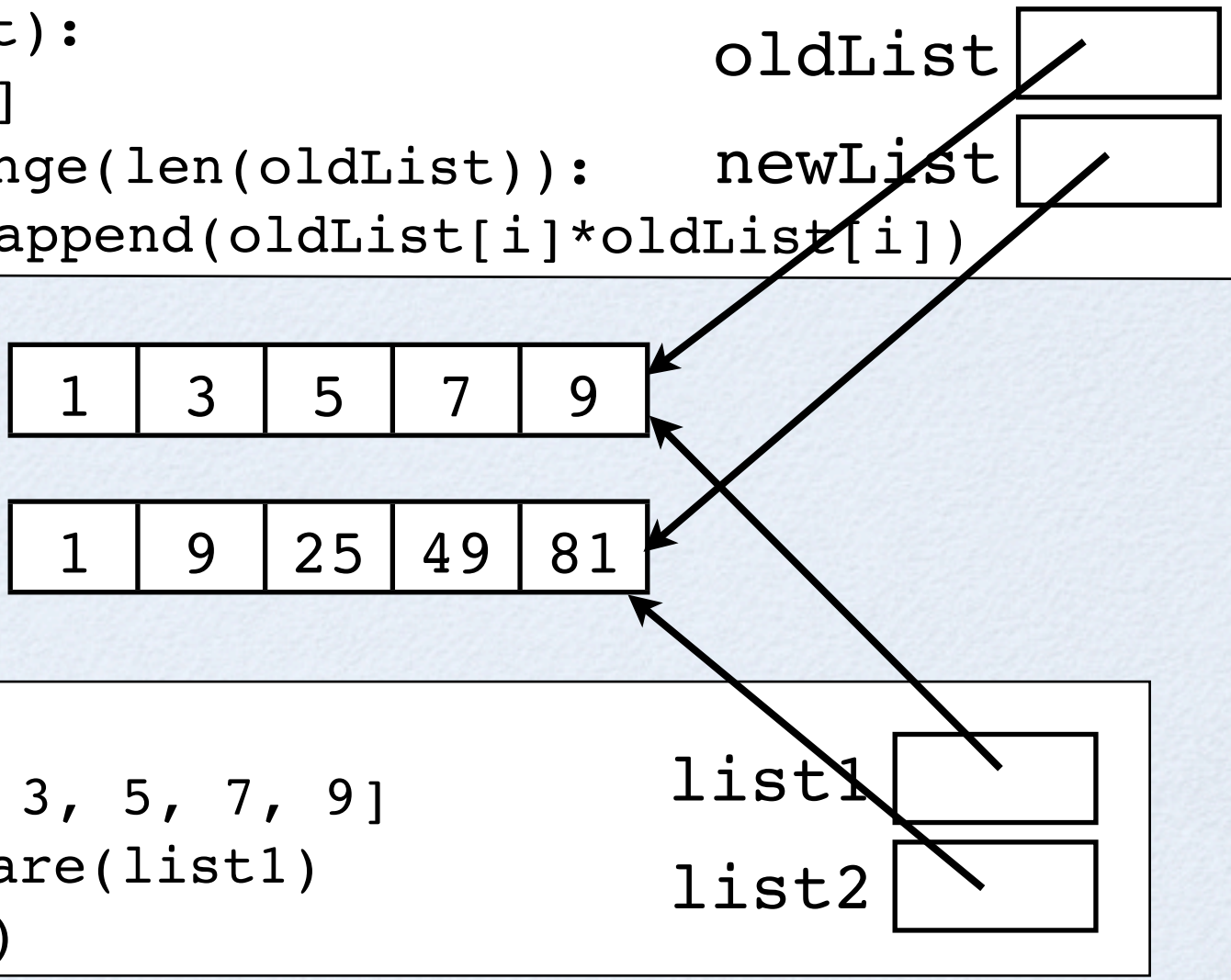
```
def myFunction(list):
    for i in range(len(list)):
        list[i] = list[i]*2

def main():
    a = [1, 3, 5, 7, 9]
    myFunction(a)
    print(a)
```

Lists and functions

- So far, we have been seeing lists that are created in the calling function (e.g. `main()`), and modified in the called function (e.g. `myFunction()`).
- Could we create a list in the called function, and return it to the calling function?

FUNCTIONS RETURNING LISTS

```
def square(oldList):
    newList = []
    for i in range(len(oldList)):
        newList.append(oldList[i]*oldList[i])
```




```
def main():
    list1 = [1, 3, 5, 7, 9]
    list2 = square(list1)
    print(list2)
```

EXERCISE

- You have a 2D list called `matrix`, which is created in `main()`. You want to pass `matrix` into a function, `columnSum()`, that will calculate the sum of each column in `matrix` and put it into a 1D list, `vector`. The resulting 1D list is then passed back to `main()`.

`matrix`

4	5	9	1	6
5	6	2	1	0
6	8	4	4	2

`vector`

15	19	15	6	8

```
def main():
  matrix = [[4, 5, 9, 1, 6],
            [5, 6, 2, 1, 0],
            [6, 8, 4, 4, 2]]

  vector = columnSum(matrix)
  print("Vector:", vector)
```