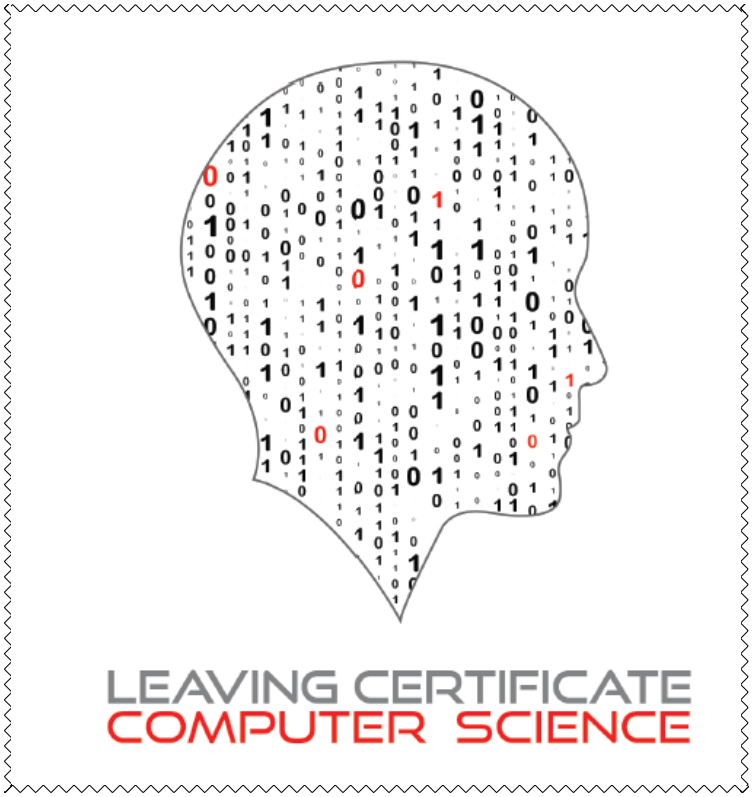
LC CS Python

Student Manual



Section 4

Lists

Name: _____

Introduction

A *list* is a collection of variables. The list type is commonly referred to as an *array* in other programming languages but as we shall see, there are some subtle differences between lists and traditional arrays.

Lists are useful because they provide us with a means of grouping several variables into a single variable. The value of each variable is known as a *list element* and these can be accessed by using the indexing/slicing techniques described in the previous section on strings.

Just think about it – so far, variables have been used to store **single values**. This has been the case regardless of the variable's datatype i.e. whether it is a string or numeric, only one value at a time can be stored in it.

Strings are a special type of list where the individual elements are the individual characters that make up the string. One key difference between lists and strings, however, is that lists are *mutable,* whereas strings are not i.e. you can overwrite a single element in a list.

The main purpose of lists it to provide a mechanism for dealing with 'a whole bunch of data' using a single variable.

KEY POINT: A list is a Python programming construct useful for modelling any real world data that can be grouped together under a common name.

Examples of lists include teacher names, schools, subjects, teams, lists of friends, a book list, a list of tweets, play lists (songs), shopping lists, a list of instructions, lists of countries, lists of capital cities, days of the week, months of the year, holiday dates, lists of numbers (e.g. lottery, ages, salaries, sales figures, heights etc.). The list of lists is endless!

Consider for example, scenarios where we needed to keep track of the number of times a particular event occurred. Let's say there are multiple possible outcomes and we need to maintain a count for each one individually. For some reason we might be asked to write a traffic survey program that counts the numbers of pedestrians, bicycles, cars, vans, HGVs

etc. passing a particular point at a particular time of the day. Without lists we would need to program separate variables for each count.

Creating Lists

The simplest kind of list is an empty list i.e. one that contains no elements. Empty lists are created by using a pair of square brackets as illustrated in the code below.

The code shown here to the right creates five different empty lists. The square brackets on the right hand side of each assignment tell Python that the datatype of the variable named on the left hand side is a list.

Remember lists are mutable. Even though these lists do not contain any data, their construction means that the program can add data to them at a later stage.

```
boysNames = []
girlsNames = []
favouriteSongs = []
fruits = []
vehicleCount = []
accountDetails = []
```

The following code snippet illustrates how to initialise lists with data.

```
boysNames = ['John', 'Jim', 'Alex', 'Fred']
girlsNames = ['Sarah', 'Alex', 'Pat', 'Mary']
favouriteSongs = ['Moondance', 'Linger', 'Stairway to Heaven']
fruits = ['Strawberry', 'Lemon', 'Orange', 'Raspberry', 'Cherry']
vehicleCount = [0, 0, 0, 0, 0, 0]
accountDetails = [1234, 'xyz', 'Alex', '1 Main Street', 827.56]
```

Note the use of square brackets on the right hand side and the use of commas to separate the individual list elements. The number of elements in the lists are 4, 4, 3, 5, 6 and 5 respectively.

Notice also that `boysNames`, `girlsNames`, `favouriteSongs` and `fruits` are all lists of strings; `vehicleCount` is a list of numbers, and `accountDetails` is a list of different datatypes.

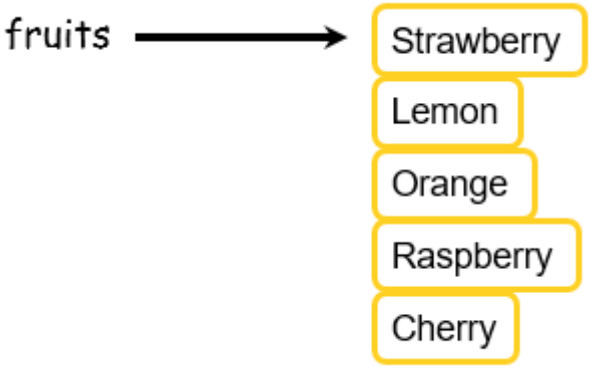
KEY POINT: A Python list can be made up of elements having different datatypes.

One of the key differences between lists and traditional arrays used in other programming languages is that a Python lists can be made up of data having a mixture of different datatypes, whereas the elements of an array must all be of the same data type.

It is useful to form a mental image of how lists are represented internally by the computer. Lists are frequently depicted in either a horizontal or vertical fashion as shown here.

A horizontal memory representation for the list `boysNames` would look like this.

A vertical memory representation for the list `fruits` would look as follows.



The key point is that the elements of the list should be envisaged in contiguous memory locations (just like the individual characters of a string as described in the previous section).

List five things you have learned about lists so far in this section.

1.

2.

3.

4.

5.

Common List Operations

Because lists and strings are both sequences, all of the common basic operations work in the same way for lists as they do for string. Specifically, lists can be added to one another (concatenation), multiplied together, indexed and sliced.

The commands `min`, `max`, and `len` also work for lists in the same way as they do for strings returning the minimum value, the maximum value and the number of elements (i.e. the length) in the list respectively.

Concatenation

Lists can also be constructed by concatenating two existing lists together. For example, we could join `boysNames` and `girlsNames` together using the concatenation operator (+) to form a new list called `names` as follows.

```
1. boysNames = ['John', 'Jim', 'Alex', 'Fred']
2. girlsNames = ['Sarah', 'Alex', 'Pat', 'Mary']
3. names = boysNames + girlsNames
4. print(names)
```

Notice that square brackets are not used on line 3.

When a list is used without square brackets like this, Python takes it that every element in the list is to be used.

The `print` command on line 4 displays the entire contents of the list. The output generated by the above code is shown here.

```
['John', 'Jim', 'Alex', 'Fred', 'Sarah', 'Alex', 'Pat', 'Mary']
```

The example below generates the exact same output, demonstrates that concatenation does not always result in a new list being created.

```
1. boysNames = ['John', 'Jim', 'Alex', 'Fred']
2. girlsNames = ['Sarah', 'Alex', 'Pat', 'Mary']
3. boysNames = boysNames + girlsNames
4. print(boysNames)
```

The list, `boysNames` is *extended* to include the list of girls' names.

Some Insights

The previous example provides us with an opportunity to gain some deeper insights into the Python programming language. Observe, that by the end of the example, the list variable `boysNames` will contain the names of four girls (at least we *think* they are girls) – *Sarah, Alex, Pat* and *Mary*. Just because we (humans) know that *Mary* and *Sarah* are definitely girls' names, it does not mean that Python knows this too. In fact, unlike humans, Python does not know the difference between a girl's name and a boy's name. (This is because the language was not designed to include any built in features to make such a distinction.). To Python, names are all just strings.

STUDENT TIP
Teachers should be aware that it is a common misconception for students to think that the name of a variable somehow determines the values that can be assigned it.

The fact that the previous program instructs Python to assign the girls' names to a variable that looks like it was named by the programmer to store the boys' names could mean one of two things. Either

a) `boysNames` was a poor choice of a variable name made by the programmer or,

b) the assignment was a mistake, again, made by the programmer

In both cases the programmer is the person responsible.

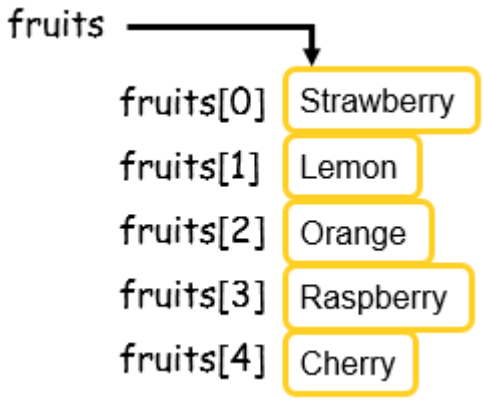
KEY POINT: Python has no understanding of natural language syntax and therefore has no way of inferring the intention of the programmer from the code.

List Indexing

Recall from the section on strings, that indexing is a technique used to access individual elements of a list. List indexing works just like string indexing.

A list element is accessed by using an index which is a zero-based positional value for that element. As was the case with strings, the index must be an integer (or an expression that evaluates to an integer), and, must be enclosed inside square brackets.

The graphic below depicts how index numbers can be included as part of the 'mental image' we formed for lists earlier.



KEY POINTS

- ✓ Every element in a list has a unique index.
- ✓ The index is the element's position in the list.
- ✓ The first element is at index position zero.
- ✓ The last element in a list of length *n* is at index position *n-1*
- ✓ Indices are enclosed in square brackets

Every list element is uniquely referenced by the list name and an index number.

Example program (fruit machine v1)

The next example program demonstrates how to combine the use of random numbers and lists to simulate the operation of a fruit machine.

- In handout, do **Task 1** (shown below).

Read the program carefully and see if you can figure out how it works.

```
1.  # Program to simulate a fruit machine!
2.  import random
3.
4.  # initialise the list of fruits - 5 elements
5.  fruits = ['Strawberry', 'Lemon', 'Orange', 'Raspberry', 'Cherry']
6.
7.  # generate three random numbers between 0 and 4 incl.
8.  selection1 = random.randint(0, 4)
9.  selection2 = random.randint(0, 4)
10. selection3 = random.randint(0, 4)
11
12. # show the results - display the fruits
13. print(fruits[selection1])
14. print(fruits[selection2])
15. print(fruits[selection3])
```

- Line 5 initialises a list of fruits.
- Lines 8, 9, and 10 each generate a random number between zero and four inclusive
- Lines 13, 14, and 15 each display an element from the list using the random numbers as the index.

Look up the online documentation for the Python `random` library.
What does the `choice` command do?
How might `choice` be used in the above program?

One final point worth noting is that when a list element is accessed, the datatype of the resulting object is the same as the datatype of the list element.

Changing the value of list elements

Lists are mutable objects. This means that the elements of a list can be changed (as well as accessed). Recall that this is not possible with strings because they are immutable.

It is therefore 'legal' to apply the index operator to a list variable on the left hand side of an assignment statement. The following short program demonstrates this.

- In handout, do **Task 2** (shown below).

```
1.  fruits = ['Strawberry', 'Lemon', 'Orange', 'Raspberry', 'Cherry']
2.  print(fruits)
3.
4.  fruits[0] = 'Apple'
5.  fruit = 'Melon'
6.  fruits[1] = fruit
7.  fruits[2] = 'Raspberry'
8.  fruits[3] = fruits[4]
9.  fruits[4] = 'Pineapple'
10.
11. print(fruits)
```

- Line 1 initialises a list of fruits and line 2 displays the contents of the list
- Lines 4-9, each make separate changes to the individual elements in the list that are 'housed' at the given index
- Line 11 displays the list again

When it is run the program displays the output shown below. Can you figure it out?

```
['Strawberry', 'Lemon', 'Orange', 'Raspberry', 'Cherry']
['Apple', 'Melon', 'Raspberry', 'Cherry', 'Pineapple']
```

Log any thoughts you have in relation to Python lists.

Index 'Out of Range' Errors

At runtime, Python always checks that index numbers lie *within bounds* for the object they are being used to access.

A list index will be *out of bounds* if it lies beyond the range of the list. If Python attempts to access a list element using an index that is out of bounds, it returns an *out of range index error* and the program will crash i.e. stop functioning.

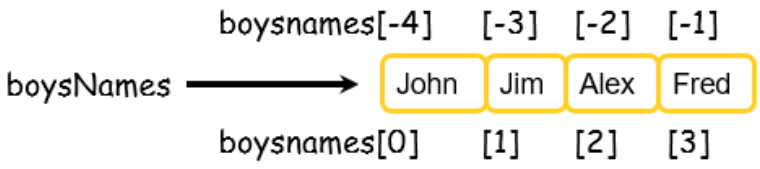
The whole idea of testing is to safeguard against system crashes happening in live (production) code.

```
1. # This is not something you want to see happening (too often!)
2. fruits = ['Strawberry', 'Lemon', 'Orange', 'Raspberry', 'Cherry']
3. print(fruits[5])
4. print("Hello World")       # This line will never get executed
5. lifeSupport = True  # Neither will this one
```

The above 'program' crashes on line 3, and lines 4 and 5 never get executed. Python displays the following error screen.

```
Traceback (most recent call last):
  File "C:\PDST\Work in Progress\Python Workshop\src\Session 4
- Lists\List1 - ops1.py", line 25, in <module>
    print(fruits[5]) # out of bounds
IndexError: list index out of range
```

Negative indices can be used on lists in just the same way as they could be used on strings. The last element of a Python list has an index of -1. Working backwards, the index of each element is one less than its predecessor. Therefore, the valid indices for a list made up of **four** elements would be -4 to 3 inclusive. This is illustrated below.



KEY POINT: In general, the index numbers of any sequence of length of n must lie within the range of $-n$ and $n-1$. So, $-n \leq index \leq n-1$

List Slicing

Lists can be quite long and sometimes we might just be interested in processing a portion of the data they contain. We can extract sub-lists from lists using the exact same technique that we used to extract substrings from strings earlier i.e. *slicing*.

A slice is a list of consecutive elements taken from another (larger) list.

Slices are created using the square brackets index operator. As was the case with strings, the colon delimits the start and end positions of the slice we are interested in extracting.

The technique of slicing is demonstrated in the program below.

```
# A program to demonstrate list slicing
fruits = ['apple', 'pear', 'orange', 'banana', 'kiwi']

print(fruits[1:3])  # ['pear', 'orange']
print(fruits[2:4])  # ['orange', 'banana']
print(fruits[2:5])  # ['orange', 'banana', 'kiwi']

print(fruits[1:])   # ['pear', 'orange', 'banana', 'kiwi']
print(fruits[:5])   # ['apple', 'pear', 'orange', 'banana', 'kiwi']
```

- In handout, do **Task 3** (code shown above).

KEY POINT: In general, the expression, **`aList[startPos:endPos]`** creates a new list or *slice* from the list identified by `aList`.

- ✓ The resulting slice starts from index position `startPos` in `aList`
- ✓ The resulting slice runs up to index position `endPos-1` in `aList`
- ✓ If `startPos` is missing it is taken to be zero
- ✓ If `endPos` is missing it is taken to be `len(aList)-1`

It is worth emphasising the point that a slice is in fact a new list.

For example, the following code results in the new list being stored in the list variable `exoticFruits`

```
fruits = ['apple', 'pear', 'orange', 'banana', 'kiwi']
exoticFruits = fruits[2:5]
```

The contents of this *new* list are: ['orange', 'banana', 'kiwi']

- In handout, do **Task 4** (shown below).

Time to experiment!
Let's say we have the following initialisation.

```
fruits = ['Strawberry', 'Lemon', 'Orange', 'Raspberry', 'Cherry']
```

Predict the output that would be displayed by each of the `print` statements in the program snippet below. Record your predictions in the left column.

```
1.  print(fruits[0])
2.  print(fruits[3])
3.  print(fruits[2])
4.  print(fruits[len(fruits)-1])
5.  print(fruits[1])
6.
7.  fruit = fruits[2+2]
8.  print(fruit)
9.  print(fruit[0])
10.
11. orange = fruits[1]
12. print(orange)
13. lemon = fruits[1]
14. print(lemon)
```

Prediction	***Actual***

Now key in the program and run it.
Record the actual output in the right column.

What value do you think the following expression would generate? Try it!

```
fruits[0][0]+fruits[1][0]+fruits[2][0]
```

_____

How does this answer shape your thinking in terms of the relationship between strings and lists?

_____

_____

_____

_____

_____

- In handout, do **Task 5** (shown below).

Can you find and suggest fixes for the syntax error contained in the code below?

```
# Initialise two lists
1.  suits = ['Hearts','Diamonds','Spades','Clubs']
2.  cardFaces = ['Ace','1','2','3','4','5','6','7','8','9','Jack','Queen','King']
3.
4.  print(cardFaces[1:10])
5.
6.  colourCards = cardFaces[10:23]
7.  print(colourCards)
8.
9.  redSuits = suits[:2]
10. blackSuits = [2:]
11.
12. print(pinStripSuits)
13. print(redSuits)
14. print(blackSuits)
```

Syntax Error:

Fix:

List Methods

As data structures go, lists are *very* flexible. Apart from the basic common operations that can be carried out on all sequences (e.g. concatenation, indexing, slicing), list objects also support a variety of additional type-specific commands (methods).

The table below introduces some of these, but a more complete reference can be found by browsing to the official Python page: https://docs.python.org/3/library/stdtypes.html#mutable-sequence-types

Method Name	Description
`aList.append(item)`	Appends the `item` to the list named `aList`
`aList.count(item)`	Returns an integer count of how many times the element `item` occurs in `aList`
`aList.extend(anotherList)`	Appends the list contained in `anotherList` to `aList`
`aList.index(item)`	Returns the index of the first occurrence of `item` in `aList`
`aList.insert(index, item)`	Inserts `item` into `aList` at offset `index`
`aList.pop()`	Removes and returns the last element from `aList`
`aList.remove(item)`	Removes `item` from the list
`aList.reverse()`	Reverses the order of all the items of `aList`
`aList.sort()`	Sorts objects of `aList` *in place* i.e. without creating a new list. A new sorted list can be created using the `sorted` built in command

The `del` keyword can also be applied to remove individual elements or entire slices from a lists. `del` is used in conjunction with the square bracket index operator as follows.

`del aList[i]`	Removes element at position `i` from `aList`
`del aList[i:j]`	Removes all elements from position `i` up to, but not including, position `j` in `aList`. (Same as `aList[i:j] = []`)
`del aList[:]`	Removes (clears) all the elements from `aList`

List Methods - Example 1

The following program demonstrates the use of several of these type-specific commands. The generated output is shown as a comment at the end of each print statement. You should read through the code and try to understand the list methods used.

- In handout, do **Task 6** (code shown below).

```
fruits = ['pear', 'apple', 'orange', 'banana', 'kiwi']
fruit = 'apple'
vegs = ['peas', 'carrots']


fruits.append(fruit)
print(fruits) # ['pear', 'apple', 'orange', 'banana', 'kiwi', 'apple']

fruits.extend(vegs)
print(fruits) # ['pear', 'apple', 'orange', 'banana', 'kiwi', 'apple', 'peas', 'carrots']

fruits.insert(2, fruit)
print(fruits) # ['pear', 'apple', 'apple', 'orange', 'banana', 'kiwi', 'apple', 'peas', 'carrots']

fruits.pop()
print(fruits) # ['pear', 'apple', 'apple', 'orange', 'banana', 'kiwi', 'apple', 'peas']

fruits.remove(fruit)
print(fruits) # ['pear', 'apple', 'orange', 'banana', 'kiwi', 'apple', 'peas']

fruits.reverse()
print(fruits) # ['peas', 'apple', 'kiwi', 'banana', 'orange', 'apple', 'pear']

fruits.sort()
print(fruits) # ['apple', 'apple', 'banana', 'kiwi', 'orange', 'pear', 'peas']

print(fruits.index(fruit)) # 0
print(fruits.count(fruit)) # 2
```

Notice that the same dot notation as was described for string methods is also used for lists. This means in order to use a list method, the programmer must type the name of the list variable followed by a dot, followed by the method's name, i.e.

```
<list-variable-name>.<method-name>
```

List Methods - Example 2

- In handout, do **Task 7** (code shown below).

```
# Sample program to build up a list of user details
userList = []
userName = input("Enter your name: ")
userList.append(userName)
userAge = int(input("What age are you "+userName+"?"))
userList.append(userAge)
userCountry = input("What is your country of birth?")
userList.append(userCountry)

print("My database for "+userName+"\n"+str(userList))
```

Two More Strings Methods (`split` and `splitlines`)

Another way to create lists is by using either of the string methods – `split` or `splitlines`.

These two methods (commands) are very similar to one another in the sense that they both break a string into separate 'tokens'. Both methods create a new list with each token becoming a separate list element.

Take for example the following code.

```
chomskyStr = "Colourless green ideas sleep furiously"
aList = chomskyStr.split()
print(aList)
```

- The first line initialises the variable `chomskyStr`

  [Aside: *Colourless green ideas sleep furiously* is a sentence composed by Noam Chomsky, (American linguist, philosopher and cognitive scientist), as an example of a sentence that is grammatically correct, but semantically nonsensical.]
- Line 2 creates a new list called `aList`. The string is tokenised and each word becomes a separate list element as shown.by the output:

```
['Colourless', 'green', 'ideas', 'sleep', 'furiously']
```

The main difference between `split` and `splitlines` is exemplified by the program below. (Notice the use of triple quotes to create the block string which spans multiple lines.)

```
seussStr = """I do not like green eggs and ham.
I do not like them Sam-I-am.
I do not like them here or there.
I do not like them anywhere.
I do not like them in a house
I do not like them with  a mouse"""
bList = seussStr.splitlines()
print(bList)
```

The string comes from a poem *Green Eggs and Ham* written by Dr Seuss, American author, political cartoonist and poet

The program creates a new list, `bList` - each line is a separate element as shown.

```
['I do not like green eggs and ham.', 'I do not like them Sam-I-am.',
'I do not like them here or there.', 'I do not like them anywhere.', '
I do not like them in a house', 'I do not like them with  a mouse']
```

- In handout, do **Task 8**.

Example Program (fruit machine v2)

The program below simulates a fruit machine. The sample outputs shown here should give a good idea of what the program does.

```
Cherry
Strawberry
Melon
```

```
Banana
Pineapple
Cherry
```

```
Melon
Pineapple
Pineapple
```

```
Strawberry
Banana
Strawberry
```

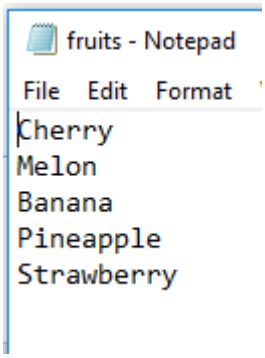
Fruit machine simulator program

```
# Program to simulate a fruit machine!
import random

# Open the fruits file (already created)
fruitFile = open("fruits.txt","r")

# Read the entire file
fileContents = fruitFile.read()

# Close the file
fruitFile.close()

# Split the content into a list
fruits = fileContents.split()

# Spin! Display three fruits
print(random.choice(fruits))
print(random.choice(fruits))
print(random.choice(fruits))
```

STUDENT TIP
Teachers should encourage students to look for recurring patterns in code.

Once such pattern is the use of the `split` and `splitlines` methods to put some structure on data such as the contents of a file.

The program works by reading the contents of a file called `fruits.txt` (shown here to the right) into a string called `fileContents`. The string is then `split` into individual tokens each of which are stored as separate elements of a list called `fruits`.



Finally, the `choice` command from the `random` library is used three times. Each time it picks a random element from the `fruits` list and displays it.

- In handout, do **Task 9**.