Lab 3 Summary

Lists and Iteration

- Study *Think Python* Chapter 10: Lists
- Study *Think Python* Chapter 7: Iteration

1. Storing and accessing data in a list

Up to this point, we have seen numbers, Boolean values, strings, functions, and a few other types. Once one of these objects has been created, it cannot be modified.

A *list* contains zero or more objects. Lists can be modified.

Consider counts of salmon and sea trout at Riding Mill on the River Tyne from 2008: https://www.gov.uk/government/statistical-data-sets/river-tyne-fish-counts

Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
27	15	26	97	691	4047	5281	4110	6138	6413	2323	158

We can use a list to keep track of the 12 int objects that contain the counts.

```
>>> fish = [ 27, 15, 26, 97, 691, 4047, 5281, 4110, 6138, 6413, 2323, 158]
>>> fish
[27, 15, 26, 97, 691, 4047, 5281, 4110, 6138, 6413, 2323, 158]
>>> type(fish)
<class 'list'>
```

The items in a list are ordered.

Each item has an *index* indicating its position in the list.

The first item in a list is at index 0, the second at index 1, and so on.

It would perhaps be more natural to use 1 as the first index, but many programming languages start indexing at 0.

To refer to a particular list item, we put the index in square brackets after a reference to the list (such as the name of a variable).

```
>>> fish[0]
27
>>> fish[1]
15
>>> fish[11]
158
>>> fish[12]
Traceback (most recent call last):
  File "<pyshell#74>", line 1, in <module>
    fish[12]
IndexError: list index out of range
>>> third = fish[2]
>>> third
26
```

Python also lets us index backward from the end of a list.

The last item is at index -1, the one before it at index -2, and so on.

Negative indices provide a way to access the last item, second-to-last item, etc, without having to figure out the size of the list.

```
>>> fish[-1]
158
>>> fish[-2]
2323
>>> fish[-12]
27
```

The empty list

An empty string is a string that doesn't contain any characters.

An empty list is a list with no items in it.

```
>>> ''
''
>>> type('')
<class 'str'>
>>> []
[]
>>> type([])
<class 'list'>
```

Lists are heterogeneous

Lists can contain any type of data, including integers, strings, and even other lists.

```
>>> person = ['Elizabeth','Queen',1926,1952,4]
>>> person
['Elizabeth', 'Queen', 1926, 1952, 4]
```

Modifying lists

Suppose we are typing a list and make a mistake with one of the elements. Rather than retyping the list we can assigned a new value to a specific element.

```
>>> footballers = ['Ronaldo','Messy','Neymar','Zidane']
>>> footballers
['Ronaldo', 'Messy', 'Neymar', 'Zidane']
>>> footballers[1] = 'Messi'
>>> footballers
['Ronaldo', 'Messi', 'Neymar', 'Zidane']
```

Lists a *mutable*, i.e., the contents of a list can be *mutated*.

Suppose L is a list.

- If L[i] is used in an expression then it means to get the value referred to at index i of list L.
- If L[i] is used on the left of an assignment statement, it means to overwrite index i of list L with the value evaluated on the right of the assignment statement into the

Numbers and strings are *immutable*, i.e.., can cannot be mutated. You cannot change a letter in string. Methods that appear to do that actually create a new string.

```
>>> name = 'Einstein'
>>> name[2] = 'x'
Traceback (most recent call last):
  File "<pyshell#19>", line 1, in <module>
    name[2] = 'x'
TypeError: 'str' object does not support item assignment
```

Operations on Lists

len(L)	Returns the number of items in list L.
max(L)	Returns the maximum value in list L.
min(L)	Returns the minimum value in list L.
sum(L)	Returns the sum of the values in list L.
sorted (L)	Returns a copy of list L where the items are in order from smallest to largest. *This does not mutate L.*

Slicing lists

```
>>> footballers = ['Ronaldo','Messi','Neymar','Zidane']
>>> footballers[0:2]
['Ronaldo', 'Messi']
>>> footballers[1:3]
```

```
['Messi', 'Neymar']
>>> footballers[1:]
['Messi', 'Neymar', 'Zidane']
>>> footballers[:3]
['Ronaldo', 'Messi', 'Neymar']
```

Copy of a list vs alias of a list

To create a copy of the entire list, omit both indices so that the slice runs from the start of the list to the end.

```
>>> footballers = ['Ronaldo','Messi','Neymar','Zidane']
>>> footballers_copy = footballers[:]
>>> footballers_copy
['Ronaldo', 'Messi', 'Neymar', 'Zidane']
>>> footballers_copy[0] = 'RRRonaldo'
>>> footballers_copy
['RRRonaldo', 'Messi', 'Neymar', 'Zidane']
>>> footballers
['Ronaldo', 'Messi', 'Neymar', 'Zidane']
```

An *alias* is an alternative name for something.

```
>>> footballers_alias = footballers
>>> footballers_alias
['Ronaldo', 'Messi', 'Neymar', 'Zidane']
>>> footballers
['Ronaldo', 'Messi', 'Neymar', 'Zidane']
>>> footballers_alias[1] = 'MMMessi'
>>> footballers_alias
['Ronaldo', 'MMMessi', 'Neymar', 'Zidane']
>>> footballers
['Ronaldo', 'MMMessi', 'Neymar', 'Zidane']
```

So if two variables refer to the same list, then any changes you make to one will be "seen" by the other.

List methods

Lists are objects and so they have methods.

Here is a useful table of list methods:
https://www.w3schools.com/python/python_ref_list.asp

```
>>> footballers = ['Ronaldo','Messi','Neymar','Zidane']
>>> footballers.reverse()
>>> footballers
['Zidane', 'Neymar', 'Messi', 'Ronaldo']
```

Warning! – Many list methods return the value None rather than creating a returning a new list. Remember this when you think a list has disappeared.

Lists of lists

```
>>> clubs = [['Neymar','PSG'],['Messi','Barcelona']]
>>> clubs
[['Neymar', 'PSG'], ['Messi', 'Barcelona']]
>>> clubs[0]
['Neymar', 'PSG']
>>> clubs[0][1]
'PSG'
```

2. Repeating code using loops

We have seen that if, if-else, if-elif and if-elif-else statements are used for making choices.

We say they are *control flow* statements because they control the way the computer executes programs.

Here we introduce another fundamental kind of control flow: repetition.

Up to now, to execute an instruction two hundred times, you would need to write that instruction two hundred times. Now you will see how to write the instruction once and use loops to repeat that code the desired number of times.

Loop over items in a list or string

```
L = [6,3,9,7,2]
for x in L:
    print('x:',x,'  2*x:',2*x)

country = 'United Kingdom'
for c in country:
    if c.isupper():
        print(c)
```

Loop over a range of numbers

```
>>> range(5)
range(0, 5)
>>> for i in range(5):
      print(i)

0
1
2
3
```

```
4
>>> list(range(5))
[0, 1, 2, 3, 4]
>>> list(range(1))
[0]
>>> list(range(0))
[]
>>> list(range(1,5))
[1, 2, 3, 4]
>>> list(range(5,10))
[5, 6, 7, 8, 9]
>>> list(range(2000,2050,4))
[2000, 2004, 2008, 2012, 2016, 2020, 2024, 2028, 2032, 2036, 2040, 2044, 2048]
>>> list(range(2050,2000,-4))
[2050, 2046, 2042, 2038, 2034, 2030, 2026, 2022, 2018, 2014, 2010, 2006, 2002]
>>> list(range(2000,2050,-4))
[]
```

Processing lists using indices

```
def find_max(L):
    """Find the largest value in the list"""
    max_value = None
    if (len(L)>=1):
        max_value = L[0]
        for i in range(1,len(L)):
            if (L[i] > max_value):
                max_value = L[i]
    return(max_value)

print(find_max([1,7,3,2]))
```

Looping until a condition is reached

for loops are useful only if you know how many iterations of the loop you need.

In some situations, it is not known in advance how many loop iterations to execute, so we use a while loop.

```
import random
def count_dice_rolls(value_wanted):
    """Count dice rolls until value wanted is rolled."""
    finished = False
    count = 0
    while (not finished):
        d = random.randint(1,6)
        count = count + 1
        print('rolled a',d)
        print('count so far is',count)
        finished = (d==value_wanted)
    return(count)

print(count_dice_rolls(3))
```

Nested loops

```
for i in range(5):
    for j in range(5):
        if ((i+j)%2==0):
            print('#', end='')
        else:
            print('.', end='')
    print('')
```

3. Sorting a list

There are many algorithms for sorting a list into order.

Here we take a look at *insertion sort*.

To get an idea of how it works, have a look at this animation:
http://www.algomation.com/algorithm/insertion-sort-animated

```
def insertion_sort(L):
    """Insertion sort algorithm"""
    i = 1
    while (i < len(L)):
        j = i
        while ((j > 0) and (L[j-1] > L[j])):
            # swap L[j] and L[j-1]
            L[j],L[j-1] = L[j-1],L[j]
            j = j - 1
        i = i + 1

L = [7,3,6,2,8,1]
insertion_sort(L)
print(L)
```

Summary

Lists

- Lists are used to keep track of zero or more objects. The objects in a list are called items or elements. Each item has a position in the list called an index and that position ranges from zero to one less than the length of the list.
- Lists can contain any type of data, including other lists.
- Lists are **mutable**, which means that their contents can be modified.
- Slicing is used to create new lists that have the same values or a subset of the values of the originals.
- When two variables refer to the same object, they are called aliases.

Iteration

- Repeating a block of code is a fundamental way to control a program's behaviour.
- A for loop can be used to iterate over the items of a list, over the characters of a string, and over a sequence of integers generated by built-in function range.
- The most general kind of repetition is the while loop, which continues executing as long as some specified Boolean condition is True. However, the condition is tested only at the beginning of each iteration. If that condition is never false, the loop will be executed forever.
- Control structures like loops and conditionals (if-else) can be nested inside one another to any desired depth.

Exercise.

The variable kingdoms is the list defined as follows.

```
kingdoms = ['Bacteria','Protozoa','Chromista','Plantae','Fungi','Animalia']
```

Using kingdoms and either slicing or indexing with positive indices, write expressions that produce the following.

(a) The first item of kingdoms.

(b) The last item of kingdoms.

(c) The list ['Bacteria','Protozoa','Chromista']

(d) The list ['Chromista','Plantae','Fungi']

(e) The list ['Fungi','Animalia']

(f) The empty list.

Exercise.

Using nested loops, print a right triangle of the character T where the triangle is one character wide at top and seven characters wide at the bottom.

T
TT
TTT
TTTT
TTTTT
TTTTTT
TTTTTTT