Data Structures

Data Structure

- So far we have used variables that can store a single item in them.
-
- if you use the random.choice([“blue”, “red”, “black”]) we are picking a random item from the list of possible options.

- One item can hold several pieces of data and in our example above, a collection of colours.
- There are several ways that collections of data can be stored as a single item in Python.
- `List`, `Strings`, `Tuples`, `Dictionary`, Deque, Heap

Lists

- List can store different types of data at the same time.
- Lists are `iterable`: can be looped over
- List can be broken in little element called index. The element are numbered sequentially with integer (index), starting from 0.

- Most common way to store a collection of data under one variable name in Python

- The square brackets define the group of data as a list.

- A list can have another list inside (see example)

Index for each element in the main list. Notice the list inside is considered as a single element with an index of 2

Our list with a list inside it

```
4
5 @author: mireilla
6 """
7            0   1     2      3  4  5
8
9 our_list = [1, 2, [3, 4, 5], 6, 7, 8]
10
11                 0  1  2
12
13
14 our_list2 = our_list[2]
15
16 our_list3 = our_list[2][0]
17
18 print(our_list)
19
20 print(our_list2)
21
22 print(our_list3)
```

Index for elements inside second list

Will display element at index 2 of our list

This will display the element at index 0 in our second list

```
In [26]: runfile('C:/Users/mireilla/.
[1, 2, [3, 4, 5], 6, 7, 8]
[3, 4, 5]
3
```

This is the output of the code above

List - Accessing individual elements

- The data in a list does not all have to be of the same type. For example, the same list can store both strings and integers or floating point numbers.

```
our_list = [1, "Orange", [3, 4, 5], 6, 7, 8, "blue", 1.4]
```

```
In [30]: runfile('C:/Users/mireilla/.spyder-py3/
[1, 'Orange', [3, 4, 5], 6, 7, 8, 'blue', 1.4]
```

- List are mutable: The content of a list can change while the programme is running.

- An individual list can be updated using an assignment statement.

The name of the list followed by the element's index enclosed in squared bracket. In this case we want to replace "Orange" which is at index 1 with "purple"

```
our_list = [1, "Orange", [3, 4, 5], 6, 7, 8, "blue", 1.4]
our_list[1] = ["purple"]
```

```
In [31]: runfile('C:/Users/mireilla/.spyder-py3/li
[1, ['purple'], [3, 4, 5], 6, 7, 8, 'blue', 1.4]
```

Loops and List

- A `for` loop executes once for each item in a collection. The collection can be a range of integer or a list.
- Sometimes lists are constructed which iterate over a list's indices instead of its value using `len` function.
- `Len` can be used with the `range` function: it will construct a collection of integers that includes all of the indices for a list

```
#Initialise data and total
data = [2.71, 2.14, 1.41, 1.62]
total = 0

#Total the values in data
for value in data:
    total = total + value

#Display the total
    print("The total is: ", total)
```

```
The total is:  2.71
The total is:  4.85
The total is:  6.26
The total is:  7.88
```

```
#Initialise data and largest_pos
data = [1.62, 1.41, 3.14, 5.74,  ]
largest_pos = 0

#Find the position of the largest element
for i in range(1, len(data)):
    if data[i] > data[largest_pos]:
        largest_pos = i

#Display the result
    print("The largest value is: ", data[largest_pos], \
          "which is at index", largest_pos)
```

The first argument is 1 and the second argument is the length of *data*, which is 4. As a result, range returns a collection of sequential integers from 1 up to and including 3, which is all the indices for all the elements except the first

Loops and List

- A `while` loop can also be used when working with lists.

```python
#Initiliase data
data = [0, -1, 4, 1, 0]

# Loop while i is a valid index and
# the value at index i is not a positive value
i = 0
while i < len(data) and data[i] <= 0:
    i = i +1

# If i is less than the length of data then
# the loop terminates beacuse a positive number was found.
# Otherwise, i will be equal to the lenght of data,
# indicating that a positive number was found.
if i < len(data):
    print("The first positive number is at index", i)
else:
    print("The list does not contain a positive number")
```

```
The first positive number is at index 2
```

Additional List Operations

- Tasks such as inserting a new element into a list and removing an element from a list are performed by applying a method to a list.

- A `method`, much like a function, is a collection of statements that can be called upon to perform a task.

- Elements can be added at the end of the list using the `append` method

- Element can be inserted at any location using the `insert` method

```
our_list.append(input("Add another colour: "))
```

```
Add another colour: lime
[1, ['purple'], [3, 4, 5], 6, 7, 8, 'blue', 1.4, 'lime']
```

This ask the user to enter a colour and will add it to the end of the list

```
our_list = [1, "Orange", [3, 4, 5], 6, 7, 8, "blue", 1.4]
our_list[1] = ["purple"]
our_list.insert(2, "I am taking over index 2 in our list")
```

```
[1, ['purple'], 'I am taking over index 2 in our list', [3, 4, 5], 6, 7, 8, 'blue', 1.4]
```

Here, our new element "I have remove.

Additional List Operations

- `del` our_list[4] – deletes item with index 4. This means the 5th item which is 7.

- any_list`.sort` – will sort the any_list into alphabetical order and saves the list in a new order.
  - This does not work if the list is storing data of different type such as the list we have been using: our_list[].

- The `pop` method is used to remove an element at a particular index from the list. If the index of the element to be removed is not given as an argument, the last element from the list is removed.

- The `remove` method can also be used to remove a value from the list. When executed, it removes the first occurrence of its argument from the list.

```
our_list = [1, "Orange", [3, 4, 5], 6, 7, 8, "blue", 1.4]

del our_list [4]   # Delete elemnt at index 4

our_list.remove("Orange")  # remove orange from the list

our_other_list = our_list.pop() # Remove the last element 1.4
```

```
[1, 'I am taking over index 2 in our list', [3, 4, 5], 7, 8, 'blue']
```

Rearranging the elements in a List

- Two elements in a list can be swapped using a series of assignments statements.

- The reverse method will reverse the order of the elements in the list.

- The sort method sorts the element in ascending order.

- Both sort and reverse can be applied without any argument.

```python
# Create a list
data1 = [1.62, 1.41, 3.14, 5.74, 98, 3.45 ]

#Swap the element at index 2 with the element at index 5
temp = data1[2]
data1[2] = data1[5]
data1[5] = temp

# Display te modified list

print(data1)
```

```
[1.62, 1.41, 3.45, 5.74, 98, 3.14]
```

```python
# Create a new empty list
values = []

# Read values from the user and store them in
# a list until a blank line is entered
line = input("Enter a number (blank line to quit): ")
while line != "":
    num = float(line)
    values.append(num)

    line = input("Enter a number (blank line to quit): ")

# Sort the values into ascending order
values.sort()

# Display the values
for v in values:
    print(v)
```

```
Enter a number (blank line to quit): 67

Enter a number (blank line to quit): 09

Enter a number (blank line to quit): 78

Enter a number (blank line to quit): 23

Enter a number (blank line to quit): 90

Enter a number (blank line to quit): 654

Enter a number (blank line to quit): 12

Enter a number (blank line to quit):
9.0
12.0
23.0
67.0
78.0
90.0
654.0
```

Searching a List

- Python's in operator allows us to determine whether or not a particular value is in the list. If presents, the expression evaluates to True, otherwise False.

- The index method is used to identify the position of a particular value with a list

```python
# Create a new empty list
more_data = []

# Read values from the user and store them in
# a list until a blank line is entered
line = input("Enter an integer(blank line to quit): ")
while line != "":
    n = int(line)
    more_data.append(n)

    line = input("Enter an integer (blank line to quit): ")

# Read additional integer from the user
y = int(input("Enter one additional integer: " ))

# Display the index of the first occurrence of y (if it is present in the list
if y in more_data:
    print ("The first", y, "is at index", more_data.index(y))
else:
    print(y, "is not in the list")
```

```
Enter an integer(blank line to quit): 89

Enter an integer (blank line to quit): 09

Enter an integer (blank line to quit): 7

Enter an integer (blank line to quit): 8945

Enter an integer (blank line to quit): 23

Enter an integer (blank line to quit): 90

Enter an integer (blank line to quit): 09

Enter an integer (blank line to quit):

Enter one additional integer: 7
The first 7 is at index 2
```

Dictionaries

- Allows you to store the data, look up a key then return a value.
- The content can be changed while the programme is running
- Unlike a list or tuple, does not have an order. It uses a key instead of index.
- Determine by the use of braces {}
- Each `value` in a dictionary must have a `key` associated with it.
- The dictionary key can be integers, floating-point numbers or strings
- The value associated with a key can be integers, floating-point numbers, strings or a Boolean value; it can also be a list or another dictionary.
- `Key-value pair`: A dictionary and its associated value
- Key must be unique but values do not have to be unique.

Example1: the key is a string and value is a floating-point number

Makes changes to the data associated with key

Example2: Creates a dictionary called "colours" where each key is assigned a value.

The key is an integer and value is a string

```
constants = {"pi": 3.14, "e": 2.71, "root": 1.41}
print(constants)
constants["e"] = 2.80
print(constants)
```

```
In [67]: runfile('C:/Users/mireilla/.s
{'pi': 3.14, 'e': 2.71, 'root': 1.41}
```

```
colours = {1: "red", 2: "blue", 3: "green"}
print(colours)
colours[3] = "purple"
print(colours)
```

```
In [84]: runfile('C:/Users/mireilla
{1: 'red', 2: 'blue', 3: 'green'}
{1: 'red', 2: 'blue', 3: 'purple'}
```

Dictionary – Accessing, Modifying and Adding Values

- Accessing a value in a dictionary is similar to accessing a value in a list.

```python
# Create a new dictionary with 2 key-value pairs
results = {"pass": 0, "fail": 0}

# Add a new key-value pair to the dictionary
results["withdrawal"] = 1

# Update two values in the dictionary
results["pass"] = 3
results["fail"] = results["fail"] + 1

# Display the values associated with fail, pass and withdrawal respectively
print(results["fail"])
print(results["pass"])
print(results["withdrawal"])
```

```
In [70]: runfile('C:
1
3
1
```

- You can access all the dictionary keys by turning the dictionaryName.keys into a list as seen in the example below.

```python
constants = {"pi": 3.14, "e": 2.71, "root": 1.41}
print(constants)

a = list(constants.keys())
print(a)
```

```
In [90]: runfile('C:/Users/mireilla/.spyd
{'pi': 3.14, 'e': 2.71, 'root': 1.41}
['pi', 'e', 'root']
```

Removing a Key-Value Pair & Additional Dictionary operations

- A key-value pair is removed using `pop` the method.
- The key to be removed must be provided as argument. When the method is executed, both the key and its value are removed.

- The `len` function determines how many key-value pairs are in a dictionary. It returns 0 if the dictionary is empty.

- The `in` operator determines whether or not a particular key or value is present in the dictionary.

- The `in` operator used with the `values` method can be used to determine whether or not a value is present in a d

```
if x in constants.values():
    print("At least one of the value in constants is: ", x)
else:
    print("none of the values in constant are: ", x)
```

Loops and Dictionaries

- A `for` loop can be used to iterate over all of the keys in a dictionary

```
# Create a dictionary
constants = {"pi": 3.14, "e": 2.71, "root 2": 1.41}

# Print all of the keys and values with nice formatting
for k in constants:
    print("The value associated with", k, "is", constants[k])
```

```
In [74]: runfile('C:/Users/mireilla/.spyde
The value associated with pi is 3.14
The value associated with e is 2.71
The value associated with root 2 is 1.41
```

- A `for` loop can also be used to iterate over the values in a dictionary instead of the keys. This is done by applying the `values` method, which does not take an argument.

```
# Create a dictionary
constants = {"pi": 3.14, "e": 2.71, "root 2": 1.41}

# Compute the sum of all the value values in the dictionary
total = 0
for v in constants.values():
    total = total + v

# Display total
print("The total is: ", total)
```

```
In [80]: runfile('C:/Us
The total is:  7.26
```

Loops and Dictionaries

- Some problems involving dictionary are better solved with a `while` loop than `for` loops.

- For example, the following programme uses a while loop to read strings from a user until 5 unique values have been entered. Then all of the strings are displayed with their counts

```python
# Create a dictionary
constants = {"pi": 3.14, "e": 2.71, "root 2": 1.41}

# Compute the sum of all the value values in the dictionary
total = 0
for v in constants.values():
    total = total + v

# Display total
print("The total is: ", total)


# Count how many times each string is entered by the user
counts = {}

# Loop until 5 distinct strings have been entered
while len(counts) < 5:
    s = input("Enter a string: ")

    # If s is already a key in the dictionary then increase its count by 1.
    #Otherwise add s to the dictionary count of
    if s in counts:
        counts[s] = counts[s] + 1
    else:
        counts[s] = 1

# Display all of the strings and their counts
for k in counts:
    print(k, "occured", counts[k], "times")
```

Tuples

- Similar to list, just few differences:
- Once you have created a tuple, it cannot be changed. `Immutable` data type.
- `Iterable` - It can be looped over.
- Similar to a string, a tuple does not support item assignment.
- Tuples are good for storing data that you don't want to be accidentally changed. They are usually used for menu items that would not need to be changed.
- The round brackets `()` determine a tuple.

Tuples

- Create and display a tuple

```
# Create a variable named fruit_tuple which stores
# for pieces of fruit within it.
fruit_tuple = ("Apple", "Banana", "Strawberry", "Avocado")
print(fruit_tuple)
```

```
In [91]: runfile('C:/Users/mireilla/.spyder-p
('Apple', 'Banana', 'Strawberry', 'Avocado')
```

- Display the index of an item in a tuple

```
# Display the index of the the item "Banana"
print(fruit_tuple.index("Banana"))
```

- Display an item given a known index in a tuple

```
# Display item with index 3 from fruit_tuple
print(fruit_tuple[3])
```

```
Avocado
```

Numeric Arrays

- Pythons arrays can only store numbers.
- These numbers can have varying ranges, but all pieces of data must have the same data type (see table below).
- When you create an array, you need to define the type of data it will contain.
- You cannot change that type while the programme is running

Type code	Common name	Description	Size in bytes
'i'	Integer	Whole number between -32,768 and 32,767	2
'l'	Long	Whole number between -2,147,483,648 and 2,147,483,648	4
'f'	Floating-point	Allows decimal places with number ranging from -10^{38} to 10^{38}	4
'd'	Double	Allows decimal places with numbers ranging from -10^{308} to 10^{308}	8

```python
# First line so that Python can use the array library
from array import *

# Creates an array called "nums". It uses the integer data type and has 6 items
nums = array('i', [45,78,345,98,12,78])
print("This is the original array: ", nums)

# Asks the user to enter a new number
# which will be added to the end of the existing array
newNumber = int(input("Enter a number: "))
nums.append(newNumber)
print("This is the array with ther new number: ", nums)

# Reverse the order of the array
nums.reverse()
print("This is the reversed array: ", nums)

# Sort the array into ascending order
nums = sorted(nums)
print(" This is the sorted array: ", nums)

#Remove the last item from the array
nums.pop()
print(" This is the array after removing the last number: ", nums)


# Create a blank array called "NewArray" which uses the integer type 'i'
newArray = array('i', [])

# Ask the user how many items they want to add and then append these new items
# to newArray.
more = int(input("How many numbers in total do you want to add? "))
for y in range (0, more):
    newValue = int(input("Enter a number: "))
    newArray.append(newValue)

print("This is the newArray array after adding the items: ", newArray)

# Join the content of newArray and nums array
nums.extend(newArray)

print("This is the nums array after joining it with newArray: ", nums)

# Ask the user to enter the item they want to get rid of and remove the first
# item that matches that value from the array.
getRid = int(input("Enter the number you want to remove: "))
nums.remove(getRid)

# Display how many time the value "78" appears in the array.
print("78 appears ",  nums.count(78), "times")
print("This is the array after removing the requested number ", nums)
```

Numeric Arrays – example code

Numeric Arrays

Output from the code we in the previous slide

```
In [27]: runfile('C:/Users/mireilla/.spyder-py3/python_arrays_code.py', wdir='C:/Users/mireilla/.
This is the original array:  array('i', [45, 78, 345, 98, 12, 78])

Enter a number: 200
This is the array with ther new number:  array('i', [45, 78, 345, 98, 12, 78, 200])
This is the reversed array:  array('i', [200, 78, 12, 98, 345, 78, 45])
 This is the sorted array:  [12, 45, 78, 78, 98, 200, 345]
 This is the array after removing the last number:  [12, 45, 78, 78, 98, 200]

How many numbers in total do you want to add? 3

Enter a number: 909

Enter a number: 808

Enter a number: 707
This is the newArray array after adding the items:  array('i', [909, 808, 707])
This is the nums array after joining it with newArray:  [12, 45, 78, 78, 98, 200, 909, 808, 707]

Enter the number you want to remove: 12
78 appears  2 times
This is the array after removing the requested number  [45, 78, 78, 98, 200, 909, 808, 707]
```

```
# Displays the array with each item appearing on a separate line
for x in nums:
    print(x)
```

2D Lists and dictionaries

	0	1	2
0	45	67	57
1	65	58	34
2	23	89	35
3	56	76	87

- A 2Dlist uses a standard Python indexing.

```python
# A two-dimensional list of student grades
grades = [[45, 67, 57],[65, 58, 34],[23, 89, 35],[56,76,87]]
print(grades)

# A two-dimensional dictionaries of student grades
grades = [{"Ma":45, "En":67, "Fr":57},{"Ma":65, "En":58, "Fr":34},{"Ma":23, "En":89, "Fr":35},{"Ma":56,"En":76,"Fr":87}]

# Print the english result for the first student
print(grades[0] ["En"])

# A two-dimensional dictionaries of student grades
grades = {"Paul":{"Ma":45,"En":67, "Fr":57}, "Mary":{"Ma":23, "En":89, "Fr":35}}

# Display the grade for Paul's Maths exam
print("The grade for Paul's maths exam is: ", grades["Paul"]["Ma"])

# Display all exam results for Mary
print("This is Mary's exam result: ", grades["Mary"])

# Change the mary's exam result to 55
grades["Mary"]["Fr"] = 55

# Display all exam results for Mary
print("This is Mary's exam result: ", grades["Mary"])

# Add another row of data to the 2 D dictionary - name would be the row
# and Maths and English would be the coloumns indexes
grades["Rita"]={"Ma":70, "En":69}
print(grades)

# Display only the name and the English grade for each student
for name in grades:
    print((name), grades[name]["En"])
```

```
In [42]: runfile('C:/Users/mireilla/.spyder-py3/two_d_list_ar
[[45, 67, 57], [65, 58, 34], [23, 89, 35], [56, 76, 87]]
67
The grade for Paul's maths exam is:  45
This is Mary's exam result:  {'Ma': 23, 'En': 89, 'Fr': 35}
This is Mary's exam result:  {'Ma': 23, 'En': 89, 'Fr': 55}
{'Paul': {'Ma': 45, 'En': 67, 'Fr': 57}, 'Mary': {'Ma': 23, '
Paul 67
Mary 89
Rita 69
```