Data Processing

Wrestling with Python

Overview

- The Story so far….
- A Data Processing Problem
  - Reading data into a list
  - Printing a list
  - Sorting a list
- A data query

THE STORY SO FAR

What we can do so far

- Store data (using variables)
- Change data (using assignments)
- Make decisions (using conditions)
- Loop round statements(using while)
- Loop through sequences(using for)
- Store collections of data(using lists)

PROCESSING DATA

Our Problem

- We want to read in a number of scores for a class test
- Then we want to print them out in ascending order
- Then we want to do things like find out how many people scored more than 50

University of Hull

Data in programs

- There are essentially two kinds of data that we might like to store
- Single values
  - The score I got in the exam
- Multiple values
  - All the scores for the class

Single Values

```
score = 99  # stores the value 99 in score
```

- This statement creates a single variable called `score` that holds the score of one student
- The Python system can work out that we are storing numeric values
  - If we assigned a string the variable would hold strings

Multiple Values

```
scoreList = []  # create a list called scoreList
```

- This statement creates a single variable called `scoreList` that we can use to hold lots of scores
- This variable is a *list* which can hold an unlimited number of items
- A list provides behaviours we can use to manage the data stored in it

Storing Values in a list

```
scoreList.append(99) # put the value 99 in scoreList
```

- This value puts the value 99 in the list at the end of it
- A program can add things to a list simply by appending them to the end using the append behaviour above
- You can put lots of things in a single list

Problem: How to we store data?

- We need to read a bunch of test scores from the user and store them in a list
  - It would be nice if the user didn't have to tell the program in advance how many numbers are being entered
- To make this work we might make the program stop reading numbers when the user gives an empty string rather than a value

Storing numbers

```
scoreList = []
while(True):
    scoreString = input("Enter value (empty string to stop):")
    if scoreString == "":
        break
    scoreValue = int(scoreString)
    scoreList.append(scoreValue)
```

- This will repeatedly read in and store values until the user enters an empty string

Storing numbers

```
scoreList = []
while(True):
    scoreString = input("Enter value (empty string to stop):")
    if scoreString == "":
        break
    scoreValue = int(scoreString)
    scoreList.append(scoreValue)
```

- This statement creates the list that will hold the data
- The program will append things to this list

Storing numbers

```
scoreList = []
while(True):
    scoreString = input("Enter value (empty string to stop):")
    if scoreString == "":
        break
    scoreValue = int(scoreString)
    scoreList.append(scoreValue)
```

- This creates a loop that will go on for ever
  - Actually it won't because the user can break out of the loop by entering an empty score value

Storing numbers

```
scoreList = []
while(True):
    scoreString = input("Enter value (empty string to stop):")
    if scoreString == "":
        break
    scoreValue = int(scoreString)
    scoreList.append(scoreValue)
```

- This reads in the text of the score value and stores it in a variable called `scoreString`

Storing numbers

```
scoreList = []
while(True):
    scoreString = input("Enter value (empty string to stop):")
    if scoreString == "":
        break
    scoreValue = int(scoreString)
    scoreList.append(scoreValue)
```

- This makes the loop stop if the user enters an empty score string
- The `break` statement causes the program to break out of the loop

Storing numbers

```
scoreList = []
while(True):
    scoreString = input("Enter value (empty string to stop):")
    if scoreString == "":
        break
    scoreValue = int(scoreString)
    scoreList.append(scoreValue)
```

- This converts the string that the user typed into an integer that we can store in the list of scores

Storing numbers

```
scoreList = []
while(True):
    scoreString = input("Enter value (empty string to stop):")
    if scoreString == "":
        break
    scoreValue = int(scoreString)
    scoreList.append(scoreValue)
```

- This appends the score value on the end of the list of scores
- This is where the scores are actually stored

Problem: Printing the data

- We have now read in the data and stored it in a list
- It might be nice to print it out next
- To make this work we need a construction that will work through a list one element at a time
  - The for loop construction would be good for this

Printing numbers

```
for score in scoreList:
    print(score)
```

- The for loop construction is very good for working through collections
- Each time round the loop the value in score is the next one in the list
- The loop stops automatically at the end of the list

Printing numbers

```
for score in scoreList:
    print(score)
```

- Set up the for loop to work through the list
- The variable `score` will take the value of each successive element in turn
- Python does all this for us automatically

Printing numbers

```
for score in scoreList:
    print(score)
```

- This prints the value in `score`
- The statement is performed repeatedly by the loop (how do we know this?)

Printing numbers

```
for score in scoreList:
    print(score)
```

- This prints the value in `score`
- The statement is performed repeatedly by the loop (how do we know this?)
  - We know this because it is indented under the loop
  - If it was on the left edge of the screen it would not do what we want

Problem: Sorting the data

- We want the program to print out the scores in ascending order
- To do this we need to sort the collection of values
- This sounds like it might be hard work
  - But fortunately Python has this behaviour "built in"

Sorting numbers

```
scoreList.sort()
```

- This statement asks the list to sort itself
- After this statement completes the list contains the values in ascending order
- If we work through the list and print the values they come out with the smallest values first

Sorting numbers in reverse order

```
scoreList.sort(reverse=True)
```

- We can change how the sort works by feeding in a parameter called `reverse` which we set to `True`
  - A parameter is something that is fed into a method to tell it what to do
- The sort will now put the largest values first
- The default value for the `reverse` parameter is `true`

A DATA QUERY

Data Queries

- Now that we have the scores data in the program we can have some fun with it:
  - Find out how many people scored zero
  - Find out how many people scored more than fifty percent
  - Work out the average score
  - Work out the highest score

Problem: How Many Zeros?

- We want to know how many scores of zero were entered
- We need to work through the list looking for zero scores
- This is a similar kind of look to the one used to print out the values

Finding Zeroes

```
zeroCount = 0
for score in scoreList:
    if score == 0:
        zeroCount = zeroCount + 1

print("Number of people who scored zero: ", zeroCount)
```

- This loop will look through the list finding all the scores of zero
- Each time it finds a zero it adds one to the count

Problem: Counting scores > 50

- Now we want to count the number of scores that are greater than fifty
  - Will this be difficult?
  - What will we base the code on?
  - Is this easier if the list has been sorted?

Summary

- We can use lists to store large amounts of data
- We can use for loops to work through them and pull out each element in turn
- A list provides built-in behaviours (for example sorting)