Lists

Wrestling with Python
Using Lists

Lists

What we can do so far...

- Store data (using variables)
- Change data (using expressions)
- Make decisions (using conditions)
- Create loops (using while and for)
- There is not much more that we need to know how to do
  - But we do need to know how to create lists

Processing Cricket Scores

- Earlier we created a program to process cricket scores
- We read each score value in turn and used them to work out the highest and lowest scores, along with the total score and average

Variables

- We can score a single score in one value:

```
score = 10
```

- This will create a variable which can hold a single integer value
- The variable has the identifier **`score`**
- The variable holds the value **`10`**

University of Hull

Handling more scores

- If we want to store more data, the simplest approach is to create more variables:

```
score1 = 5
score2 = 10
score3 = 0
score4 = 30
```

University of Hull

Manipulating data

- However, this makes the data hard to work with:

```
if (score1 > score2 and
    score1 > score3 and
    score1 > score4 )
{
   print (score1)
}
```

Manipulating data

```
if(score1 > score2 and score1 > score3 and 
score1 > score4):
     print(score1)
elif(score2 > score1 and score2 > score3 and 
score2 > score4):
     print(score2)
elif(score3 > score1 and score3 > score2 and 
score3 > score4):
     print(score3)
else:
     print(score4)
```

A need for lists

- It is obvious that simple variables are no good when we are working with large amounts of data
- We need something that lets us store collections of values
- Python provides lists to do just this

Creating an empty list

- You can create an empty list

```
scoreList = []
```

- At the moment this list holds nothing at all
- The program can add items into the list and the list will keep track of them for us

Appending to a list

```
scoreList = []
scoreList.append(6)
scoreList.append(7)
```

- Once you have your list you can append things to it
- `scoreList` now holds two values
- We can append as many items as we like

Working through a list

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

- The for loop in Python is designed to work through a list of items
- We can give it a list and it will work through each item in turn

Reading and Printing a List

PRACTICAL BREAK 1

List elements

```
print (scoreList[0])
```

- You can access individual elements in a list by giving their *subscript* in brackets
- The above statement would print out the first element stored in the list
  - It would of course fail if the list was empty

Subscripts Etiquette

- Subscripts start at 0
- If you try to access an element which is not in the list (perhaps by using a subscript which is too large) your program will fail
- Subscripts should be checked as your program runs so that our programs never "fall off the end of a list"

Creating a list

```
scoreList = [5, 10, 0, 30]
```

- You can specify data in the list when you create

```
nameList = ["Rob", "Kevin", "Fred"]
```

- Lists can keep track of strings as well as numbers

UNIVERSITY OF Hull

Finding the length of a list

- You can find the length of a list like this:

```
>>> scoreList = [5, 10, 0, 30]
>>> len(scoreList)
4
```

Working with Lists

```
if scoreList[0] > scoreList [1]:
     temp=scoreList[0]
     scoreList[0] = scoreList[1]
     scoreList[1] = temp
```

- What does this do?

Working with Lists

```
if scoreList[0] > scoreList [1]:
     temp=scoreList[0]
     scoreList[0] = scoreList[1]
     scoreList[1] = temp
```

- It compares the top two elements in the list and swaps their order if they are the wrong way round
- This is the basis of sorting

Sorting a list

PRACTICAL BREAK 2

Summary

- Lists are the last thing that we need to know how to write every program in the world
- The allow us to store huge amounts of data and search and sort it
- The key to the power of a list is the use of variables as subscripts