COMP1021
Introduction to Computer Science

Lists, Tuples and Strings

David Rossiter and Gibson Lam

Outcomes

- After completing this presentation, you are expected to be able to:

  1. Create a collection of things using a list or a tuple in Python

  2. Manage a collection of things in a list

  3. Explain the big difference between lists and tuples

  4. Create a two dimensional structure using lists and tuples

 5. Explain the Python representation of strings

Storing a Collection of Data

- You have seen the use of variables to store simple data such as a number or a string (=a piece of text)

- Instead of a single piece of data it will be useful if you can store a collection of data in a variable inside a program

- For example, in a shooting game, if you can store a collection of monsters in one place, you will be able to manage them a lot easier than having them stored separately

```
monsters =
```

Lists

- Lists are a way to store a collection of items

| Item 0 | Item 1 | Item 2 | Item 3 | • • • |
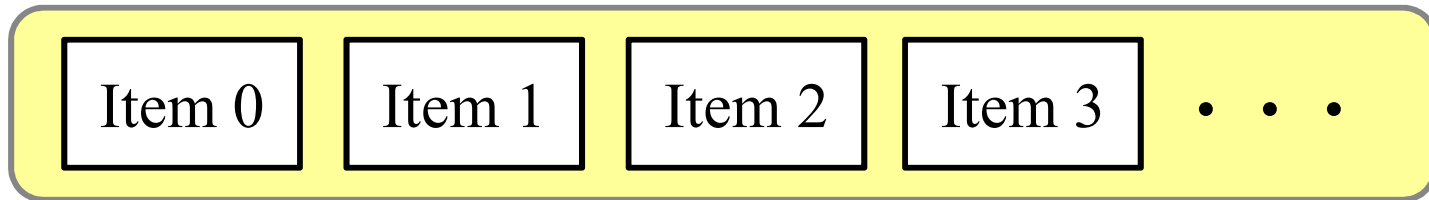
- They can be used to store many items together

- The items can be of any type

- For example, you can store a collection of numbers and text together in a list:

| "John" | 18 | "CSE" | 180 | • • • |

Using a List

- To create a list in Python you use a pair of square brackets to enclose the items you need, for example:

```
girlfriends = ["Gwen", "Lois", "Mary"]
```

- To check a particular item in the list, you need to say which item you want by using an *index*

- For example, to print the first item from the above list, you can use this code:

```
print(girlfriends[0])
```

Important! The first item has an index of 0, not 1!

0   *1*   *2*

```
>>> girlfriends = ["Gwen", "Lois", "Mary"]
>>> print(girlfriends[0])
Gwen
```

The Length of a List

- You can use `len( name of a list )` to tell you how many things are in the list

- For example:

```
>>> girlfriends = ["Gwen", "Lois", "Mary"]
>>> print(len(girlfriends))
3
```

Going Through Everything in a List

- There's 2 ways to go through everything in a list
- The first way is shown here:

```
>>> girlfriends = ["Gwen", "Lois", "Mary"]
>>> for thisgirlfriend in girlfriends:
        print(thisgirlfriend)

Gwen
Lois
Mary
```

Going Through Everything in a List

- The second way is by using *range* to generate the the index numbers, then using the index numbers:

```
>>> girlfriends = ["Gwen", "Lois", "Mary"]
>>> for index in range(len(girlfriends)):
        print(girlfriends[index])

Gwen
Lois
Mary
```
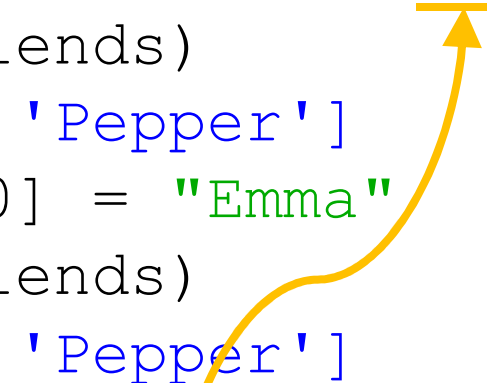
- The first way is simpler than the second way, but sometimes you need to use the second way

Changing an Item in a List

- You can change any item in a list, for example:

```
>>> girlfriends = ["Gwen", "Lois", "Mary"]
>>> girlfriends[2] = "Pepper"
>>> print(girlfriends)
['Gwen', 'Lois', 'Pepper']
>>> girlfriends[0] = "Emma"
>>> print(girlfriends)
['Emma', 'Lois', 'Pepper']
```

You can use either " ... " or ' ... '
to enclose a piece of text – see later

Insert, Remove and Append a List

- You can insert/remove/append items at any time, for example: ← *'append' means 'put it at the end'*

```
>>> comment = ["Dave", "is", "human"]
>>> print(comment)
['Dave', 'is', 'human']
>>> comment.insert(2, "not")
>>> print(comment)
['Dave', 'is', 'not', 'human']
>>> comment.remove("human")
>>> print(comment)
['Dave', 'is', 'not']
>>> comment.append("evil!")
>>> print(comment)
['Dave', 'is', 'not', 'evil!']
```

Insert before item 2 (the third item)

If there was more than one 'human' in the list, this would only remove the first one

Other Things You Can Do with Lists

- sort – sorts the list in increasing number/letter order

```
>>> words = ["cat", "dog", "apple", "bat"]
>>> words.sort()
>>> print(words)
['apple', 'bat', 'cat', 'dog']
```

- reverse – reverses the content of the list

```
>>> words = ["cat", "dog", "apple", "bat"]
>>> words.reverse()
>>> print(words)
['bat', 'apple', 'dog', 'cat']
```

- count – counts how many times something appears

```
>>> ages = [20,21,19,20,19,22,20,20,20,18]
>>> ages.count(20)
5
```

Other Things You Can Do with Lists

- index – finds the first occurrence of something

```
>>> ages = [20,21,19,20,19,22,20,20,20,18]
>>> ages.index(19)
2
```

- extend – appends another list to this list

```
>>> happywords = ["I", "love", "you"]
>>> print(happywords)
['I', 'love', 'you']
>>> happywords.extend(["if","you","lend","me","money"])
>>> print(happywords)
['I', 'love', 'you', 'if', 'you', 'lend', 'me', 'money']
```

- There are other things you can do with lists, we may look at those later in the course if we have time

Two Dimensional Structures

- In some cases a one dimensional (1D) structure (things that are arranged in one direction) is not enough

- For example, a digital camera image is a 2D structure

  - You need both the X location and Y location of a point to read its colour in an image

Vertical direction (Y)

Horizontal direction (X)
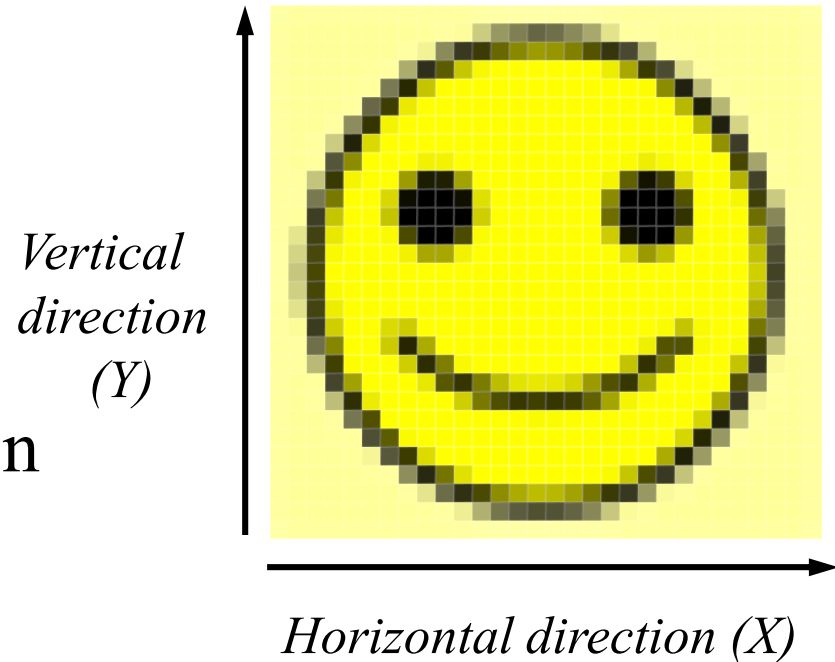
Two Dimensional Structures

- A Python list is a 1D data structure

- What if you want to use a 2D structure?

  – Then you need to use lots of lists inside another list

- We call this a 'list of lists'

  – The outer list is one of the dimensions, and the inner lists are the other dimension

Outer list

Inner list 1

[…]

Inner list 2

[…]

Inner list 3

[…]
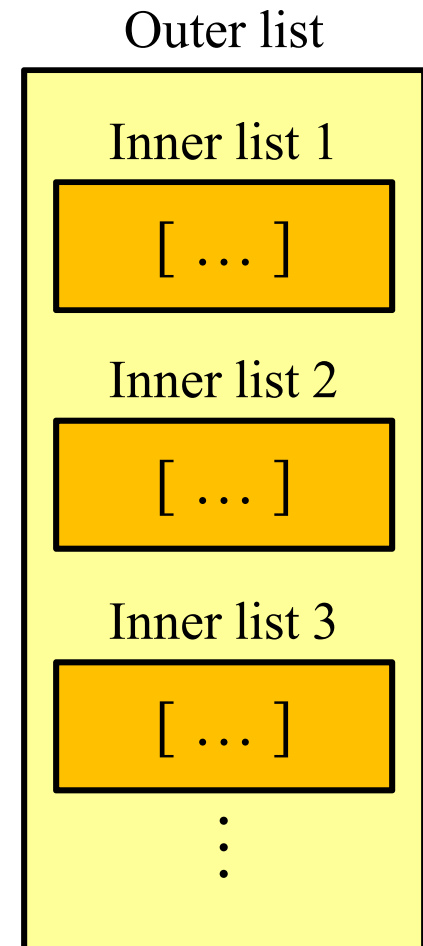
Examples of a 1D List and a 2D List

- 1D example:

```
things = ["j","o","k","e","s"]
letter_o = things[1] # the "o" in second col
```

things

j	o	k	e	s

- 2D example:

```
things =
  [["h","a","r","r","y"],
   ["l","i","k","e","s"],
   ["j","o","k","e","s"]]
letter_o = things[2][1]
# the "o" in the third row second column
```

things[0]

h	a	r	r	y

things[1]

l	i	k	e	s

things[2]

j	o	k	e	s

Be careful! The general idea is `[row][col]`, not `[col][row]`!

What About Tuples?

- Basically, a tuple is the same idea as a list, **but** you can't change anything in a tuple after it is created

- To create a tuple you use a pair of parentheses (not square brackets) for the items, like this:

```
myfriend = ("John", 18, "CSE", 180, 70)
```

- To get a particular item from a tuple it's the same as a list, for example:

```
print( "Name of my friend:", myfriend[0] )
```

```
>>> myfriend = ("John", 18, "CSE", 180, 70)
>>> print("Name of my friend:", myfriend[0])
Name of my friend: John
>>>
```

You still use [] *when getting an item from a tuple*

Trying to Change an Item in a Tuple

- If you try and change something inside a tuple, the program crashes:

```
>>> myfriend = ("John", 18, "CSE", 180, 70)
>>> myfriend[2] = "ECE"
Traceback (most recent call last):
  File "<pyshell#16>", line 1, in <module>
    myfriend[2] = "ECE"
TypeError: 'tuple' object does not support item assignment
>>> print(myfriend)
('John', 18, 'CSE', 180, 70)
>>>
```

- Using the language of computing, we say lists are *mutable* (can be changed) and tuples are *immutable* (cannot be changed)

What Works With Tuples?

- `len()`, `count()` and `index()` work because they don't try to change the tuple content

- Creating a 2D structure in a tuple is fine e.g.

```
things = (("h","a","r","r","y"),
          ("l","i","k","e","s"),
          ("j","o","k","e","s"))
```

- `insert()`, `remove()`, `append()`, `sort()`, `reverse()`, `extend()` all **don't work** because they try to change the tuple content

When to Use Tuples

- You may use only lists in your programming because they are 'more powerful' than tuples

- However, there are situations where it's better to use tuples e.g.:

  – You want to ensure some data can't be changed

  – There are some not common situations where a list doesn't work but a tuple does, e.g. sometimes in a dictionary (this may be discussed later in another presentation)

A String is a Tuple of Letters

- In the world of computing, a *string* is the name for text

- Python thinks of a string as a tuple of letters

- For example:

```
>>> mytext="Hello"
>>> print(mytext[1])
e
```

- Just like a tuple, you can't directly change the content of a string:

```
>>> mytext[1]="a"
Traceback (most recent call last):
  File "<pyshell#18>", line 1, in <module>
    mytext[1]="a"
TypeError: 'str' object does not support item assignment
```

Using Quotes " or '

- As we have mentioned before, you can use either
  " (double quotes) or ' (single quotes)
  to enclose the content of a string

- For example:

  `'Hello!'` is the same as `"Hello!"`

- However, you cannot mix them up like this:

```
>>> greeting = "Hello!'
SyntaxError: EOL while scanning string literal
```

- The error means Python does not understand the
  string, because it starts with a " and ends with a '
  which confuses Python

What Works With Strings?

- Just like a tuple, with a string you can use `len()`, `count()` and `index()`

```
>>> mytext="Hello"
>>> len(mytext)
5
>>> mytext.count("l")
2
>>> mytext.index("o")
4
```

- Just like a tuple, with a string `insert()`, `remove()`, `append()`, `sort()`, `reverse()`, `extend()` all **don't work** because they try to change the string content