COMP1021
Introduction to Computer Science

Sequences
– Lists, Tuples and Strings

David Rossiter

Outcomes

- After completing this presentation, you are expected to be able to:
  1. Create a sequence of things using a list/ tuple/ string
  2. Understand and use index numbers
  3. Access things stored in a list/ tuple/ string
  4. Add to and change things stored in a list

Storing a Sequence of Things

- Often you need to store a sequence of things in a variable inside a program
- For example, in a shooting game, if you can store a sequence of monsters in one place, you will be able to manage them a lot easier than having them stored separately

`monsters =` 

- In this presentation we are talking about 3 ways to use a *sequence* of things

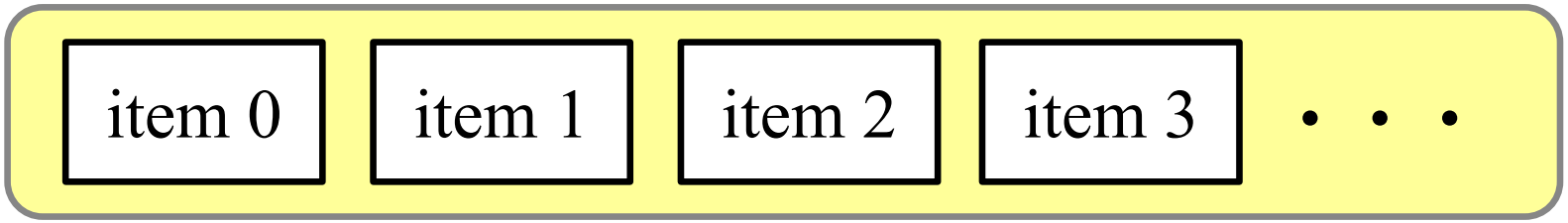
Three Types of Sequence

- In Python there are 3 basic ways to store a sequence:
  - Use a list
  - Use a tuple
  - Use a string (this is a sequence of letters)

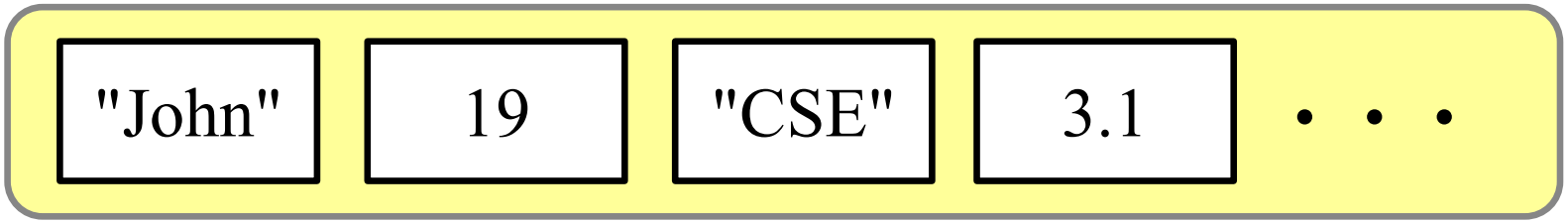
- A *string* is the computer name for text

A List

- A list is the most common way to store a sequence



- A list can store many items together
- The items can be almost anything
- For example, you can store a collection of numbers and text in a list:



Using a List

- To create a list in Python you use a pair of square brackets `[ ]`, for example:

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

```
                  0          1          2
friends = ["Chan", "May", "Peter"]
```

index numbers

- To access an item you can use the index number

```
print( friends[0] )   ⇒   Chan
print( friends[1] )   ⇒   May
print( friends[2] )   ⇒   Peter
```

Another Example

```
                 0          1        2        3
info = ["John", 19, "CSE", 3.1]
```

index numbers

```
print( info[0] )   ⇒   John
print( info[1] )   ⇒   19
print( info[2] )   ⇒   CSE
print( info[3] )   ⇒   3.1
```

The Length of a Sequence

- `len(` *name of the sequence* `)` will tell you how many things are in the sequence
- For example:

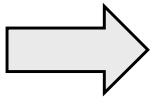
```
mylist = [ "cat" ]
print( len(mylist) )          ⇨   1

mylist = [ 48, 60, 65, 68 ]
print( len(mylist) )          ⇨   4

mylist = []
print( len(mylist) )          ⇨   0
```

- `[]` is called an empty list

Going Through a Sequence 1

- There's two ways to go through everything in a sequence, the first way is demonstrated here:

```
all_ages = [20, 19, 22, 24, 21, 20]

for this_age in all_ages:
    print(this_age)
```



```
20
19
22
24
21
20
```

Going Through a Sequence 2

- The second way is by using `range()` to generate the index numbers, then you use the index numbers:

```
all_ages = [20, 19, 22, 24, 21, 20]

for this_index in range(len(all_ages)):
    print(all_ages[this_index])
```

```
20
19
22
24
21
20
```

- The first way (last slide) is simpler than this way, but sometimes you need to use the indicies

Changing an Item in a List

- You can use the index number to change something in a list, for example:

```
all_ages = [20, 19, 22, 24, 21, 20]

print(all_ages)  ⇨  [20, 19, 22, 24, 21, 20]

all_ages[1] = 21

print(all_ages)  ⇨  [20, 21, 22, 24, 21, 20]
```

- The second item has changed

Some Commands to Change a List

- `insert()` – put something into the list at a particular position
- `remove()` – remove something from the list
- `append()` – put something at the end of the list

Insert, Remove and Append a List

```
words = ["an", "apple", "is", "tasty"]

words.insert(3, "not")
      ⇨ ["an", "apple", "is", "not", "tasty"]
words.remove("apple")
                ⇨ ["an", "is", "not", "tasty"]
words.insert(1, "ant")
         ⇨ ["an", "ant", "is", "not", "tasty"]
words.append("probably")
         ⇨
 ["an", "ant", "is", "not", "tasty", "probably"]
```

Other Things You Can Do with Lists

`reverse()` – reverses the content of the list

```
words = ["mouse", "goes", "squeak"]

print( words ) ⇨ ["mouse", "goes", "squeak"]

words.reverse()

print( words ) ⇨ ["squeak", "goes", "mouse"]
```

Other Things You Can Do with Lists

`sort()` – sort the list in increasing number/letter order

```
all_ages = [20, 19, 22, 24, 21, 20]
all_ages.sort()
print(all_ages)    ⇨  [19, 20, 20, 21, 22, 24]
all_names = ["cat", "ant", "bat"]
all_names.sort()
print(all_names)   ⇨  ["ant", "bat", "cat"]
```

Other Things You Can Do with Lists

`count()` – count how many times something is in the sequence

```
results = ["pass", "pass", "fail", "pass"]

print( results.count("pass") )   ⇨   3

print( results.count("fail") )   ⇨   1

print( results.count("incomplete") )   ⇨   0
```

Other Things You Can Do with Lists

`index()` – gives you the index number of the first occurrence of something

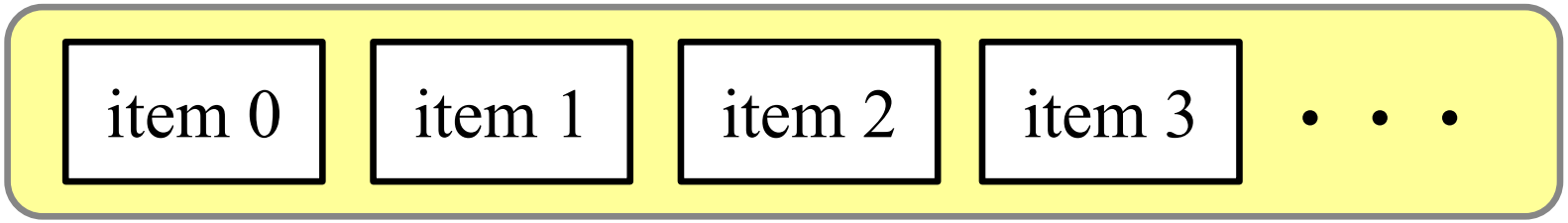
```
results = ["C-", "B", "B+", "C+", "B+"]

print( results.index("B+") )    ⇨    2

print( results.index("B") )     ⇨    1

print( results.index("D") )
```

⇨

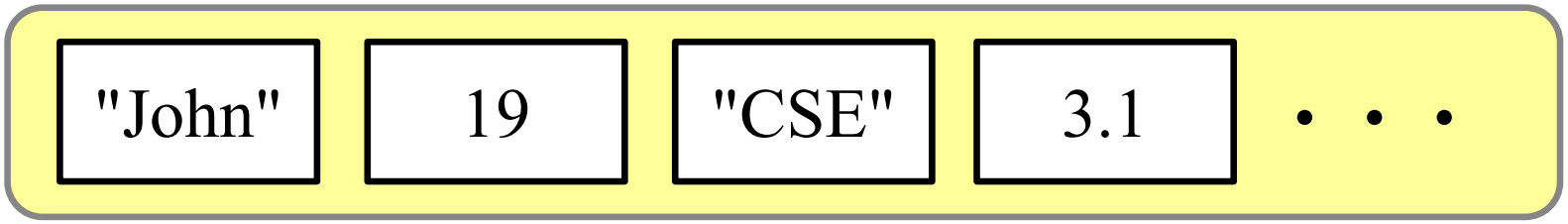
```
Traceback (most recent call last):
  File "<pyshell#19>", line 1, in <module>
    results.index( "D" )
ValueError: 'D' is not in list
```

A Tuple

- A tuple is another way to store a sequence of items



- A tuple can store many items together
- The items can be almost anything
- For example, you can store a collection of numbers and text in a tuple:



- A tuple is similar to a list
- However, after a tuple is created **you cannot change anything** in the tuple

Using a Tuple

- To create a tuple in Python you use a pair of parentheses ( ) e.g:

In a sequence, the first item has an index of 0

0 *1* *2* ***index numbers***

```
friends = ("Chan", "May", "Peter")
```

- To access an item you can use the index number
- For example:

You still use `[]` *when using an index number for a tuple*

```
print( friends[0] )   ⇒   Chan
print( friends[1] )   ⇒   May
print( friends[2] )   ⇒   Peter
```

Another Example

```
                 0          1        2        3
info = ("John", 19, "CSE", 3.1)
```

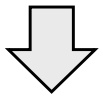
index numbers

```
print( info[0] )   ⇒   John
print( info[1] )   ⇒   19
print( info[2] )   ⇒   CSE
print( info[3] )   ⇒   3.1
```

Trying to Change Something in a Tuple

- If you try to change something inside a tuple, Python will crash:

```
info = ("John", 19, "CSE", 3.1)
info[3] = 3.2
```



```
Traceback (most recent call last):
  File "<pyshell#24>", line 1, in <module>
    info[3]=3.2
TypeError: 'tuple' object does not support item assignment
```

Going Through Everything in a Tuple

- For a tuple, you can go through everything using the same techniques that you use for lists

```
all_ages = (20, 19, 22, 24, 21, 20)

for this_age in all_ages:
    print(this_age)


for this_index in range(len(all_ages)):
    print(all_ages[this_index])
```

- Both loops produce the same result:

```
20
19
22
24
21
20
```

What Works With a Tuple?

- For a tuple, commands such as `len()`, `count()` and `index()` work the same way that they work for lists
- Those commands don't try to change the tuple content

```
data = ("circle", 10, 10, 300)
print( len(data) )          ⇒  4
print( data.count(10) )     ⇒  2
print( data.index(300) )    ⇒  3
```

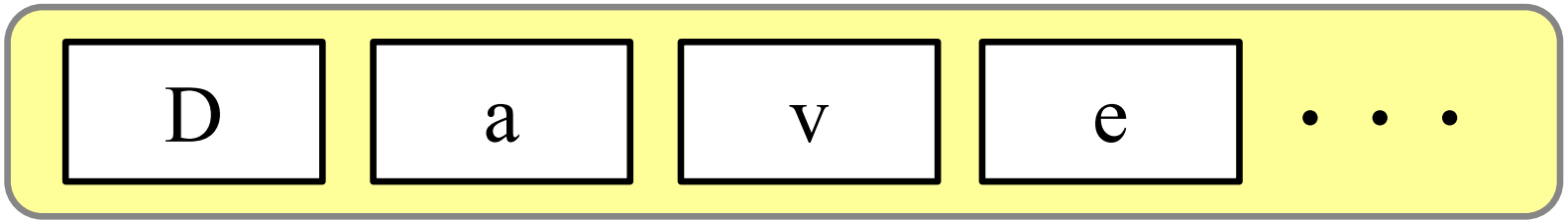
- For a tuple, commands such as `insert()`, `remove()`, `append()`, `sort()`, `reverse()` all **don't work** because they try to change the tuple content

Why Use a Tuple?

- Often, you would choose to use a list rather than a tuple because a list is ‘more powerful’ than a tuple
- However, there are situations where it’s better to use a tuple than a list e.g.:
  - You would prefer that the program crashes instead of letting some special data be changed
  - There are some situations where a list doesn’t work but a tuple does, e.g. sometimes when you use a *dictionary* (to be discussed in another presentation)

A String

- A string is a way to store text



- Basically, a string can contain all the things you can type on your computer keyboard, such as letters and digits and special characters
- A string is only for storing text
- After a string is created the string cannot be changed

Using a String

- To create a string you use a pair of speech marks e.g:

  0 1 2 3] ***index numbers***

```
my_friend = "Dave"
```

  or:

```
my_friend = 'Dave'
```

- To access one thing in the string you can use the index number, for example:

```
print( my_friend[0] )   ⇒   D
print( my_friend[1] )   ⇒   a
print( my_friend[2] )   ⇒   v
```

Trying to Change Something in a String

- If you try to change something inside a string, Python will crash:

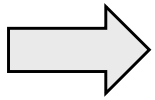
- *A space counts as text*

```
>>> my_city = "Hong Kong"
>>> my_city[6]="i"
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    my_city[6]="i"
TypeError: 'str' object does not support item assignment
```

The Length of a String

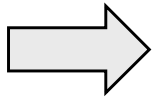
- `len(` *name of the string* `)` will tell you how many letters are in the string
- For example:

```
mytext = "lion"
print( len(mytext) )
```



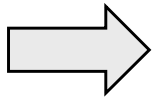
4

```
mytext = "apple"
print( len(mytext) )
```



5

```
mytext = ""
print( len(mytext) )
```



0

- `""` is called an empty string

Going Through Everything in a String

- For a string, you can go through everything using the same techniques that you use for lists/ tuples

```
message = "Hello!"
```

- *Special characters are also text*

```
for this_letter in message:
    print(this_letter)
```

```
for this_index in range(len(message)):
    print(message[this_index])
```

- Both loops produce the same result:

```
H
e
l
l
o
!
```

What Works With a String?

- For a string, commands such as `len()`, `count()` and `index()` work the same way as for lists/ tuples
- Those commands don't try to change the string content

```
mytext = "Hello!"
print( len(mytext) )          ⇒  6
print( mytext.count("l") )    ⇒  2
print( mytext.index("!") )    ⇒  5
```

- For a string, commands such as `insert()`, `remove()`, `append()`, `sort()`, `reverse()` all **don't work** because they try to change the string content

Quick Summary

- A list

  0 1 2] *index numbers*

  `friends = ["Chan", "Peter", "Mary"]`

  - A list can contain almost anything
  - After a list is created, it can be changed

- A tuple

  0 1 2] *index numbers*

  `friends = ("Chan", "Peter", "Mary")`

  - A tuple can contain almost anything
  - After a tuple is created, it cannot be changed

- A string

  0 1 2 3] *index numbers*

  `my_friend = "Dave"`

  - A string only contains text
  - After a string is created, it cannot be changed

Changing a Sequence

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