COMP1021
Introduction to Computer Science

More on Sequences

David Rossiter

Outcomes

- After completing this presentation, you are expected to be able to:
  1. Use + for a sequence
  2. Use * for a sequence
  3. Use slicing to access or change data
  4. Create a 2D or 3D structure
  5. Use negative indices for a sequence

Reminder – 3 Types of Sequence

- A list

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

index numbers

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

- A tuple

```
              0         1         2
friends = ("Chan", "Peter", "Mary")
```

index numbers

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

- A string

```
                 0123
my_friend = "Dave"
```

index numbers

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

Reminder - Handling of a Sequence

- Read something in the sequence:
  list_name `[` *item_number* `]`

- Count something in the sequence:
  list_name`.count(` *thing_you_want_to_count* `)`

- Find the index of something in the sequence:
  list_name`.index(` *thing_you_are_searching_for* `)`

- The above don't change the data
- So they work for tuples and strings, as well as lists

Reminder - For Lists

- Changing a value in the list:
  list_name `[` *item_number* `]` = *new_thing*

- Inserting a value into the list:
  list_name`.insert(` *index_number*, *new_thing* `)`

- Removing something from the list:
  list_name`.remove(` *thing_you_want_to_remove* `)`

- Put something new at the end:
  list_name`.append(` *thing_you_want_to_append* `)`

- Sorting the list:
  list_name`.sort()`

- Reversing the order of the things in the list:
  list_name`.reverse()`

Using +

- You can use + if both the left side and right side are the same type of data

- Using + for lists

```
old_friends = ["Chan", "Mary"]
new_friends = ["May", "Wong"]
friends = old_friends + new_friends
print(friends) ➡ ["Chan", "Mary", "May", "Wong"]
```

- Using + for tuples

```
previous = ("B-", "C+")
new_grades = ("B+", "A-")
grades = previous + new_grades
print(grades)  ➡ ("B-", "C+", "B+", "A-")
```

- Using + for strings

```
surname = "Rossiter"
other_names = "David"
name = surname + ", " + other_names
print(name)    ➡  Rossiter, David
```

- The computer word for sticking things together is *concatenate*

Using *

- `thing * n` will concatenate *n* copies of `thing`

- Using * for lists

```
steps = ["left", "right"]
all = steps * 2
print(all) → ["left", "right", "left", "right"]
```

- Using * for tuples

```
stages = ("n", "e")
all = stages * 3
print(all) → ("n", "e", "n", "e", "n", "e")
```

- Using * for strings

```
mytext = "fun"
result = mytext * 4
print(result)   →   funfunfunfun
```

Slicing

- Slicing will be discussed in more detail in another presentation. This is the basic idea:

```
mydata[ start_index : target_index ]
```

gives you access to part of a sequence

```
mydata[ start_index : target_index : step_number ]
```

gives you access to part of a sequence, using a step number

When you do slicing the target index is not included in the result

- Examples of these are on the next slide

Slicing to Access Data

- Slicing for lists

```
info = [21, 19, 18, 25, 20, 26]
print(info[1:3])        ➡  [19, 18]
print(info[0:4:2])      ➡  [21, 18]
```

- Slicing for tuples

```
data = (11, 10, 12, 14, 13, 16, 15)
print(data[2:5])        ➡  (12, 14, 13)
print(data[1:4:2])      ➡  (10, 14)
```

When you do slicing the thing at the position of the target index is not included in the result

- Slicing for strings

```
word = "chairs"
print(word[1:5])        ➡   "hair"
print(word[0:5:2])      ➡   "car"
```

- Slicing is discussed more in another presentation

Slicing to Change Data

- You can use slicing to change data

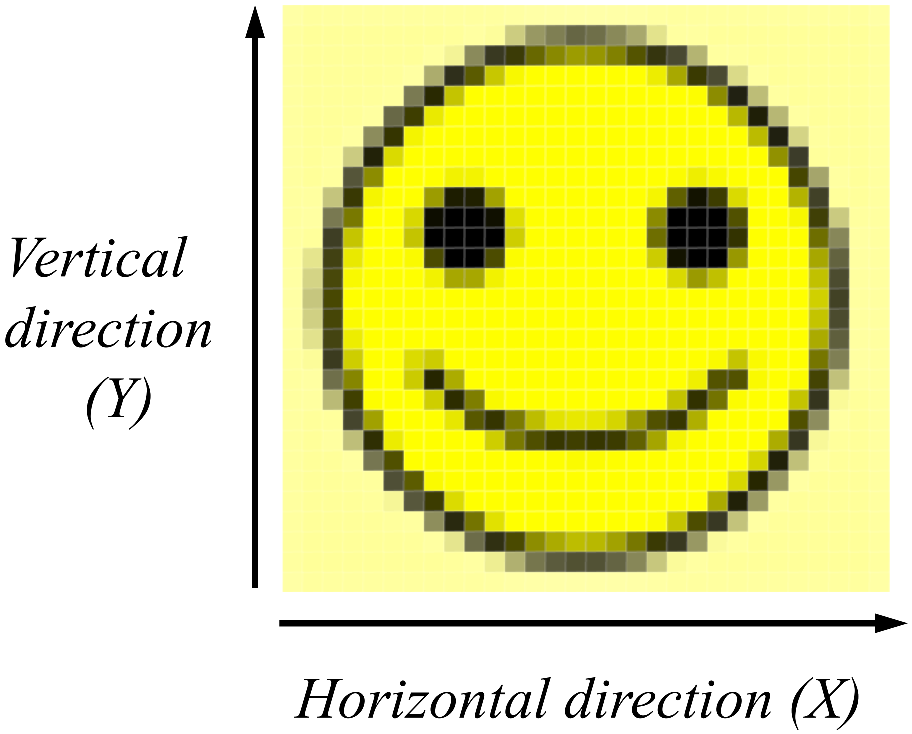
```
info = [21, 19, 18, 21, 20, 19]

info[1:3] = [25, 27]

print(info) ➡ [21, 25, 27, 21, 20, 19]
```

- You can't change tuples or strings, so this technique won't work for them

Two Dimensional Structures

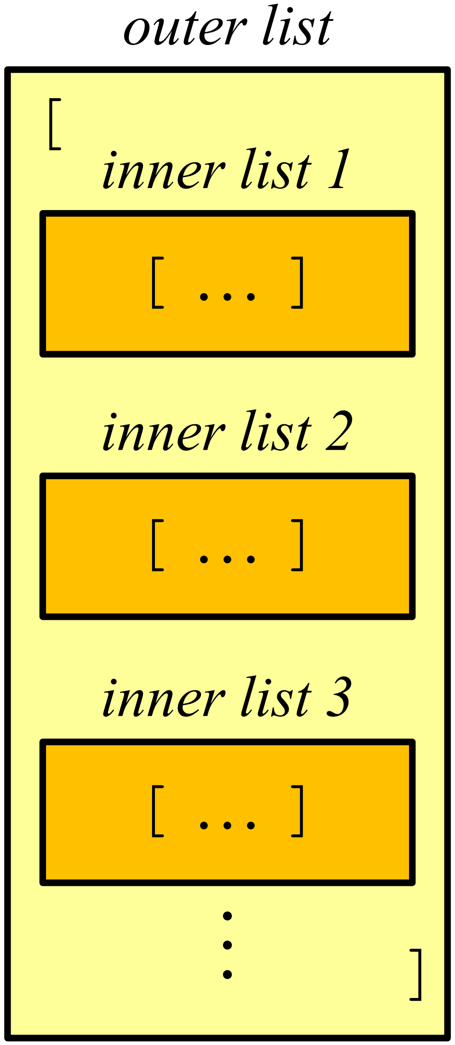
- Sometimes a one dimensional (1D) structure (things that are arranged in one direction) is not enough

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



Two Dimensional (2D) 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
- This is called a ‘list of lists’
  - The outer list is one of the dimensions, and the inner lists are the other dimension



A 2D List

- You can put almost anything in a list, so you can make a 2D structure in a list e.g.

```
things = [[20, 20, 19, 18, 22],
          [18, 19, 20, 18, 17],
          [21, 22, 24, 22, 25]]
```

things[0]

20	20	19	18	22

things[1]

18	19	20	18	17

things[2]

21	22	24	22	25

`print(things[2][1])` → `22`

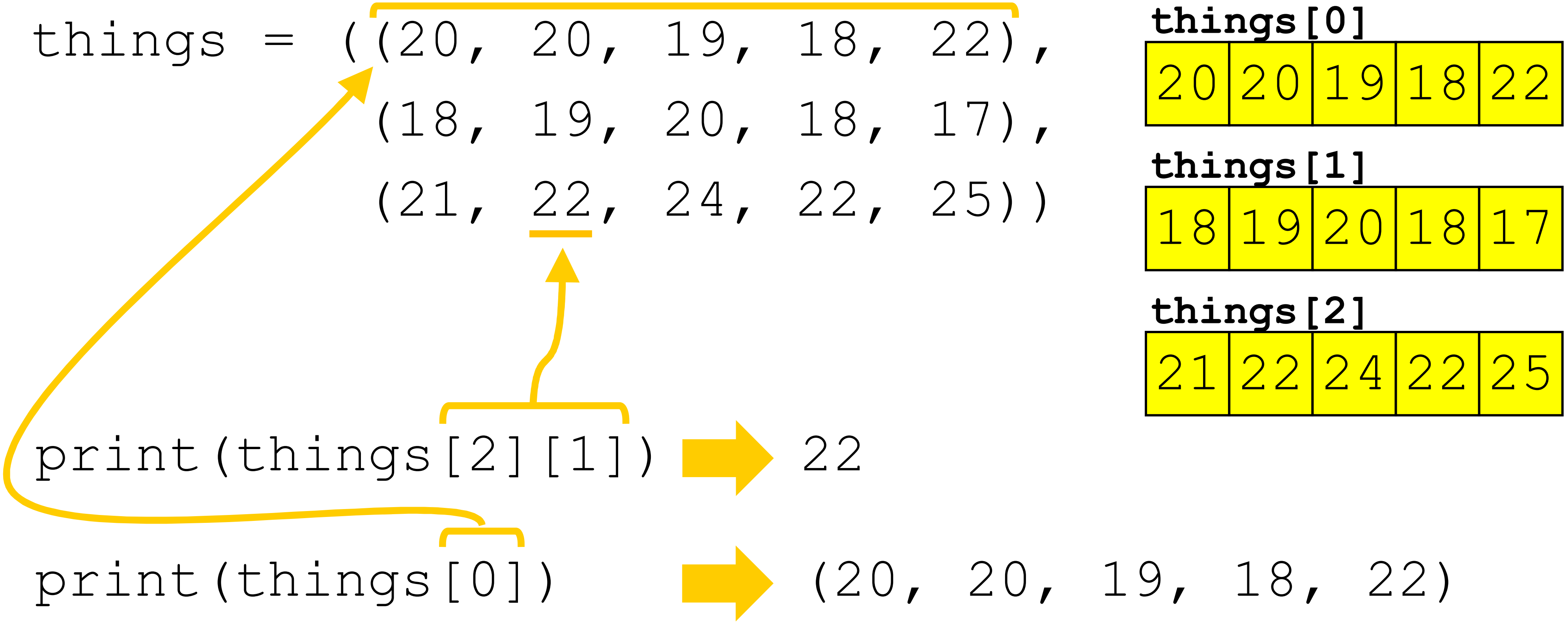
Be careful! The general idea is `[`*row*`][`*col*`]`, not `[`*col*`][`*row*`]` !

So it's more similar to `[` *y* `][` *x* `]` than `[` *x* `][` *y* `]`

`print(things[1])` → `[18, 19, 20, 18, 17]`

A 2D Tuple

- Like a list, you can put almost anything in a tuple, so you can make a 2D structure in a tuple e.g.



```
things = ((20, 20, 19, 18, 22),
          (18, 19, 20, 18, 17),
          (21, 22, 24, 22, 25))



print(things[2][1])  ➡  22

print(things[0])     ➡  (20, 20, 19, 18, 22)
```

things[0]

20	20	19	18	22

things[1]

18	19	20	18	17

things[2]

21	22	24	22	25

The Length of a 2D Sequence

```
things = [ [2004, 2003, 2006, 2005],
           [2001, 2000, 2004, 2006] ]

print(len(things))  ➡  2

things = [ [16, 12],
           [21, 22, 19, 24],
           [28],
           [5, 7, 6, 8] ]

print(len(things))  ➡  4
```

- `len()` doesn't count inside the lists which are inside the list

A 3D Example

- You could make a 3D structure e.g.

```
things =  [ [ [1, 2],   [3, 4]    ],
            [ [5, 6],   [7, 8]    ],
            [ [9, 10],  [11, 12] ] ]

print( things[1][0][1] )  ➡  6
```

Negative List Indices

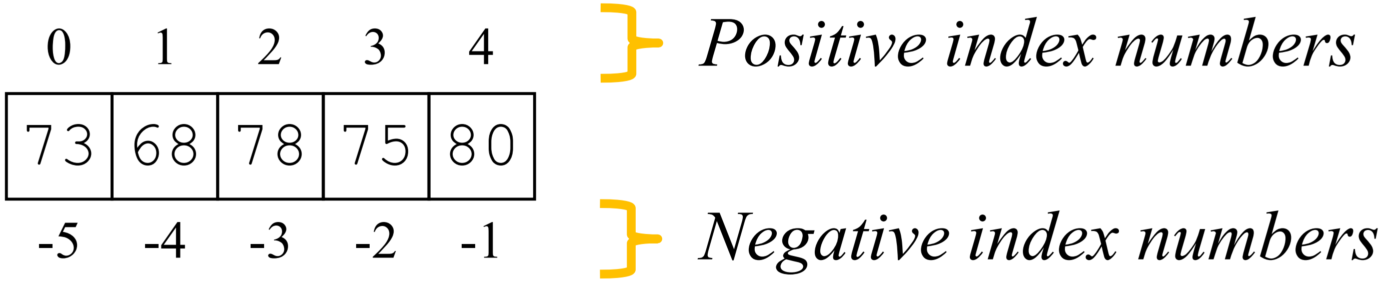
- In Python, you can use a **negative** index number

  list_name`[-1]` means the last one

  list_name`[-2]` means the next to last one

  and so on

```
x = [ 73, 68, 78, 75, 80 ]
```



0	1	2	3	4
73	68	78	75	80
-5	-4	-3	-2	-1

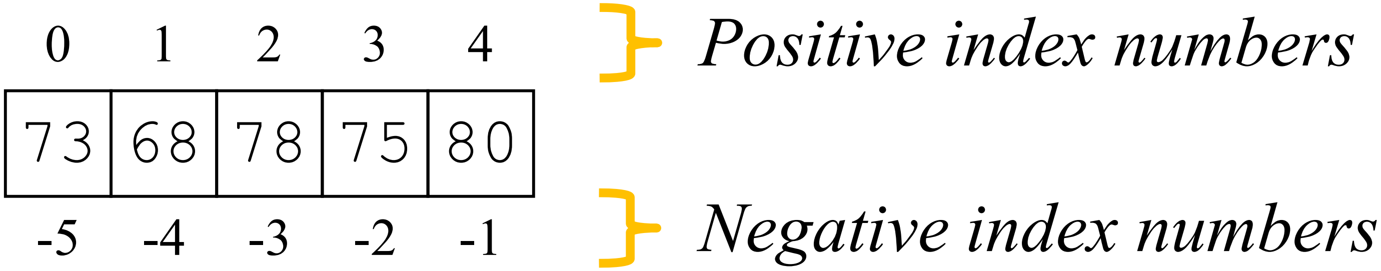
Positive index numbers

Negative index numbers

- In the above example `x[0]` and `x[-5]` refer to the same thing (73)

Negative List Indices

- You can use negative index numbers for all three types of sequence we have looked at
- Lists `x=[ 73, 68, 78, 75, 80 ]`



0	1	2	3	4	*Positive index numbers*
73	68	78	75	80	
-5	-4	-3	-2	-1	*Negative index numbers*

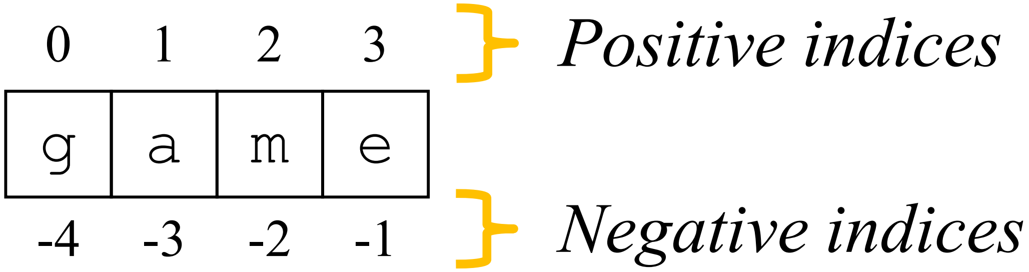
```
print( x[-1] ) ➡ 80
```

- Tuples `x=( 73, 68, 78, 75, 80 )`

```
print( x[-1] ) ➡ 80
```

- Strings `x="game"`

```
print( x[-1] ) ➡ e
```



0	1	2	3	*Positive indices*
g	a	m	e	
-4	-3	-2	-1	*Negative indices*