COMP1021
Introduction to Computer Science

More on
Lists and Strings

David Rossiter

Outcomes

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

    1. Manipulate a string as a collection of characters

    2. Use *len* to know how many items are in a list or a string

    3. Use negative indicies for a list or string

    4. Apply slicing techniques to a string

    5. Use the + operator and the * operator on strings and lists

A String in Python

- A *string* is the computer word for a piece of text

- In Python, a string can be thought of as a collection of letters/digits/symbols, which we generally call *characters*

- For example, the string 'funny' in Python is a collection of five characters: 'f', 'u', 'n', 'n', 'y'

- As mentioned before,
  Python treats "..." exactly the same as '...'

Simple Handling of Strings

- So you can write a string using either single or double quotes:

  `'funny'` or `"funny"` are both OK

- A string can be stored in a variable, e.g.

  `myword = "funny"`

- You can add strings together to produce a new string e.g:

  `two_words = "pretty" + " umbrellas"`

- The word for sticking text together is *concatenate*

- Python thinks of a string as a list of letters, so some of the techniques for handling lists also work for strings

Reminder of Handling Lists

- Creating a list:
  list_name = [*first_thing*, *second_thing*, …]

- Reading a value from the list:
  list_name [*item_number*]

  Works for strings as well

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

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

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

- Adding 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()

- Counting something in the list:
  list_name.count(*thing_you_want_to_count*)

- Searching for something in the list:
  list_name.index(*thing_you_are_searching_for*)

- Adding another list at the end of the list:
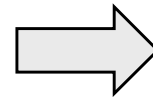 *list_name*.extend(*another_list*)

Works for strings as well

General Methods We Know which Can be Used to Handle a List

- Go through the list in a `while` loop
- Go through the list in a `for` loop
  - Usually with `range`
- Using slicing to copy values from a list

- These 3 methods can be applied to strings also (as long as you don't try to change the string)
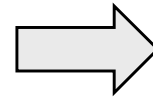
How to Know the Length of a List

- `len(` *name_of_the_list* `)`
  - tells you how many things are in the list

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

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

mylist = []
print( len(mylist) )                    ⟹        0

mylist = [ ["Dave", 3554], ["Gibson", 3553] ]
print( len(mylist) )                    ⟹        2
```
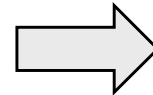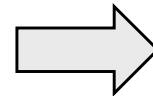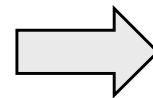
- It's the same idea for strings:

```
mytext = "warm"
print( len(mytext) )                    ⟹        4
```

Negative List Indicies

- You can use a negative number to refer to items

$$x=[\; 73, \; 68, \; 78, \; 75, \; 80 \;]$$

```
     0    1    2    3    4        Index
   ┌────┬────┬────┬────┬────┐     numbers
   │ 73 │ 68 │ 78 │ 75 │ 80 │
   └────┴────┴────┴────┴────┘
    -5   -4   -3   -2   -1        Index
                                  numbers
```

- *list_name*[-1] means the last one (here, it is 80)
- *list_name*[-2] means the second from last one (here, it is 75)
- In this example x[0] and x[-5] are both 73

Individual Items in a String

- Handling a string is like handling a list

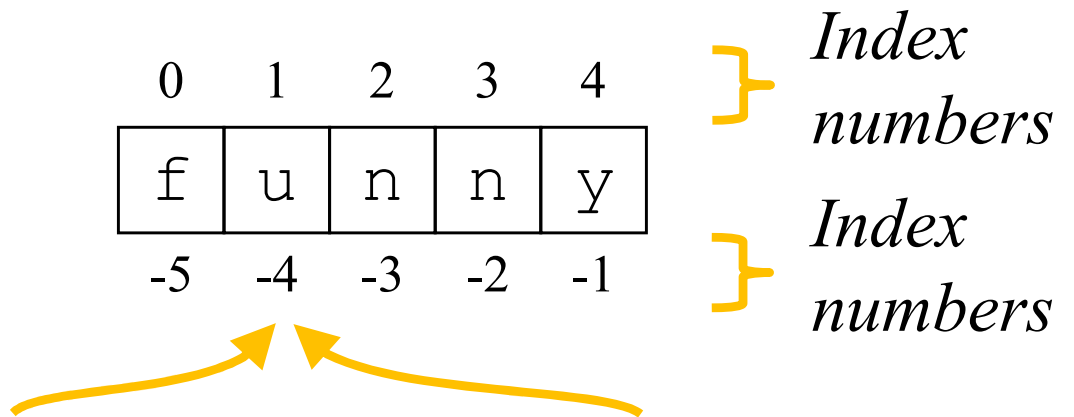- For example, the string 'funny' has indexing like this:

f	u	n	n	y
0	1	2	3	4

} *Index numbers for the characters*

- If `myword` is the variable storing the above string, `print(myword[1])` will produce 'u'

Using a Negative Index

- Like a list, you can use negative indices for strings

```
   0   1   2   3   4
 ┌───┬───┬───┬───┬───┐
 │ f │ u │ n │ n │ y │
 └───┴───┴───┴───┴───┘
  -5  -4  -3  -2  -1
```

Index numbers

Index numbers

- For example, `myword[1]` and `myword[-4]` refer to the same character 'u', in this example

Slicing for Strings

f	u	n	n	y
0	1	2	3	4

- You can do slicing for strings, just like lists

- Some examples:

  - Printing the second and third characters:

    ```
    print(myword[ 1 : 3 ]) # This is "un"
    ```

  - Printing all the characters except the last one:

    ```
    print(myword[ 0 : 4]) # This is "funn"
    ```

    Remember when you do slicing the target number is not included in the result

Omitting Numbers with Slicing

f	u	n	n	y
0	1	2	3	4

- As you know, you can omit numbers with slicing

- Examples:

  – From the start of the string:

    ```
    print(myword[ : 3 ])   # This is "fun"
    ```

  – To the end of the string:

    ```
    print(myword[ 3 : ])   # This is "ny"
    ```

Slicing Using a Step Value

- Like lists, you can include a step value:

  name_of_string[*start* : *target* : *step*]

- As you know, a step value means you do not move one step at a time but move *step* characters at a time

Slicing Using a Step Value

f	u	n	n	y
0	1	2	3	4

- The following examples use this:

  - To print the second and the fifth characters:

    ```
    print(myword[1:5:3]) # outputs "uy"
    ```

  - To print alternate characters:

    ```
    print(myword[::2]) # outputs "fny"
    ```

  - To print the string in reverse order:

    ```
    print(myword[::-1])# outputs "ynnuf"
    ```

You Can't Change a String

- You can 'read' characters in a string using the techniques we have just looked at

- You can't *change* the content of a string after it's created

- For example:

```
lunch = "I love to eat a pineapple bun!"

lunch[-2] = "g"      # Not allowed!

lunch[2:6] = "hate" # Also not allowed!
```

More Operations

- You can add and multiply lists and strings:

```
s + t  # Concatenate lists strings
```
  - String example: `"so " + "funny" = "so funny"`
  - List example: `[2,4,6] + [8,10] = [2,4,6,8,10]`

```
s * n  # Concatenate n copies
```
  - String example: `"fun" * 3 = "funfunfun"`
  - List example: `[2,4,6] * 2 = [2,4,6,2,4,6]`