CMPT 120
Intro to CS & Programming I

WEEK 9 (Mar. 10-14)

— *Jérémie O. Lumbroso* —

Lecture 22:
Common Programs on Lists

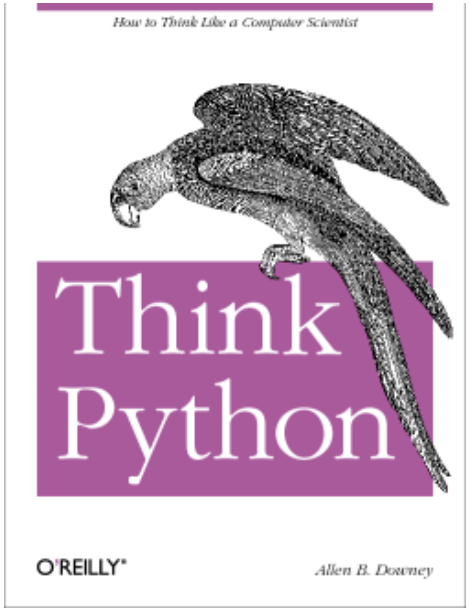
http://www.sfu.ca/~jlumbros/Courses/CMPT120/

Required Reading

Chapter 10, "Lists" of Think Python

for the next few lectures

`http://www.greenteapress.com/thinkpython/`



Some examples of indexes and slices

INDEXES AND SLICES

At the End of Last Lecture

Test if any row has a line of the same character **using iteration**: here using **element iteration**

```
boxes = [ [ "O", "X", "X" ],
          [ "X", "O", "O" ],
          [ "X", "O", "O" ] ]

for row in boxes:
  first = row[0]                          # first element
  aligned_on_this_row = True              # flag variable

  for elt in row[1:]:
    if elt != first:
      aligned_on_this_row = False

  if aligned_on_this_row:
    print first, "has lined up on a row!"
```

Another Way

Test if any row has a line of the same character **using iteration**: here using **position iteration**

```
boxes = [ [ "O", "X", "X" ],
          [ "X", "O", "O" ],
          [ "X", "O", "O" ] ]

for i in range(len(boxes)):
  aligned_on_this_row = True          # flag variable

  for j in range(1, len(boxes)):      # starts at 1
    if boxes[i][0] != boxes[i][j]:
      aligned_on_this_row = False

  if aligned_on_this_row:
    print first, "has lined up on a row!"
```

Lists of List or Matrix

- In the previous two examples, we have boxes which is a list of **lists of the same size** (all length 3 here), or a **matrix** (a rectangle table)
- We can
  - either iterate over the rows, and then manipulate these as lists
  - or use double indexing
    - `boxes[0]` gets the **list** that is an element of the list
    - `boxes[0][1]` accesses an **element** of that inner list

Some Examples

```
boxes = [ [ "O", "X", "X" ],
          [ "X", "O", "O" ],
          [ "X", "O", "O" ] ]
```

```
boxes = [ [ "O", "X", "X" ],
          [ "X", "O", "O" ],
          [ "X", "O", "O" ] ]
```

boxes[0] # first row

```
boxes = [ [ "O", "X", "X" ],
          [ "X", "O", "O" ],
          [ "X", "O", "O" ] ]
```

boxes[0][0]

```
boxes = [ [ "O", "X", "X" ],
          [ "X", "O", "O" ],
          [ "X", "O", "O" ] ]
```

boxes[1] # second row

boxes[0][1]

```
boxes = [ [ "O", "X", "X" ],
          [ "X", "O", "O" ],
          [ "X", "O", "O" ] ]
```

```
boxes = [ [ "O", "X", "X" ],
          [ "X", "O", "O" ],
          [ "X", "O", "O" ] ]
```

boxes[1:] # all but first row
[["X", "O", "O"],
["X", "O", "O"]]

boxes[1][1:] # ["O", "O"]

So What's This One?

```
boxes = [ [ "O", "X", "X" ],
          [ "X", "O", "O" ],
          [ "X", "O", "O" ] ]
```

`boxes[1:][1:]`

A `[ [ "O", "O" ] ]`

B `[ [ "O", "O"], [ "O", "O" ] ]`

C `[ "O", "O" ]`

D `[ "X", "O", "O" ]`

E `[ [ "X", "O", "O" ] ]`

Solution

```
boxes = [ [ "O", "X", "X" ],
          [ "X", "O", "O" ],
          [ "X", "O", "O" ] ]

boxes[1:] # all but first row

boxes[1:] = [ [ "X", "O", "O" ],
              [ "X", "O", "O" ] ]

boxes[1:][1:]  # all but first row
               # (of the new list)

boxes[1:][1:] = [ [ "X", "O", "O" ] ]
```

And This One?

```
boxes = [ [ "O", "X", "X" ],
          [ "X", "O", "O" ],
          [ "X", "O", "O" ] ]
```

`boxes[:1][0]`

A `[ [ "O", "X", "X" ] ]`

B `[ "O", "X", "X" ]`

C `[ "X", "X" ]`

D `"X"`

E `[ "X" ]`

Important to Remember

- Suppose `a` contains a list
  - a slice of a list, `a[1:]` or `a[:-1]` or `a[2:4]` or `a[:]` is **also a list** (a subset of the original list)
  - but using an index of a list, `a[1]` or `a[-1]` or `a[2]` or `a[4]` gets **an element**

- When dealing with a list of list, the element can also be a list, but with one less depth

Creating a new list

FIRST TEMPLATE ON LIST

What We Won't Do This Part

- We have already seen how to compute a value from a list (with the example calculating the average)
- The following CodeWrite exercise use this: `sum_of_list`, `product_of_list`, `average_of_list`

A I get the concept used in those exercises

B I did not understand those exercises

C I have not yet looked at those exercises

In This Part

- We don't look at how to compute a value from a list
- Instead we look at how to create a new list from an existing list
- The "in place" modification aspect is important but will not be mentioned this time around (leaves something for next time!)

Creating New Strings

Write a function `J_string(s)` which takes a string `s` and returns a new string where each character that is `"J"` or `"j"` has been replaced by `"A"` and all other characters by `"B"`

For example

- `J_string("Jazelle")  # returns "ABBBBBB"`
- `J_string("JJ")  # returns "AA"`
- `J_string("!!!???")  # returns "BBBBBB"`

A Done!

B Nope

Possible Solution

```
def J_string(s):
  new_s = ""
  for ch in s:
    if ch == "J" or ch == "j":
      new_s = new_s + "A"
    else:
      new_s = new_s + "B"
  return new_s
```

auxiliary variable initialized to the **empty string**

we build the string using **concatenation**

once done, we **return** the result

Creating a New List

- As we have seen, strings and lists are similar in many respects
- Writing functions that create new lists works in exactly the same way
- Differences
  - our auxiliary variable is initialized usually to the **empty list** `[ ]`
  - **concatenation** has to be between two lists, so when we add a single element we have to put it in between brackets

```
new_List = new_List + [ new_element ]
```

J_List

Function `J_List(L)` takes a list `L` and returns a new list where an element has been replaced by `"A"` if it was a `"J"` or `"j"` and `"B"` or otherwise

```
def J_List(L):
  new_L = []
  for el in L:
    if el == "J" or el == "j":
      new_L = new_L + [ "A" ]
    else:
      new_L = new_L + [ "B" ]
  return new_L
```

auxiliary variable initialized to the **empty list**

we build the list using **concatenation**

once done, we **return** the result

Some Expressions

```
Lst = [ ]
```

We consider the expression above has been entered:

```
>>> Lst = Lst + 3                    # what does Lst contain?
```

A [3] B [[3]] C Other D Nothing E ERROR

```
>>> Lst = Lst + [ 3, 4 ]             # what does Lst contain?
```

A [3, 4] B [[3, 4]] C Other D Nothing E ERROR

```
>>> print len(Lst)                   # what does this print?
```

A 1 B 2 C Other D 7 E ERROR

```
>>> Lst = Lst + [ [1, 2] ]
```

A [3, 4, 1, 2] B [3, 4, [1, 2]] C Other D Nothing E ERROR

```
>>> print len(Lst)
```

A 4 B 3 C Other D 10 E ERROR

Your Turn

```
boxes = [ [ "O", "X", "X" ],
          [ "X", "O", "O" ],
          [ "X", "O", "O" ] ]
```

```
>>> mat = [ range(3) ] * 3
>>> mat
[[0, 1, 2], [0, 1, 2], [0, 1, 2]]
```

How to check that a variable `L` contains a matrix?

- the variable must be a list
- its elements must be lists (`type(L) is list`)
- they must be of the same size (use `len`)

A Done!

B Nope

CodeWrite Exercises

- This concept of creating a new list is used in the following CodeWrite exercises of this week
  - `cumulative_sum`
  - `larger_than`
  - `reverse`
- If you have not yet tried those exercises, now is a good time!

- **TUTORIAL:** Thur., 12:30-2:20 in TASC1 9204

Pacing and Understanding

How well did you understand today?

A. Too easy, this lecture is way below my abilities

B. Everything went at a good pace, and I am fine

C. Too fast, but I will catch up on my own

D. Too fast, and I need you to slow down

E. I really do not think I can handle this