# CMPT 120
# Intro to CS & Programming I

## WEEK 9 (Mar. 10-14)

— *Jérémie O. Lumbroso* —

**Lecture 21:**
Manipulating Lists, is Just Like Strings

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/`

# Illustrative Purpose of Lists?

- Data type (or data structure) to store an **ordered sequence of elements**
- Can be useful to handle arbitrarily large collections
- Small collection: name of parents

```
father = "Malcolm"
mother = "Sienna"
```

- Large collection: i.e. name of students, this is unmanageable:

```
student0 = "Gedwinna"
student1 = "Jasper"
student2 = "Juliett"
student3 = "Philister"
…
```

- Better (to write, to access):

```
students = [ "Gedwinna", "Jasper", "Juliett",
             "Juliett", "Philister", … ]
```

# Lists in Python

- **Syntax:** brackets + elements separated by comma
- **Empty list:** `[]`
- Example: `[1, 10, "word", 4.5]`

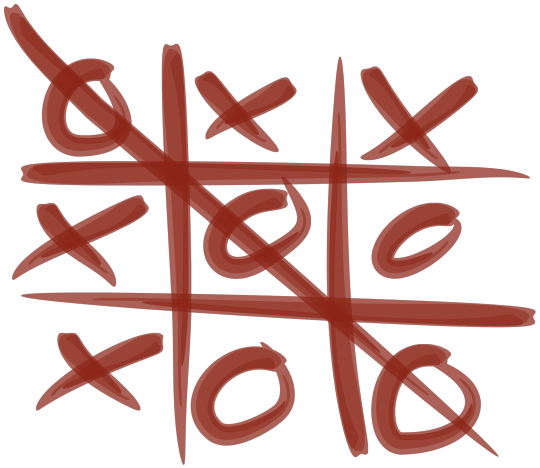
# Lists Can…

- Lists can be **heterogenous** (contain values of different types): `[1, 10, "word", 4.5]`
- Lists can be **nested** (a list can contains itself a list): `[[1,3],[5,3]]`

- It can be determine if the variable `a` is a list using the `type(...)` function:

```
if type(a) is list:
  print "var a contains a list"
else:
  print "var a does not contain a list"
```

# Other Example
# Tic-Tac-Toe



- Without lists (each on one line):

```
box1 = "O"  box4 = "X"  box7 = "X"
box2 = "X"  box5 = "O"  box8 = "O"
box3 = "X"  box6 = "O"  box9 = "O"
```

- With nested lists:

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

- Second option:
  - easier to manipulate (iterate, reset, store, etc.)
  - easier to extend (larger grids, etc.)
  - better overall

# Add an Element to a List

- `myL = [ 1, 1, 2, 3, 5 ]`
- I want to **add an element to the list**
- One possible way (more later):

```
myL = myL + [ 8 ]
print myL
# shows [ 1, 1, 2, 3, 5, 8 ]
```

- **Warning:** note brackets around the single element, this would cause an error:

```
myL = myL + 8    # brackets missing
                 # around 8
```

# Adding Example

```
myL = []              # initialize to empty list
while True:           # loop forever (or until break)
  s = raw_input("Type a number (or something else to exit: ")
  if s.isdigit():
    num = int(s)          # convert the string s into an integer
    myL = myL + [ num ]   # add element num to the list
  else:
    print "Your input was not a number."
    print "Exiting the input sequence."
    break      # exit the loop

print "You have typed the following numbers:"
print myL
```

# Slicing Lists (Just Like Strings)

- Lists can be sliced: an element or range of elements can be accessed using the following syntax
  - `myL[k]` gives element #k of the lsit
  - `myL[a:b]` gives range from element #a to #b
  - `myL[a:]` gives range from element #a to end
  - `myL[:b]` gives range from beginning to #b

- **Important:** lists (and everything else in Python) are indexed in 0; this means that the first character of a `myL` is `myL[0]` **not** `myL[1]`

# Question 1

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

Test if a row (for ex. the first one) has a line of the same character **using only indexing**?

A Got it!

B Nope

# Summary of Operations

`lstA = [1, 10, "word", 4.5]` `lstB = ["aa", 3]`

| operation | syntax | meaning (+ example) |
|---|---|---|
| indexing | `[ ]` | access an element of a list<br>`lstA[2]` gives `"word"` |
| concatenation | `+` | combine lists together<br>`lstA + lstB` gives `[1, 10, "word", 4.5, "aa", 3]` |
| repetition | `*` | repeats list multiple times<br>`[ 0 ] * 4` gives `[0, 0, 0, 0]` |
| membership | `in` | determines if an element is in a list (but is VERY slow)<br>`1 in lstA` gives `True`; `"word" in lstB` returns `False` |
| length | `len` | gives the length of a list<br>`len(lstA)` gives `4` and `len(lstB)` gives `2` |
| slicing | `[ : ]` | extracts a sequence of elements from a list<br>`lstA[1:3]` gives `[10, "word"]` (elements from `1` to `3 - 1 = 2`) |

## Isn't this familiar?

# Lists and Strings

- You know the elementary data types: `str`, `int`, `float`, `bool` and now `list`
- Lists (`list`), and strings (`str`) belong to the same category of data types in Python, called "**sequences**"
- The main difference between strings and lists is that:
  - strings cannot be modified; instead, they are copied (we always build a `new_s` when we want to alter a string)
  - list can be modified; lists can also be copied

# How to Uppercase the First Character of a String

```
mys = "green eggs and ham"

mys[0] = "G"    # gives an error
print mys       # still "green..."

mys[0].upper() # returns 'G'...
print mys       # ... but does not modify mys

# we must create a new string
new_mys = mys[0].upper() + mys[1:]
print new_mys  # works!
```

# How to Uppercase the First Element of a List

```
myL = ["green", "eggs", "and", "ham"]

myL[0] = "Green"    # no error
print myL           # [ "Green", ... ]

myL = ["green", "eggs", "and", "ham"] # reset
new_myL = [ "Green" ] + mys[1:]
print new_myL       # also works!
```

# Modifying a List: Be Careful

- Lists can be modified by making a copy

```
myL = [1, 2, 3]
myL2 = [5] + myL[1:]
print myL2                    # [5, 2, 3]
print myL                     # [1, 2, 3]
```

- Lists can be modified **without** making a copy: this is called **in-place** modification (and all "references" to the object are changed)

```
myL = [1, 2, 3]
myL2 = myL
myL2[0] = 5
print myL2                    # [5, 2, 3]
print myL                     # [5, 2, 3]
```

# Iterating over Lists

- Like with strings, two ways:
  - iterate through elements, for elt in myL:
  - iterate through positions, for i in range(len(myL)):
- In the second case, the list can be modified

# Question 2

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

Test if any row has a line of the same character **using iteration**?

A Got it!

B Nope

# 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