CISC101 Reminders & Notes

- Test 3 is being graded
  - Marks will be posted in Moodle
  - Tests will be returned in next week's tutorial
- Will schedule exam review sessions
  - Date, place and time will be announced

Today

- From last time …
  - Insertion sort
    - Slides 28-30
- Bubble Sort
- GUIs with Tkinter
  - Widgets
  - Placement of widgets

Bubble Sort

- Is best envisioned as a vertical column of numbers as bubbles
  - The larger bubbles gradually work their way to the top
  - The smaller ones are pushed down to the bottom

- *Loop through array from $i=0$ to length of array*
  - *Loop down from the last element in the array to i*
    - *Swap adjacent elements if they are in the wrong order*

Bubble Sort - Cont.

```
def bubbleSort(numsList):
    size = len(numsList)
    for i in range(0, size):
        j = size - 1
        while j > i :
            if numsList[j] < numsList[j - 1]:
                swap(numsList, j, j - 1)
            j = j - 1
```

Bubble Sort – A Slight Improvement

```
def bitBetterBubbleSort(numsList):
    size = len(numsList)
    i = 0
    isSorted = False
    while i < size and not isSorted:
        j = size - 1
        isSorted = True
        while j > i :
            if numsList[j] < numsList[j - 1]:
                swap(numsList, j, j - 1)
                isSorted = False
            j = j - 1
        i = i + 1
```

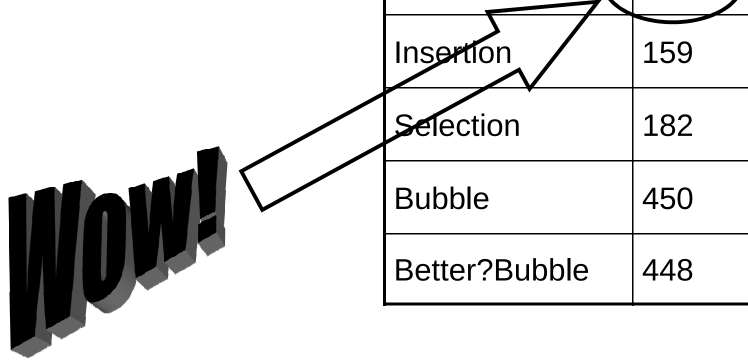
Bubble Sort - Cont.

- The improvement will (potentially) reduce the number of iterations
- Possibly the simplest sorting algorithm to code
- Also the slowest sorting algorithm!
  - Assume that there are *n* elements
  - On average, bubble sort makes *n* times more moves than selection or insertion sort

Timing Summary

- 1000 data points
  - Between 1 and 1000

Sort	Millisec
Python `sort()`	0.62
Insertion	159
Selection	182
Bubble	450
Better?Bubble	448



Next Slide – Quicksort

- You are not responsible for knowing this algorithm
  - It is included here strictly for interest

```
def quickSort(numsList, start, finish) :
    lower = start + 1
    upper = finish
    swap(numsList, start, (start + finish) / 2)
    pivot = numsList[start]
    while lower <= upper :
        while numsList[lower] < pivot :
            lower = lower + 1
        while numsList[upper] > pivot :
            upper = upper - 1
        if lower < upper:
            swap(numsList, lower, upper)
            upper = upper - 1
        lower = lower + 1

    swap(numsList, upper, start)
    if upper - start > 1 :
        quickSort(numsList, start, upper - 1)
    if finish - upper > 1 :
        quickSort(numsList, upper + 1, finish)
```

GUI with Tkinter

- Stands for “Tk Interface”
- A Python interface to the Tk GUI toolkit
  - Maintained by ActiveState
    - `www.activestate.com`
    - They also distribute a free Python development tool called ActivePython, which is an alternative to IDLE
- Tk consists of a bunch of components or *widgets*
  - Tk can be used with other languages like Perl or Ruby
- See Section 24.1 in the Python Library Reference

Some Tkinter Widgets

- Button
  - Something to click on
- Canvas
  - To draw on or display graphics
- Checkbutton
  - A checkbox (on or off)
- Entry
  - Single line text entry

Some Tkinter Widgets - Cont.

- Frame
  - Container for other widgets
- Label
  - Displays one line of text that the user cannot change
- Listbox
  - Drop down list for user selection
- Menu
  - A list of choices displayed by a Menubutton

Some Tkinter Widgets - Cont.

- Message
  - Shows multiple lines of text
- Radiobutton
  - Appear in groups where the user can only select one
- Scale
  - A slider
- Scrollbar
  - Allows scrolling for other widgets
- Text
  - User can enter multiple lines of text

Using the Tkinter Module

- It is important to note that IDLE is programmed itself using Tkinter
  - This might cause some strange results
  - Does not seem to be a problem with Python 3.1 …
- You must import the Tkinter module with one of

```
import tkinter
from tkinter import *
```

- You may have to install the module for this to work
  - For Windows Python installs it should already be there

Using Tkinter - Cont.

- Create your main or *root* window using

```
top_window = tkinter.Tk()
```

- Then start the main loop using

```
tkinter.mainloop()
```

  - This assumes you used the first **`import`** statement

Using Tkinter - Cont.

- The first line **`top_window = tkinter.Tk()`** creates a Tk *object*
  - Calls a *constructor* to create the object
  - Assigns the object to variable **`top_window`**
- The main loop is the "listener" loop for the window
  - It will wait very patiently for the user to do something
- Demo: WindowBasic.py
  - Note the functionality built into even this simple window!

Adding a Label

- Demo: WindowLabel.py

```
helloWorldLabel = tkinter.Label(top, text="HelloWorld!")
helloWorldLabel.pack()
```

- The `pack()` method is invoked to place the label into the first available position in the window
- Pretty small font …
  - Demo: WindowLabelFont.py
- How can we get away from this top-down stacking of widgets in the window?

The "Packer"

```
pack(side="top/right/bottom/left", expand=0/1,
  anchor="n/nw/w...", fill="x/y/both")
```

- All of these arguments are optional

- The "packer" is very simple but just jams stuff in
  - Displays each widget in its own row or column
  - Must use *frames* to get more than one widget in a single row or column
    - We won't be covering frames

The Grid Manager

- The Grid Manager is an alternative to the packer
  - Just a bit harder to use
- Specify a row and column position for widgets instead
- **Do not mix the packer and grid together**
  - **Use one or the other!**

The Grid Manager - Cont.

```
grid(row=?, column=?, rowspan=?, columnspan=?,
  sticky="news", ipadx=?, ipady=?, padx=?, pady=?)
```

- **`"news"`** is North, East, West, and/or South
  - Dictates which side the widget "sticks" to in its portion of the grid
  - Can set **`sticky`** to a subset of **`"news"`**
    - *e.g.*, **`"n", "e", "ws"`**
- Other dimensions are in *pixels*
  - What is a pixel, anyways?

The Grid Manager - Cont.

- Demo: WindowGrid.py
- Note how the columns and rows are sized to the width and height of the largest widget in that row or column
- Widgets are centre-aligned by default
  - Change this using the **`sticky`** option
- They can span multiple rows and columns
  - Use **`rowspan`** or **`columnspan`** (or both!)
- Add extra "padding" using **`padx`** and **`pady`**

Widget Options

- ***`widget_name`*****`.keys()`**
  - Gives you a list of dictionary keys
  - Each is an option that can be changed for that widget
- Any set of these keys can be set when you create the widget
- ***`widget_name`*****`.configure()`**
  - Gives you the dictionary with keys and values

Widget Options - Cont.

- ***`widget_name`*****`.cget(`*****`option`*****`)`**
  - Returns the value of the specified option as a string
- ***`widget_name`*****`.configure(`*****`option`*****`=?,`** ***`option`*****`=?, …)`**
  - Changes as many options as you want all at once
- Demo: WindowWidgetOptions.py

Button Widget

- A button allows the user to initiate an event
  - Most often to invoke some function in your program

```
button = Tkinter.Button(master, text=?, command=?)
```

- ***`master`*** is the window's name
  - Must be present when you create any widget

Button Widget - Cont.

- Demo: WindowButton.py
- The **`command`** option is given the name of a function
  - It is not a string
  - It cannot have brackets or parameters
- That window re-sizing thing is really annoying!
  - How do we fix it?

Grid Manager Methods

- Use the **`columnconfigure(…)`** or **`rowconfigure(…)`** methods on the master

```
columnconfigure(column, option=value, ...)
```

- Options
  - **`minsize`**
  - **`pad`**
  - **`weight`**
- Let's add some padding too
  - Demo: WindowGridConfigure.py

GUI Design

- A great deal of the time spent building a GUI program goes into design
- Two primary considerations
  - The appearance and functionality of the components
  - The window's layout, *i.e.*, where the components will sit in the window

Colours!

- Demo: WindowColours.py
- Colours are specified using the RGB system
  - Red, green and blue
- The intensity of each colour lies between 0x00 and 0xFF in hex
  - 0 and 255 in decimal
  - Possible range of 256*256*256 = 16,777,216 colours!
- Windows uses 32 bit colour
  - The extra byte is used to specify transparency

Colours - Cont.

- You can also specify colours as a parameter
  - `"black"`, `"red"`, `"green"`, `"blue"`, `"cyan"`, `"yellow"`, and `"magenta“`
  - There may be others such as `“lightblue”`, *etc.*
- The demo program is also using a colour chooser dialog box called `tkinter.colorchooser`
  - This is a GUI window that has already been constructed for us

Other Dialog Boxes

- `tkinter.commondialog`
  - Base class for the dialogs defined in the other modules listed here
- `tkinter.filedialog`
  - Common dialog for opening or saving a file
- `tkinter.messagebox`
  - Access to standard Tk dialog boxes
- `tkinter.simpledialog`
  - Basic dialogs and convenience functions
- Demo: WindowDialogs.py

Entry Widget

- Allows the user to provide information to your program by typing it into the text box
- Specify a width (**`width=?`**) in addition to the **`master`** and the **`font`**, as a minimum when you create one
- Use the **`get()`** method on the Entry widget to get the contents
  - Returned as a string only
- Demo: WindowEntry.py

Radiobutton Widget

- Allow the user to make a single choice from a list of options
  - Like the station buttons on old car radios …
- Demo: WindowRadiobutton.py
- Note …
  - the use of `IntVar()` to create a “container” `radioChoices` that stores the buttons’ value
  - the use of a *callback* function with `command=`…
  - how the colour of `choiceLabel` is set to red

Common Widgets Not Covered Here

- Canvas
- Checkbutton
- Listbox
- Menu and Menubutton
- Message
- Scale
- Scrollbar
- Text

More Info on Tkinter

- There are lots of web resources
- A good starting place are the links collected at

`http://wiki.python.org/moin/TkInter`