Take-Home Task 1: Icon Stacker

(15%, due Wednesday, September 2nd)

Overview

One of the most basic functions of any IT system is to process a given data set to produce some form of human-readable output. This task requires you to produce a visual image based on data stored in a list. It tests your abilities to:

- Process sequences of data values;
- Perform arithmetic operations;
- Display information in a visual form; and
- Produce reusable code.

These same skills, which you will acquire by completing this task at home, will subsequently be tested under time pressure in the Week 7 "in-class test."

Goal

You are required to write a Python program which processes data stored in a list to display a collection of distinct icons packed into a rectangular frame. You must produce four distinct icons, all belonging to some common theme. The icons are each of a different size. Most importantly, it must be possible to display each icon in different positions and in different orientations.

A rectangular grid is supplied to guide the positioning of the icons. Data is provided in the form of a list, where each list element is a quadruple, specifying the following properties for each icon to be drawn:

- Which style of icon to draw, either *A*, *B*, *C* and *D*.
- The column number in which the icon's bottom-left corner must appear.
- The row number in which the icon's bottom-left corner must appear.
- The icon's orientation, pointing north, south, east or west.

NB: When we refer to the icon's "bottom-left" corner we mean whichever corner appears at the bottom-left when the icon is oriented in the specified direction. North is assumed to mean that the icon is standing upright.

You are required to use Turtle graphics to draw each icon in the given data set. Each icon drawn must precisely match the dimensions specified, must be clearly distinct, and it must be possible to display the icon in any of the specified orientations. Your program must work for all of the given data sets, and any other similar data sets in the same format.

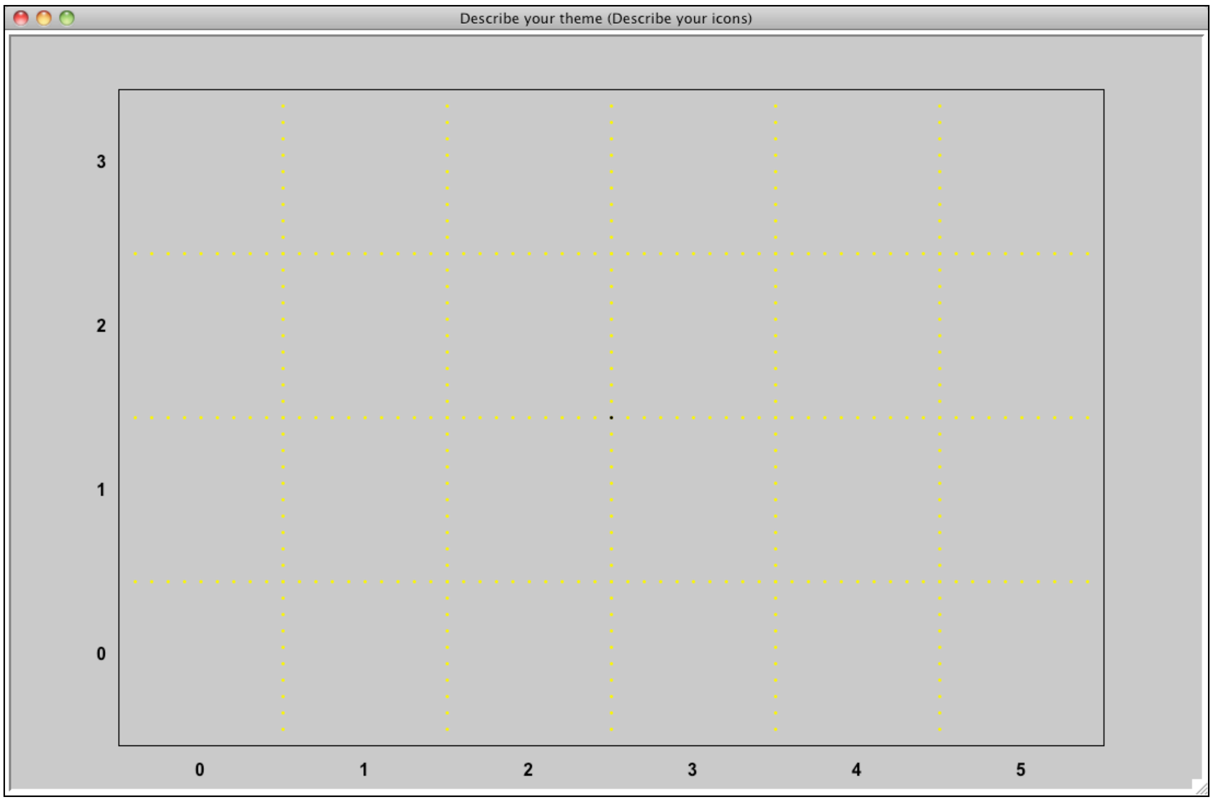
You have a free choice of icons to be displayed, as long as they all belong to some common "real world" theme and are clearly recognisable as such. The icons must be non-trivial, i.e., they cannot just be simple geometric shapes, and they must be asymmetric so that it's clear which direction they are pointing in.

The size and positioning of the icons is expressed in terms of squares in the provided grid. The four icons each have a different size in terms of their horizontal-by-vertical dimensions when upright (pointing north). The four icon sizes are as follows:

- Icon style *A*: 1 × 1 squares
- Icon style *B*: 1 × 2 squares
- Icon style *C*: 2 × 2 squares
- Icon style *D*: 2 × 3 squares

Examples

To get you started, you will find a Python template `icon_stacker.py` accompanying these instructions. When you run this program it will produce an empty grid as shown below. Notice that there is a rectangular frame in which all icons must appear, dotted grid lines to mark the individual squares, and a numeric scale for the columns and rows (counting from zero as per Python convention!).



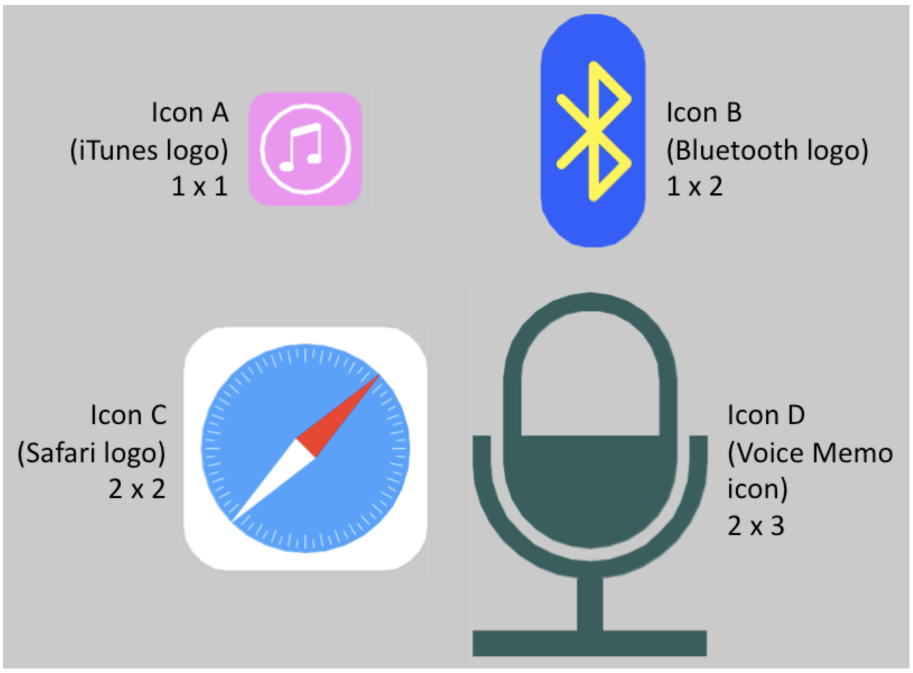
The exact centre of the drawing screen, i.e., coordinate (0, 0) is marked with a black dot. Each of the grid squares is 150 pixels wide and high.

In the Python template file you will also find several data sets in the form of lists, e.g.,

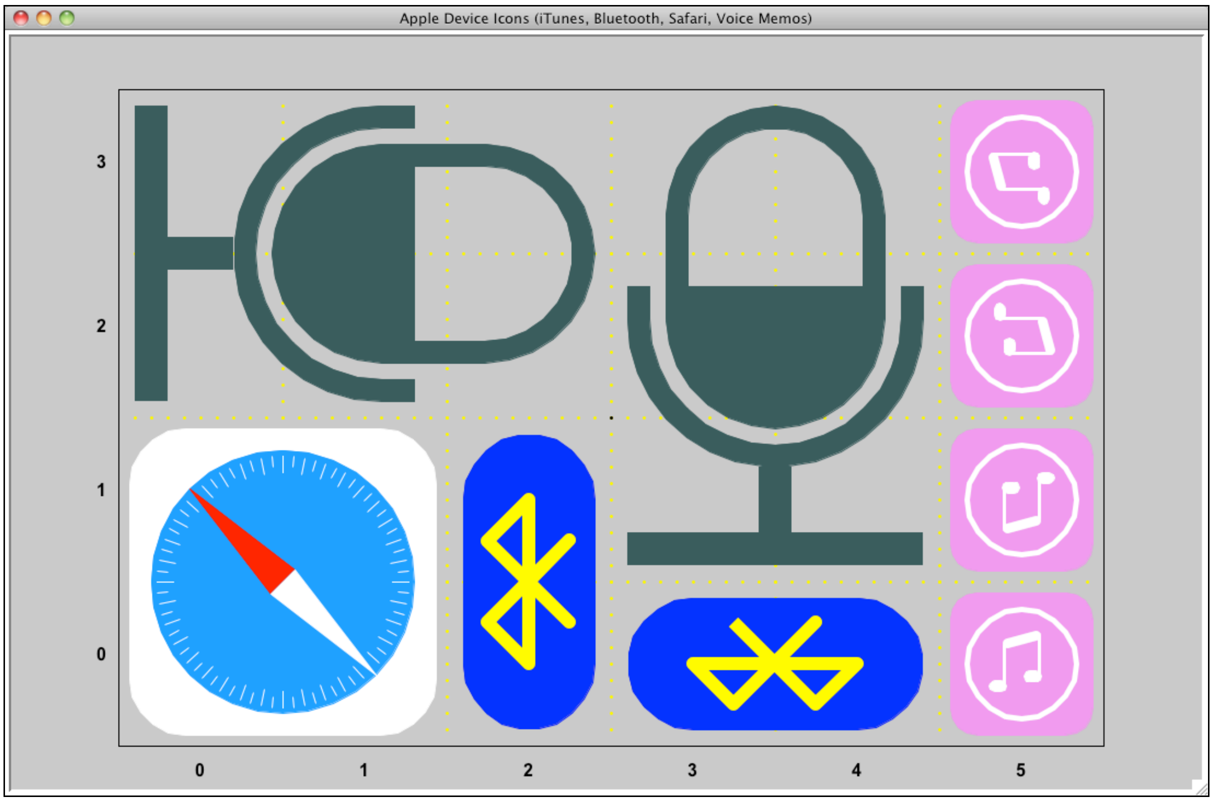
```
# All four icons in different orientations
data_set_08 = [['A', 5, 0, 'N'], ['A', 5, 1, 'S'],
               ['A', 5, 2, 'E'], ['A', 5, 3, 'W'],
               ['B', 2, 0, 'S'], ['B', 3, 0, 'E'],
               ['C', 0, 0, 'W'],
               ['D', 0, 2, 'E'], ['D', 3, 1, 'N']]
```

Each element of the list is itself a list which specifies the choice of icon to be displayed, the column and row in which the icon's bottom-left corner must appear, and its orientation. For instance, the first item in `data_set_08` above requires us to draw a copy of icon *A*, with its bottom-left corner in column 5, row 0, and pointing north (i.e., upright). Similarly, the eighth specification in the list says to draw a copy of icon *D*, with bottom-left corner in column 0, row 2, and pointing east.

Your first challenge is to design your set of icons. For the purposes of illustration, we choose here to draw symbols that are used in Apple computer products. Our four icons of choice, and their respective sizes on the grid, are as follows.



Given a data set like `data_set_08` above, you are required to draw icons in the grid of the specified types at the specified positions and in the specified orientations. For instance, the figure overleaf shows the image we drew using our four icons as specified by `data_set_08`.

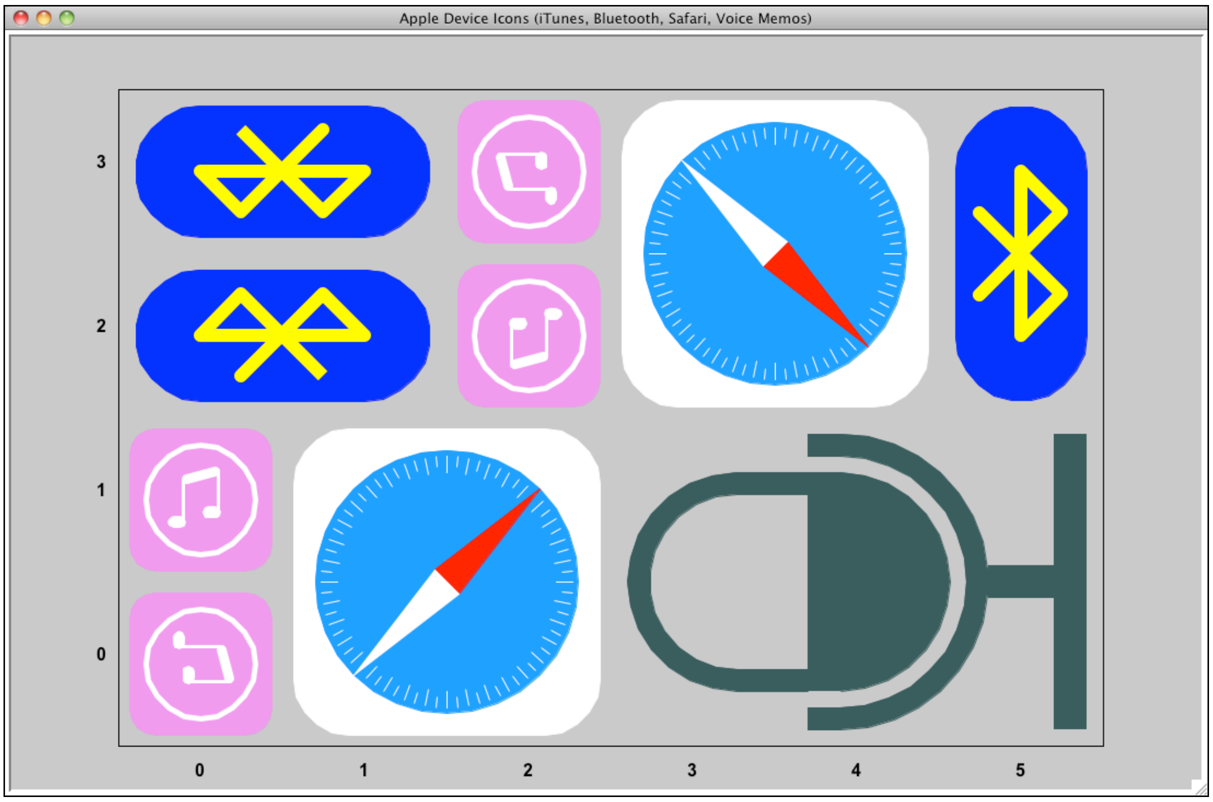


Notice that each icon fills the appropriate number of squares, is oriented as specified, and its bottom-left corner appears in the specified column and row position (regardless of its orientation). For instance, `data_set_08` says there must be an instance of icon *A*, which is the iTunes logo in our case, at column 5, row 3, pointing west. This icon can be seen in the top-right corner above. Similarly, `data_set_08` says there must be an instance of icon *D*, which is the Voice Memo symbol in our case, at column 0, row 2, pointing east. This icon appears in the top-left corner above, occupying six squares in the grid.

As well as providing several data sets to help you test your code, the given template allows you to choose whether or not to display the grid in the background. As another example, consider the following data set.

```
# Yet another arrangement with all four icons
data_set_10 = [['A', 0, 0, 'E'], ['A', 0, 1, 'N'],
               ['A', 2, 2, 'S'], ['A', 2, 3, 'W'],
               ['B', 0, 3, 'E'], ['B', 0, 2, 'W'],
               ['B', 5, 2, 'N'],
               ['C', 1, 0, 'N'], ['C', 3, 2, 'E'],
               ['D', 3, 0, 'W']]
```

The image overleaf shows the result of displaying this data set using our four example icons, but with the dotted lines in the background switched off. (The numbers and rectangular frame can also be disabled if desired.)



Once again, notice that the positioning and orientation of the icons matches the specifications in the data set. For instance, `data_set_10` says there must be a copy of icon *B*, the Bluetooth symbol in our case, pointing west, at column 0, row 2. This can be seen second from the top on the left.

When designing your icons they must occupy all of the specified number of grid squares. You may leave a *small* margin, as we have done above, if you don't want the icons touching one another. However, in cases of "tile-like" symbols, you may want to make the icons go right up to the edge of the squares.

Finally, notice that we have set the title of the drawing window to explain what the theme of our diagram is, and to list the individual icons we can draw.

General requirements

To complete this task you are required to extend the provided `icon_stacker.py` template file by completing the function `draw_icons` so that it can draw images of multiple icons packed into a frame by following a given data set that specifies the type, position and orientation of the icons to be displayed.

To get started, you first need to choose a "theme" with four icons, logos or symbols to draw. These must all be related in some way, must clearly be evocative of the theme they are intended to represent, and must be non-trivial. The icons must be asymmetric so that it is possible to tell which way they are pointing. Simple geometric shapes, such as squares, circles, etc, are *not* acceptable. Suggested sources of ideas for sets of icons include the following.

- Comic or cartoon characters
- Sporting team logos
- Corporate logos, e.g., airlines, banks, etc
- Political party logos
- Logos for commercial products, e.g., soft drinks, fast food, etc
- Automobile makes
- Icons for games
- Educational institution logos, e.g., universities or high schools
- Media brands, e.g., television, newspapers, magazines, etc

Other ideas are welcome, provided that they involve a set of non-trivial, clearly-distinct and asymmetric icons of appropriate shapes.

Specific requirements

Your submitted solution will consist of a **single Python file**, and must have at least the following features.

- You code must set the drawing window's title to **clearly identify your chosen theme and each of your icons**.
- Your program must be **capable of drawing four different icon**s, all belonging to some common theme, and each of which is clearly recognisable.
  - The icons must be **clearly distinct**.
  - There must be one icon for each of the four **different sizes** described above, 1 × 1, 1 × 2, 2 × 2 and 2 × 3 grid squares. Each icon must fill the available space, but you can leave a small margin as we did in the example above if it would not be appropriate for the icons to touch one another.
  - The icons must be **non-trivial**, i.e., they must not be simple geometric shapes such as circles and squares. We would expect that each icon would comprise *at least* three distinct shapes, typically more.
  - The icons must be clearly **asymmetric** so that it it is obvious which way they are pointing, north, south, east or west.
- Your code must be capable of drawing each of the four icons in different locations and **each icon must be rotatable to point in different directions**. Importantly, all features of each icon must rotate properly when the icon is drawn pointing different ways. When the icons are rotated there must be *at least* two distinct shapes within the icon which change position or angle, typically more.
- When your `draw_icons` function is called with a particular data set, the icons must be drawn to precisely **match the specifications in the data set** in terms of their style, position and orientation. In particular, the icons should look exactly the same in each of their four possible orientations. The provided data sets numbered 5 to 10 each

specify layouts in which all four icons appear, all squares in the rectangular frame are occupied, and no icons overlap or appear outside the frame.

- Your solution **must work for all of the provided data sets** and any other data sets in the same format. You cannot "hardwire" your solution for specific data sets and you may not change the data sets provided. (You can add additional sets if you like, as long as your code works with all the ones provided in the template file.)
- Your program code **must be presented in a professional manner**. See the coding guidelines in the *IFB104 Code Marking Guide* (on Blackboard under *Assessment*) for suggestions on how to achieve this. In particular, given the obscure and repetitive nature of the code needed to draw complex images using Turtle graphics, each significant code segment must be *clearly commented* to say what it does, e.g., what feature of the icon the corresponding code produces.

You must complete the task using **basic Turtle graphics and maths functions only**. You may not import any additional modules or files into your program other than those already included in the given `icon_stacker.py` template. In particular, you may not import any image files for use in creating your icons.

Most importantly, you are *not* required to copy the set of icon styles shown in this document. Instead you are strongly encouraged to **be creative** and to choose your own set of icons from some theme that interests you personally.

Development hints

- This is not a difficult task, but due to the need to create multiple icon styles that can be drawn in different orientations, it would be very repetitive if you tried to code the whole solution using 'brute force.' Instead you are strongly encouraged to design reusable *parameterised functions* to draw the icons to reduce the amount of code you need to write.
- There are *two basic strategies* for drawing icons that can be rotated. The first is to draw the icons using only *relative* movements ('go forward', 'turn left', etc) from the initial position, so that the icon's orientation will be different depending on which direction the cursor was facing at the very beginning. The second is to use *absolute* movements ('go to' a specific coordinate), where the *x*-*y* coordinates are calculated to be positive or negative depending on the orientation. Either approach is possible, so you can use whichever one you are more comfortable with.
- If you are unable to complete the whole task, just submit whatever parts you can get working. You will receive *partial marks for incomplete solutions*.
- To help you debug your code we have provided several data sets, numbered 1 to 4, which just draw one style of icon. You can use these to help you *create the code for each icon separately*.

Deliverable

You should develop your solution by completing and submitting the provided Python template file `icon_stacker.py` as follows. **Do not submit any other files! Do not submit a zip archive!**

1. Complete the "statement" at the beginning of the Python file to confirm that this is your own individual work by inserting your name and student number in the places indicated. **We will assume that submissions without a completed statement are not your own work and they will not be marked.**
2. Complete your solution by developing Python code at the place indicated. You must complete your solution using **only the modules already imported by the provided template**. You may **not** use or import any other modules to complete this program. In particular, you may **not** import any image files into your solution.
3. Submit **a single Python file** containing your solution for marking. Do **not** submit an archive containing several files. Only a single file will be accepted, so you cannot accompany your solution with other files or pre-defined images.

Apart from working correctly your program code must be well-presented and easy to understand, thanks to (sparse) commenting that explains the *purpose* of significant code segments and *helpful* choices of variable and function names. **Professional presentation** of your code will be taken into account when marking this task.

If you are unable to solve the whole problem, submit whatever parts you can get working. You will receive **partial marks for incomplete solutions**.

How to submit your solution

A link will be created on Blackboard under *Assessment* for uploading your solution file close to the deadline (midnight on Wednesday, September 2nd).