CSC 242 NOTES
Fall 2019/2020: Anthony Zoko

Week of Tuesday, April 27th, 2020

Graphical user interfaces

Graphical user interfaces (GUIs) give a better overview of what an application can do and make it easier to carry out application tasks.

Providing parameters to a single handler

A more sophisticated calculator can be seen in **calc.py**.

This has **several types of buttons**:

- The **digits 0 through 9**: Clicking here causes the digits to be displayed in the Entry widget
- The **operators** *, + , -, and /: Clicking one of these causes the operator to be displayed in the Entry widget
- =: This causes the function to attempt to **evaluate the expression** currently displayed in the Entry widget. If it can be evaluated, the expression is cleared and the result displayed. If it can't be evaluated, an error is displayed.
- C: This causes the Entry widget to be **cleared**

This is also the first example that shows how you can use a **loop to place items into a widget using the grid**() function.

Placing all of these options individually would be very time consuming, so placing the items and attaching the event handler in a loop is an improvement in efficiency.

Exercise: Modify the calculator GUI to be a class.

GUIs as classes review

To make a GUI application (or any other program) reusable, it should be developed as a component (a function or class) that encapsulates all the implementation details and all the references to data (and widgets) defined in the program.

A first example

In the first example of an object-oriented GUI, we re-do the hello world application using a class.

See **oop_gui1.py** for an example.

The **HelloClass** class uses a new enclosing organizational structure for building GUIs.

The GUI is built as a **subclass of the Frame class/widget**.

This means that self is of type HelloClass and of type Frame.

The Frame class **__init__** constructor method is extended to include a Button object/widget inside the HelloClass object/widget.

The (button click) **event handler function hello is a method of HelloClass**.

So event handling function calls are executed in the namespace of the HelloClass widget instance.

Customizing GUI widgets with classes

Instance variables provide a natural way to remember state between handling events.

Class callback handler methods then manipulate the state.

For an example, see the **oop_gui2.py** file.

In this example the instance variable self.data retains its state between key presses.

In general, using classes for GUIs provides a couple of **benefits**:

- **State information is retained** between events by assigning to attributes of self, and they are visible to callback methods in the class.
- It's easy to make **multiple copies** of such a GUI component because each class instance is a distinct namespace.

In the **oop_gui2.py** example, the extended **__**init**__** method takes an (optional) parent argument.

This allows Hello widgets to be embedded in other enclosing widgets. See the example in **oop_gui3.py**.

We can do the same thing by extending the Hello class, which you can see in the **oop_gui4.py** example.

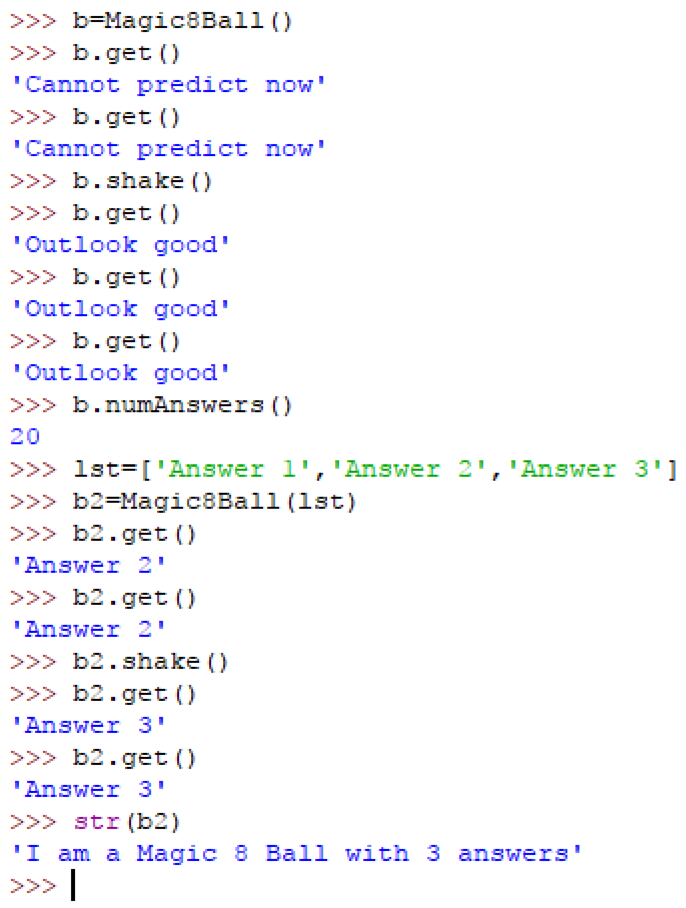
Tkinter GUIs can be coded without ever writing a single new class, but using classes to structure your GUI code makes it much more reusable in the long run.

In Class Exercise:

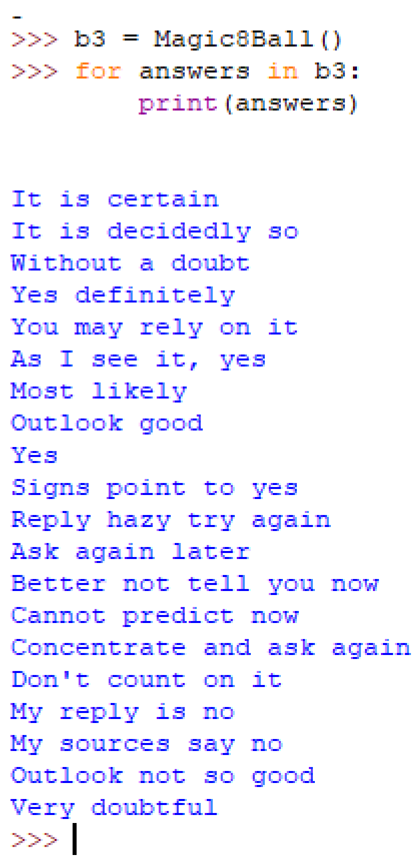
Let's work on this two-part problem together. You will find the Magic8Ball template in the class zip file.

1. Implement a class **Magic8Ball** (that is a subclass of the object class) representing an implementation of the Magic 8 Ball. The Magic 8 Ball has 20 possible answers to a question. You can find the history and a list of the 20 questions here: https://en.wikipedia.org/wiki/Magic_8-Ball . You need to implement a ball with supporting six methods:
    a. **__init__**() which either takes no parameters or takes a list representing answers. If nothing is passed in the constructor, the original 20 answers will be used. Otherwise, you can pass in a different list of answers. I recommend you set default list as empty and then populate it in the constructor if the list is empty.
    b. **get**() which returns the current answer.
    c. **shake**() which simulates the shaking of the 8 ball to get a random answer. Only shake will change the answer. In other words, after you **shake**() the ball, get will return the same answer until a new shake. Note: Per the Wikipedia article the intention was to turn the ball but not shake. We'll keep using the shake idea ☺
    d. **numAnswers**() which returns the number of answers in the ball..
    e. **__str__**() which returns a friendly string with information about how many answers it contains.
    f. **__iter__():** Allows you to iterate through all the answers in the Magic8Ball.

The following shows how the **Magic8Ball** class and its methods could be used:



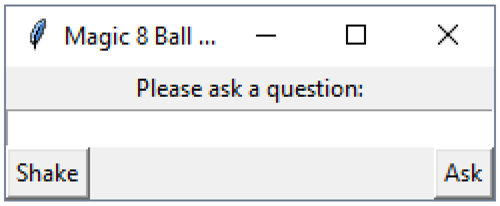
```
>>> b=Magic8Ball()
>>> b.get()
'Cannot predict now'
>>> b.get()
'Cannot predict now'
>>> b.shake()
>>> b.get()
'Outlook good'
>>> b.get()
'Outlook good'
>>> b.get()
'Outlook good'
>>> b.numAnswers()
20
>>> lst=['Answer 1','Answer 2','Answer 3']
>>> b2=Magic8Ball(lst)
>>> b2.get()
'Answer 2'
>>> b2.get()
'Answer 2'
>>> b2.shake()
>>> b2.get()
'Answer 3'
>>> b2.get()
'Answer 3'
>>> str(b2)
'I am a Magic 8 Ball with 3 answers'
>>> 
```



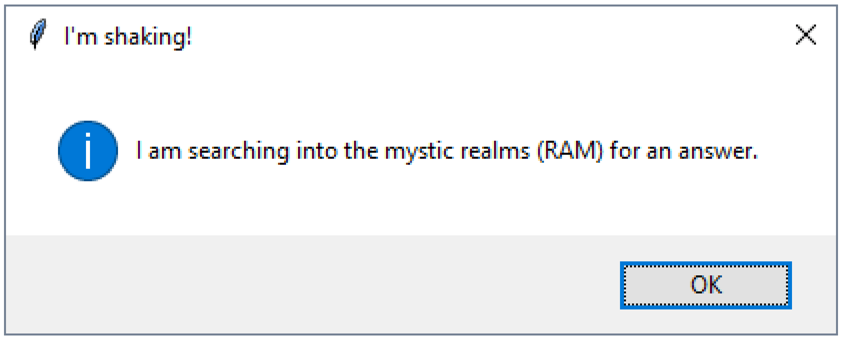
```
-
>>> b3 = Magic8Ball()
>>> for answers in b3:
        print(answers)


It is certain
It is decidedly so
Without a doubt
Yes definitely
You may rely on it
As I see it, yes
Most likely
Outlook good
Yes
Signs point to yes
Reply hazy try again
Ask again later
Better not tell you now
Cannot predict now
Concentrate and ask again
Don't count on it
My reply is no
My sources say no
Outlook not so good
Very doubtful
>>> 
```

2 Write a GUI class **Magic8BallGame** that implements the magic 8 ball game you did in part 1 but as a GUI. The GUI should start by asking the user a question:

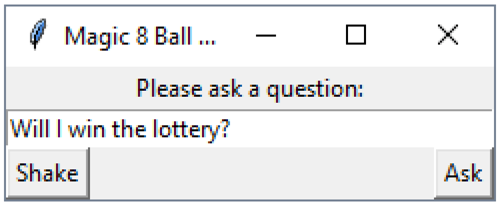


When you clock “Shake” you get the following popup. Shake forces the 8 Ball to get a new response:

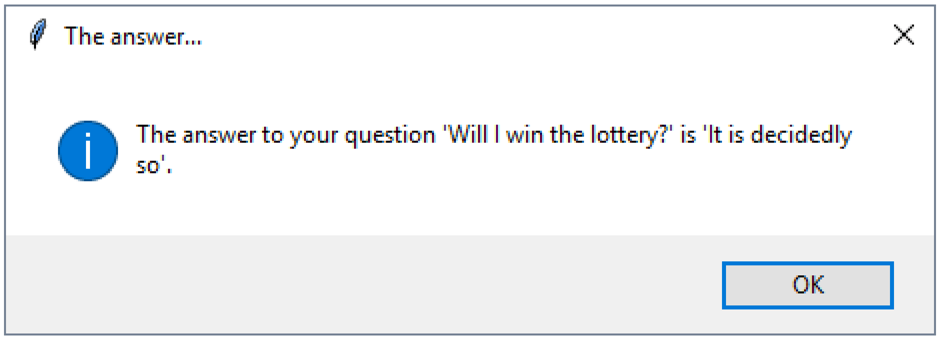

I'm shaking!

I am searching into the mystic realms (RAM) for an answer.

When you click as for the following question:

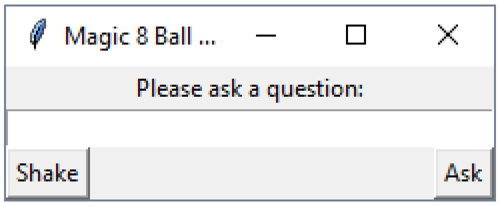


Then click “Ask” you get a response in the following format in a popup. Notice bot the answer and the question are displayed.



The answer to your question 'Will I win the lottery?' is 'It is decidedly so'.

When the answer is done, the question is cleared for the user and the text box is reset (HINT: self.entry.delete(0,END)) :



You should start with the template provided in the template file. It shows the organization of the Magic8BallGame class, including the methods you are expected to write. **Widgets should be arranged using pack()**. **The constructor has been implemented for you and shouldn't be changed**. You should test your class by writing: **Magic8BallGame().mainloop()**