

Graphical User Interface II

SQLite

#13

Serdar ARITAN

Biomechanics Research Group,
Faculty of Sports Sciences, and
Department of Computer Graphics
Hacettepe University, Ankara, Turkey





Using a `StringVar` to **connect a text-entry** box and a **label** is the first step toward **separating models** (How do we **represent** the data?), **views** (How do we **display** the data?), and **controllers** (How do we **modify** the data?), which is the key to building larger GUIs (as well as many other kinds of applications).

View is something that displays information to the user, like `Label`. Many views, like `Entry`, also accept input, which they display immediately. The key is that they don't do anything else.

Models, on the other hand, store data, like a piece of text or the current inclination of a telescope. They also don't do calculations; their job is simply to keep track of the application's **current state**.

Controllers are the pieces that convert **user input into calls on functions** in the model that manipulate the data.



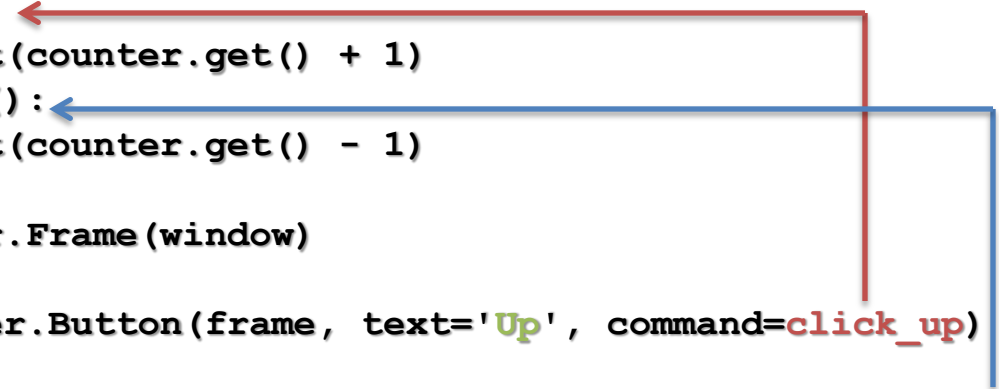
```
import tkinter
# The controller.
def click(): ←
    counter.set(counter.get() + 1)
if __name__ == '__main__':
    window = tkinter.Tk()
    # The model.
    counter = tkinter.IntVar()
    counter.set(0)
    # The views.
    frame = tkinter.Frame(window)
    frame.pack()
    button = tkinter.Button(frame, text='Click', command=click)
    button.pack()
    label = tkinter.Label(frame, textvariable=counter)
    label.pack()
    # Controllers -> Start the machinery!
    window.mainloop()
```

The first two arguments used to construct the Button should be familiar by now. The third, `command=click`, tells it to call function click each time the user presses the button. This makes use of the fact that in Python a function is just another kind of object and can be passed as an argument like anything else.

Function click in the previous code **does not have any parameters** but **uses variable counter**, which is defined outside the function. Variables like this are called global variables, and their use should be avoided, since they make programs hard to understand.



```
import tkinter
window = tkinter.Tk()
# The model.
counter = tkinter.IntVar()
counter.set(0)
# Two controllers.
def click_up(): ←
    counter.set(counter.get() + 1)
def click_down(): ←
    counter.set(counter.get() - 1)
# The views.
frame = tkinter.Frame(window)
frame.pack()
button = tkinter.Button(frame, text='Up', command=click_up)
button.pack()
button = tkinter.Button(frame, text='Down', command=click_down)
button.pack()
label = tkinter.Label(frame, textvariable=counter)
label.pack()
window.mainloop()
```



```
import tkinter
window = tkinter.Tk()
# The model.
counter = tkinter.IntVar()
counter.set(0)
# General controller.
def click(var, value):
    var.set(var.get() + value)
# The views.
frame = tkinter.Frame(window)
frame.pack()
button = tkinter.Button(frame, text='Up', command=lambda: click(counter, 1))
button.pack()
button = tkinter.Button(frame, text='Down', command=lambda: click(counter, -1))
button.pack()
label = tkinter.Label(frame, textvariable=counter)
label.pack()
window.mainloop()
```



```
import tkinter
class Counter:
    """A simple counter GUI using object-oriented programming."""
    def __init__(self, parent):
        """Create the GUI."""
        # Framework.
        self.parent = parent
        self.frame = tkinter.Frame(parent)
        self.frame.pack()
        # Model.
        self.state = tkinter.IntVar()
        self.state.set(1)
        # Label displaying current state.
        self.label = tkinter.Label(self.frame, textvariable=self.state)
        self.label.pack()
        # Buttons to control application.
        self.up = tkinter.Button(self.frame, text='up', command=self.up_click)
        self.up.pack(side='left')
        self.right = tkinter.Button(self.frame, text='quit',
        command=self.quit_click)
        self.right.pack(side='left')
    def up_click(self):
        """Handle click on 'up' button."""
        self.state.set(self.state.get() + 1)
    def quit_click(self):
        """Handle click on 'quit' button."""
        self.parent.destroy()
if __name__ == '__main__':
    window = tkinter.Tk()
    myapp = Counter(window)
    window.mainloop()
```



```
from tkinter import *  
class Application(Frame):  
    def __init__(self, master=None):  
        Frame.__init__(self, master)  
        self.grid()  
        self.create_widgets()  
        self.count_value = 0  
  
    def create_widgets(self):  
        self.count_label = Label(self, text="Count: 0")  
        self.count_label.grid(row=0, column=1)  
        self.incr_button = Button(self, text="Increment",  
                                command=self.increment_count)  
        self.incr_button.grid(row=0, column=0)  
        self.quit_button = Button(self, text="Quit",  
                                command=self.master.destroy)  
        self.quit_button.grid(row=1, column=0)  
    def increment_count(self):  
        self.count_value += 1  
        self.count_label.configure(text='Count: ' + str(self.count_value))  
  
app = Application()  
app.mainloop()
```

Creates application class
From Tkinter's Frame class

Create instance app

Runs app



Events and Bindings : Tkinter application spends most of its time inside an event loop (entered via the `mainloop` method). Events can come from various sources, including key presses and mouse operations by the user, and redraw events from the window manager (indirectly caused by the user, in many cases). Tkinter provides a powerful mechanism to let you deal with events yourself. For each widget, you can bind Python functions and methods to events.

```
widget.bind(event, handler)
```



Capturing clicks in a window

```
from tkinter import *
```

```
window = Tk()
```

```
def callback(event):  
    print ("clicked at", event.x, event.y)
```

```
frame = Frame(window, width=100, height=100)  
frame.bind("<Button-1>", callback)  
frame.pack()
```

```
window.mainloop()
```




PYTHON PROGRAMMING

GUI : Using Module Tkinter

Capturing keyboard events : Keyboard events are sent to the widget that currently owns the keyboard focus. You can use the `focus_set` method to move focus to a widget:

```
from tkinter import *  
window = Tk()  
def key(event):  
    print ("pressed", repr(event.char))  
  
def callback(event):  
    frame.focus_set()  
    print ("clicked at", event.x, event.y)  
  
frame = Frame(window , width=100, height=100)  
frame.bind("<Key>", key)  
frame.bind("<Button-1>", callback)  
frame.pack()  
window.mainloop()
```

Events : Events are given as strings, using a special event syntax

Event Formats

<Button-1> A mouse button is pressed over the widget. Button 1 is the leftmost button, button 2 is the middle button, and button 3 the rightmost button. When you press down a mouse button over a widget, Tkinter will automatically “grab” the mouse pointer, and subsequent mouse events (e.g. Motion and Release events) will then be sent to the current widget as long as the mouse button is held down, even if the mouse is moved outside the current widget. The current position of the mouse pointer (relative to the widget) is provided in the x and y members of the event object passed to the callback.

You can use **ButtonPress** instead of **Button**, or even leave it out completely:

<Button-1>, **<ButtonPress-1>**, and **<1>** are all synonyms.

<B1-Motion>The mouse is moved, with mouse button 1 being held down (use B2 for the middle button, B3 for the right button). The current position of the mouse pointer is provided in the x and y members of the event object passed to the callback.

<ButtonRelease-1>Button 1 was released. The current position of the mouse pointer is provided in the x and y members of the event object passed to the callback.

<Double-Button-1>Button 1 was double clicked. You can use Double or Triple as prefixes. Note that if you bind to both a single click (<Button-1>) and a double click, both bindings will be called.

```
<Button-1> <Double-Button-1>
```

```
from tkinter import *
```

```
def hello(event):  
    print("Single Click, Button-1")
```

```
def quit(event):  
    print("Double Click, so let's stop")  
    import sys; sys.exit()
```

```
widget = Button(None, text='Mouse Clicks')  
widget.pack()  
widget.bind('<Button-1>', hello)  
widget.bind('<Double-1>', quit)  
widget.mainloop()
```

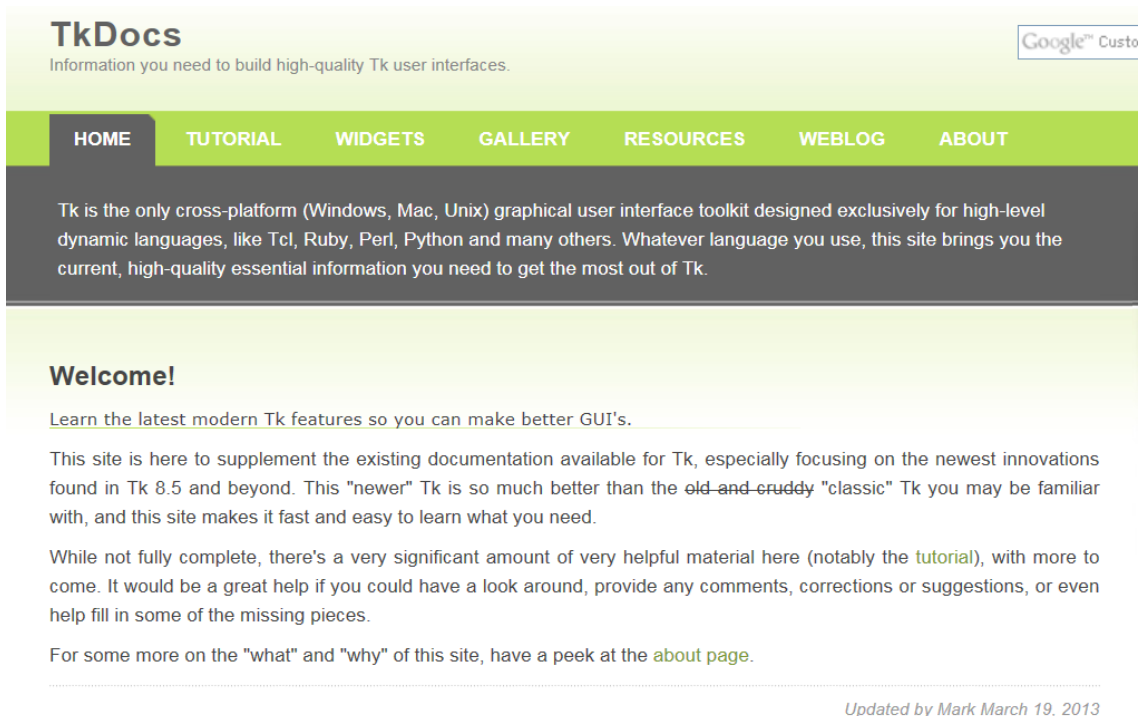
<Enter> The mouse pointer entered the widget (this event doesn't mean that the user pressed the Enter key!).

<Leave> The mouse pointer left the widget.

<FocusIn> Keyboard focus was moved to this widget, or to a child of this widget.

<FocusOut> Keyboard focus was moved from this widget to another widget.

<http://www.tkdocks.com/index.html>



The screenshot shows the TkDocs website. At the top left is the TkDocs logo, and at the top right is a Google Custom Search bar. Below the header is a green navigation bar with links: HOME, TUTORIAL, WIDGETS, GALLERY, RESOURCES, WEBLOG, and ABOUT. The main content area has a dark grey header with the text: "Tk is the only cross-platform (Windows, Mac, Unix) graphical user interface toolkit designed exclusively for high-level dynamic languages, like Tcl, Ruby, Perl, Python and many others. Whatever language you use, this site brings you the current, high-quality essential information you need to get the most out of Tk." Below this is a "Welcome!" section with the text: "Learn the latest modern Tk features so you can make better GUI's." followed by a paragraph about the site's purpose and a paragraph about the tutorial. At the bottom right, it says "Updated by Mark March 19, 2013".

TkDocs
Information you need to build high-quality Tk user interfaces.

Google™ Custom Search

HOME TUTORIAL WIDGETS GALLERY RESOURCES WEBLOG ABOUT

Tk is the only cross-platform (Windows, Mac, Unix) graphical user interface toolkit designed exclusively for high-level dynamic languages, like Tcl, Ruby, Perl, Python and many others. Whatever language you use, this site brings you the current, high-quality essential information you need to get the most out of Tk.

Welcome!

Learn the latest modern Tk features so you can make better GUI's.

This site is here to supplement the existing documentation available for Tk, especially focusing on the newest innovations found in Tk 8.5 and beyond. This "newer" Tk is so much better than the ~~old and cruddy~~ "classic" Tk you may be familiar with, and this site makes it fast and easy to learn what you need.

While not fully complete, there's a very significant amount of very helpful material here (notably the [tutorial](#)), with more to come. It would be a great help if you could have a look around, provide any comments, corrections or suggestions, or even help fill in some of the missing pieces.

For some more on the "what" and "why" of this site, have a peek at the [about page](#).

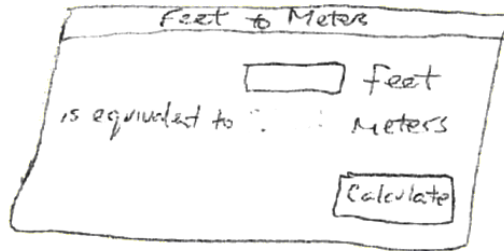
Updated by Mark March 19, 2013

<http://www.tkdocks.com/tutorial/firstexample.html#walkthrough>



Design

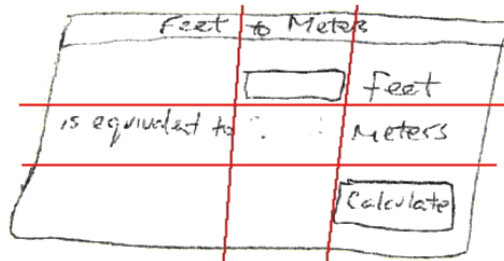
The example we'll use is a simple GUI tool that will convert a number of feet to the equivalent number of meters. If we were to sketch this out, it might look something like this:



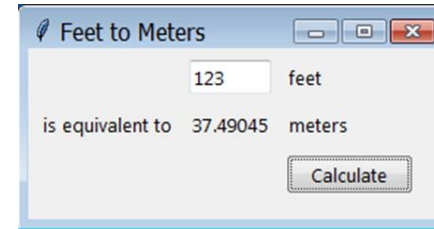
A sketch of our feet to meters conversion program.

So it looks like we have a short text entry widget that will let us type in the number of feet, and a 'Calculate' button that will get the value out of that entry, perform the calculation, and then put the resulting number of meters on the screen just below where the entry is. We've also got three static labels ("feet", "is equivalent to", and "meters") which help our user figure out how to use the interface.

In terms of layout, things seem to naturally divide into three columns and three rows:



The layout of our user interface, which follows a 3 x 3 grid.



Event Bindings : For events that don't have a command callback associated with them, you can use Tk's "**bind**" to capture any event, and then (like with callbacks) execute an arbitrary piece of code.

```
from tkinter import *
from tkinter import ttk
root = Tk()
lb =ttk.Label(root, text="Starting...")
lb.grid()
lb.bind('<Enter>', lambda e: lb.configure(text='Moved mouse inside'))
lb.bind('<Leave>', lambda e: lb.configure(text='Moved mouse outside'))
lb.bind('<1>', lambda e: lb.configure(text='Clicked left mouse button'))
lb.bind('<Double-1>', lambda e: lb.configure(text='Double clicked'))
lb.bind('<B3-Motion>', lambda e: lb.configure(text=f'right button drag to
{e.x}, {e.y}'))
root.mainloop()
```

A **canvas widget** manages a 2D collection of graphical objects — lines, circles, images, other widgets and more.

```
from tkinter import *
from tkinter import ttk

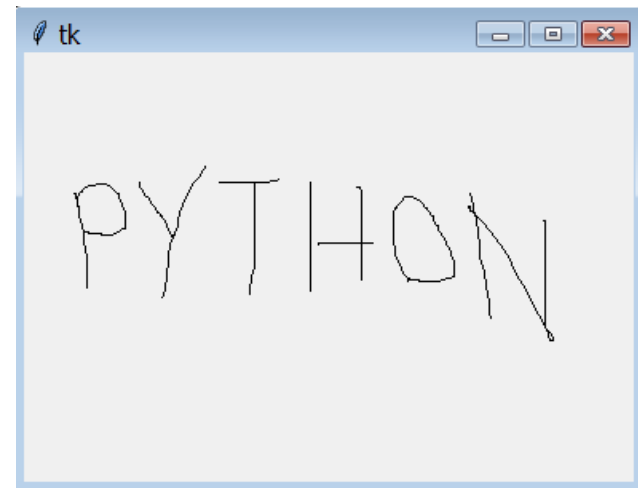
lastx, lasty = 0, 0

def xy(event):
    global lastx, lasty
    lastx, lasty = event.x, event.y

def addLine(event):
    global lastx, lasty
    canvas.create_line((lastx, lasty, event.x, event.y))
    lastx, lasty = event.x, event.y

root = Tk()
root.columnconfigure(0, weight=1)
root.rowconfigure(0, weight=1)
canvas = Canvas(root)
canvas.grid(column=0, row=0, sticky=(N, W, E, S))
canvas.bind("<Button-1>", xy)
canvas.bind("<B1-Motion>", addLine)

root.mainloop()
```



```
from tkinter import *
from tkinter import ttk

root = Tk()

h = ttk.Scrollbar(root, orient=HORIZONTAL)
v = ttk.Scrollbar(root, orient=VERTICAL)
canvas = Canvas(root, scrollregion=(0, 0, 1000, 1000), yscrollcommand=v.set, xscrollcommand=h.set)
h['command'] = canvas.xview
v['command'] = canvas.yview
ttk.Sizegrip(root).grid(column=1, row=1, sticky=(S,E))

canvas.grid(column=0, row=0, sticky=(N,W,E,S))
h.grid(column=0, row=1, sticky=(W,E))
v.grid(column=1, row=0, sticky=(N,S))
root.grid_columnconfigure(0, weight=1)
root.grid_rowconfigure(0, weight=1)

lastx, lasty = 0, 0

def xy(event):
    global lastx, lasty
    lastx, lasty = canvas.canvasx(event.x), canvas.canvasy(event.y)

def setColor(newcolor):
    global color
    color = newcolor
    canvas.dtag('all', 'paletteSelected')
    canvas.itemconfigure('palette', outline='white')
    canvas.addtag('paletteSelected', 'withtag', 'palette%s' % color)
    canvas.itemconfigure('paletteSelected', outline='#999999')
```

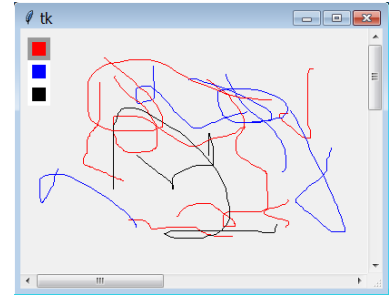
```
def addLine(event):
    global lastx, lasty
    x, y = canvas.canvasx(event.x), canvas.canvasy(event.y)
    canvas.create_line((lastx, lasty, x, y), fill=color, width=5, tags='currentline')
    lastx, lasty = x, y

def doneStroke(event):
    canvas.itemconfigure('currentline', width=1)

canvas.bind("<Button-1>", xy)
canvas.bind("<B1-Motion>", addLine)
canvas.bind("<B1-ButtonRelease>", doneStroke)

id = canvas.create_rectangle((10, 10, 30, 30), fill="red", tags=('palette', 'palette' + 'red'))
canvas.tag_bind(id, "<Button-1>", lambda x: setColor("red"))
id = canvas.create_rectangle((10, 35, 30, 55), fill="blue", tags=('palette', 'palette' + 'blue'))
canvas.tag_bind(id, "<Button-1>", lambda x: setColor("blue"))
id = canvas.create_rectangle((10, 60, 30, 80), fill="black", tags=('palette', 'palette' + 'black',
'paletteSelected'))
canvas.tag_bind(id, "<Button-1>", lambda x: setColor("black"))

setColor('black')
canvas.itemconfigure('palette', width=5)
root.mainloop()
```



There is no universal answer to how a widget should react to a user trying to enter bad data. The validation logic found in various GUI toolkits can differ greatly;

Tkinter's **validation system** is one of those parts of the toolkit that is less than intuitive. It relies on three configuration arguments that we can pass into any input widget:

- validate**: This option determines which type of event will trigger the validation callback.
- validatecommand**: This option takes the command that will determine if the data is valid.
- invalidcommand**: This option takes a command that will run if validatecommand returns **False**.

The `validate` argument specifies what kind of event triggers the validation. It can be one of the following string values:

Value	Trigger event
<code>none</code>	Never. This option turns off validation.
<code>focusin</code>	The user selects or enters the widget.
<code>focusout</code>	The user leaves the widget.
<code>focus</code>	Both <code>focusin</code> and <code>focusout</code> .
<code>key</code>	The user presses a key while in the widget.
<code>all</code>	Any of the <code>focusin</code> , <code>focusout</code> , or <code>key</code> events

How do we get a reference to a Tcl/Tk function? We just need to pass a Python callable to the `register()` method of any Tkinter widget. This returns string reference that we can use with `validatecommand`.

```
import tkinter as tk

def always_good():
    return True
def no_t_for_me(proposed):
    return 't' not in proposed

win = tk.Tk()
isim = tk.Entry(win)
isim.grid()

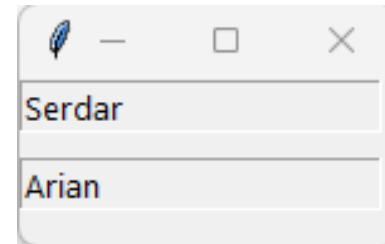
isim_ref = win.register(always_good)

isim.configure( validate='all', validatecommand =(isim_ref,))

soyisim = tk.Entry(win)
soyisim.grid(pady=10)

soyisim_ref = win.register(no_t_for_me)
soyisim.configure( validate='all', validatecommand =(soyisim_ref, '%P'))

win.mainloop()
```



we're passing the %P substitution code into our validatecommand tuple so that our callback function will be passed the proposed new value for the widget (that is, the value of the widget if the keystroke is accepted). In this case, we're going to return False if the proposed value contains the t character.





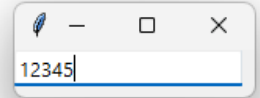
```
import tkinter as tk
import tkinter.ttk as ttk
import re
```

```
def check_num(newval):
    print(f'check_num({newval=!a})')
    return re.match('^[0-9]*$', newval) is not None and len(newval) <= 5
```

```
root = tk.Tk()
check_num_wrapper = (root.register(check_num), '%P')

num = tk.StringVar()
e = ttk.Entry(root, textvariable=num, validate='key',
              validatecommand=check_num_wrapper)
e.grid(column=0, row=0, sticky='we')
```

```
=== RESTART: C:/Lectures/BCO 601 Python Programming/G
>>> check_num(newval='1')
check_num(newval='12')
check_num(newval='123')
check_num(newval='1234')
check_num(newval='12345')
check_num(newval='123455')
check_num(newval='123456')
check_num(newval='123456')
check_num(newval='123456')
```



Code	Value passed
<code>%d</code>	A code indicating the action being attempted: 0 for delete, 1 for insert, and -1 for other events. Note that this is passed as a string, and not as an integer.
<code>%P</code>	The proposed value that the field would have after the change (key events only).
<code>%s</code>	The value currently in the field (key events only).
<code>%i</code>	The index (from 0) of the text being inserted or deleted on key events, or -1 on non-key events. Note that this is passed as a string, not an integer.
<code>%S</code>	For insertion or deletion, the text that is being inserted or deleted (key events only).
<code>%v</code>	The widget's validate value.
<code>%V</code>	The event type that triggered validation, one of focusin, focusout, key, or forced (indicating the widget's variable was changed).
<code>%W</code>	The widget's name in Tcl/Tk, as a string.



Calculate Your Body Mass Index

Body mass index (BMI) is a measure of body fat based on height and weight that applies to adult men and women.

How to **calculate** your **Body Mass Index** **(BMI)** value

$$\text{BMI} = \frac{\text{mass}_{\text{kg}}}{\text{height}_{\text{m}}^2} = \frac{\text{mass}_{\text{lb}}}{\text{height}_{\text{in}}^2} \times 703$$

Calculate Your Body Mass Index

0	1	2	
BMI			
Body Mass Index Calculator			
Metric		Imperial	
Weight :		kilograms	0
Height :		meters	1
Calculate BMI Result			2
			3
			4
			5
			6
BMI CATEGORY		CLASSIFICATION	
Below 18.5		Underweight	
18.5-24.9		Normal weight	
25.0-29.9		Pre-obesity	
30.0-34.9		Obesity class I	
35.0-39.9		Obesity class II	
Above 40		Obesity class III	

Calculate Your Body Mass Index

MODEL

```
import tkinter as tk

class Application(tk.Tk):

    def __init__(self):
        super().__init__()

        self.title("BMI")
        self.resizable(False, False)
        self.iconphoto(False, tk.PhotoImage(file="logo.png" ))
        self.config(bg='lightcyan')
        self.unit = tk.IntVar()
        self.unit.set(1) # initializing the choice, i.e. Metric
        self.unit_weight = tk.StringVar()
        self.unit_weight.set('kilograms')
        self.unit_height = tk.StringVar()
        self.unit_height.set('meters')
        self.result = tk.StringVar()
        self.result.set('')

        self.create_widgets()
```



```
def create_widgets(self):
    # Unit selection: Radio Button
    self.title_label = tk.Label(self, text="Body Mass Index Calculator", bg='lightcyan')
    self.title_label.grid(row=0, columnspan=3, sticky=tk.E + tk.W)
    self.radio_button1 = tk.Radiobutton(self, text="Metric", bg='lightcyan', variable=self.unit, value=1, command=self.sel)
    self.radio_button1.grid(row=1, column=1, padx=10)
    self.radio_button2 = tk.Radiobutton(self, text="Imperial", bg='lightcyan', variable=self.unit, value=2, command=self.sel)
    self.radio_button2.grid(row=1, column=2, padx=10)
    #
    self.weight_label = tk.Label(self, text="Weight : ", bg='lightcyan')
    self.weight_label.grid(row=2, column=0, padx=10, sticky=tk.W)
    self.height_label = tk.Label(self, text="Height : ", bg='lightcyan')
    self.height_label.grid(row=3, column=0, padx=10, sticky=tk.W)
    # Input for height and weight
    self.weight_entry = tk.Entry(self)
    self.weight_entry.grid(row=2, column=1, padx=10, sticky=tk.W+tk.E)
    self.height_entry = tk.Entry(self)
    self.height_entry.grid(row=3, column=1, padx=10, sticky=tk.W+tk.E)
    # Unit for height and weight
    self.weight_unit_label = tk.Label(self, textvariable= self.unit_weight, bg='lightcyan')
    self.weight_unit_label.grid(row=2, column=2, padx=10, sticky=tk.W)
    self.height_unit_label = tk.Label(self, textvariable= self.unit_height, bg='lightcyan')
    self.height_unit_label.grid(row=3, column=2, padx=10, sticky=tk.W)
    self.calculate_button = tk.Button(self, text="Calculate BMI Result", command=self.calculate)
    self.calculate_button.grid(row=4, columnspan=3, padx=10, sticky=tk.E + tk.W)
    self.result_label = tk.Label(self, textvariable= self.result , bg='lightcyan')
    self.result_label.grid(row=5, columnspan=3, padx=10, sticky=tk.E + tk.W)
    self.bmiGraph = tk.PhotoImage(file = "bmicalculator.png")
    self.bmiLabel = tk.Label(self, image=self.bmiGraph)
    self.bmiLabel.grid(row=6, columnspan=3)
```




```
def sel(self):
    print( "You selected the option " + str(self.unit.get()))
    if self.unit.get() == 1:
        self.unit_weight.set('kilograms')
        self.unit_height.set('meters')
    else:
        self.unit_weight.set('pounds')
        self.unit_height.set('inches')

def calculate(self):
    print( "You selected the option " + str(self.unit.get()))
    if self.unit.get() == 1:
        result = float(self.weight_entry.get())/float(self.height_entry.get())**2
        print(result)
    else:
        result = 703*float(self.weight_entry.get())/float(self.height_entry.get())**2
        print(result)
    # show the result
    self.result.set('Your BMI result is {:.2f}'.format(result))
    # clear the entry
    self.weight_entry.delete(0, tk.END)
    self.height_entry.delete(0, tk.END)

if __name__ == "__main__" :

    app = Application()
    app.mainloop()
```

Calculate Your Body Mass Index

 BMI

Body Mass Index Calculator

☒ Metric

☐ Imperial

Weight :

Height :

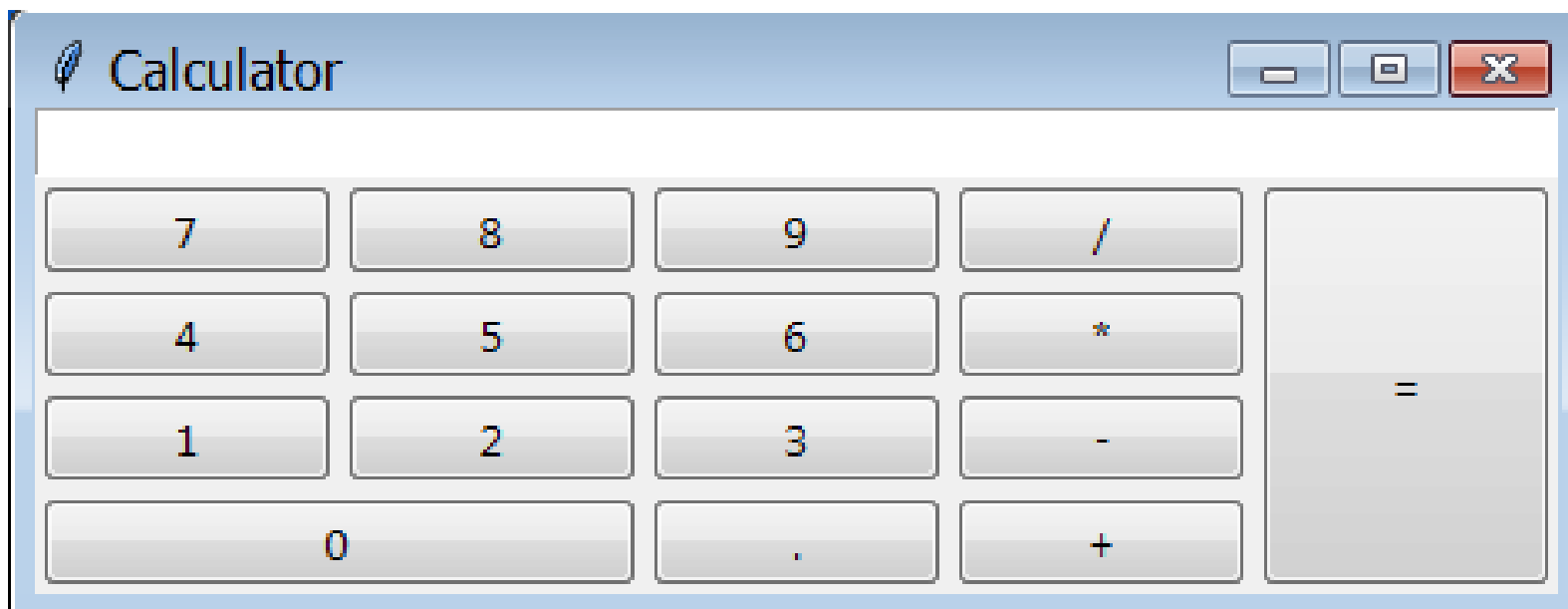
kilograms

meters

Calculate BMI Result

BMI CATEGORY	CLASSIFICATION
Below 18.5	Underweight
18.5-24.9	Normal weight
25.0-29.9	Pre-obesity
30.0-34.9	Obesity class I
35.0-39.9	Obesity class II
Above 40	Obesity class III

Designing a Calculator Application Homework



Designing a Calculator Application

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

```
root = Tk()  
root.title('Calculator')
```

```
style = ttk.Style()  
style.map("C.TButton",  
    foreground=[('pressed', 'red'), ('active', 'blue')],  
    background=[('pressed', '!disabled', 'black'), ('active', 'white')])
```

```
Entry(root).grid(row=0, column=0, sticky='nswe', columnspan = 6)  
ttk.Button(root, text="=", style="C.TButton").grid(row=1, column=5, padx=2, pady=2,  
sticky='nswe', rowspan=4)
```

```
ttk.Button(root, text="7", style="C.TButton").grid(row=1, column=0, padx=2, pady=2,  
sticky='we')  
ttk.Button(root, text="8", style="C.TButton").grid(row=1, column=1, padx=2, pady=2,  
sticky='we')  
ttk.Button(root, text="9", style="C.TButton").grid(row=1, column=2, padx=2, pady=2,  
sticky='we')
```

```
root.mainloop()
```

- **SQLite** is an open source embedded database. The original implementation was designed by D. Richard Hipp.
- Hipp was designing software used on board guided missile systems and thus had limited resources to work with.
- The resulting design goals of **SQLite** were to allow the program to be operated without a database installation or administration.
- In 2000 version 1.0 of SQLite was released. This initial release was based off of GDBM (GNU Database Manager). Version 2.0 replaced GDBM with a custom implementation of B-tree data structure.
- Version 3.0 added many useful improvements such as internalization and manifest typing.

Adobe - Uses SQLite in Photoshop and Acrobat\Adobe reader. The Application file Format of SQLite is used in these products.

Apple - Several functions in Mac OS X use SQLite:

- Apple Mail,
- Safari Web Browser,
- Aperture

The iPhone and iPad platforms may also contain SQLite implementations.

<http://www.sqlite.org/famous.html>

SQLite Major Users

Mozilla - Uses SQLite in the Mozilla Firefox Web Browser. SQLite is used in Firefox to store metadata.

Google - Google uses SQLite in Google Desktop and in Google Gears. SQLite is also used in the mobile OS platform, Android.

McAfee- Uses SQLite in its various Anti-virus programs

PHP - Php comes with SQLite 2 and 3 built in.

Python - SQLite is bundled with the Python programming language.

Specifications for SQLite

- (+) Portable - uses only ANSI-standard C and VFS, file format is cross platform (little vs big endian, 32 vs 64 bit)
- (+) Reliable – has 100% test coverage, open source code and bug database, transactions are ACID even if power fails
- (+) Small – 300 kb library, runs in 16kb stack and 100kb heap
- (-) High concurrency – reader/writer locks on the entire file
- (-) Huge datasets – DB file can't exceed file system limit or 2TB
- (-) Access control – there isn't any

SQLite is a C library that provides a lightweight disk-based database that doesn't require a separate server process and allows accessing the database using a nonstandard variant of the SQL query language. Some applications can use SQLite for internal data storage. SQLite stands alone and doesn't require any special dependencies. All the database interaction is done through whatever program needs to access the information, and the database itself is stored in a single file. Because of this, there is no confusing configuration involved, and SQLite is small and, in many applications, quite fast. It's also been released in the public domain.

```
>>> import sqlite3
>>> sqlite3.version          #the version of the pysqlite
DeprecationWarning: version is deprecated and will be removed
in Python 3.14
'2.6.0'
>>> sqlite3.sqlite_version   #the version of the SQLite
database library
'3.45.3'
>>>
```

Creating a database in SQLite is really easy, but the process requires that you know a little SQL to do it. Here's some code that will create a database to hold music albums:

```
import sqlite3 # Import the sqlite3 module
# Call connect function to obtain a Connection
conn = sqlite3.connect("mydatabase.db")
# or use :memory: to put it in RAM
# conn = sqlite3.connect(":memory:")
cursor = conn.cursor()
#Ask the Connection object to give you a Cursor object:
```

First we have to import the sqlite3 library and create a connection to the database. You can pass it a file path, file name or just use the special string “:memory:” to create the database in memory. In our case, we created it on disk in a file called `mydatabase.db`. Next we create a cursor object, which allows you to interact with the database and add records, among other things.



Here we use SQL syntax to create a table named albums with 5 text fields: title, artist, release_date, publisher and media_type. SQLite only supports five data types : null, integer, real, text and blob.

```
# create a table
```

```
cursor.execute("""CREATE TABLE albums
                  (title text, artist text,
                   release_date text, publisher text,
                   media_type text)
                """)
```

```
<sqlite3.Cursor object at 0x0000000003333B90>
```



Let's build on this code and insert some data into our new table!

```
# insert some data
```

```
cursor.execute("INSERT INTO albums VALUES ('Glow', 'Andy  
Hunter', '7/24/2012', 'Xplore Records', 'MP3')")
```

```
# save data to database
```

```
conn.commit()
```

Here we use the INSERT INTO SQL command to insert a record into our database. Note that each item had to have single quotes around it. This can get complicated when you need to insert strings that include **single quotes** in them. Anyway, to save the record to the database, we have to commit it .

insert multiple records using the more secure "?" method

```
albums = [('Exodus', 'Andy Hunter', '7/9/2002', 'Sparrow Records', 'CD'),  
          ('Until We Have', 'Red', '2/1/2011', 'Essential Records', 'CD'),  
          ('The End', 'Thousand Foot', '4/17/2012', 'TFKmusic', 'CD'),  
          ('Life', 'Trip Lee', '4/10/2012', 'Reach Records', 'CD')]
```

```
cursor.executemany("INSERT INTO albums VALUES (?, ?, ?, ?, ?)", albums)
```

```
conn.commit()
```

The code shows how add multiple records at once be using the cursor's **executemany** method. Note that we're using question marks (?) instead of string substitution (%s) to insert the values. Using string substitution is NOT safe and should not be used as it can allow **SQL injection attacks** to occur. The question mark method is much better because it does all the escaping for you so you won't have to mess with the annoyances of converting embedded single quotes into something that SQLite will accept.

Updating Records

Being able to update your database records is key to keeping your data accurate.

```
import sqlite3
conn = sqlite3.connect("mydatabase.db")
cursor = conn.cursor()
sql = """ UPDATE albums SET artist = 'John Doe' WHERE artist = 'Andy
Hunter' """
cursor.execute(sql)
conn.commit()
```

we use SQL's **UPDATE** command to update our albums table. You can use SET to change a field, so in this case we change the artist field to be "John Doe" in any record WHERE the artist field is set to "Andy Hunter". Note that if you don't commit the changes, then your changes won't be written out to the database.

Deleting Records

```
import sqlite3
conn = sqlite3.connect("mydatabase.db")
cursor = conn.cursor()

sql = """ DELETE FROM albums
          WHERE artist = 'John Doe' """

cursor.execute(sql)
conn.commit()
```

The SQL is only 2 lines! In this case, all we had to do was tell SQLite which table to delete from (albums) and which records to delete using the WHERE clause. Thus is looked for any records that had “John Doe” in its artist field and deleted it.

Basic SQLite Queries : Queries in SQLite are pretty much the same as what you'd use for other databases, such as MySQL or Postgres.

```
import sqlite3
conn = sqlite3.connect("mydatabase.db")
#conn.row_factory = sqlite3.Row
cursor = conn.cursor()
sql = "SELECT * FROM albums WHERE artist=?"
cursor.execute(sql, [("Red")])
print (cursor.fetchall()) # or use fetchone()
```

The first query we execute is a `SELECT *` which means that we want to select all the records that match the artist name we pass in, which in this case is “Red”. Next we execute the SQL and use `fetchall()` to return all the results. You can also use `fetchone()` to grab the first result. If you un-comment `row_factory`, the results will be returned as Row objects.

Basic SQLite Queries :

```
import sqlite3
conn = sqlite3.connect("mydatabase.db")

print ("\nHere's a listing of all the records in the table:\n")

for row in cursor.execute("SELECT rowid, *FROM albums ORDER BY artist"):
    print (row)
```

The second query is much like the first, but it returns every record in the database and orders the results by the artist name in ascending order. This also demonstrates how we can loop over the results.

Basic SQLite Queries :

```
import sqlite3
conn = sqlite3.connect("mydatabase.db")

print ("\nResults from a LIKE query:\n")
sql = """ SELECT * FROM albums
          WHERE title LIKE 'The%' """
cursor.execute(sql)
print (cursor.fetchall())
```

The last query shows how to use SQL's LIKE command to search for partial phrases. In this case, we do a search of the entire table for titles that start with "The". The percent sign (%) is a wildcard operator.



```
import sqlite3 as lite
import sys
con = None

try:
    con=  lite.connect("mydatabase.db")
    cur = con.cursor()
    cur.execute("SELECT SQLITE_VERSION() ")
    data = cur.fetchone()
    print ("SQLite version: %s" % data)

except (lite.Error, e):
    print ("Error %s:" % e.args[0])
    sys.exit(1)
finally:
    if con:
        con.close()
```

```
con = None
```

We initialize the con variable to None. In case we could not create a connection to the database (for example the disk is full), we would not have a connection variable defined. This would lead to an error in the finally clause.

```
except (lite.Error, e):  
    print ("Error %s:" % e.args[0])  
    sys.exit(1)
```

In case of an exception, we print an error message and exit the script with an error code 1.

```
finally:  
    if con:  
        con.close()
```

In the final step, we release the resources.

```
import sqlite3 as lite

con = lite.connect("F:\Lectures\Python\mydatabase.db")
# with is used when working with unmanaged resources

with con:

    cur = con.cursor()
    cur.execute("SELECT SQLITE_VERSION() ")

    data = cur.fetchone()

    print ("SQLite version: %s" % data)
```

With the `with` keyword, the Python interpreter automatically releases the resources. It also provides error handling. The `with` statement clarifies code that previously would use `try...finally` blocks to ensure that clean-up code is executed.



```
# create database
import sqlite3 as lite

with lite.connect('mydb.db') as conn:
    cur = conn.cursor()
    # create table
    cur.execute('''CREATE TABLE my_db
                  (id TEXT, my_var1 TEXT, my_var2 INT)''')

    # insert one row of data
    cur.execute("INSERT INTO my_db VALUES ('ID_2352532','YES', 4)")

    # insert multiple lines of data
    multi_lines =[ ('ID_2352533','YES', 1),
                    ('ID_2352534','NO', 0),
                    ('ID_2352536','YES', 9),
                    ('ID_2352535','YES', 3),
                    ('ID_2352537','YES', 10)
                  ]
    cur.executemany('INSERT INTO my_db VALUES (?,?,?)', multi_lines)

    # save (commit) the changes
    conn.commit()

    # close connection
    conn.close()
```




```
# querying database
import sqlite3 as lite

# open existing database
with lite.connect('mydb.db') as conn:

    cur = conn.cursor()
    # print all lines ordered by integer value in my_var2
    for row in cur.execute('SELECT * FROM my_db ORDER BY my_var2'):
        print(row)
    print('---- ordered by integer value in my_var2 ----')

    # print all lines that have "YES" as my_var1 value
    # and have an integer value <= 7 in my_var2
    t = ('YES', 7,)
    for row in cur.execute('SELECT * FROM my_db WHERE my_var1=? AND my_var2 <= ?', t):
        print(row)
    print('have YES" as my_var1 value and have an integer value <= 7 in my_var2')

    # print all lines that have "YES" as my_var1 value
    # and have an integer value <= 7 in my_var2
    t = ('YES', 7,)
    cur.execute('SELECT * FROM my_db WHERE my_var1=? AND my_var2 <= ?', t)
    rows = cur.fetchall()
    for r in rows:
        print(r)

# close connection
conn.close()
```

```
# update database
import sqlite3 as lite

with lite.connect('mydb.db') as conn:

    cur = conn.cursor()
    # update field
    t = ('NO', 'ID_2352533', )
    cur.execute("UPDATE my_db SET my_var1=? WHERE id=?", t)
    print ("Total number of rows changed:", conn.total_changes)

    # delete rows
    t = ('NO', )
    cur.execute("DELETE FROM my_db WHERE my_var1=?", t)
    print ("Total number of rows deleted: ", conn.total_changes)
    # add column
    cur.execute("ALTER TABLE my_db ADD COLUMN 'my_var3' TEXT")
    # save changes
    conn.commit()
    # print column names
    cur.execute("SELECT * FROM my_db")
    col_name_list = [tup[0] for tup in cur.description]
    print (col_name_list)

# close connection
conn.close()
```