# Adding GUI & Widget

Chapter 7

# Revision

- Using Lists

```
>>> bob = ['Bob Smith', 42, 30000, 'software']
>>> sue = ['Sue Jones', 45, 40000, 'hardware']


>>> bob[0].split()[-1]         # what's bob's last name?
'Smith'
>>> sue[2] *= 1.25             # give sue a 25% raise
>>> sue
['Sue Jones', 45, 50000.0, 'hardware']
```

- Using Dictionaries

```
>>> bob = {'name': 'Bob Smith', 'age': 42, 'pay': 30000, 'job': 'dev'}
>>> sue = {'name': 'Sue Jones', 'age': 45, 'pay': 40000, 'job': 'hdw'}

>>> bob['name'], sue['pay']           # not bob[0], sue[2]
('Bob Smith', 40000)

>>> bob['name'].split()[-1]
'Smith'

>>> sue['pay'] *= 1.10
>>> sue['pay']
44000.0
```

# Revision

- **Using Classes - 1**

```
class Person:
    def __init__(self, name, age, pay=0, job=None):
        self.name = name
        self.age  = age
        self.pay  = pay
        self.job  = job
    def lastName(self):
        return self.name.split()[-1]
    def giveRaise(self, percent):
        self.pay *= (1.0 + percent)

if __name__ == '__main__':
    bob = Person('Bob Smith', 42, 30000, 'software')
    sue = Person('Sue Jones', 45, 40000, 'hardware')
    print(bob.name, sue.pay)

    print(bob.lastName())
    sue.giveRaise(.10)
    print(sue.pay)
```

-

```
Bob Smith 40000
Smith
44000.0
```

# Revision

- **Using Classes -2 :** Encapsulated in the form of classes customizable implementations of our records and their processing logic

```
from person import Person

class Manager(Person):
    def giveRaise(self, percent, bonus=0.1):
        self.pay *= (1.0 + percent + bonus)

if __name__ == '__main__':
    tom = Manager(name='Tom Doe', age=50, pay=50000)
    print(tom.lastName())
    tom.giveRaise(.20)
    print(tom.pay)
```

When run, this script's self-test prints the following:

```
Doe
65000.0
```

# Revision

- **Using make_db_classes**

```
import shelve
from person import Person
from manager import Manager

bob = Person('Bob Smith', 42, 30000, 'software')
sue = Person('Sue Jones', 45, 40000, 'hardware')
tom = Manager('Tom Doe',  50, 50000)

db = shelve.open('class-shelve')
db['bob'] = bob
db['sue'] = sue
db['tom'] = tom
db.close()
```

- **Using dump_db_classes**

```
import shelve
db = shelve.open('class-shelve')
for key in db:
    print(key, '=>\n  ', db[key].name, db[key].pay)

bob = db['bob']
print(bob.lastName())
print(db['tom'].lastName())
```

```
bob =>
   Bob Smith 30000
sue =>
   Sue Jones 40000
tom =>
   Tom Doe 50000
Smith
Doe
```

- **Using update_db_classes**

```
import shelve
db = shelve.open('class-shelve')

sue = db['sue']
sue.giveRaise(.25)
db['sue'] = sue

tom = db['tom']
tom.giveRaise(.20)
db['tom'] = tom
db.close()
```

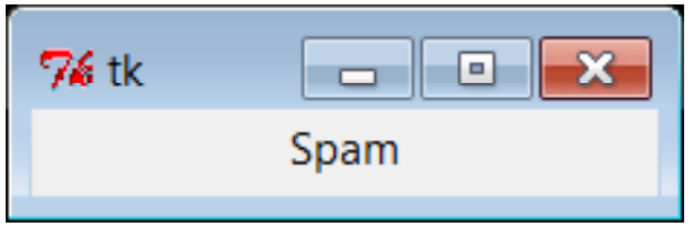
```
bob =>
   Bob Smith 30000
sue =>
   Sue Jones 50000.0
tom =>
   Tom Doe 65000.0
Smith
Doe
```

# GUI Basics

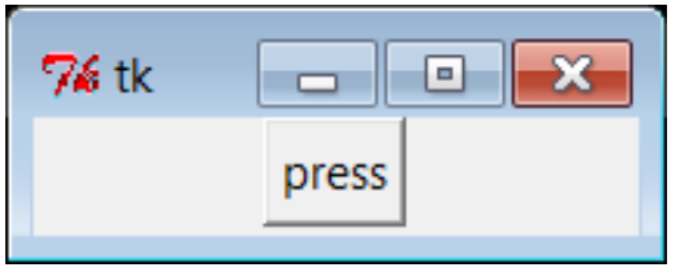
- GUI toolkits and builders are available for Python programmers:
  - tkinter
  - wxPython
  - PyQt
  - PythonCard
  - Dabo
- **tkinter** is a lightweight toolkit and so meshes well with a scripting language such as Python
- Also portable across Windows, Linux/Unix, and Macintosh
- simply copy the source code to the machine on which you wish to use your GUI.

# Two basic examples

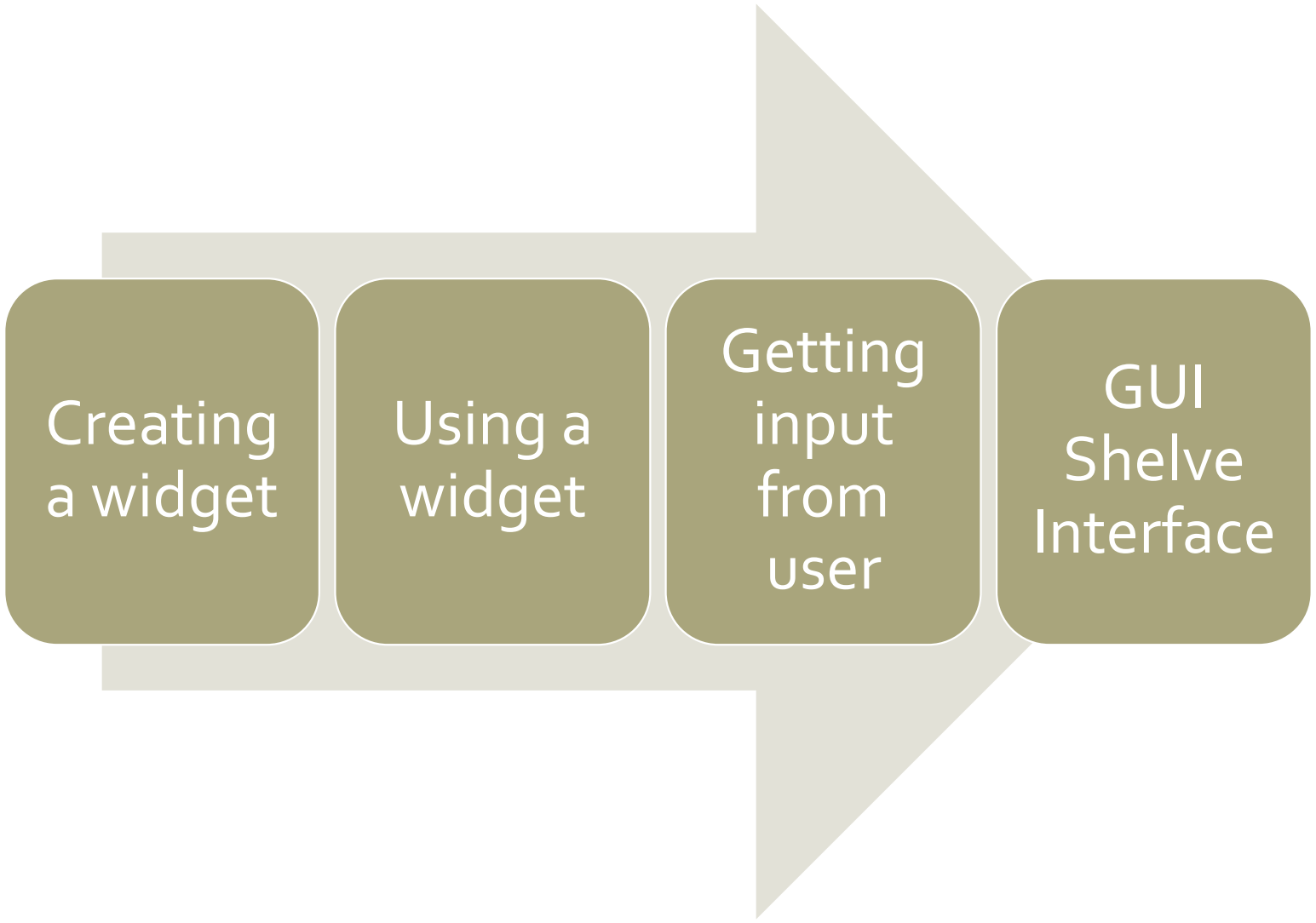
```
from tkinter import *
Label(text='Spam').pack()
mainloop()
```




```
from tkinter import *
from tkinter.messagebox import showinfo

def reply():
    showinfo(title='popup', message='Button pressed!')

window = Tk()
button = Button(window, text='press', command=reply)
button.pack()
window.mainloop()
```



# Process Flow



# Using OOP for GUIs – Creating a Widget

- In larger programs, it is often more useful to code a GUI as a subclass of the tkinter Frame widget—a container for other widgets.
- Single-button GUI recorded in this way as a class

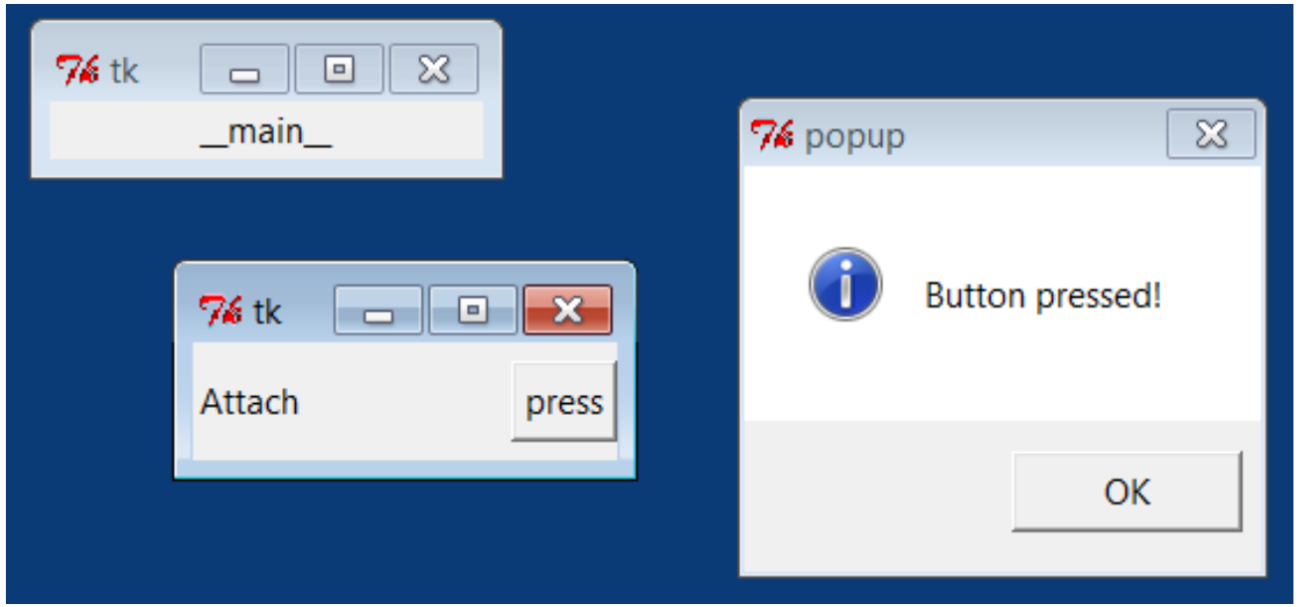
```
from tkinter import *
from tkinter.messagebox import showinfo

class MyGui(Frame):
    def __init__(self, parent=None):
        Frame.__init__(self, parent)
        button = Button(self, text='press', command=self.reply)
        button.pack()
    def reply(self):
        showinfo(title='popup', message='Button pressed!')

if __name__ == '__main__':
    window = MyGui()
    window.pack()
    window.mainloop()
```

# Example – Using a widget

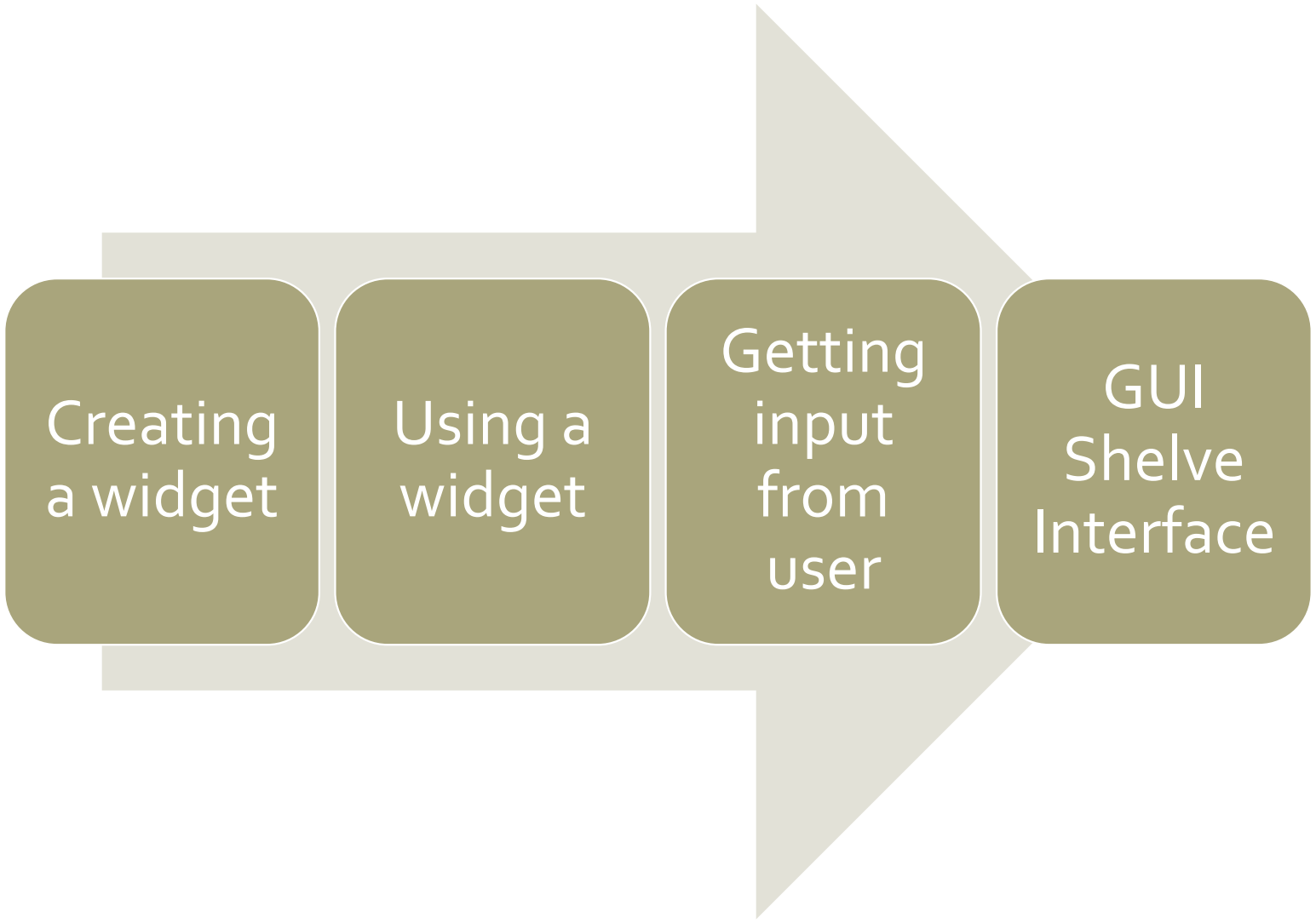
- This example attaches our one-button GUI to a larger window, here a Top-level popup window created by the importing application and passed into the construction call as the explicit parent.
- Our one-button widget package is attached to the right side of its container this time.

```
from tkinter import *
from tkinter102 import MyGui

# main app window
mainwin = Tk()
Label(mainwin, text=__name__).pack()

# popup window
popup = Toplevel()
Label(popup, text='Attach').pack(side=LEFT)
MyGui(popup).pack(side=RIGHT)                    # attach my frame
mainwin.mainloop()
```

# Example – Using a widget



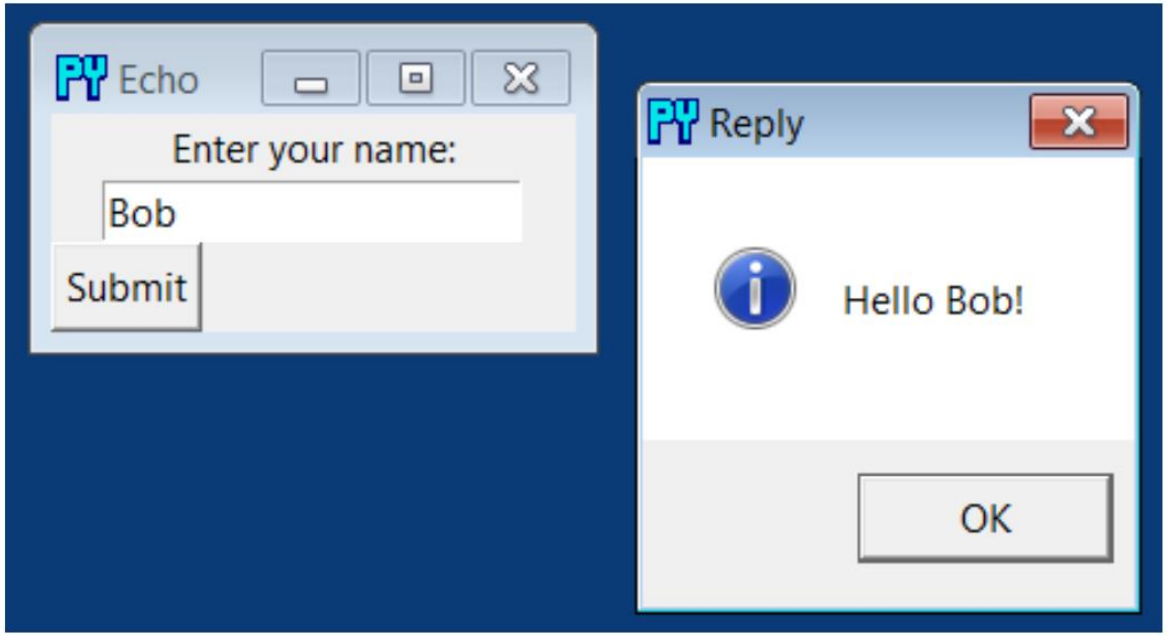
# Process Flow



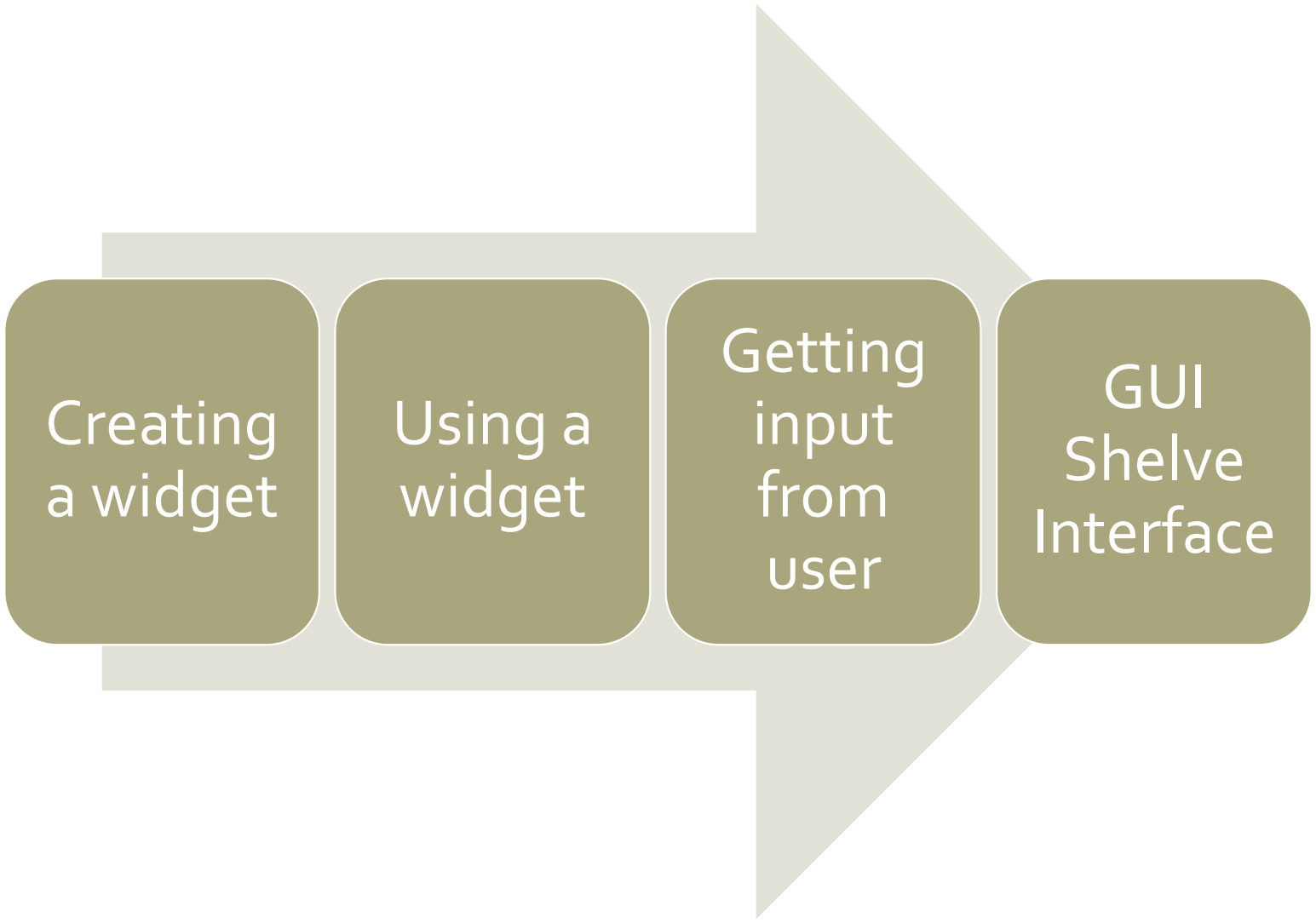
# Getting input from user

```
from tkinter import *
from tkinter.messagebox import showinfo

def reply(name):
    showinfo(title='Reply', message='Hello %s!' % name)

top = Tk()
top.title('Echo')
top.iconbitmap('py-blue-trans-out.ico')

Label(top, text="Enter your name:").pack(side=TOP)
ent = Entry(top)
ent.pack(side=TOP)
btn = Button(top, text="Submit", command=(lambda: reply(ent.get())))
btn.pack(side=LEFT)

top.mainloop()
```

# Getting input from user



# Process Flow

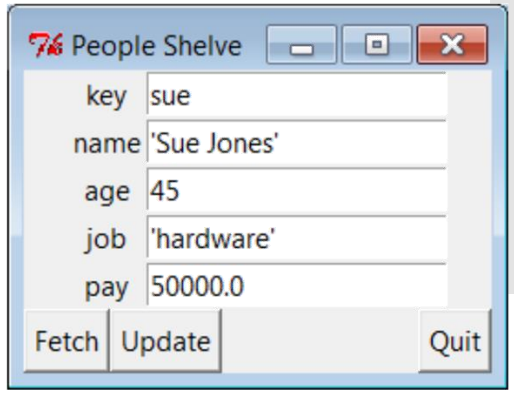


# GUI Shelve Interface

- Console-based interface approach of the preceding section works, and it may be sufficient for some users assuming that they are comfortable with typing commands in a console window.
- A view on the shelve file and allows us to browse and update the file without typing any code.

# GUI Shelve Interface

```
"""
Implement a GUI for viewing and updating class instances stored in a shelve;
the shelve lives on the machine this script runs on, as 1 or more local files;
"""

from tkinter import *
from tkinter.messagebox import showerror
import shelve
shelvename = 'class-shelve'
fieldnames = ('name', 'age', 'job', 'pay')


def makeWidgets():
    global entries
    window = Tk()
    window.title('People Shelve')
    form = Frame(window)
    form.pack()
    entries = {}
    for (ix, label) in enumerate(('key',) + fieldnames):
        lab = Label(form, text=label)
        ent = Entry(form)
        lab.grid(row=ix, column=0)
        ent.grid(row=ix, column=1)
        entries[label] = ent
    Button(window, text="Fetch",  command=fetchRecord).pack(side=LEFT)
    Button(window, text="Update", command=updateRecord).pack(side=LEFT)
    Button(window, text="Quit",   command=window.quit).pack(side=RIGHT)
    return window
```



# GUI Shelve Interface

```
def fetchRecord():
    key = entries['key'].get()
    try:
        record = db[key]                        # fetch by key, show in GUI
    except:
        showerror(title='Error', message='No such key!')
    else:
        for field in fieldnames:
            entries[field].delete(0, END)
            entries[field].insert(0, repr(getattr(record, field)))

def updateRecord():
    key = entries['key'].get()
    if key in db:
        record = db[key]                        # update existing record
    else:
        from person import Person               # make/store new one for key
        record = Person(name='?', age='?')      # eval: strings must be quoted
    for field in fieldnames:
        setattr(record, field, eval(entries[field].get()))
    db[key] = record

db = shelve.open(shelvename)
window = makeWidgets()
window.mainloop()
db.close() # back here after quit or window close
```