1.9 Looking at variables

Variables

To implement some of its functions, Tkinter uses a very special kind of variable called an **observable variable**. This variable works like a regular variable (i.e., it's able to store values which are accessible to the outside world) but there is something more – any change of the variable's state can be **observed** by a number of external agents. For example, the `Entry` widget can use its own observable variable to inform other objects that the contents of the input field have been changed.

From a technical point of view, such a variable is an object of the **container class**. This means that a variable of that kind has to be **explicitly created and initialized**.

There is another important difference – these variables are **typed**. You have to be aware of what type of value you want to store in them, and don't change your mind during the variable's life.

Note: you can only create an observable variable **after the main window initialization**. Don't forget this – you've been warned!

There are **four** kinds (types) of observable variable used by `tkinter`:

- BooleanVar
- DoubleVar
- IntVar
- StringVar

The names you see are also the constructors' names, so if you want to use any of the variables, you must invoke the proper constructor and save the returned object.

Note: the newly created variables are set to:

- integer `0` for `IntVar`;
- float `0.0` for `DoubleVar`;
- Boolean `False` for `BooleanVar`;
- string «» for `StringVar`.

For example, if a widget is able to serve an observable variable of the string type, you'll create it in the following way:

```
s = StringVar()
```

If you want to **assign a value** to an observable variable, you have to invoke its method, named `set()`, and pass an argument to it. The argument should be of a type compatible with the variable's kind.

The example shows how to assign a string to a variable of the `StringVar` kind:

```
strng.set("To be or not to be")
```

If you need to **use the value** stored in a variable, you have to use the variable method named `get()`:

```
sn = strng.get()
```

The method returns the value of the type compatible with the variable's kind.

Each observable variable can be **enriched** with a number of **observers**. An observer is a **function** (a kind of callback) which will be invoked automatically each time a specified event occurs in the variable's life.

The number of observers is not limited.

Adding an observer to a variable is done by a method named `trace()`:

```
obsid = variable.trace(trace_mode, observer)
```

The method takes two arguments:

- a string describing which events should **trigger an observer** – the possible values are:
  - «r» – if you want to be aware of the variable reads (accessing its value through `get()`)
  - «w» – if you want to be aware of the variable writes (changing its value through `set()`)
  - «u» – if you want to be aware of the variable's annihilation (removing the object through `del`)
- a reference to a function which will be invoked when the specified event occurs.

The function returns a string which is a unique observer **identifier**. Don't try to interpret its contents. You don't want to know its meaning.

The observer should be declared as a **three-parameter function**:

```
def observer(id, ix, act):
:
:
```

- `id` – an internal observable variable identifier (unusable for us);
- `ix` – an empty string (always – don't ask us why, it's `tkinter`'s business)
- `act` – a string informing us what happened to the variable or, in other words, what reason triggered the observer ('r', 'w' or 'u')

If you don't need any of the arguments, you can declare the observer as: `def obs(*)`:

Removing the observer is done with a method named `trace_vdelete()`:

```
variable.trace_vdelete(trace_mode,obsid)
```

Its arguments' meanings are as follows:

- `trace_mode` – the mode in which the observer has been created;
- `obsid` – the observer's identifier obtained from the previous `trace()` invocation.

We've prepared a simple snippet showing how the observable variables **cooperate** with their observers. Take a look at it, we've provided it in the editor.

```
import tkinter as tk


def r_observer(*args):
    print("Reading")


def w_observer(*args):
    print("Writing")


dummy = tk.Tk()    # we need this although we won't display any windows
```

```
variable = tk.StringVar()
variable.set("abc")
r_obsid = variable.trace("r", r_observer)
w_obsid = variable.trace("w", w_observer)
variable.set(variable.get() + 'd')  # read followed by write
variable.trace_vdelete("r", r_obsid)
variable.set(variable.get() + 'e')
variable.trace_vdelete("w", w_obsid)
variable.set(variable.get() + 'f')
print(variable.get())
```

The code creates one observable variable of type `StringVar` and assigns two observers to it – one for reading and one for writing. The observers send a line to `stdout` when invoked.

Trace the code's execution and try to explain its behavior. We're sure you can do it.