Home

If you're new to Python and VPython: Introduction

A VPython tutorial

Pictures of 3D objects

Choose a 3D object

Work with 3D objects

Displays/Events

What's new in VPython 6

VPython web site
VPython license
Python web site
Math module (sqrt etc.)
Numpy module (arrays)

Keyboard Interactions

The simplest keyboard interaction is to wait for the user to press a key before proceeding in the program. Suppose the 3D display is in scene, the default window created by VPython. Here is a way to wait for a keypress:

```
key = scene.kb.getkey() # wait for and get keyboard info
```

If len(key) == 1, the input is a single printable character such as 'b' or 'B' or new line ('\n') or tab ('\t'). Otherwise key is a multicharacter string such as 'escape' or 'backspace' or 'f3'. For such inputs, the ctrl, alt, and shift keys are prepended to the key name. For example, if you hold down the shift key and press F3, key will be the character string 'shift+f3', which you can test for explicitly. If you hold down all three modifier keys, you get 'ctrl+alt+shift+f3'; the order is always ctrl, alt, shift.

Multicharacter names include delete, backspace, page up, page down, home, end, left, up, right, down, numlock, scrlock, f1, f2, f3, f4, f5, f6, f7, f8. Windows and Linux also have f9, f11, f12, insert.

Pressing the enter/return key produces the code '\n' ("new line").

Starting with VPython 6, an alternative to waiting for a keypress is to wait for various kinds of keyboard or mouse events:

```
scene.waitfor('keydown') # wait for keyboard key press
scene.waitfor('keyup')   # wait for keyboard key release
scene.waitfor('click keydown') # click or keyboard
scene.waitfor('click')     # wait for a click
scene.waitfor('mousedown') # wait for mouse button press
scene.waitfor('mouseup') # wait for mouse button release
scene.waitfor('mousemove') # wait for mouse to be moved
scene.waitfor('mousedown mousemove') # either event
```

The event 'keydown' or 'keyup' occurs when you press or release a key on the keyboard (if you hold down a key, you may get multiple 'keydown' events due to repeats).

You can obtain a package of information about the event that caused the end of the wait:

```
from visual import *
box()
ev = scene.waitfor('click keydown')
if ev.event == 'click':
    print('You clicked at', ev.pos)
else:
    print('You pressed key '+ev.key)
```

Polling and callback

There are two different ways to get a keyboard event, "polling" and "callback". In polling, you continually check **scene.kb.keys** to see whether any events are waiting to be processed, and you use **scene.kb.getkey()** to get the next event to process. Prior to VPython 6, this was the only way you could handle mouse or keyboard events.

If you use the callback method, you specify a function to be executed when a specific type of event occurs, and the function is sent the event information when the specified type of event occurs. For many purposes this is a better way to handle keyboard and mouse events, and we will discuss it first. Programs that use polling will continue to work, but you cannot mix polling and callback approaches: you must use one or the other in a program.

Handling events with callbacks

Here is a test routine using callbacks that lets you type text into a label.

```
from visual import *
prose = label() # initially blank text

def keyInput(evt):
    s = evt.key
    if len(s) == 1:
        prose.text += s # append new character
    elif ((s == 'backspace' or s == 'delete') and
            len(prose.text)) > 0:
```

```
        if evt.shift:
            prose.text = '' # erase all text
        else:
            prose.text = prose.text[:-1] # erase letter

scene.bind('keydown', keyInput)
```

We define a "function" named "keyInput". Then we "bind" this function to 'keydown' events occurring in the display named "scene". Whenever VPython detects that a 'keydown' event has occurred, VPython calls the bound function, which in this case adds the input to the text of the label object.

Note that **evt.ctrl**, **evt.alt**, and **evt.shift** are True if the corresponding key is down at the time of the event. On the Macintosh, **evt.cmd** is True if the Command key is down.

This operation is called a "callback" because with scene.bind you register with VPython that you want to be called back any time there is a 'keydown' event. Here are the built-in events that you can specify in a bind operation:

```
Keyboard: keydown, keyup
Mouse:    click, mousedown, mousemove, mouseup
Other:    redraw, draw_complete
```

The event 'keydown' or 'keyup' occurs when you press or release a key on the keyboard. The events 'click', 'mousedown', 'mousemove', and 'mouseup' are discussed in the **mouse** section. A 'redraw' event occurs just before the 3D scene is redrawn on the screen, and a 'draw_complete' event occurs just after the redrawing (these event have rather technical uses such as timing how often redrawings occur, or how much time they take).

You can bind more than one event to a function. The following will cause the callback function to be executed whether you click with the mouse or press a key on the keyboard:

```
scene.bind('click keydown', myFunction)
```

With the following statement, click event will no longer be sent to myFunction:

```
scene.unbind('click', myFunction)
```

The example program eventHandlers.py illustrates the callback method for handling many kinds of events.

Handling events with polling

The following information on how to handle events using polling is still valid, but you are encouraged to consider using the more powerful callback approach when writing new programs. Remember that you cannot mix the two schemes. You can use either callback or polling in a program, but not both.

Here is a test routine using polling that lets you type text into a label. If scene.kb.keys is nonzero, one or more keyboard events have been stored, waiting to be processed. If one or more events are available, you can execute key = scene.kb.getkey() to obtain the oldest keyboard input and remove it from the input queue.

```
from visual import *
prose = label() # initially blank text
while True:
    rate(30)
    if scene.kb.keys: # event waiting to be processed?
        s = scene.kb.getkey() # get keyboard info
        if len(s) == 1:
            prose.text += s # append new character
        elif ((s == 'backspace' or s == 'delete') and
              len(prose.text)) > 0:
            prose.text = prose.text[:-1] # erase letter
        elif s == 'shift+delete':
            prose.text = '' # erase all text
```

Note that mouse events also provide information about the ctrl, alt, and shift keys, which may be used to modify mouse actions.

Pausing for keyboard or mouse input

Often you want to pause for either mouse or keyboard input. You can copy the following function into your program, and then insert **pause()** wherever you want to pause.

```
def pause():
    while True:
        rate(30)
```

```
if scene.mouse.events:
    m = scene.mouse.getevent()
    if m.click == 'left': return
elif scene.kb.keys:
    k = scene.kb.getkey()
    return
```

As of VPython 6, an alternative to this function is simply to write **scene.waitfor('click keydown')**.