2 Development Platforms for Multimedia Programming

2.1 Introduction to Python

2.2 Multimedia Frameworks for Python

2.3 Document-Based Platforms: SMIL, OpenLaszlo

2.4 Multimedia Scripting Languages: JavaFX, Processing

2.5 Authoring Tools: Flash

Literature:

G. van Rossum and F. L. Drake, Jr., An Introduction to Python - The Python Tutorial (version 2.5), Network Theory 2006
http://www.network-theory.co.uk/docs/pytut/

How to Choose a Development Platform?

- Who will be the developer?
  - Experienced programmer: Brings some knowledge to build upon
  - Unexperienced programmer: Wants a simple language to grow with (Python)
  - Non-programmer: Wants to avoid programming at all (why?)
- What will be required from the program?
  - Operating systems to run on?
  - Functional features: Sound, graphics, communications, …
  - Non-functional features: memory usage, execution speed, security, …

- Guido van Rossum, 1991, CWI Amsterdam
- Targeted at programming novices
- Characteristics:
  - Interpreted scripting language
  - Compiled to intermediate byte code (similar to Java)
  - Multi-paradigm language:
    imperative/structured, object-oriented, functional, aspect-oriented
  - Dynamic typing
  - Automatic garbage collection
- Do you really understand all these terms?

Java to Python: Imperative Example (Java)

```
public class Main {

    public static int sequentialSearch(int q, int[] a) {
        for(int i = 0; i < a.length; i++) {
            if(a[i]==q) {
                return i;
            }
        }
        return -1;
    }

    public static void main(String[] args) {

        int[] a = {11, 22, 33, 44, 55, 66};
        System.out.println("Array a: "+a);
        System.out.println("Search for 55: "+sequentialSearch(55,a));
        System.out.println("Search for 23: "+sequentialSearch(23,a));

    }

}
```

Java to Python: Imperative Example (Python)

```
def sequentialSearch (q, a):
  for i in range(0,len(a)):
     if a[i]==q:
          return i
  return -1

a = [11, 22, 33, 44, 55, 66]
print "Array a: ", a
print "Search for 55: ",sequentialSearch(55,a)
print "Search for 23: ",sequentialSearch(23,a)
```

First Observations on Python

- Very compact code
- Data types are not specified
- Powerful but simple built-in list datatype

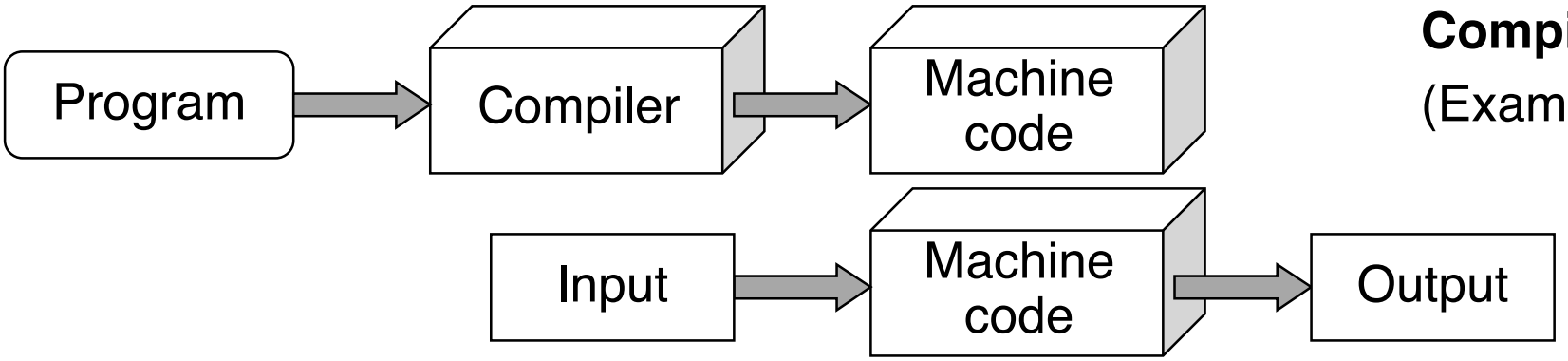
- Indentation (white space) is important for program semantics !!!
  - Block levels given by indentation
  - What is done in Java with {} brackets, is done here with indentation
- Example: A different (wrong!) algorithm:

```
def sequentialSearch (q, a):
     for i in range(0,len(a)):
          if a[i]==q:
               return i
          return -1
```

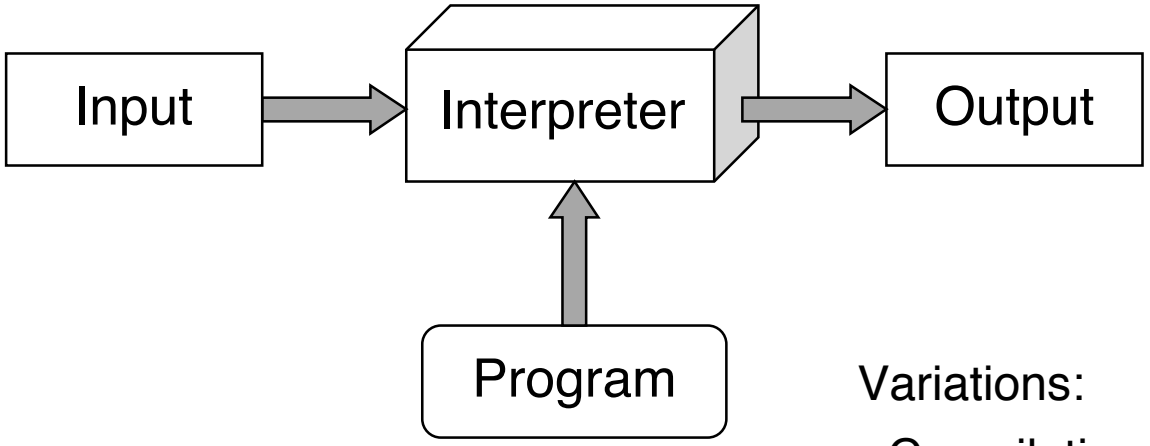
Scripting Language

- Traditionally:
  A *scripting language* is a programming language that is used to control some application software
  - Command languages for operating systems (*batch* and *shell* languages)
  - Scripts for task automatisation in user interfaces
  - Scripts executed in Web browsers, word processors, spreadsheet software, …
- Historically, scripting languages were considered slow in execution and limited in program size
- Modern general-purpose scripting languages
  - Have inherited many features from traditional scripting languages
  - Are considered as full application programming languages:
  - Examples: Rexx, Perl, **Python**, Ruby

Compilation, Interpretation and Others



Compilation
(Example: C)

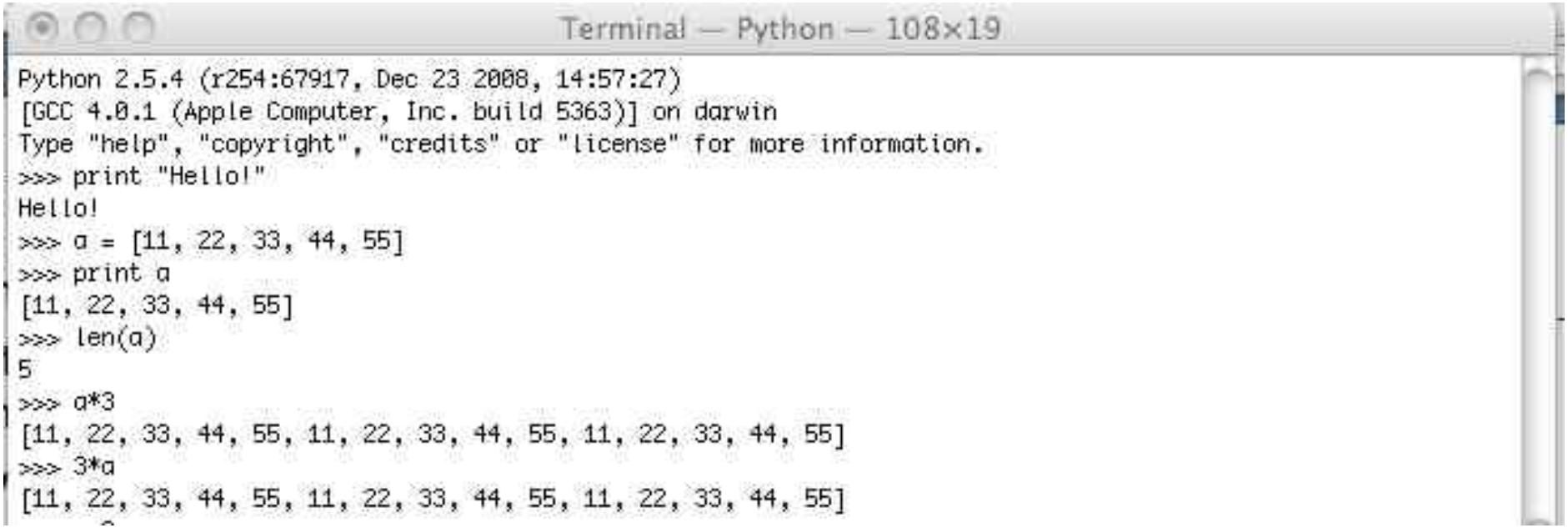


Interpretation
(Example: Python)

Variations:

- Compilation to intermediate code (Java)
- Just-in-time compilation

Interactive Interpreter



- Interpreted languages can easily by executed line-by-line
- Interactive execution is helpful for understanding
  - See BASIC, Logo etc.

Static and Dynamic Typing

- Type checking:
  - Simple, automatically executable form of proof for program correctness
  - Avoids operations to be applied to unsuitable arguments
- Static typing:
  - Type information is checked **before execution** of program (at compile time)
  - Program code has to specify types for all variables
  - Examples: Java, Pascal, C, Standard ML
- Dynamic typing:
  - Type information is checked **during execution** of program (at run time)
  - Type information for variables only exists after value assignment
  - Examples: Smalltalk, Python, JavaScript
- In practice, static and dynamic tying are sometimes mixed:
  - See the dynamic type check for *downcast* operations in Java!

Strong and Weak Typing

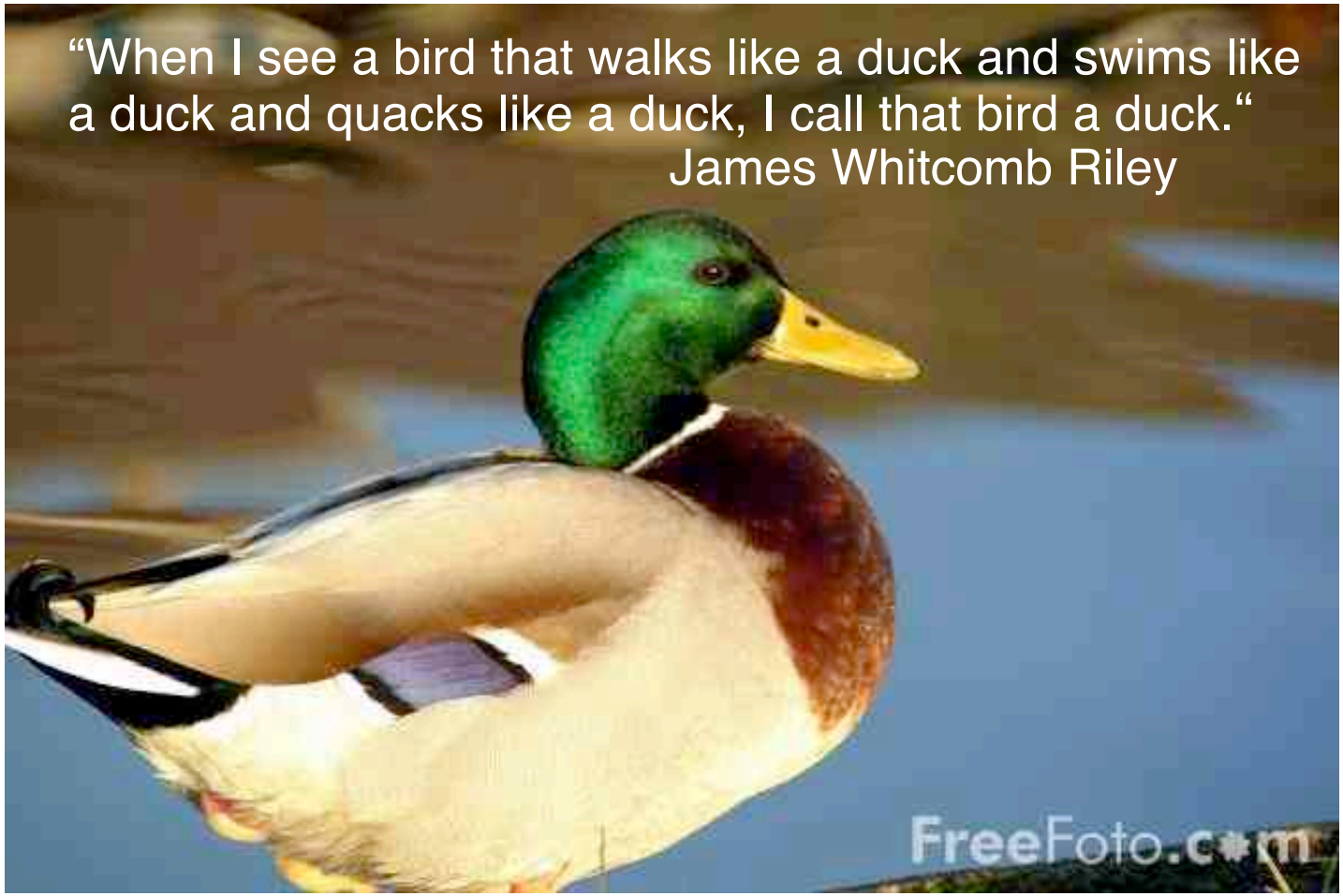
- Surprisingly ill-defined terms!
- Strong typing:
  - Basic idea: “Strong” typing provides no (or only very limited) possibility to evade the restrictions of the type system
  - Examples of strongly typed languages:

    Java, Pascal, Standard ML, **Python**
- Weak typing:
  - Implicit type conversions
  - Type conversions with undefined result
  - Examples of weakly typed languages:

    Visual Basic, C
- Do not confuse extended operator signatures with weak typing!
  - Python can multiply strings with numbers:

```
>>> 3*'abc'
'abcabcabc'
```

Duck Typing



- The type of an object is determined only by the fact whether it has the features required from it.
- Appropriate for object-oriented programming languages with dynamic types - like Python.

String Operations in Python

Operations vlaid for all sequence types:

- Indexing: **`str[5]`**
- Negative indexing: **`str[-5]`** (counting from the end)
- Slicing: **`str[2:5], str[:5], str[2:6:2], str[::-1]`**
  - Omitted index is begin or end, third value is step size (covers reversion)
- Operations:

  `len(str), min(str), max(str), x in str`

Numerous methods specific for strings like:

- **`capitalize()`**
- **`count(substr)`**
- **`find(substr)`**
- **`isalpha()`**
- **`partition(sep)`**
- **`replace`**
- **`split(sep)`**
- **`upper()`**
- **`title()`**

Lists in Python

- List: Sequential collection of objects (of arbitrary, also varying type)
- Can be easily used as stack or queue data structures
- Flexible creation of lists e.g. by *list comprehension:*

```
l = [3*x for x in range(1,4)]
```

- Lists are mutable (can be even changed through slices)
- List methods:
  - **`append`**
  - **`count`**
  - **`extend`**
  - **`index`**
  - **`insert`**
  - **`pop`**
  - **`remove`**
  - **`reverse`**
  - **`sort`**

Sets in Python

- Set: Unordered collection without duplicates
- Constructor
  - **`set`** builds a set from a list
- Basic mathematical operations for sets:
  - Union (|)
  - Intersection (&)
  - Difference (-)
  - Symmetric difference (^)
- Example:

```
set('multimedia') & set('programming')
```

Java to Python: Imperative Example (Python)

```
def sequentialSearch (q, a):
  return q in a

a = [11, 22, 33, 44, 55, 66]
print a
print "Array a: ", a
print "Search for 55: ",sequentialSearch(55,a)
print "Search for 23: ",sequentialSearch(23,a)
```

Tuples and Dictionaries in Python

- Tuple: immutable collection of objects (of arbitrary type)

  ```
  N = ('max','muster')
  N = 'max','muster'
  ```

  Strange: One-element tuple written as `'max',`
- Easy unpacking of tuples:

  ```
  vorname, nachname = ('max','muster')
  ```

- Dictionary: Mutable collection of object maps (of arbitrary type)

  ```
  age = {'anna':23, 'max':22}
  ```

  – Key entries can only be of immutable type (strings, numbers, tuples)
  – Key entries must be *hashable*
  – Main purpose: indexed access `age['anna']`
- Constructor accepts lists or *generator expressions*:

  ```
  dict((x, x*x) for x in range(0,5))
  ```

Java to Python: Object-Oriented Example (Java)

```
public class Counter {

  private int k = 0;

  public void count () {
      k++;
  }

  public void reset () {
      k = 0;
  }

  public int getValue () {
      return k;
  }
}
```

Java to Python: Object-Oriented Example (Python)

```
class Counter:

    def __init__(self):
        self.k = 0
    def count(self):
        self.k += 1
    def reset(self):
        self.k = 0
    def getValue(self):
        return self.k
```

Initialization (constructor)

Instance variable k

“Self” parameter is implicit in method calls but explicitly mentioned in declaration

Constructing Objects, Invoking Methods

- Example:

```
c = Counter()
print c.getValue()
c.count()
c.count()
c.count()
print c.getValue()
```

Inheritance in Python

```
class LimitCounter(Counter):

    def __init__(self, limit):
        self.k = 0
        self.limit = limit
    def count(self):
        if self.k != self.limit:
            self.k += 1
```

In contrast to Java, Python allows *multiple inheritance*!

Python Modules

- Module: A file containing Python definitions and statements
  - File name is module name with suffix `.py`
  - Module name is available as global variable `__name__`
  - Statements in a module are executed when the module is imported (initialization)
- Importing a module `m`:

  ```
  import m
  ```

  - Accessing a definition `f()` in `m`:

    ```
    m.f()
    ```

  ```
  from m import *
  ```

  - Accessing a definition `f()` in `m`:

    ```
    f()
    ```

2 Development Platforms for Multimedia Programming

2.1 Introduction to Python

2.2 Multimedia Frameworks for Python 

2.3 Document-Based Platforms: SMIL, OpenLaszlo

2.4 Multimedia Scripting Languages: JavaFX, Processing

2.5 Authoring Tools: Flash

Literature: W. McGugan, Beginning Game Development with Python and Pygame, Apress 2007

Multimedia Frameworks for Python

- Python has a very active open source community
- Frameworks have been developed for many purposes
  - See http://wiki.python.org !
  - GUI frameworks: Tkinter, wxPython
  - Web frameworks: Zope, Django
  - …
- Several frameworks address multimedia issues separately:
  - Python Imaging Library (PIL)
    for decoding, encoding, processing images
  - PyMedia
    for decoding, encoding, playing, analysing audio and video
    (based on ffmpeg)
- *Game development* frameworks comprise all multimedia aspects
  - Pygame: well known and frequently used

History of Pygame

- Sam Lantinga, 1998:
  Simple DirectMedia Layer (SDL) framework, to simplify porting games among platforms
  - Common and simple way to create displays and process input abstracting from platform particularities
  - Originally written in C
- Pygame is a *language binding* for SDL to the Python language
  - Use the SDL library from Python code
- Pygame and SDL are open source projects
  - Constantly being refined
  - Version 1.8.1 (July 2008) is current version
  - www.pygame.org
- Documentation :
  - www.pygame.org/docs
- Pygame is just an ***example*** here! (No “holy cow”!)

Modules in the Pygame Package

`pygame.cdrom`	Controls CD drives
`pygame.cursors`	Loads cursor images
`pygame.display`	Accesses display
`pygame.draw`	2D vector graphics
`pygame.event`	External events
`pygame.font`	Uses System fonts
`pygame.image`	Loads and saves an image
`pygame.joystick`	Special input
`pygame.key`	Keyboard input
`pygame.mixer`	Loads and plays sounds
`pygame.mouse`	Manages mouse
`pygame.movie`	Plays movie files

`pygame.music`	Works with music and streaming audio
`pygame.overlay`	Advanced video overlays
`pygame`	High level functions
`pygame.rect`	Manages areas
`pygame.sndarray`	Manipulates sound data
`pygame.sprite`	Manages moving images
`pygame.surface`	Manages images and the screen
`pygame.surfarray`	Manipulates image pixel data
`pygame.time`	Manages timing and frame rate
`pygame.transform`	Resizes and moves images

Slide Show Example (1)

```
import pygame
from pygame.locals import *
from sys import exit

background = pygame.Color(255,228,95,0)
sc_w = 356
sc_h = 356

pygame.init()

# Create program display area
screen = pygame.display.set_mode((sc_w,sc_h),0,32)
pygame.display.set_caption("Simple Slide Show")

# Set background color by drawing a rectangle
pygame.draw.rect(screen,background,pygame.Rect(0,0,sc_w,sc_h),0)

# Load slide and show it on the screen
slide = pygame.image.load('pics/tiger.jpg').convert()
screen.blit(slide,(50,50))
pygame.display.update()
...
```

Flags

Thickness

Bild auf screen kopieren
(bit block image transfer)

Display Setup

`pygame.display.set_mode(`*`rect`*`,`*`flags`*`,`*`depth`*`)`

- **`Rect:`** Size of the display window (pixels)
- **`Flags:`** Properties of the display which can be switched on/off
  - **`FULLSCREEN`**
  - **`DOUBLEBUF`** Double buffering
  - **`HWSURFACE`** Hardware-accelerated display (must be full screen)
  - **`OPENGL`** OpenGL rendering
  - **`RESIZABLE`**
  - **`NOFRAME`**
- **`Depth:`** Bit depth of display
  - 8: 256 colors
  - 15: 32,768 colors
  - 16: 65,536 colors
  - 24: 16,7 million colors
  - 32 (eight spare bits): 16,7 million colors

Slide Show Example (2)

```
...
pygame.time.wait(4000)

# Load slide and show it on the screen
slide = pygame.image.load('pics/butterfly.jpg').convert()
screen.blit(slide,(50,50))
pygame.display.update()
pygame.time.wait(4000)
...
# Event loop
while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            exit()
```

Interactive Slide Show – Keyboard Control

```
slides = []
slides.append(pygame.image.load('pics/tiger.jpg').convert())
...
slides.append(pygame.image.load('pics/butterfly.jpg').convert())

slideindex = 0

# Event loop
while True:
    newslide = True
    for event in pygame.event.get():
        if event.type == QUIT:
            exit()
        if event.type == pygame.KEYDOWN:
            if event.key in [K_SPACE,K_RIGHT]:
                if slideindex+1 < len(slides):
                    slideindex += 1
                    newslide = True
            if event.key == K_LEFT:
                if slideindex > 0:
                    slideindex -= 1
                    newslide = True
            if event.key == K_q:
                exit()
    if newslide:
        screen.blit(slides[slideindex],(50,50))
        pygame.display.update()
```

List

Key pressed

Individual keys

Event Attributes in Pygame

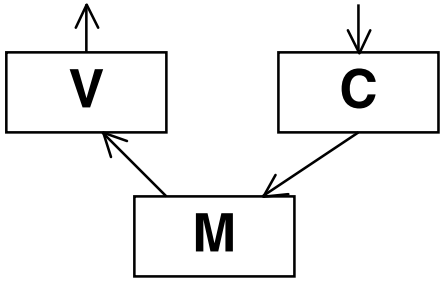
Excerpt from Pygame Documentation:π

> “All Event objects contain an event type identifier in the `Event.type` member. You may also get full access to the Event's member data through the `Event.dict` method. All other member lookups will be passed through to the Event's dictionary values.”

Equivalent expressions (only in this case, not generally in Python!):

```
event.key
event.dict['key']
```

Model-View-Controller Architecture (MVC)



- Model:
  - Domain model of information, as independent of user interface as possible
  - *observable*
- View:
  - Representation of information in user interface
  - Observer of model
  - Inquires (at "update“) required data from model
- Controller:
  - Modifies values in model
  - Is often tied to certain elements of "View" (e.g. Buttons)
  - Reacts to events and creates appropriate method calls to handle events

Using MVC in Python: Main Program/Controller

```
from EventManagement import EventManager
from SlideShowView import *
from SlideShowModel import *

pygame.init()

evmanager = EventManager()
model = SlideShowModel(evmanager)
view = SlideShowView(evmanager,model)

# Event loop
while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            exit()
        if event.type == pygame.KEYDOWN:
            if event.key in [K_SPACE,K_RIGHT]:
                model.forward()
            if event.key == K_LEFT:
                model.backward()
            if event.key == K_q:
                exit()
```

Using MVC in Python: Model

```
import EventManagement
class SlideShowModel:

   def __init__(self,evmanager):
        # Preload slide files
        self.slides = []
        self.slides.append(pygame.image.load('pics/tiger.jpg')
            .convert()) ...
        self.evmanager = evmanager
        self.slideindex = 0

    def currentPicture(self):
        return self.slides[self.slideindex]

    def forward(self):
        if self.slideindex+1 < len(self.slides):
            self.slideindex += 1
            self.evmanager.setChanged()
        self.evmanager.notify()

    def backward(self):
        ...
```

Using MVC in Python: View

```
import EventManagement
import SlideShowModel
import SlideShowView

background = pygame.Color(255,228,95,0)
sc_w = 356
sc_h = 356
screen = pygame.display.set_mode((sc_w,sc_h),0,32)
pygame.display.set_caption("Simple Slide Show")
pygame.draw.rect(screen,background,pygame.Rect(0,0,sc_w,sc_h),0)

class SlideShowView:

    def __init__(self,evmanager,model):
        self.model = model
        evmanager.addObserver(self)
        self.update()

    def update(self):
            image = self.model.currentPicture()
            screen.blit(image,(50,50))
            pygame.display.update()
```

Using MVC in Python: Event Management

```
class EventManager:

    def __init__(self):
        self.observers = []
        self.changed = False

    def addObserver(self,obs):
        self.observers.append(obs)

    def delObserver(self,obs):
        self.observers.remove(obs)

    def setChanged(self):
        self.changed = True

    def notify(self):
        if self.changed:
            for obs in self.observers:
                obs.update()
            self.changed = False
```