Multimedia-Programmierung

Heinrich Hußmann
Ludwig-Maximilians-Universität München
Sommersemester 2010

Deutsch und Englisch

- Im Hauptstudium sind viele aktuelle Materialien nur in englischer Sprache verfügbar.
- Programmiersprachen basieren auf englischem Vokabular.
- Austausch von Materialien zwischen Lehre und Forschung scheitert oft an der deutschen Sprache.
- Konsequenz:
  - Die wichtigsten Lehrmaterialien zu dieser Vorlesung (v.a. Folien) sind in englischer Sprache gehalten!
  - Der Unterricht findet (noch?) in deutscher Sprache statt.

Multimedia Programming

- Multimedia Programming:
  - Creating programs which make use of “rich media” (images, sound, animation, video)
- Key issue in multimedia programming: mixture of skills
  - Programmers are not interested in creative design
  - Designers are intimidated by programming
- Mainstream solution in industry:
  - Multimedia runtime system, plus authoring tool, plus scripts (e.g. Adobe Flash & MS Silverlight)
- Questions (to be covered in this lecture):
  - Which ways exist to bridge between creative design and programming?
    - » Different platforms and tools
    - » Which tool to chose for which purpose?
  - What is the most efficient way of developing multimedia applications?
    - » Which techniques exist to make multimedia programming easier?
    - » What is an adequate development process for multimedia programs?

(Not) Covered Topics

- This lecture does ***not*** cover:
  - Treatment of multimedia data on low system levels (operating system, networks)
  - Production of media products which are consumed in a linear, non-interactive way (like movies)
- The focus of the lecture is on:
  - Graphical representations and (2D-)animation
  - Integration of sound and video into programs
  - Interaction techniques for rich media
  - Development process in teamwork using recent software technologies
- Various example development environments will be covered:
  - Script languages with multimedia frameworks (based on Python)
  - Multimedia scripting languages (JavaFX, Processing)
  - Document-based platforms (SMILE, OpenLaszlo)
  - To a limited extent:
    - » Authoring tools (Flash)
    - » Java multimedia frameworks

Organisatorisches

Ausnahmsweise auf Deutsch:

- Die Lehrveranstaltung (2V+3Ü) ist eine Mischung aus:
  - Vorlesung (12 Doppelstunden)
  - Klassische Übungen (incl. Hausaufgaben)
  - Eigene Freiarbeit

- *Keine Projektphase* wie in früheren Jahren!
  - Siehe Blockpraktikum Multimediaprogrammierung

Scheinkriterien und Bonuspunkte

Diplom:

- Keine Klausur
- Scheinkriterium: Bearbeitung der Übungsblätter (50% der Punkte pro ÜB)
- 2 „Joker“, d.h. zwei Abgaben können gestrichen werden

Bachelor:

- Klausur
- Bearbeiten der Übungsblätter **keine** Klausurvoraussetzung
- Bonuspunkte für Klausur durch Übungsblätter:
  - >75% der Punkte eines ÜBs => 1 Bonuspunkt für Klausur
  - max. 15% Bonus für Klausur

MMP im Nebenfach:

- Trennung zwischen Programmier- und Verständnisaufgaben
- Eine Programmieraufgabe für alle und spezielle wählbare Aufgaben je nach Studium

Um- und Ausbau des Lehrangebots zum Thema

- Vorlesung Multimediaprogrammierung (2 SWS, 4 ECTS)
  - Diese Veranstaltung, Sommersemester
  - Überblick über Techniken der Multimedia-Programmierung
- Übung Multimediaprogrammierung (3 SWS, 2 ECTS)
  - Begleitende Übungen zu dieser Vorlesung, Sommersemester
  - Übungen zur Programmierung mit Python und anderen Programmiersprachen in Kombination mit Multimedia-Frameworks
- Blockpraktikum Multimediaprogrammierung (4 SWS, 6 ECTS)
  - 2-wöchige Blockveranstaltung
  - Intensives Programmierprojekt, voraussichtlich mit Adobe Flash oder Flex
  - Zwischen Sommer- und Wintersemester, voraussichtlich Anfang September
- Für Diplom-Studierende:
  - Bis zu 9 SWS mit diesem Thema möglich (davon aber 7 Übung/Praktikum)
  - Siehe nächste Folie
- Für Bachelor-Studierende:
  - Hauptfach Medieninformatik: Bis zu zwei mal „Vertiefendes Thema“ (jeweils 6 ECTS)
  - Hauptfach Kunst und Multimedia: Teil des Pflichtmoduls „Medienpraxis“
    - » Achtung: Bei Problemen mit den Programmiergrundlagen bitte melden!

Einbringung in Diplom-Studiengänge

Einbringung der erbrachten Leistung im Diplomstudium Informatik oder Medieninformatik:

- Ohne Schein:
  - 2 SWS Prüfungsstoff
- Mit Schein zu den begleitenden Übungen:
  - 5 SWS als Prüfungsstoff (wenn Schein erlangt), davon 3 SWS Übung
  - Alternativ Pflichtschein für MM-Säule (Medieninformatik) + 2 SWS Prüfungsstoff
- Mit Schein zum Blockpraktikum Multimediaprogrammierung:
  - Zusätzlich 4 SWS Prüfungsstoff (Übung/Praktikum)
  - D.h. maximal 9 SWS, davon 7 Übung/Praktikum
  - Für 12-SWS-Prüfung: 2 weitere Vorlesungen (zu je 2 SWS) nötig
  - Für 18-SWS-Prüfung: 3 weitere Vorlesungen (zu je 2 SWS) nötig

Outline (Preliminary)

1. Development Platforms for Multimedia Programming
   - 1.1 Introduction to Python
   - 1.2 Multimedia Frameworks for Python
   - 1.3 Document-Based Platforms: SMIL, OpenLaszlo
   - 1.4 Multimedia Scripting Languages: JavaFX, Processing
   - 1.5 Authoring Tools: Flash
2. Challenges in Multimedia Programming
   - 2.1 Historical Background
   - 2.2 Classification of Development Platforms
   - 2.3 Typical Features of Multimedia Development Platforms
4. Programming with Images
5. Programming with Vector Graphics and Animations
6. Programming with Sound
7. Programming with Video
8. Design Patterns for Multimedia Programs
9. Development Process for Multimedia Projects
10. Modelling of Multimedia Applications

1 Development Platforms for Multimedia Programming

1.1 Introduction to Python

1.2 Multimedia Frameworks for Python

1.3 Document-Based Platforms: SMIL, OpenLaszlo

1.4 Multimedia Scripting Languages: JavaFX, Processing

1.5 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/

Multimedia

- Literal definition: Combination of several media types
- Steinmetz/Nahrstedt 2004:
  “A multimedia document is characterized by information which is coded in at least one continuous (time-dependent) and one discrete (time-independent) medium.”
- Boll 2001:
  “A multimedia document is a media element that forms the composition of continuous and discrete media elements into a logically coherent multimedia unit.”

Observations:

- Multimedia is about *composing* and *integrating* (mono-)media.
- There is a soft borderline between *multimedia applications* and *multimedia documents*.
- A multimedia application/document always has a temporal aspect (i.e. is time-dependent).

Example: Variety of Development Tools

- Example application:
  - Simple slide show, showing a sequence of bitmap pictures (photos)
- Same application behaviour and appearance
- Different development environments
- Will be further analyzed in future lectures…

- First platform (to be used mainly in the tutorials): ***Python + Pygame***
  - Open source software
  - Very simple to use
  - Radically different from industrial mainstream (Flash/Silverlight)

- 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

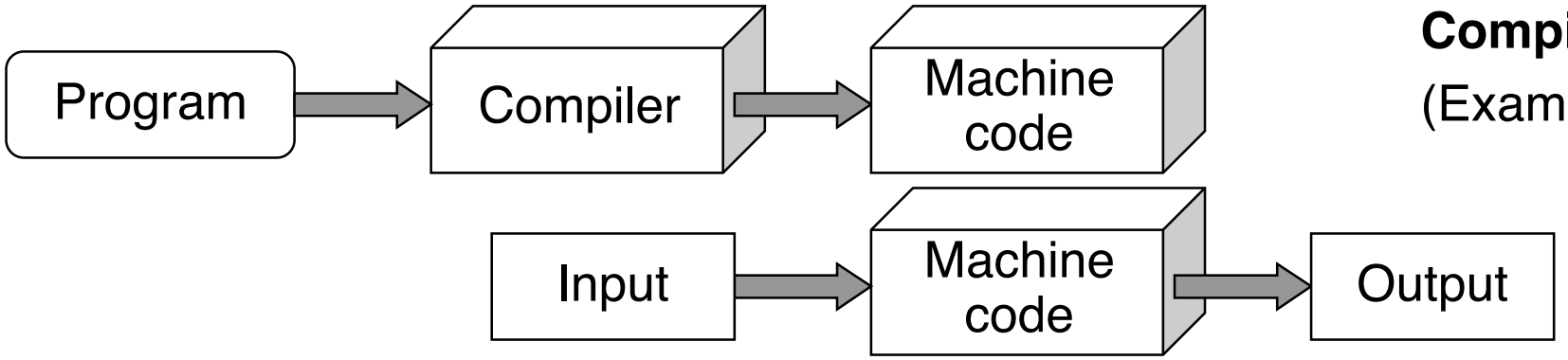
- Indentation (white space) is important for program semantics !!!
  - Block levels given by indentation
  - What is done in Java with {} brackets, is done here with indentation
- Example: A different (wrong!) algorithm:

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

Scripting Language

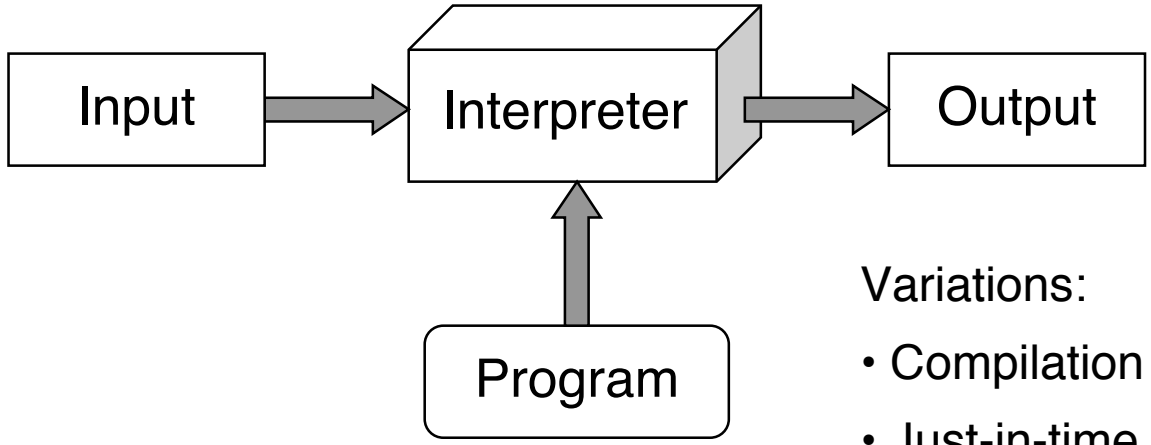
- Traditionally:
  A *scripting language* is a programming language that is used to control some application software
  - Command languages for operating systems (*batch* and *shell* languages)
  - Scripts for task automatisation in user interfaces
  - Scripts executed in Web browsers, word processors, 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)



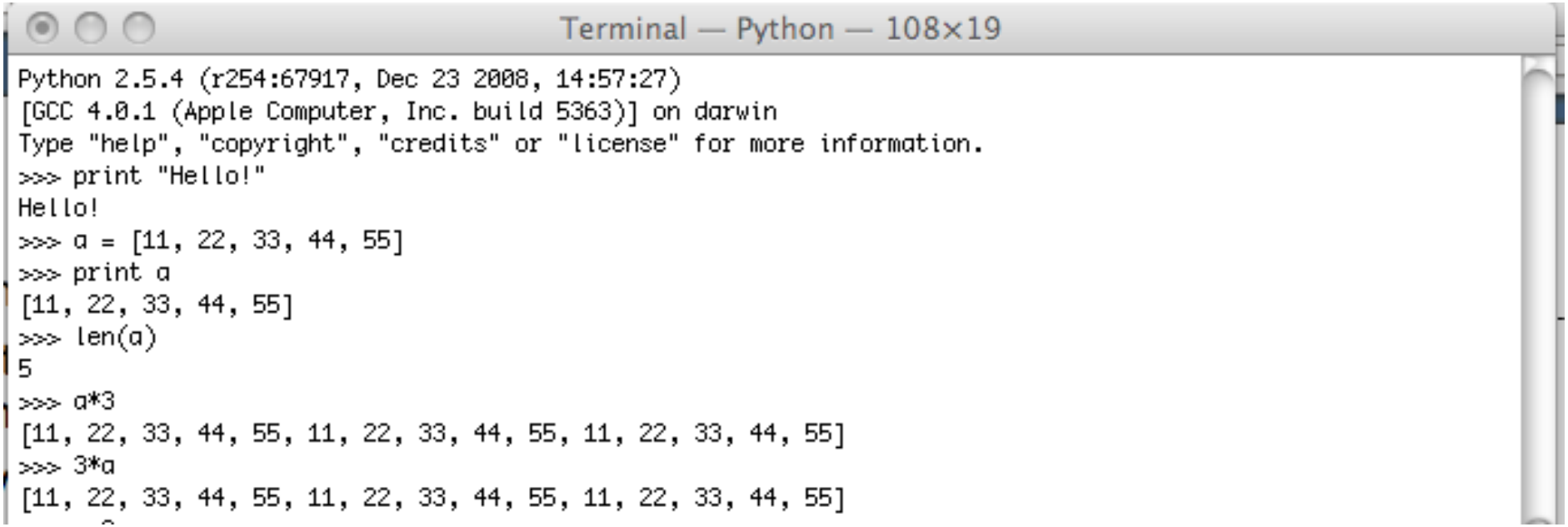
Interpretation

(Example: Python)

Variations:

- Compilation to intermediate code (Java)
- Just-in-time compilation
- Internal compilation to intermediate code in interpreter (Python/ CPython)

Interactive Interpreter



- Interpreted languages can easily be 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 (in certain limited respects)
  - 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 (explicitly or implicitly) 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 valid for all sequence types:

- Indexing: ***`str`*`[5]`** (*str* is the string object)
- 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()`

1 Development Platforms for Multimedia Programming

1.1 Introduction to Python

1.2 Multimedia Frameworks for Python 

1.3 Document-Based Platforms: SMIL, OpenLaszlo

1.4 Multimedia Scripting Languages: JavaFX, Processing

1.5 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 1.9.1 (August 2009) 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
screen.fill(background)

# 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

Copy image to screen
(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()
```

Event Loop

- Why ask for events so late?
  - Premature termination not possible
  - Event loop (for..., ***not*** surrounding while) has to be rpeated elsewhere
    - » Between images
  - Redundant and lengthy code using same code for each individual slide
- Why ask only for QUIT events?
  - There are for instance kes with arrows...
- Next version:
  - Uses generic code to deal with all slides
  - Switches interactively between slides (arrow keys)

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

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()
    newslide = False
```

List

Key pressed

Individual keys

What Will Happen Now in the Lecture

- This will not become a Python / Pygame lecture!
- Lecture:
  - Comparison with other platforms
  - Examples from many platforms (including Flash)
  - Principles transferrable between platforms
- Tutorials:
  - Will go into more detail on Python / Pygame
  - Will do some "excursions" to other platforms
- Practical Course (Blockpraktikum); later:
  - Will concentrate on a different platform (most likely Flash)

References

- S. Boll, “Zyx - towards flexible multimedia document models for reuse and adaptation”. Dissertation Technische Universität Wien, 2001 http://medien.informatik.uni-oldenburg.de/index.php?id=31
- R. Steinmetz, K. Nahrstedt, *Multimedia Applications*, 1st ed. Berlin: Springer 2004