Multimedia-Programmierung
Übung 1

Ludwig-Maximilians-Universität München

Sommersemester 2014

Good to Know

- Informatiker Forum
  http://www.die-informatiker.net/
- Mimuc Twitter Account (inoffiziell)
  http://twitter.com/mimuc
- Medieninformatik LMU Facebook Gruppe (inoffiziell)
  https://www.facebook.com/groups/36775131102/

Übungsbetrieb

- Informationen zu den Übungen: http://www.medien.ifi.lmu.de/mmp
- Anmeldung über Uniworx https://uniworx.ifi.lmu.de/?action=uniworxCourseWelcome&id=277
- Zwei Stunden pro Woche
- Praktische Anwendungen zum Gebiet Multimediaprogrammierung
- Vorbereitung auf die Übungsblätter
- Wöchentliche Übungsblätter
- Spieleprojekt zum Abschluss

Scheinkriterien und Bonuspunkte

Bachelor:

- Klausur
- Bearbeiten der ÜBs **keine** Klausurvoraussetzung
- Keine Bonuspunkte für ÜBs
- Größeres Bonusblatt am Ende des Semesters (bis zu 10% Bonus auf Klausur)

MMP im Nebenfach:

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

Plagiate

Das Abschlussprojekt wird auf Plagiate geprüft
Plagiat führt zum Verlust der Bonuspunkte

Today

What is Python?

- Programming language
- Supports object oriented as well as functional programming
- Fully dynamic type system
- Runs on all major operating systems

- Goal: create a **simple**, **efficient** and **easy-to-learn** programming language

“Wer hat’s erfunden?”
“Die Holländer!"

Guido van Rossum. Programmer of Python.
Source: Doc Searls

For this lecture

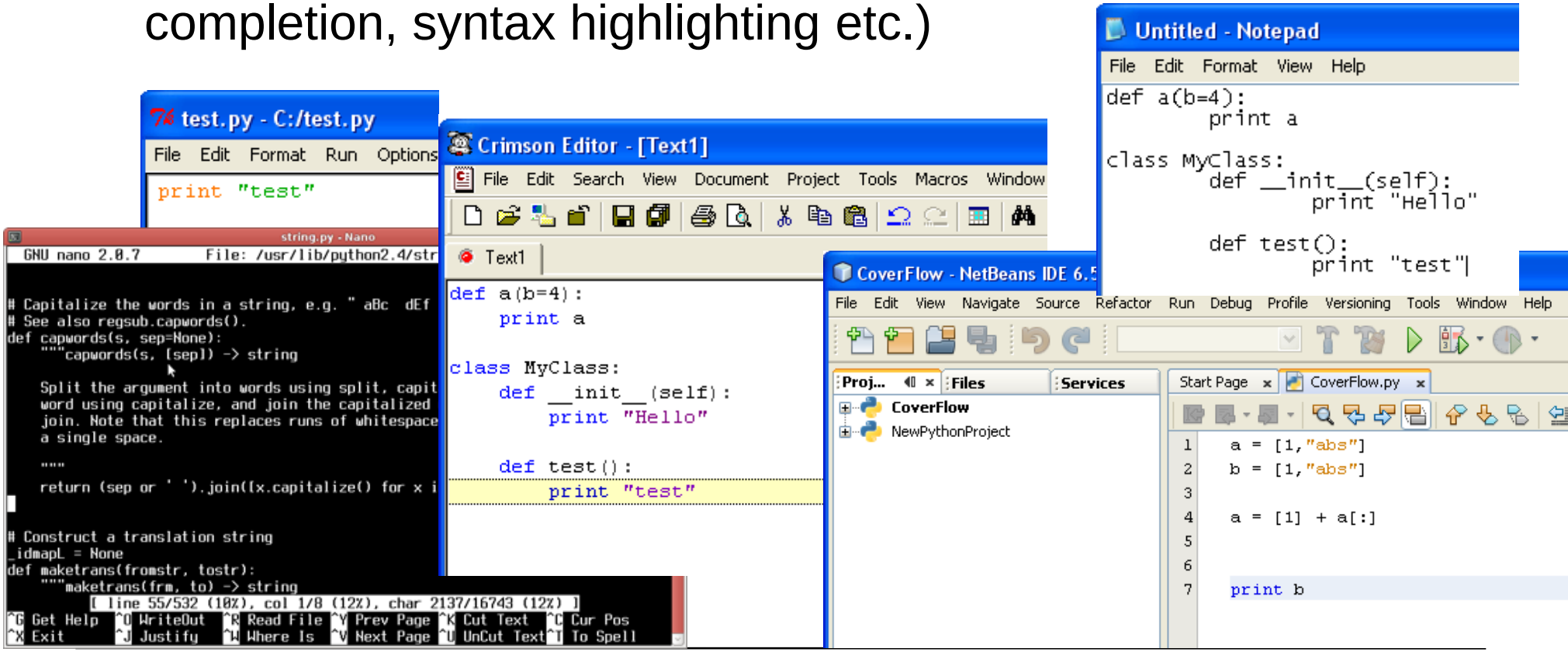
- Python 2.7.6 http://www.python.org/download/
- Pygame 1.9.1 http://www.pygame.org/download.shtml

- Recommended IDE:
  - Netbeans 8.0 (incl. JDK 8)
  - support http://www.oracle.com/

- Installation:
  - Install & start Netbeans (incl. JDK 8)
  - Add Python support: https://blogs.oracle.com/geertjan/entry/python_in_netbeans_ide_71
  - Select all Python plugins and install
  - Choose Tools > Python Platforms > New (Navigate to Python 2.7. Installation path and select e.g. python.exe on Windows)
  - Select Python 2.7. Platform > Make Default

Writing Python Code

- Python scripts are **text files**
- Thus they can be written using **any text editor**
- **IDEs** provide additional support (debugging, code completion, syntax highlighting etc.)



Python code is compact


```
public class Hello {

    public static void main (String args[]) {
        System.out.println("Hello World!");
    }

}
```


```
print “Hello World!“
```

Python code is intuitive


```
String[] a = [“test1“];
String[] b = [“test2“];

String[] c = ArrayUtils.addAll(a, b);

or

String[] a = [“test1“];
String[] b = [“test2“];
String[] c = new String[a.length+b.length];
System.arraycopy(a, 0, c, 0, a.length);
System.arraycopy(b, 0, c, a.length,
b.length);
```


```
a = [“test1“]

b = [“test2“]


c = a + b
```

Python code is fun


```
String a = “test“;

String b = ““;

for(int i = 0; i<5; i++) {
    b = b + a;
}
```

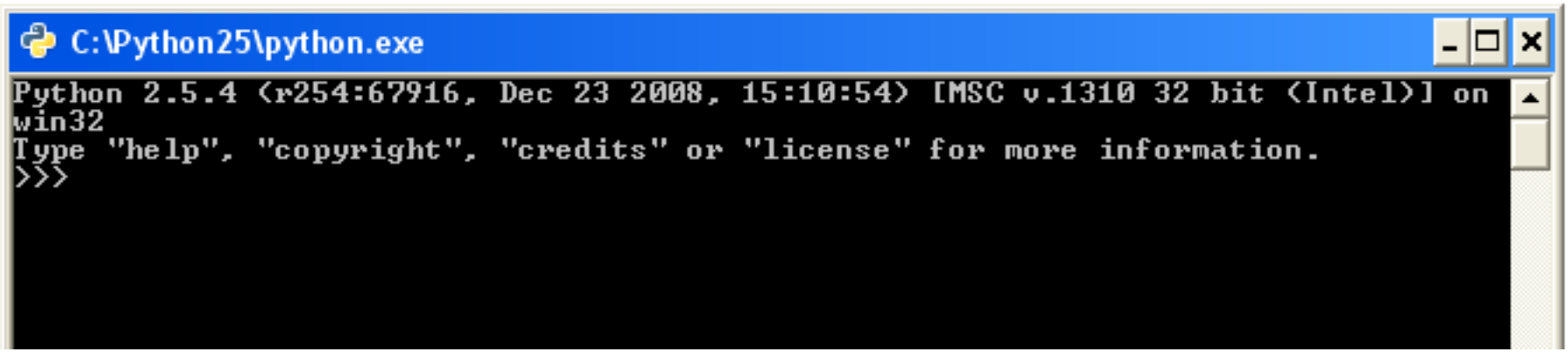

```
a = “test“

b = a * 5
```

Executing Python Code

Interactive Mode

- Lines of Python code can be directly interpreted by the Python interpreter
- Results are immediately visible

- Comes with all standard Python installations
- Mac OS X/Linux: type “python” in the command shell/Terminal
- Windows: e.g. start python.exe from your Python folder



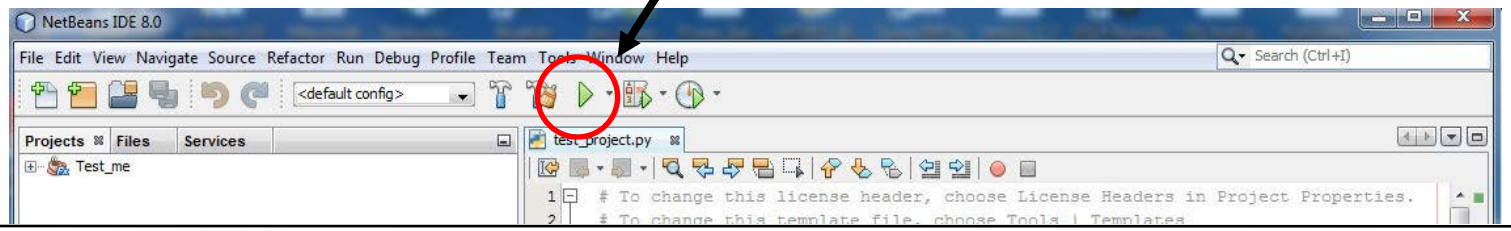
Executing Python Code

Python Scripts

- Python programs are usually called scripts
- Script files end on .py, sometimes .pyw in Windows
- To execute a script use the python interpreter followed by the location of the script

- For example: `python helloworld.py`

- In Netbeans just click the “run” button



Where the %$& § are my delimiters?

- Python does not use special characters as delimiters (e.g. ‘{‘ and ‘}’ in Java)
- Blocks are delimited by indentations/whitespaces

```
a = 1
b = 2

if a > b:
    a = 10
    print a
else:
    a = 100
    print a
```

- editor support recommended
- forces the programmer to write clean and readable code
- a line of code cannot exceed several lines

allowed:

```
a = 1 + 2
```

forbidden:

```
a = 1
+ 2
```

allowed:

```
a = 1 \
+ 2
```

Everything's an Object

with Consequences

Define:

```
def b():
    x = 0
    print x

b()
b = 4
b()
```

Output:

```
0
…
TypeError: 'int' object is not callable
```

id() returns the identifier of the object

is can be used to check whether two objects are the same

Everything's an Object

Types

Define:

```
def b():
    x = 0
    print x

print type(b)
b = 4
print type(b)

print isinstance(b,int)
```

Output:

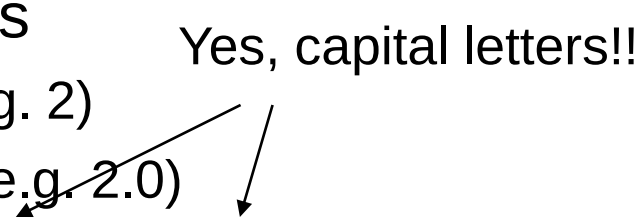
```
<type 'function'>
<type 'int'>
True
```

type() can be used to get the type of an object

isinstance() returns true if an object has a specific type

Types - Examples

- None
  - None
- Numbers
  - int (e.g. 2)
  - float (e.g. 2.0)
  - bool (True and False)
- Sequences
  - str (e.g. “zwei”)
  - tuple (e.g. (1,2))
  - List (e.g. [1,2])
- Callable types
  - functions
  - methods



and many many more …

Comments

or: Being a Good Programmer

```
print "Who stole my Monkey?“ # weird but I‘ll let it in
a = 1
b = 2
print a + b # I hope it‘ll output 3

# print "bye“
```

NebeansTip:

str+shift+c comments the whole selection

Output:

```
Who stole my Monkey?
3
```

Documentation

or: Being a Good Programmer 2

```
def a():
    """This is function a"""
    return 1
print a.__doc__
```

Output:

```
This is function a
```

Functions

Define:

```
def a():
    print "I am function a“

def b(text):
    return "I don‘t like "+text
```

Use:

```
a()
print b("function a")
```

Output:

```
I am function a
I don‘t like function a
```

Functions

Default Parameters

Define:

```
def test(a=1,b=2,c=3):
    print a+b+c

test(1)
test(2,2)
test(c=2)
```

Output:

```
6
7
5
```

Keyword arguments can be used to manipulate specific parameters only.

Namespaces

Local and Global Variables I

Define:

```
def b():
   x = 0
   print x

x = 2

b()
print x
```

Output:

```
0
2
```

Namespaces

Local and Global Variables II

Define:

```
def b():
   global x
   x = 0
   print x

x = 2

b()
print x
```

Output:

```
0
0
```

Namespaces

Local and Global Variables - Episode III

Define:

```
def b():
   x = 0
   print locals()

b()
```

Output:

```
{'x': 0}
```

The functions locals() and globals() can help to get an overview.

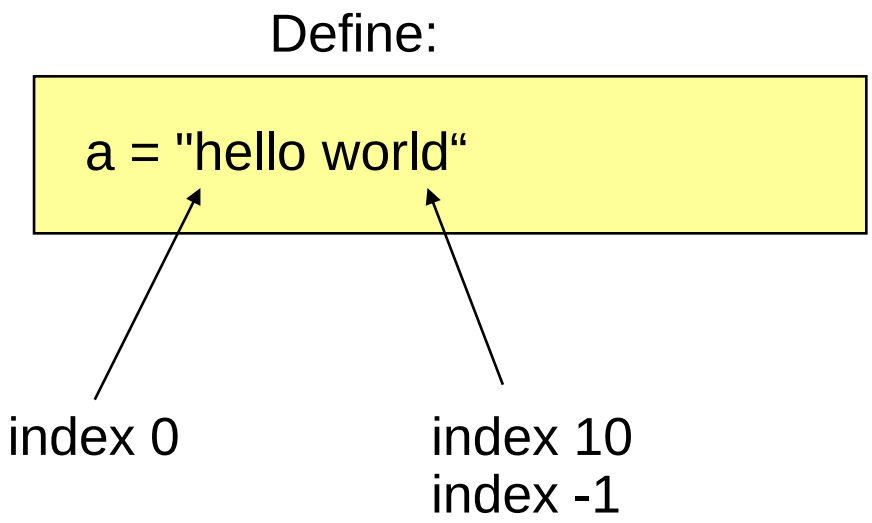
Strings

Range Slice

The range slice notation can be used to access substrings.

string_name[x:y]
x: “from” index starting from 0 (included)

y: “to” index starting from 0 (excluded)



Strings

Examples

Define:

```
a = "hello"
print a[0]
print a[0:]
print a[0:2]
print a[0:len(a)]
print a[2:]
print a[:2]
print a[2:4]
print a[-1]
```

Output:

```
h
hello
he
hello
llo
he
ll
o
```

Attention: strings are immutable!

```
a[2] = "c"
```

```
...
TypeError: 'str' object does
not support item assignment
```

Strings

Formatted Text

Define:

```
print """lalala
test:
        aha"""
```

Output:

```
lalala
test:
        aha
```

Formatted strings are defined using """.

Strings

raw Strings

Define:

```
print "lalala\ntest"
```

Output:

```
lalala
test
```

```
print r"lalala\ntest"
```

```
lalala\ntest
```

Adding an “r” to the string creates a raw string.

Lists a.k.a. Arrays

Define:

```
a = [1,3,"a","b"]
print a
print a[0]

a[0] = 2
print a

print 2 * a
```

Output:

```
[1, 3, 'a', 'b']
1
[2, 3, 'a', 'b']
[2, 3, 'a', 'b',2, 3, 'a', 'b']
```

Lists can contain any types (even mixed).

Dictionaries

Define:

```
priceDict = {'mehl': 99, 'butter': 78}

print priceDict['mehl']
print priceDict.keys()

priceDict['oel'] = 112

print 'oel' in priceDict
```

Output:

```
99
['butter', 'mehl']
True
```

Dictionaries store key-value-pairs.

IF-Statement

Define:

```
a = 0
if a > 0:
    print "a>0"
elif a == 0:
    print "a=0"
else:
    print "none"
```

Output:

```
a=0
```

if...elif...else

Loops

Define:

```
a = [1,3,"a","b"]

for x in a:
    print x

while True:
    print "This will never end. :-s"
```

Don‘t try this at home!

Output:

```
1
3
a
b
This will never end. :-s
...
```

break stops a loop

continue skips to the next part of the loop

Classes

Constructor and Methods

Define:

```
class HelloWorld:
    def __init__(self):
        print "Hello World"

    def test(self):
        print “test"
```

Use:

```
a = HelloWorld()
a.test()
```

Output:

```
Hello World
test
```

Modules

File test.py:

```
def a():
    print "there we are"

def b():
    print "function b"
```

Use:

```
import test

test.a()
```

Or:

```
from test import a

a()
```

Output:

```
there we are
```

Working with Files

Reading Lines

example.txt:

```
line1
line2
cheese cake
cat
```

Open File:

```
file = open("example.txt", "r")
print file.readline()
print file.readline()
file.close()
```

Output:

```
line1
line2
```

open(filename,mode)

mode: ‘r’ for read, ‘w’ for write

‘a’ for append

Working with Files

Iterating all Lines

example.txt:

```
line1
line2
cheese cake
cat
```

Open File:

```
file = open("example.txt", "r")
for line in file:
print line
```

Output:

```
line1
line2
cheese cake
cat
```

Command Line Arguments

Console:

```
python test.py argument1
```

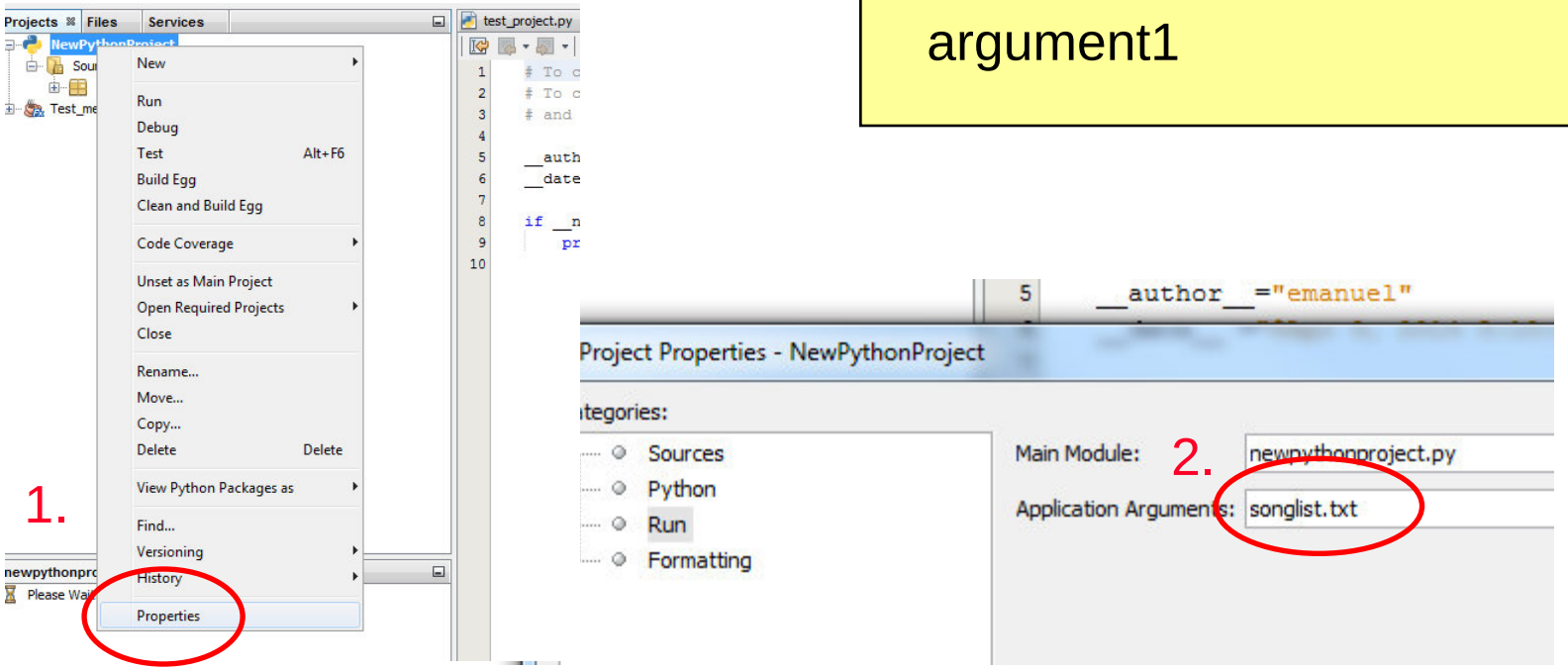
Use:

```
import sys
print sys.argv[1]
```

Output:

```
argument1
```

Netbeans:



Reading Input from the Command Line

Console:

```
a = raw_input("Name:")
```

Output:

```
Name:
```

Waits for user input. If necessary it waits forever. ;-)

input(prompt) is used to get input that is already converted to a type (e.g. an integer)

Useful Links

- Python 2.7.6 documentation
  http://docs.python.org/release/2.7.6/
- Python 2.7.6 tutorial
  http://docs.python.org/release/2.7.6/tutorial/index.html

- File objects
  http://docs.python.org/release/2.7.6/library/stdtypes.html#file-objects

- String methods
  http://docs.python.org/release/2.7.6/library/stdtypes.html#string-methods