

# Multimedia-Programmierung

## Übung 1

Ludwig-Maximilians-Universität München  
Sommersemester 2016

# 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=510>
- Zwei Stunden pro Woche
- Praktische Anwendungen zum Gebiet  
Multimediatechnologie
- Vorbereitung auf die Übungsblätter
- Wöchentliche Übungsblätter
- Spieleprojekt zum Abschluss

# Bonuspunkte und Klausur

## Bewertung:

- Klausur
- Keine Klausurvoraussetzungen, keine Bonuspunkte für Übungsblätter
- Bonuspunkte für Klausur durch Abschlussprojekt gegen Ende der Übungen (max. 10% Bonus für Klausur)

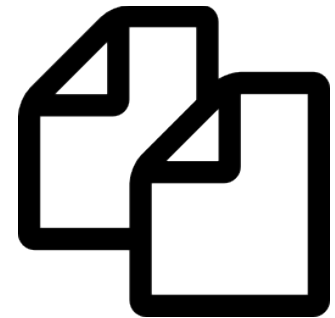
## MMP im Nebenfach (bspw. KuM):

- Angepasste Bewertung bei der Klausur
- Angepasste Projektaufgabe (für Bonuspunkt)

# Plagiate

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

Zudem sind weitere Konsequenzen auf Grund  
des Betrugsversuchs möglich.



# Overview



...Events, Animations, Physics Simulations, Sound...

**Final Project:** Erstes eigenes Spiel!

# Vorläufige Agenda

---

18.04.2016	Overview, Einführung in Python
25.04.2016	Game Design Workshop
02.05.2016	Einführung in PyGame /SDL
09.05.2016	Einführung in C++ / Cocos2d-x
16.05.2016	Pfingsten (keine Übungen)
23.05.2016	Einführung in Cocos2d-x 2
30.05.2016	Animationen in PyGame / Cocos
06.06.2016	Sprites in PyGame / Cocos
13.06.2016	Physics in Cocos / Start Final Project
20.06.2016	Sound in PyGame / Cocos
27.06.2016	TBA
04.07.2016	Klausurvorbereitung

---

# 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 3.5.1 (oder 2.7.6 für Linux) <http://www.python.org/download/>
- Pygame 1.9.2 (oder 1.9.1) <http://www.pygame.org/download.shtml>
  
- Recommended IDEs:
  - Netbeans 8.0 or higher (incl. JDK 8)
  - Eclipse 3.5 or higher
  - Atom
  
- Up-to-date installation recommendations:  
[http://kidscancode.org/blog/2015/09/pygame\\_install/](http://kidscancode.org/blog/2015/09/pygame_install/)

# Installation of Python (3.5.1) and Pygame (1.9.2)

## Example: Windows

### Python:

- ✓ Download Python version, e.g. 2.7.6 oder 3.5.1, from: <https://www.python.org/downloads/>
- ✓ Run the file and select „Add Python 3.5 to PATH“.
- ✓ Click „Customize Installation“ und select every checkbox for “Optional Features“.
- ✓ You do not need to make changes on the „Advanced Options. Run the installation.

### Pygame installieren:

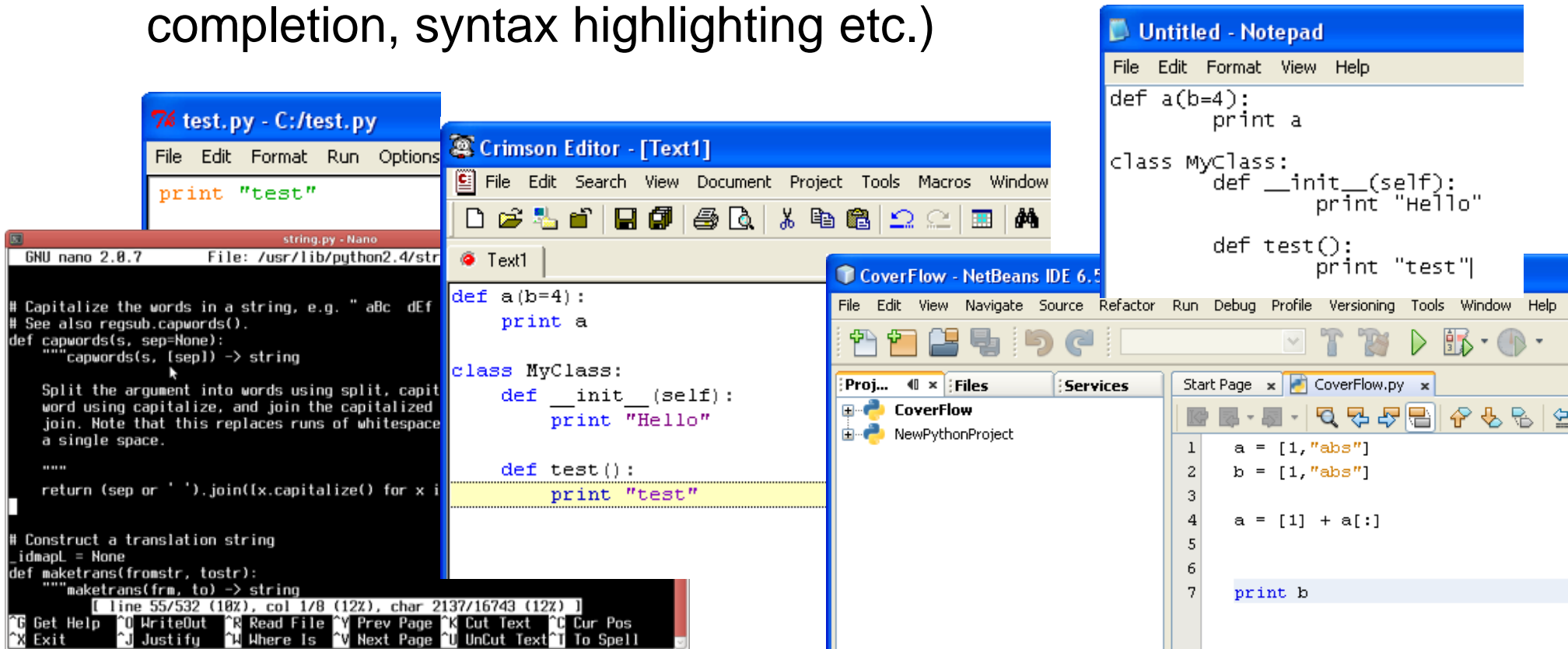
- ✓ Download a suitable insaller from [www.lfd.uci.edu/~gohlke/pythonlibs/#pygame](http://www.lfd.uci.edu/~gohlke/pythonlibs/#pygame)
- ✓ Start the command line (e.g. by Win Key + R → input cmd and press return) and enter
- ✓ `C:\Users\name\> cd Downloads`
- ✓ `C:\Users\name\Downloads> pip install pygame-1.9.2a0-cp35-none-win_amd64.whl`
- ✓ Test the installation with the following command:  
`C:\Users\name\Downloads> python`  
And run pygame  
`>>> import pygame`
- ✓ If you did not receive an error message, the installation worked.

# Python Integration into Eclipse (3.5 or higher)

- More detailed installation manual: [http://www.pydev.org/manual\\_101\\_install.html](http://www.pydev.org/manual_101_install.html)
- Click Help → Install New Software...
- Click Add and enter:  
Name: Pydev and Pydev Extensions  
Location: <http://pydev.org/updates>
- „Pydev“ and „Pydev Mylyn Extension“ should appear now → check „Pydev“ and go on (2x).
- Deselect „Contact all update sites during install to find required software“ and go on.
- Accept the Licence Agreement and go on.
- Restart Eclipse.

# 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.)



The image displays four overlapping windows, each showing Python code in a different editor or IDE:

- test.py - C:/test.py**: A simple text editor window showing `print "test"`.
- Crimson Editor - [Text1]**: A text editor window showing a Python function and class:

```
def a(b=4):  
    print a  
  
class MyClass:  
    def __init__(self):  
        print "Hello"  
  
    def test():  
        print "test"
```
- GNU nano 2.8.7**: A terminal-based text editor window showing a docstring and a function:

```
def capwords(s, sep=None):  
    """capwords(s, [sep]) -> string  
  
    Split the argument into words using split, capit  
word using capitalize, and join the capitalized  
join. Note that this replaces runs of whitespace  
a single space.  
  
    """  
    return (sep or ' ').join([x.capitalize() for x i
```
- CoverFlow - NetBeans IDE 6.5**: An IDE window showing a project structure and a code editor with the same Python code as the Crimson Editor window:

```
def a(b=4):  
    print a  
  
class MyClass:  
    def __init__(self):  
        print "Hello"  
  
    def test():  
        print "test"
```
- Untitled - Notepad**: A text editor window showing a Python function and class:

```
def a(b=4):  
    print a  
  
class MyClass:  
    def __init__(self):  
        print "Hello"  
  
    def test():  
        print "test"
```

# Python – which version should I use? (Probably 3.x)

Aspect	Python 2	Python 3
Print function	<code>print 'Hello, World!'</code>	<code>print('Hello, World!')</code>
Integer division	<code>3 / 2 = 1</code>	<code>3 / 2 = 1.5</code>
Exceptions	<code>raise IOError, "file error"</code>	<code>raise IOError("file error")</code>
Error handling	<code>except NameError, err:</code>	<code>except NameError as err:</code>
Next function	<code>next(my_generator)</code> <code>my_generator.next()</code>	<code>next(my_generator)</code>

## Further improvements:

- for-loop variables don't leak into the global namespace
- Unicode by default
- `input()` stores strings by default

## Sources:

- [http://sebastianraschka.com/Articles/2014\\_python\\_2\\_3\\_key\\_diff.html](http://sebastianraschka.com/Articles/2014_python_2_3_key_diff.html)
- <https://blog.appdynamics.com/devops/the-key-differences-between-python-2-and-python-3/>
- <https://wiki.python.org/moin/Python2orPython3>

# Python code is compact



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



```
print "Hello World!"
```

v. 2.x

```
print ("Hello World!")
```

v. 3.x

# 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;  
}
```

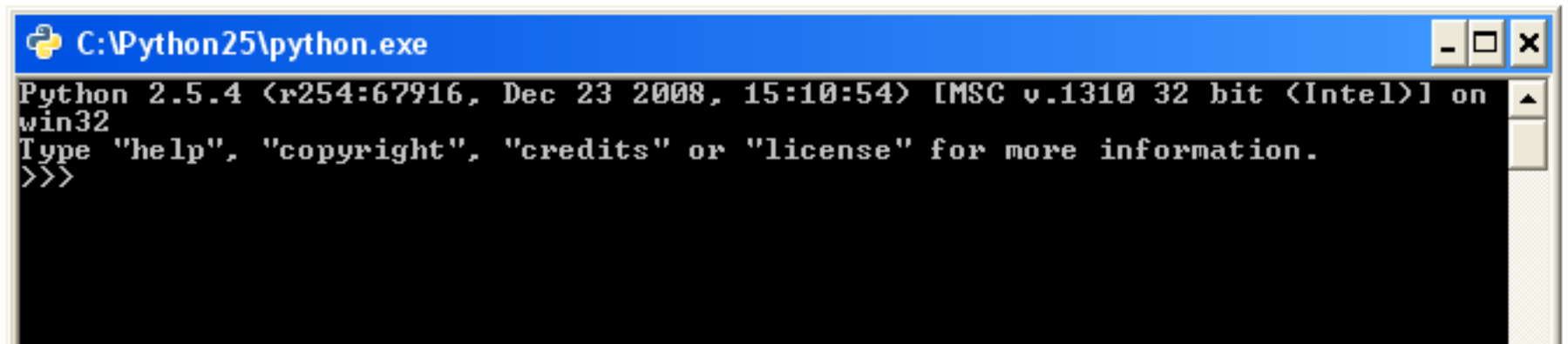


```
a = "test"  
b = a * 5
```

# 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

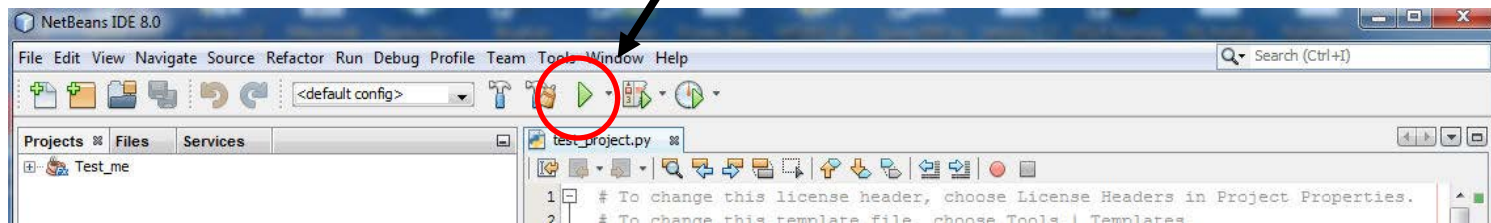


```
C:\Python25\python.exe
Python 2.5.4 (r254:67916, Dec 23 2008, 15:10:54) [MSC v.1310 32 bit (Intel)] on
win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

# 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
```

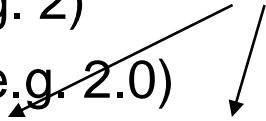
`type()` can be used to get the type of an object

`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 (**T**True and **F**False)
- Sequences
  - str (e.g. "zwei")
  - tuple (e.g. (1,2) )
  - List (e.g. [1,2])
- Callable types
  - functions
  - methods

Yes, capital letters!!



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__
```



“Good  
Boy”

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)

Define:

```
a = "hello world"
```

index 0

index 10

index -1

# 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"
```

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

Output:

```
lalala  
test
```

```
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
```



# Random Module

- The module `random` contains functions to create random numbers, lists etc.
- `randint(a,b)` creates a random number of the interval `[a,b]`
- `random()` creates a random float of the interval `[0.0,1.0]`
- `shuffle(list)` randomly shuffles a list
- Etc.
- Object `Random()` contains all those functions as well

```
import random

test = random.Random()
print test.random()
print random.randint(0,3)
```

# 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
```

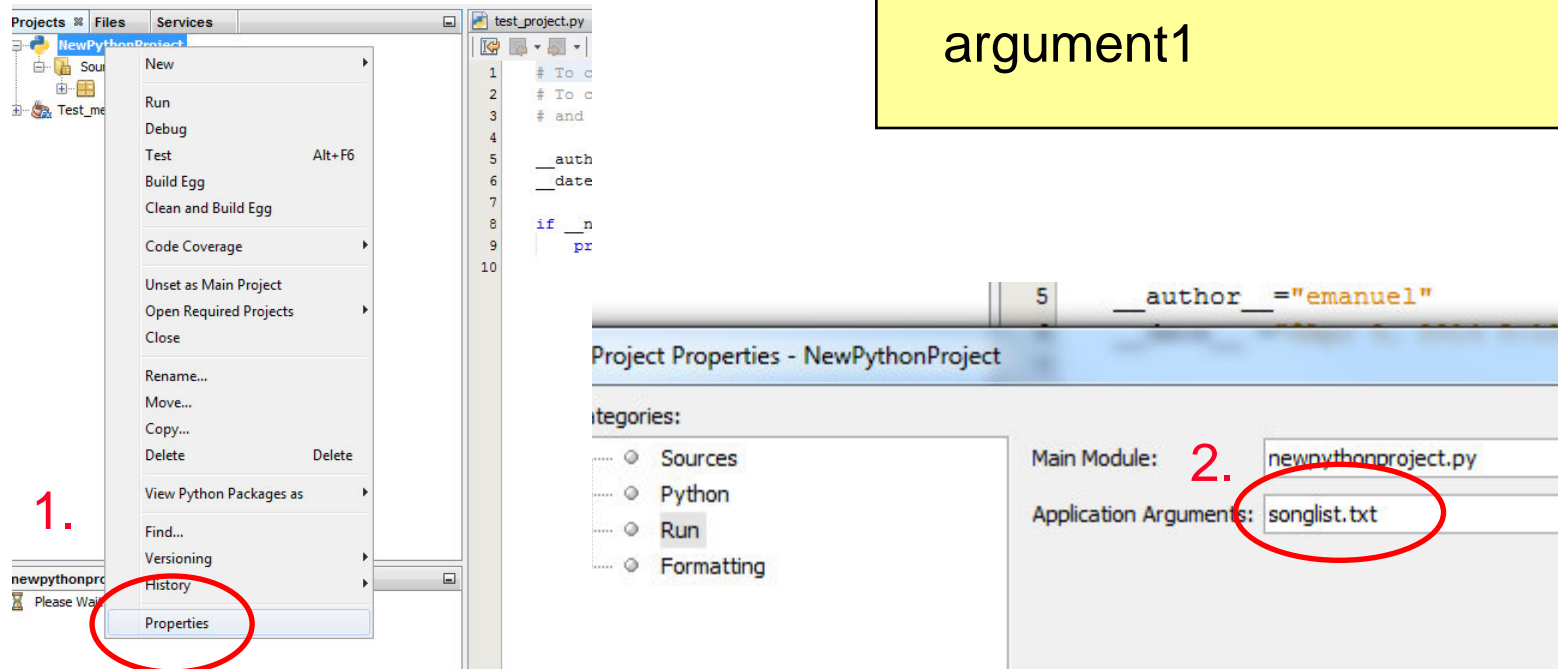
Use:

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

Output:

```
argument1
```

Netbeans:



The screenshot shows the NetBeans IDE interface. On the left, the 'Project Properties' dialog is open for 'NewPythonProject'. The 'Application Arguments' field is set to 'songlist.txt' and is circled in red. A red circle with the number '1.' is around the 'Properties' button in the bottom left. A red circle with the number '2.' is around the 'Main Module' field, which is set to 'newpythonproject.py'. The background shows a code editor with a Python script:

```
1 # To c
2 # To c
3 # and
4
5 __auth
6 __date
7
8 if __n
9
10 pr
```

# 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)

# Exceptions

- Baseclass `BaseException`
- Own exceptions should be extended from class `Exception`
- Exceptions can be raised:

```
raise NameError("unknown name")
```

- `try ... except` to handle exceptions

```
try:  
    test = open("test.txt", "r")  
except IOError:  
    print "file doesn't exist"
```

# Useful Links

- Python:
  - <http://docs.python.org/>
  - <https://docs.python.org/2.7/>
- Tutorials
  - <http://www.learnpython.org>
  - <https://docs.python.org/3/tutorial/>
  - <https://docs.python.org/2/tutorial/>