Introduction to Python

Alastair Burt  Andreas Eisele  Christian Federmann
Torsten Marek  Ulrich Schäfer

Universität des Saarlandes

October 5th, 2009
Today's Topics:

1. Course Overview
2. Overview of Python
3. Data Types & Functions
4. Control Flow
5. Input/Output
6. Regular Expressions
Part I

Course Overview
You will learn how to...

- write small to medium-size programs
- use built-in data types
- process text files
- use object-oriented techniques
- do NLP with the NLTK
You will learn how to...

- write small to medium-size programs
- use built-in data types
- process text files
- use object-oriented techniques
- do NLP with the NLTK

Course Focus

- What can be done, and how
- very little . . .
  - . . . technobabble (red boxes)
  - . . . implementation/machine details
  - . . . comparison to other programming languages
Course Schedule

Monday  - Introduction, Data Structures, Control Flow & I/O
Tuesday - Data Processing
Wednesday - Object-Oriented Programming
Thursday - NLTK
The Daily Routine

- Monday - Thursday: 2 lectures from 14:00 to 15:30 and 16:00 to 16:45
- Tuesday - Thursday: Maria Staudte will help with exercises
Part II

Overview of Python
What is Python?

Author

- Guido van Rossum

Key Features

- executable pseudo-code
- batteries included
- multi-platform
- open source
- interpreted
What is Python?

The Zen of Python, by Tim Peters

1. Beautiful is better than ugly.
2. Explicit is better than implicit.
3. Simple is better than complex.
4. Complex is better than complicated.
5. Flat is better than nested.
6. Sparse is better than dense.
7. Readability counts.
8. Special cases aren’t special enough to break the rules.

...
Running Python Programs

Running A Program with the Interpreter

user@machine:~% python hello.py
Hello, World!
user@machine:~%
Running Python Programs

Running A Program with the Interpreter

```
user@machine:~% python hello.py
Hello, World!
user@machine:~%
```

Using the Python Console

```
user@machine:~% python
Python 2.6.2 (release26-maint, Apr 19 2009, 01:56:41)
[GCC 4.3.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> "Hello, World!"
'Hello, World!'
>>> 1 + 1
2
```

On the console, you can do (almost) everything, like defining new functions, classes, etc.
IPython, An Enhanced Python Console

Additional IPython Features

- Access help for functions via `some_func`?
- Create and store macros
- Conveniently edit already defined functions
- TAB completion
- Enjoy colored output

...and much more
No Dead Code

I will use IPython in this lecture to show code examples live rather than just having them on the slides.
No Dead Code
I will use IPython in this lecture to show code examples live rather than just having them on the slides.

For Reference
All the example code (with comments!) will be available as supplement to the slides. See the homepage for details.
Part III

Data Types & Functions
Natural numbers: `int`

Some facts

- represents an arbitrarily large natural number
- everything you would expect given primary school math

Overflow

There is no integer overflow. Numbers can be as large as your memory affords.
Real numbers: float

Some facts

- representation of real numbers
- approximately 16 digits precision
- literals for scientific notation: 6.673e−11
- everything you would expect given highschool school math
Some facts

- type for boolean logic
- literals True and False
- propositional logic
Numeric Operators

Numeric operators

- Simple mathematical formulas can be used without adaption
- Same precedence rules apply
Numeric Operators

Numeric operators

- Simple mathematical formulas can be used without adaption
- Same precedence rules apply

Special Operators

- \(2 ** 4\) Exponentiation
- \(5 // 2\) Integer division
- \(5 \% 2\) Division Remainder
Operators, continued

Comparison Operators

Expressions with comparison operators return True or False:

- `==`, `!=` check for equality / inequality
- `<`, `<=` less than
- `>=`, `>` greater than

Mixing Integers and Reals

If integers and reals are mixed, the result will always be a real:

```python
>>> 1 / 2
0
>>> 1 / 2.0
0.5
```
Comparison Operators

Expressions with comparison operators return True or False:

- `==`, `!=` check for equality / inequality
- `<`, `<=` less than
- `>=`, `>` greater than

Mixing Integers and Reals

If integers and reals are mixed, the result will always be a real:

```python
>>> 1 / 2
0
>>> 1 / 2.0
0.5
```
Boolean Operators

Propositional Logic, anyone?

\[
\begin{align*}
  a \text{ and } b & \quad \text{boolean and} \\
  a \text{ or } b & \quad \text{boolean or} \\
  \text{not } a & \quad \text{boolean negation}
\end{align*}
\]
Propositional Logic, anyone?

\[
\begin{align*}
\text{a and b} & \quad \text{boolean and} \\
\text{a or b} & \quad \text{boolean or} \\
\text{not a} & \quad \text{boolean negation}
\end{align*}
\]

Short-Circuit Evaluation

Boolean operators are short-circuiting, i.e. evaluation of an expression will stop as soon as its final value is known.
A New Variable

Storing a Value in a Name

To declare a new variable `<name>`, just write

```
<name> = <expression>
```
A New Variable

Storing a Value in a Name

To declare a new variable `<name>`, just write

```
<name> = <expression>
```

References

All variables are references to objects. They can be freely introduced and deleted:

```
>>> a = 0
>>> a
0
>>> del a
>>> a
NameError: name 'a' is not defined
```
Storing texts

Texts are stored in a data structure called “string”, a list of characters.

legacy strings : str
- only English script supported

Unicode strings: unicode
- support for all larger scripts/alphabets in the world
- more about text encodings later
Strings literals

**Strings: str**

```python
hello = "Hello!"
```
Strings literals

Strings: `str`

```python
hello = "Hello!"
```

More than just A-Z: `unicode`

```python
goodbye = u"Hallole"
```
Access to Single Characters

Single characters can be accessed with []:

```python
>>> hello[0]
"H"
```
Operations on Strings

Access to Single Characters

Single characters can be accessed with []:

```python
>>> hello[0]
"H"
```

Substrings

Substrings can be extracted using the slicing syntax:

```python
>>> hello[1:4]
"ell"
```
String Operators

Operators for Strings

+ Concatenation
* Repetition
== != Equivalence, Non-equivalence
< > >= <= Ordering
in Containment
Determining Length

The length of a string is determined with the function `len`:

```python
>>> len("abc")
3
```
Determining Length

The length of a string is determined with the function `len`:

```python
>>> len("abc")
3
```

Function Calls

A function call has the following syntax (note the parentheses):

```python
<function-name>(<arguments>)
```

If a function takes several arguments, they are separated by commas.
Methods, Anyone?

Object Primer

Strings are not just a list of characters, they also define some functions specific to strings.

- functions defined by objects are called methods
- methods are bound to objects (and their data)
Methods, Anyone?

Object Primer

Strings are not just a list of characters, they also define some functions specific to strings.

- functions defined by objects are called methods
- methods are bound to objects (and their data)

Calling Methods

To find out if a string starts with a certain prefix, we use the object method `str.startswith`:

```python
>>> hello.startswith("Hell")
True
```
More String Methods

**String Methods**

- `tolower` Case conversion
- `split` Splitting strings
- `find` Exact matching in strings
- `replace` Replacing substrings
  ...
list Primer

A list is a collection of one or more elements.
**list Primer**

A list is a collection of one or more elements.

**list Literals**

```python
counts = [1, 2, 3, 4, 5, 6]
strings = ["do", "re", "mi"]
lists = [[1, 2], [3, 4], [5, 6]]
```
List Operations

Lists and Strings are Sequences

- indexed element access with []
- operators for concatenation, repetition, equivalence
- `len` function
List Operations

Lists and Strings are Sequences

- indexed element access with []
- operators for concatenation, repetition, equivalence
- len function

Lists are Mutable

- insertion, deletion and changing of elements
- in-place sorting and reversal
More About Slicing

Slicing Syntax

Accessing a slice of a sequence has the following syntax:

\[
\text{<seq>[<start>:<end>:<step>]}\]

Explanations

- **<start>**: the start index of the slice, included
- **<end>**: the end index, excluded
- **<step>**: the increment
More About Slicing

Slicing Syntax

Accessing a slice of a sequence has the following syntax:

`<seq>[<start>:<end>:<step>]`

Explanations

- `<start>` the start index of the slice, included
- `<end>` the end index, excluded
- `<step>` the increment
More about Slicing

- negative indices will be interpreted relative to the end of a sequence
- the step part is optional, default is 1
- start and stop can be omitted. Default values:
  
  ```
  start : 0
  stop  : len(seq)-1
  ```
- slicing always returns a copy
- assignments can be made to slices
Slicing Examples

With Slices, You Can...

- create a copy of a list
- delete parts of a list
- get a reversed copy of a list
- create a list that only contains every second, third etc element of another list
Slicing Examples

With Slices, You Can...

- create a copy of a list
- delete parts of a list
- get a reversed copy of a list
- create a list that only contains every second, third etc element of another list

The range Function

The function range(start, end, step) returns the list that contains all numbers from start to end (excluded). The step argument is optional.
Overview

A tuple is a light-weight record object that bundles several related objects with different types and semantics:

```
person = ("Fry", ["Philip", "J."], 25)
```
Overview

A tuple is a light-weight record object that bundles several related objects with different types and semantics:

```
person = ("Fry", ["Philip", "J."], 25)
```

Field Contents

- 0 Surname
- 1 List of forenames
- 2 Age
Lists vs Tuples

**Lists**
- arbitrarily large
- mutable
- new items can be added to the end without much overhead
- elements are assumed to behave similar

**Tuples**
- no size restrictions, but usually short
- immutable
- adding items requires a new tuple
- heterogenous elements
## Lists vs Tuples

### Lists
- arbitrarily large
- mutable
- new items can be added to the end without much overhead
- elements are assumed to behave similar

### Tuples
- no size restrictions, but usually short
- immutable
- adding items requires a new tuple
- heterogenous elements

### Some Remarks
- Homogeneous ≠ same type
dict Primer

Overview

A dictionary is an associative array where each element is accessed by a name.
dict Primer

Overview

A dictionary is an associative array where each element is accessed by a name.

dict Literals

```python
>>> word_frequencies = {
    "good": 1,
    "news": 2,
    "everyone": 1,
    "I": 1,
    "have": 1,
    "bad": 1
}

>>> word_frequencies["news"]
2
```
Dictionary Operations

Element Access

- named element access with []
- containment check with in
Dictionary Operations

Element Access

- named element access with []
- containment check with in

Keys, Values and Items

Dictionaries have methods to get the list of all keys, values or items.
So... What About Dictionaries?

For wordlists

- frequency counts
- probabilities
- synonyms
So... What About Dictionaries?

For wordlists

- frequency counts
- probabilities
- synonyms

...as Records

Instead of tuples, use dictionaries with “known” keys.
So... What About Dictionaries?

For wordlists

- frequency counts
- probabilities
- synonyms

...as Records

Instead of tuples, use dictionaries with “known” keys.

Efficiency

Dictionaries are used throughout Python and are said to be one of the fastest existing general hash map implementations.
set Primer

set

A datatype for sets:

```python
>>> a = set([1, 2, 3, 4, 6])
>>> 1 in a
True
>>> 5 in a
False
```
set Primer

set

A datatype for sets:

>>> a = set([1, 2, 3, 4, 6])
>>> 1 in a
True
>>> 5 in a
False

Operations

- intersection, union, difference
- subset, superset
More on sets

- only one copy of equivalent elements in a set
- all elements must be immutable
- sets are not ordered
  function sorted can be used to get items in order
Part IV

Control Flow
A Simple Algorithm

The problem

For a given number $x$, find the (integer) logarithm $l$ to the base 10. So the following holds for $l$:

$$10^l \leq x < 10^{l+1} \quad (1)$$
A Simple Algorithm

The problem

For a given number $x$, find the (integer) logarithm $l$ to the base 10. So the following holds for $l$:

$$10^l \leq x < 10^{l+1}$$ (1)

Highlevel Solution

Count the number of times $x$ can be divided by 10 until $x < 10$. 
The Algorithm

```python
x = 123
l = 0
while x >= 10:
    l = l + 1
    x = x // 10
print l
```

Explanations

1-2 variable initializations
3-5 block statement (indented with 4 spaces)
6 result output
Block statements

The Algorithm

```python
x = 123
l = 0
while x >= 10:
    l = l + 1
    x = x // 10
print l
```

Explanations

1-2 variable initializations
Block statements

The Algorithm

```python
x = 123
l = 0
while x >= 10:
    l = l + 1
    x = x // 10
print l
```

Explanations

1-2 variable initializations
3-5 block statement (indented with 4 spaces)
The Algorithm

```python
x = 123
l = 0
while x >= 10:
    l = l + 1
    x = x // 10
print l
```

Explanations

1-2 variable initializations
3-5 block statement (indented with 4 spaces)
6 result output
What If?

Conditional execute with `if`

Branch on the result of a boolean expression:

```python
if a != 0:
    print "a != 0"
else:
    print "a == 0"
```
Nested ifs?

It is not possible to put else and if onto one line - one "physical" line may only contain exactly one clause *header.*
Nested ifs?

It is not possible to put else and if onto one line - one "physical" line may only contain exactly one clause header.

The solution: `elif`

```python
if a < 0:
    ...
elif a == 0:
    ...
else:
    ...
```
## Boolean Contexts

### Variables in Boolean Expressions

As shorthands, built-in types can also be evaluated in a boolean contexts.

<table>
<thead>
<tr>
<th></th>
<th>True</th>
<th>False</th>
</tr>
</thead>
<tbody>
<tr>
<td>numbers</td>
<td><code>a != 0</code></td>
<td><code>a == 0</code></td>
</tr>
<tr>
<td>containers</td>
<td><code>len(a) &gt; 0</code></td>
<td><code>len(a) == 0</code></td>
</tr>
<tr>
<td>variables</td>
<td><code>a is None</code></td>
<td><code>a is not None</code></td>
</tr>
</tbody>
</table>
Loops for containers

For each element in a container, execute a given suite of statements.
for Loops

Loops for containers

For each element in a container, execute a given suite of statements.

Loop Syntax

```python
phrase = ["colorless", "green", "ideas"]
l = 0
for word in phrase:
    l = l + len(word)
```
More About For Loops

But for Used to Be about Numbers, Right?
Remember `range(start, stop, step)`?
More About For Loops

But for Used to Be about Numbers, Right?

Remember \texttt{range(\texttt{start}, \texttt{stop}, \texttt{step})}?

More to Come

We will revisit our example tomorrow and find out how to express it more elegantly.
Controlling Execution

Escaping

break statements cause innermost loop to be exited:

```python
for x in range(10):
    if x != 5:
        print x
    else:
        break
```
Controlling Execution

Escaping

break statements cause innermost loop to be exited:

```
for x in range(10):
    if x != 5:
        print x
    else:
        break
```

Discarding iterations

continue statements can be used to discard the current iteration and continue with the next one:

```
for x in words:
    if words[0].isdigit():
        continue
```
Functions: Motivation

Reusable code

- we don’t want to write out a complete algorithm every time
- provide code to others
- DRY: Don’t repeat yourself
Function Declarations

Functions are declared with the def keyword:

```python
def print_hello():
    print "Hello, World!"
```
Function Declarations

Functions are declared with the `def` keyword:

```python
def print_hello():
    print "Hello, World!"
```

Invoking Functions

No difference to built-in functions:

```python
>>> print_hello()
"Hello, World!"
```
Return Values

Mathematical Definition

A function $f$ takes a value $v$ from the domain and assigns to it a value from the range.
Return Values

Mathematical Definition

A function $f$ takes a value $v$ from the domain and assigns to it a value from the range.

\[ \log_{10} \]

```python
def log10(x):
    l = 0
    while x >= 10:
        x = x // 10
        l = l + 1
    return l
```
Argument Variations

Motivation

log algorithm should take the base, but default to 10.
Motivation

log algorithm should take the base, but default to 10.

\[ \log_b \]

```python
def log10(x, base=10):
    l = 0
    while x >= base:
        x = x // base
        l = l + 1
    return l
```
Multiple Return Values

... Are Quite Easy!

To return multiple values from a function, tuples are used:

```python
def first_last(string):
    return (string[0], string[-1])
```
Motivation

- `log` is not defined for number $\leq 0$
- our implementation just returns 0
- calling `log` with a negative number might be a bug in a program
- base *must* be a positive integer
- never swallow errors
  ... without a good reason
def log(x, base=10):
    if x <= 0:
        raise ValueError("log is only defined positive numbers")
    if base < 1:
        raise ValueError("base must be a positive integer")
    ...

Report an Error: Exceptions
"Catching" Errors

Error handling is done with try...except blocks:

```
try:
    log(-5)
except ValueError:
    print "negative value caused error."
```
Handling Errors

“Catching” Errors

Error handling is done with `try...except` blocks:

```
try:
    log(-5)
except ValueError:
    print "negative value caused error."
```

Exception Behavior

- exceptions cause functions to exit immediately
- if no handler is found, program execution is stopped with a nasty error message
Part V

Input/Output
Printing values

The print Statement

All built-ins can be printed using the print statement:

```python
>>> a = [1,2,3]
>>> print a
[1, 2, 3]
```
Printing values

The `print` Statement

All built-ins can be printed using the `print` statement:

```python
>>> a = [1,2,3]
>>> print a
[1, 2, 3]
```

Printing Multiple Values

```python
>>> print "Result: ", 42, "kg"
"Result: 42 kg"
```

Gets quite unreadable for more complex output
Built-in String Formatting

The String Format Operator

The `%` operator takes a string with format specifiers and an argument tuple:

```python
>>> print "Word: %s, frequency: %i" % ("green", 1)
"Word: green, frequency: 1"
```
Built-in String Formatting

The String Format Operator

The `%` operator takes a string with format specifiers and an argument tuple:

```python
>>> print "Word: %s, frequency: %i" % ("green", 1)
"Word: green, frequency: 1"
```

Format Syntax

- `%s` strings
- `%i` integer number
- `%.2f` real with two significant digits after the point
File I/O in Python

The file type

- Files are built-in objects
- Supports character- and line-based reading
- Supports reading binary files
- Encoded files with the codecs module
Reading

Opening

Opening a file is done with the "r" flag:

```python
infile = open("some_file", "r")
```
Opening

Opening a file is done with the "r" flag:

```python
infile = open("some_file", "r")
```

File Objects Support Iteration

Printing all lines in a file:

```python
for line in open("some_file", "r"):
    print line
```
Writing

Opening

To open a file for output, the "w" flag is used:

```python
out = open("some_file", "w")
```
Opening

To open a file for output, the "w" flag is used:

```python
out = open("some_file", "w")
```

Writing Values

Strings can be written to files with the `write` method. After use, files must be closed so that the content is actually written.

```python
out.write("some text")
out.close()
```
It’s a mess, but it’s getting better

1967  ASCII
- 7 bit, 128 characters
- English alphabet only
It’s a mess, but it’s getting better

1967 ASCII
- 7 bit, 128 characters
- English alphabet only

70s-00s Many standards
- usually 8 bit, 256 characters
- language-dependant
**Encoding**

It's a mess, but it's getting better

1967  ASCII
- 7 bit, 128 characters
- English alphabet only

70s-00s  Many standards
- usually 8 bit, 256 characters
- language-dependant

1991  Unicode
- several encodings
- most popular: UTF-8
- one character set for all languages
### The First Module

#### Modules

To read encoded files, we need to use a module:

```python
import codecs
```

#### About Modules

- add functionality
- new types
- new functions
- need to be added explicitly with `import` statements
The First Module

Modules

To read encoded files, we need to use a module:

```python
import codecs
```

About Modules

- add functionality
- new types
- new functions
- need to be added explicitly with `import` statements
Opening Encoded Files

Different function for opening, ...

```
infile = codecs.open("some_file", "r", "UTF-8")
```
Opening Encoded Files

Different function for opening, ...

infile = codecs.open("some_file", "r", "UTF-8")

... same behavior

- same methods
- can be iterated
- ... but returns Unicode strings
Part VI

Regular Expressions
The Shortest Possible Regex Intro

What are Regexes?

Special syntax to describe (regular) languages/patterns:

- `?`: 0 or one, `*`: 0 or more, `+`: one or more occurrences
- `^`: negation, `{n}`: `n`, `{m, n}`: between `m` and `n` times
- `.`: any (single) character, `\s`: whitespace
- all one-digit integers: `\d` (equivalent to `[0–9]`
- all non-digit characters: `\D` (equivalent to `[~0–9]`
- either A or B or c: `A|B|c`
- all English vowels: `[aeiou]`
The Shortest Possible Regex Intro

What are Regexes?

Special syntax to describe (regular) languages/patterns:

- `?`: 0 or one, `*`: 0 or more, `+`: one or more occurrences
- `^`: negation, `{n}`: `n`, `{m,n}`: between `m` and `n` times
- `.`: any (single) character, `\s`: whitespace
- all one-digit integers: `\d` (equivalent to `[0-9]`)
- all non-digit characters: `\D` (equivalent to `[\^0-9]`)
- either A or B or c: `A|B|c`
- all English vowels: `[aeiou]`

Pattern Matching

- extract patterns: dates, addresses, etc
- useful for simple parsing tasks
Regular Expressions in Python

Some Facts

- regular expressions have no built-in syntax
- largely compatible to Perl, i.e. Python regexes will work in Perl
- Unicode support
Using Regular Expressions

Basic Usage

```python
import re
pp = re.compile(ur"[0-9]+\.[0-9]{2}\s*EUR")
pp.findall('0.99 EUR, 56 EUR, or 109.75 EUR')
['0.99 EUR', '109.75 EUR']
```
Using Regular Expressions

Basic Usage

```python
import re
pp = re.compile(ur"[0-9]+\.[0-9]{2}\s*EUR")
pp.findall('0.99 EUR, 56 EUR, or 109.75 EUR')
['0.99 EUR', '109.75 EUR']
```

Explanations

- preceding `r` is needed for simple backslashes
- compilation of regular expressions is optional, but recommended
Searching

Searching Functions

- `match` only finds a match at the beginning of the string
- `search` scans the whole string
- `findall` returns a list of all matches in a string
- `finditer` returns an iterator of all matches
Searching

Searching Functions

- `match` only finds a match at the beginning of the string
- `search` scans the whole string
- `findall` returns a list all matches in a string
- `finditer` returns an iterator of all matches

Other Functions

- `split` uses the regex as a splitter
- `sub`, `subn` substitution patterns
Groups

Instead of referring to groups in regular expressions via their index, it is also possible to name groups with the following syntax:

```python
(?P<number>[0-9]+)
```
Named Groups

Groups

Instead of referring to groups in regular expressions via their index, it is also possible to name groups with the following syntax:

```
(?P<number>[0-9]+)
```

Using Named Groups

```python
>>> p = re.compile(r"(?P<number>[0-9]+)"
>>> p.match("42").groupdict()
{"number": "42"}
```
A Word of Caution

Calling to The Authorities

Some people, when confronted with a problem, think “I know, I’ll use regular expressions.” Now they have two problems.
– Jamie Zawinski, in comp.lang.emacs
A Word of Caution

Calling to The Authorities

Some people, when confronted with a problem, think “I know, I’ll use regular expressions.” Now they have two problems.
– Jamie Zawinski, in comp.lang.emacs

Advice

For a lot of tasks, it is possible to use string functions instead of using regular expressions, which are faster and more readable.
Exercises

Tomorrow

1. Look at today’s example code, experiment with it
2. Get the exercise sheet
3. Try to solve the exercises
Exercises

Tomorrow

1. Look at today’s example code, experiment with it
2. Get the exercise sheet
3. Try to solve the exercises

Exercises

- Designed to scale up to different levels of previous knowledge
- Ordered by increasing difficulty, so try to solve them in order