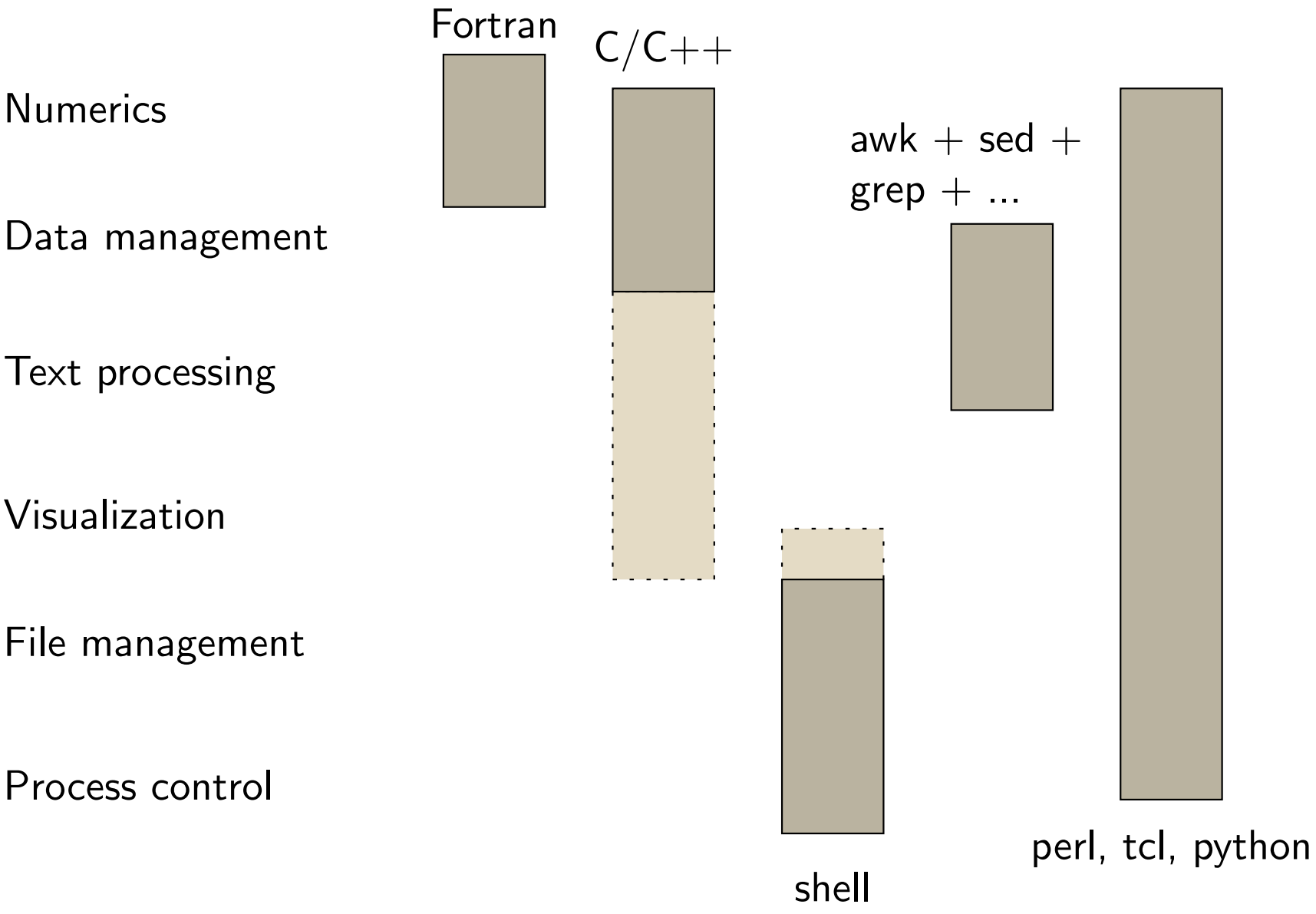
The Informal Python Boot Camp
Lesson 1

1. **Zoology of programming languages**
2. **Reptile show**
3. **Primitives**
    - 3.1 **Numbers**
    - 3.2 **Strings**
    - 3.3 **Lists**

What the average geophysicist does with a computer

- Numerics
- Data management
- Text processing
- Visualization
- File management
- Process control

Different tasks - different tools



The Unix philosophy

Write tools that do one thing and do it well.
Write tools to work together.

- Small programs tied together with shell scripts.
- Library functions tied together by a short main program.
- Library functions and external programs tied together by a high level script.

Some popular high level script/programming languages

- **Lisp** (traditional)
- **Perl** (1987)
- **Tcl** (1988)
- **Python** (1991)
- **Dylan** (early 1990s)
- **Java** (1995)
- **Ruby** (1995)

Why are these languages so popular? (1)

Less code = less errors

And faster development, quicker understanding, faster typing, faster finding errors, better overview, easier to modify.

Why are these languages so popular? (2)

More like brain language

Easier to develop, understand, maintain, remember, ...

Why are these languages so popular? (3)

No (separate) compilation step

- No compiler problems
- No makefiles
- No linker problems
- Faster development cycles

Why are these languages so popular? (4)

Great standard libraries included

- Platform independence
- One place to look first for a proven solution
- Reuse instead of reinvent

Inventor of Python

Guido van Rossum

`http://www.python.org/~guido/`

The Zen of Python, by Tim Peters

```
% python
>>> import this
```

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren’t special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
...

Example 1: See if there is Bratwurst in the mensa today

STUDIERENDENWERK HAMBURG

Speisekarte

	Mittwoch, 05.09.2007	Preise**	
		Studierende	übrige Gäste
Beliebt und gerne gegessen	**Indisches Tandoori Chicken (1,20) mit Krautsalat (3,5) und Fladenbrot als Beilage (14)**	**1,80 €**	**2,75 €**
Die Alternative	**Ägyptische Falafel auf Minz-Joghurt-Soße (20) dazu Reis und Krautsalat (3,5)**	**2,00 €**	**3,00 €**
Campus Spezial	**Lammkeule in Thymian-Rotweinsoße (1,4,11,14,16,20,22,23) dazu Princessböhnchen natur und Kartoffeltaler (1,14,16,19,20,22)**	**3,35 €**	**3,95 €**

= Fleischloses Gericht

= mit Schweinefleisch

= mit Alkohol

*) Bio-Gericht aus biologischem Anbau bzw. Aufzucht

Example 1: See if there is Bratwurst in the mensa today

```
import urllib, os

# the daily mensa menu page
url = 'http://www.studentenwerk-hamburg.de' + \
      '/essen/tag.php?haus=Geomatikum'

# get the web page
page = urllib.urlopen( url ).read()

# check for availability of bratwurst
if page.lower().find('bratwurst') != -1:
    # tell everybody
    wall = os.popen( 'wall', 'w' )
    wall.write("wurst! wuuuurst!!!")
```

Example 2: Check for recent earthquakes with magnitude larger than 5.8

```
import urllib, re

# a page with a list of recent earthquakes
url = 'http://earthquake.usgs.gov' + \
      '/eqcenter/recenteqsww/Quakes/quakes_big.php'

# get page from web
page = urllib.urlopen( url ).read()

# remove html tags
text = re.sub(r'<[^>]+>', '', page)
```

(continued...)

```
for line in text.splitlines():

    # tokens in interesting lines are now
    # separeted by a ' ' or '   '
    tokens = re.split(r'( \s*)+',line)[::2]

    if len(tokens) < 7:
        continue  # skip lines without data

    magnitude, date, lat, lon = tokens[1:5]

    # do something if magnitude is large
    if float(magnitude) > 5.8:
        print magnitude, date, lat, lon
```

Primitives

We will now loosely follow the official Python Tutorial chapter 3: *An Informal Introduction to Python*:

- `http://docs.python.org/tut/`

Interactive Python

You can use python interactively.

```
% python
Python 2.3.5 ...
>>> 1+1
2
>>> print "1 + 1 =", _
1 + 1 = 2
```

Python as a calculator

```
>>> from math import *  # to get math functions
>>> sin(pi/2.)
1.0
```

Integer division returns floor():

```
>>> 5/2
2
>>> 5/-2
-3          # unlike in C!
```

Arithmetic type casting as usual:

```
>>> 5./2
2.5
>>> 5/2.
2.5
```

Complex numbers

Complex arithmetic is built right into the language:

```
>>> 1j**2
(-1+0j)
>>> a = (3+4j)
>>> a.real
3.0
>>> a.imag
4.0
>>> abs(a)
5.0
```

Variables

A local variable is created simply by assigning it a value:

```
>>> earth_radius = 6.371e6
>>> print earth_radius * 2.
12742000.0
```

Note: Technically, instead of assigning a number to a variable, the identifier `earth_radius` is bound to an immutable number object here. This is a subtle difference to other languages and we must revisit this issue later!

Strings

Strings are defined with single or double quotes:

```
>>> print ’Indisches Tandoori Chicken’
Indisches Tandoori Chicken
>>> print "Indisches Tandoori Chicken"
Indisches Tandoori Chicken
```

Strings

Escape characters work as in other languages:

```
>>> print '\'\''
''
>>> print 'X\nX'
X
X
>>> print '\\'
\
```

Long strings

Triple quotes may be used to define a string spanning several lines:

```
>>> usage = """Usage: attack [OPTIONS] host ...
...     -a  Try all attacks
...     -v  Verbose
... """
```

Basic string operations

Concatenation:

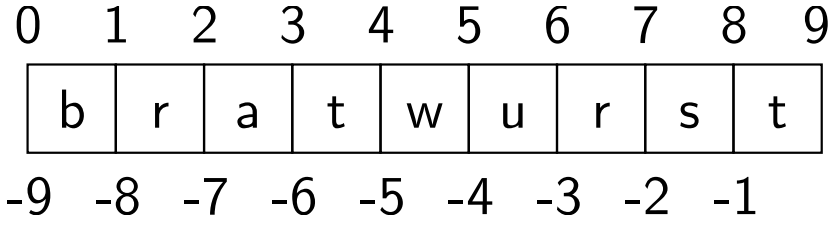
```
>>> 'sp' + 'am'
'spam'
>>> 'spam' * 10
'spamspamspamspamspamspamspamspamspamspam'
```

Basic string operations

Substrings:

```
>>> food = 'bratwurst'
>>> food[0]
'b'
>>> food[0:1]
'b'              # different than in other languages!
>>> food[1:4]
'rat'
>>> food[-5:]
'wurst'
```

Think like this, when using slices:



Strings (and numbers) are immutable

Strings in Python cannot be changed!

```
>>> food = 'bratwurst'
>>> food[0:4] = 'curry'        # impossible!
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: object doesn't support slice assignment
```

But, creating a new string is easy and efficient:

```
>>> food = 'bratwurst'
>>> food = 'curry' + food[-5:]
>>> print food
currywurst
```

String methods

Strings are objects with many useful methods:

```
>>> food = 'bratwurst'
>>> food.find( 'wurst' )  # search for substring
4
```

```
>>> food = '  bratwurst  '
>>> food.lstrip()  # remove leading whitespace
'bratwurst  '
>>> food.strip()   # remove whitespace at both ends
'bratwurst'
```

More string methods

```
>>> 'Indisches Tandoori Chicken'.split()
['Indisches', 'Tandoori', 'Chicken']
```

```
>>> ' und '.join(["Schnitzel", "Pommes", "Salat"])
'Schnitzel und Pommes und Salat'
```

There are more useful string methods like `startswith`, `endswith`, `lower`, `upper`, `ljust`, `rjust`, `center`, ...
Always have an eye on the Python Library Reference when doing something with strings:

- `http://docs.python.org/lib/string-methods.html`

String length

Use `len(string)` to get the length of a string:

```
>>> len('bratwurst')
9
```

Lists

A list is created with brackets `[]`:

```
>>> foods = [ "Schnitzel", "Pommes", "Salat" ]
```

List elements may be of different types:

```
>>> print [ 'One', 2, 3.0, foods ]
['One', 2, 3.0, ['Schnitzel', 'Pommes', 'Salat']]
```

Empty list:

```
>>> empty = []
```

Lists

A list is created with brackets `[]`:

```
>>> foods = [ "Schnitzel", "Pommes", "Salat" ]
```

List elements may be of different types:

```
>>> print [ 'One', 2, 3.0, foods ]
['One', 2, 3.0, ['Schnitzel', 'Pommes', 'Salat']]
```

Empty list:

```
>>> empty = []
```

Lists

Lists, like strings, are *sequence types*.
So, many things work the same:

```
>>> foods = ['Schnitzel', 'Pommes', 'Salat']
>>> len(foods)
3
>>> foods[0:2]
['Schnitzel', 'Pommes']
```

Modifying lists

But unlike strings (which are *immutable*), lists are of *mutable sequence type*.

```
# Replace first two elements
>>> foods[0:2] = [ "Bratwurst", "Mayo" ]
>>> print foods
['Bratwurst', 'Mayo', 'Salat']

# Remove last two elements
>>> foods[-2:] = []
>>> print foods
['Bratwurst']
```

Modifying lists

```
# Insert new elements at beginning
>>> foods[0:0] = [ "Bier", "Korn" ]
>>> print foods
['Bier', 'Korn', 'Bratwurst']

# But ...
>>> foods[0] = [ "Bier", "Korn" ]
>>> print foods
[['Bier', 'Korn'], 'Korn', 'Bratwurst']
```

Things to remember from today's lesson

- Strings and numbers are *immutable*.
- Lists are *mutable*.
- Strings and lists are *sequence types* and share many methods.

Keep these pages open, when programming in Python:

- Tutorial:
  `http://docs.python.org/tut/`
- Library Reference:
  `http://docs.python.org/lib/lib.html`