

```
\providecommand\classoptions{,notes,titlepage}
\libusetheme{FAU}
\gdefIWGSsubtitle{IWGS-1: Programming, Documents, Web Applications}
```

```
{document}
\maketitle
{npargraph}
This document contains the course notes for the course “\useSGvar{coursetitle}
(short \useSGvar{courseacronym}-1”) held at FAU Erlangen-N rnberg in the Winter Semesters
2018/19 ff.
```

```
{npargraph}
File: [courses/FAU/IWGS/course]{course/snip/otherparts.en}}
```

```
{document}
{sparagraph}
Other parts of the lecture notes can be found at
\url{http://kwarc.info/teaching/IWGS/notes-*.pdf}.
{sparagraph}
```

```
{document}
```

```
\mhinput{course/notes/frontmatter.en}
File: [courses/FAU/IWGS/course]{course/sec/preliminaries.en}}
```

```
{document}
{sfragment}[id=sec.prelim]{Preliminaries}
```

```
File: [courses/FAU/IWGS/course]{course/sec/admin.en}}
{document}
```

```
{sfragment}[id=IWCS-admin]{Administrativa}
```

```
File: [courses/FAU/IWGS/course]{course/snip/admin-intro.en}}
{document}
```

```
{sparagraph}[style=introduction]
```

```
\usemodule[smgglom/computing]{mod?efficient}
```

We will now go through the ground rules for the course. This is a kind of a social contract between the instructor and the students. Both have to keep their side of the deal to make learning as \sn{efficient} and painless as possible.

```
{sparagraph}
```

```
{document}
```

```
File: [courses/FAU/IWGS/course]{course/slides/prerequisites.en}}
```

```
{document}
```

```
{frame}
```

```
{Prerequisites}
```

```
{itemize}
```

```
\ifSGvar{semester}{2}{%
```

```
{sparagraph}[title=Formal Prerequisite]
```

```
IWGS-1\lec{If you did not take it, read the notes}
```

```
{sparagraph}}
```

```
{sparagraph}[title=General Prerequisites]
```

```
Motivation, interest, curiosity, hard work.\
```

```
\red{nothing else}!\ifSGvar{semester}{2}{\lec{apart from IWGS-1}}\par}
```

```
We will teach you all you need to know
```

{sparagraph}
You can do this course if you want! \lec{we will help}
{itemize}
{frame}
{document}

File: [courses/FAU/IWGS/course]{course/slides/grading.en}]

{document}
{smodule}{IWGS-grading}
{nparagraph}

Now we come to a topic that is always interesting to the students: the grading scheme:
The short story is that things are complicated. We have to strike a good balance between
what is didactically useful and what is allowed by Bavarian law and the FAU rules.

{nparagraph}

{frame}
{Assessment, Grades}
{itemize}

{sparagraph}[title=Grading Background/Theory]

Only modules are graded! \lec{by the law}

{itemize}
Module “DH-Einf hrung” (DHE) \hateq courses IWGS1/2, DH-Einf hrung.
DHE module grade \ergo pass/fail determined by “portfolio” \hateq collection
of contributions/assessments.

{itemize}
{sparagraph}

{sparagraph}[title=Assessment Practice]

The IWGS assessments in the “portfolio” consist of

{itemize}
weekly homework assignments, \lec{practice IWGS concepts and tools}
60 minutes exam directly after lectures end: \useSGvar{examdate}.
{itemize}
{sparagraph}

{sparagraph}[title=Retake Exam]

60 min exam at the end of the exam break. \lec{\useSGvar{retakedate}}

{sparagraph}
{itemize}
{frame}

{nparagraph}
Homework assignments, %quizzes,
and end-semester exam may seem like a lot of work -- and indeed
they are -- but you will need practice (getting your hands dirty) to master the
concepts. We will go into the details next.

{nparagraph}
{smodule}
{document}

File: [courses/FAU/IWGS/course]{course/slides/homeworks.en}]

{document}

{frame}
\usemodule[smglom/arithmetics]{mod?percentage}
 {\useSGvar{courseacronym} Homework Assignments}
{itemize}

{sparagraph}[title=Homeworks]
will be small individual problem/programming/system assignments
{itemize}
 but take time to solve\lec{at least read them directly \ergo questions}
 group submission if and only if explicitly permitted.
 {itemize}
{sparagraph}

{sparagraph}[style=warning]
Without trying the homework assignments you are unlikely to pass the exam.
{sparagraph}

{sparagraph}[title=Admin]
To keep things running smoothly
{itemize}
 Homeworks will be posted on \href{https://www.studon.fau.de/studon}{StudOn}.
 Sign up for \useSGvar{courseacronym} under \useSGvar{studon}.
 Homeworks are handed in electronically there. \lec{plain text, program files, PDF}
 \red{Go to the tutorials, discuss with your TA!}\lec{they are there for you!}
 {itemize}
{sparagraph}

{sparagraph}[title=Homework Discipline]
{itemize}
 \red{Start early!}\lec{many assignments need more than one evening's work}
 Don't start by sitting at a blank screen \lec{talking \& study group help}
 Humans will be trying to understand the text/code/math when grading it.
 {itemize}
 {sparagraph}
 {itemize}
{frame}

{nparagraph}
It is very well-established experience that without doing the homework assignments (or something similar) on your own, you will not master the concepts, you will not even be able to ask sensible questions, and take nothing home from the course. Just sitting in the course and nodding is not enough!
{nparagraph}
{document}

File: [courses/FAU/AI/course]{course/snip/tutorials-intro.en}]
{document}
{nparagraph}

If you have questions please make sure you discuss them with the instructor, the teaching assistants, or your fellow students. There are three sensible venues for such discussions: online in the lecture, in the tutorials, which we discuss now, or in the course forum -- see below. Finally, it is always a very good idea to form study groups

with your friends.

{nparagraph}

{document}

File: [courses/FAU/IWGS/course]{course/slides/uebungen.en}]

{document}

{frame}

\usemodule[courses/Jacobs/GenICT/course]{python/slides/nutshell?python-nutshell}

{\useSGvar{courseacronym} Tutorials}

{itemize}

Weekly tutorials and homework assignments\lec{first one in week two}

{columns}[c]

{column}{8cm}

{sparagraph}[title=Tutor]\lec{Doctoral Student in

\sn{computer-science?CS}}

{itemize}

Jonas Betzendahl: \url{jonas.betzendahl@fau.de}

{itemize}

They know what they are doing and really want to help you learn!\lec{dedicated to DH}

{sparagraph}

{column}

{column}{2cm}

{column}

{columns}

{sparagraph}[title=Goal 1]

Reinforce what was taught in class \lec{important pillar of the

\useSGvar{courseacronym} concept}

{sparagraph}

{sparagraph}[title=Goal 2]

Let you experiment with \python\lec{think of them as Programming Labs}

{sparagraph}

{sparagraph}[title=Life-saving Advice]

\red{go to your tutorial, and prepare it by having looked at the slides and the homework assignments}

{sparagraph}

{sparagraph}[title=Inverted Classroom]

\usemodule[courses/FAU/AI/course]{course/slides?flipped-classroom}

the latest craze in didactics\lec{works well if done right}

\titleemph{in \useSGvar{courseacronym}}: Lecture + Homework assignments + Tutorials

\hateq \sr{flipped-classroom?flipped classroom}{inverted classroom}

{sparagraph}

{itemize}

{frame}

{document}

File: [courses/FAU/IWGS/course]{course/snip/discussion.en}]

{document}
 {nparagraph}\usemodule[smglom/cs]{mod?computer-science}
 Do use the opportunity to discuss the \useSGvar{courseacronym} topics with others. After
 all, one of the non-trivial inter/transdisciplinary skills you want to learn in the
 course is how to talk about \sr{computer-science?CS}{computer science} topics --
 maybe even with real \sns{computer-science?computer scientist}. And that takes
 practice, practice, and practice.
 {nparagraph}
 {document}

File: [courses/FAU/IWGS/course]{course/slides/resources.en}]
 {document}
 {smodule}{resources}

{nparagraph}
 But what if you are not in a lecture or tutorial and want to find out more about the
 \useSGvar{courseacronym} topics?
 {nparagraph}

{frame}[label=slide.resources]
 {Textbook, Handouts and Information, Forums, Videos}
 {itemize}

{sparagraph}[title=No Textbook]
 but lots of online python tutorials on the web.
 {sparagraph}
 Course notes will be posted at \url{http://kwarc.info/teaching/IWGS}\lec{see
 references}
 {itemize}
 I mostly prepare/adapt/correct them as we go along
 please e-mail me any errors/shortcomings you notice \lec{improve for the group}
 {itemize}
 The lecture videos of WS 2020/21 are at \useSGvar{fautvURL}\lec{not much changed}
 Matrix chat at \useSGvar{chaturl} (via
 IDM)\lec{\href{https://www.anleitungen.rrze.fau.de/serverdienste/matrix-an-der-fau/erste-schritte/}}{instructions}}

{sparagraph}[title=StudOn Forum]
 \useSGvar{studon} for
 {itemize}
 announcements, homeworks\lec{my view on the forum}
 questions, discussion among your fellow students\lec{your forum too, use it!}
 {itemize}
 {sparagraph}
 If you become an active discussion group, the forum turns into a valuable
 resource!
 {itemize}
 {frame}
 {smodule}
 {document}

File: [courses/FAU/meta-inf]{admin/slides/elearning.en}]

{document}
{smodule}{eLearning-KWARC}
{nparagraph}

Next we come to a special project that is going on in parallel to teaching the course. I am using the course materials as a research object as well. This gives you an additional resource, but may affect the shape of the course materials (which now serve double purpose). Of course I can use all the help on the research project I can get, so please give me feedback, report errors and shortcomings, and suggest improvements.

{nparagraph}

{frame}[label=slide.elearning]
{Experiment: Learning Support with \sn{KWARC} Technologies}
{itemize}

{sparagraph}[title=My research area]
Deep representation formats for (mathematical) knowledge
{sparagraph}

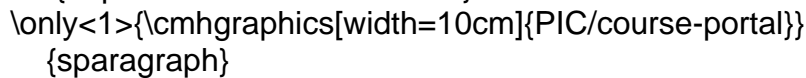
{sparagraph}[title=One Application]
Learning support systems\lec{represent knowledge to transport it}
{sparagraph}

{sparagraph}[title=Experiment]
Start with this course \lec{Drink my own medicine}
{enumerate}
Re-represent the slide materials in \LaTeX (Open Mathematical Documents)
Feed it into the \sn{ALeA} system \lec{\url{http://courses.voll-ki.fau.de}}
Try it on you all \lec{to get feedback from you}
{enumerate}
{sparagraph}
Research tasks%\lec{I cannot pay you for this}
{itemize}
help me complete the material on the slides\lec{what is missing/would help?}
I need to remember “what I say”, examples on the board.\lec{take notes}
{itemize}
Benefits for you \lec{so why should you help?}
{itemize}
you will be mentioned in the acknowledgements\lec{for all that is worth}
you will help build better course materials\lec{think of next-year’s students}
{itemize}
{itemize}
{frame}
{smodule}
{document}

File: [talks/voll-ki]{slides/course-portal.en}]
{document}
{smodule}{course-portal}

{frame}
{\sn{VoLL-KI} Portal at \url{https://courses.voll-ki.fau.de}}
{itemize}

{sparagraph}[title=Portal for \ALeA Courses]

<https://courses.voll-ki.fau.de>


AI-1 in ALeA

<https://courses.voll-ki.fau.de/course-home/ai-1>

- All details for the course.
- recorded syllabus
- keep track of material covered in course
- syllabus of the last semester (for over/preview)

ALeA Status

The ALeA system is deployed at FAU for over 1000 students taking six courses

- (some) students use the system actively
- our logs tell us
- reviews are mostly positive/enthusiastic
- error reports pour in

The VoLL-KI course portal (and the AI-1) home page is the central entry point for working with the ALeA system. You can get to all the components of the system, including two presentations of the course contents (notes- and slides-centric ones), the flash cards, the localized forum, and the quiz dashboard.

File: [talks/voll-ki/slides/flashcards.en](#)

use module [smglom/voll-ki/mod?learner-model](#)

use module [smglom/voll-ki/mod?learning-task](#)

use module [smglom/voll-ki/mod?flash-card](#)

New Feature: [Sn\[post=ing\]{drill}](#) with [Sns{flash-card?flashcard}](#)

[Sns{flashcard?flashcard}](#) challenge you with a [sn{task}](#) (term/problem) on the [sn{front}](#)

definition/answer is on the [sn{back}](#).

[Sn\[post=ment\]{self-assess}](#) updates the [sn{learner model}](#)

Idea

Challenge yourself to a [sn{card stack}](#), keep drilling [sn\[post=ing\]{assess}](#) [sns{flash-card?flashcard}](#) until the [sn{learner model}](#) eliminates all.

{sparagraph}[title=Bonus]
\Sns{flash-card?flashcard} can be generated from existing semantic markup\lec{educational
equivalent to free beer}
{sparagraph}
{itemize}
{frame}

{nparagraph}
We have already seen above how the \sn{learner model} can drive the \sn[post=ing]{drill}
with \sns{flash-card?flashcard}. It can also be used for the configuration of \sns{card stack} by
configuring a domain \inlineex{e.g. a section in the course materials and a
\sn{competency} threshold}.
{nparagraph}
{document}

File: [courses/FAU/IWGS/course]{course/slides/resources-guide.en}]
{document}
{frame}
{Practical recommendations on Lecture Videos}
{itemize}

{sparagraph}[title=Excellent Guide]
\cite{NorKueRob:lcprs18} (german Version at \cite{NorKueRob:vnas18})
\cmhgraphics[width=9cm]{course/PIC/lecture-capture-guide.pdf}
{sparagraph}
Normally intended for “offline students” \hateq everyone during Corona times.
{itemize}
{frame}
{document}

File: [courses/FAU/IWGS/course]{course/slides/tools.en}]
{document}
{smodule}{tools}

{frame}
{Software/Hardware tools}
{itemize}
You will need \sn{computer} access for this course
we recommend the use of standard software tools
{itemize}
find a \sn{text editor} you are comfortable with\lec{get good with it} A
\sn{text editor} is a program you can use to write
\sns{file-type?text file}. \lec{not \$MSWord\$}
any \sr{operating-system?OS}{operating system} you like \lec{I can only
help with \$unixOS\$}
Any browser you like \lec{I use \$firefoxbrowser\$: less spying}
{itemize}

{sparagraph}[title=Advice]
\red{learn how to touch-type NOW}\lec{reap the benefits earlier, not later}
{itemize}
you will be typing multiple hours/week in the next decades

touch-typing is about twice as fast as “system eagle”.

you can learn it in two weeks\lec{good programs}

{itemize}

{sparagraph}

{itemize}

{frame}

File: [courses/FAU/meta-inf]{admin/snip/touchtype.en}]

{document}

\usemodule{admin/mod?typingaids}

{sparagraph}[title=Touch-typing]

You should not underestimate the amount of time you will spend typing during your studies. Even if you consider yourself fluent in two-finger typing, touch-typing will give you a factor two in speed. This ability will save you at least half an hour per day, once you master it. Which can make a crucial difference in your success.

Touch-typing is very easy to learn, if you practice about an hour a day for a week, you will re-gain your two-finger speed and from then on start saving time. There are various free typing tutors on the network. At

{\url{http://typingsoft.com/all_typing_tutors.htm}} you can find about programs, most for windows, some for linux. I would probably try \$Ktouch\$ or \$TuxType\$

Darko Pesikan (one of the previous TAs) recommends the \$TypingMaster\$ program. You can download a demo version from

{\url{http://www.typingmaster.com/index.asp?go=tutordemo}}

You can find more information by googling something like "learn to touch-type". (goto {\url{http://www.google.com}} and type these search terms).

{sparagraph}

{document}

{smodule}

{document}

{sfragment}

{document}

File: [courses/FAU/IWGS/course]{course/sec/overview.en}]

{document}

{sfragment}[id=IWGS-outline]{Goals, Culture, \& Outline of the Course}

File: [courses/FAU/IWGS/course]{course/slides/goals.en}]

{document}

{frame}

\usemodule[srnglom/cs]{mod?computer-science}

{Goals of “\useSGvar{courseacronym}”}

{itemize}

{sparagraph}[title=Goal]

giving students an overview over the variety of digital tools and methods

{sparagraph}

{sparagraph}[title=Goal]

explaining their intuitions on how/why they work (the way they do).

{sparagraph}

{sparagraph}[title=Goal]

empower students for their for the emerging field \enquote{digital humanities and social sciences}.

{sparagraph}

{sparagraph}[title=NON-Goal]

Laying the \sn{mathematical} and computational foundations which will become useful in the long run.

{sparagraph}

{sparagraph}[title=Method]

introduce methods and tools that can become \emph{useful in the short term}

{itemize}

generate immediate success and gratification,

alleviate the “programming shock” (the brain stops working when in contact with \sr{computer-science?CS}{computer science} tools or

\sns{computer-science?computer scientist}) common in the humanities and social sciences.

{itemize}

{sparagraph}

{itemize}

{frame}

{document}

File: [courses/FAU/IWGS/course]{course/slides/cs-culture.en}]

{document}

{smodule}{cs-culture}

{nparagraph}

One of the most important tasks in an inter/trans-disciplinary enterprise -- and that what “digital humanities” is, fundamentally -- is to understand the disciplinary language, intuitions and foundational assumptions of the respective other side. Assuming that most students are more versed in the “humanities and social sciences” side we want to try to give an overview of the “\sr{computer-science?CS}{computer science} culture”.

{nparagraph}

{frame}

{Academic Culture in Computer Science}

{itemize}

{sdefinition}

The \definame{academic culture} is the overall style of working, research, and discussion in an academic field.

{sdefinition}

{sassertion}[style=observation]

There are significant differences in the \sn{academic culture} between \sr{computer-science?CS}{computer science}, the humanities and the social sciences.

{sassertion}

\sr{computer-science?CS}{Computer science} is an \red{engineering discipline}\lec{we build things}

{itemize}

- given a problem we look for a (mathematical) model, we can think with
- once we have one, we try to re-express it with fewer “primitives” (concepts)
- once we have, we generalize it \lec{make it more widely applicable}
- only then do we \sn{implement} it in a program\lec{ideally}

{itemize}

Design of versatile, usable, and elegant tools is an important concern

Almost all technical literature is in English.\lec{technical vocabulary too}

\sr{computer-science?computer scientist}{CSlings} love shallow hierarchies.\lec{no personality cult; alle per Du}

{itemize}

{frame}

{nparagraph}

Please keep in mind that -- self-awareness is always difficult -- the list below may be incomplete and clouded by mirror-gazing.

{nparagraph}

{smodule}

{document}

File: [courses/FAU/IWGS/course]{course/snip/outline-trans.en}]

{document}

{nparagraph}

\usemodule[smglom/cs]{mod?computer-science}

\usemodule[smglom/computing]{mod?efficient}

We now come to the concrete topics we want to cover in \useSGvar{courseacronym}. The guiding intuition for the selection is to concentrate on techniques that may become useful in day-to-day DH work -- not \sn{computer-science?CS} completeness or teaching \sn{efficiency}.

{nparagraph}

{document}

File: [courses/FAU/IWGS/course]{course/slides/outline.en}]

{document}

{frame}[label=slide.iwgs-outline]

\usemodule[courses/Jacobs/GenICT/course]{python/slides/nutshell?python-nutshell}

\usemodule[courses/Jacobs/GenCS/course]{www/slides?CSS}

\usemodule[courses/Jacobs/GenCS/course]{www/slides?javascript}

\usemodule[smglom/www]{mod?webapp}

{Outline of \useSGvar{courseacronym} 1:}

{itemize}

- \Sn{programming} in \python:\lec{main tool in \useSGvar{courseacronym}}

{itemize}

- Systematics and culture of \sn{programming}
- Program and control structures
- Basic data strutures like numbers and strings, character encodings, unicode, and regular expressions

{itemize}

- Digital documents and document processing:

{itemize}

text files
markup systems, \sn{html?HTML}, and \sn{CSS?CSS}
\sn{xml?XML}: Documents are trees.
{itemize}
Web technologies for \sn{interactive} documents and
\sns{web application}
{itemize}
\sn{internet} infrastructure: web browsers and servers
serverside computing: bottle routing and
client-side \sn[post=ion]{interact}: dynamic \sn{html?HTML},
\sn{javascript?JavaScript}, \sn{html?HTML} forms
{itemize}
\Sn{web application} project\lec{fill in the blanks to obtain a working web app}
{itemize}
{frame}
{document}

{sfragment}
{document}

File: [courses/FAU/AI/course]{course/slides/attendance.en}]

{document}
{nparagraph}

What I am going to go into next is -- or should be -- obvious, but there is an important point I want to make.

{nparagraph}

{frame}[label=slide-attendance]
{Do I need to attend the lectures}

{itemize}
<1-> Attendance is not mandatory for the \useSGvar{courseacronym} lecture
<2-> There are two ways of learning \useSGvar{courseacronym}:\lec{both are OK, your mileage may vary}

{itemize}
Approach \blue{B}: Read a \blue{Book}
\ifSGvar{courseacronym}{AI}{\cite{RusNor:AIMA09} or follow~\cite{NorTru:itai}}
Approach \blue{I}: come to the lectures, be \blue{involved}, interrupt me whenever you have a question.
{itemize}

The only advantage of \blue{I} over \blue{B} is that books do not answer questions
\lec{yet! \ogre we are working on this in AI research}

<3-> Approach \blue{S}: come to the lectures and \blue{sleep} \red{does not work}!
<4->

{sparagraph}[title=I really mean it]

If you come to class, be involved, ask questions, challenge me with comments, tell me about errors, \ldots

{itemize}
<5-> I would much rather have a lively discussion than get through all the slides

<6-> You learn more, I have more fun \lec{Approach \blue{B} serves as a backup}
<7-> You may have to change your habits, overcome shyness, \ldots\lec{please do!}
{itemize}
{sparagraph}

<8-> This is what I get paid for, and I am more expensive than most books\lec{get your money's worth}

{itemize}
{frame}
{document}

{sfragment}
{document}

File: [courses/FAU/IWGS/course]{progintro/sec/progintro.en}]

{document}
{sfragment}[id=sec.progintro]{Introduction to Programming}

File: [courses/FAU/IWGS/course]{progintro/sec/intro.en}]

{document}
{sfragment}[id=sec.programming-intro]{What is Programming?}

File: [courses/Jacobs/GenICT/course]{programming/snip/intro.en}]

{document}
{sparagraph}
\usemodule{python/slides/nutshell?python-nutshell}

\Sn{programming} is an important and distinctive part of “\useSGvar{coursetitle}”

-- the topic of this course. Before we delve into learning \python, we will review some of the basics of computing to situate the discussion.

{sparagraph}

{document}

File: [courses/Jacobs/GenICT/course]{programming/slides/hardware-software-programming.en}]

{document}
{smodule}{hardware-software-programming}
{nparagraph}

To understand \sn{programming}, it is important to realize that

\sns{computer} are universal machines. Unlike a conventional tool e.g a spade -- which has a limited number of purposes/behaviors -- digging holes in case of a spade, maybe hitting someone over the head, a \sn{computer} can be given arbitrary\footnote{as long as they are “computable”, not all are.} purposes/behaviors by specifying them in form of a \sn{program}.

{nparagraph}

{nparagraph}

\usemodule[smglom/formal-methods]{mod?formal-verification}

This notion of a \sn{program} as a behavior specification for an universal machine is so powerful, that the field of \sr{computer-science?CS}{computer science} is centered around studying it -- and what we can do with \sns{program}, this includes

{enumerate}{\em i\rm})]

storing and manipulating data about the world,
encoding, generating, and interpreting \sr{digital image}{image}, audio, and video,
transporting information for communication,
representing knowledge and reasoning,
transforming, optimizing, and \sn[post=ing]{verify} other \sns{program?program},
learning patterns in data and predicting the future from the past.

```

{enumerate}
{nparagraph}

{frame}
{Computer Hardware/Software \& Programming}
{itemize}

{sdefinition}
\Definame[post=s]{computer?computer} consist of \sn{computer?hardware} and
\sn{computer?software}.
{sdefinition}

{sdefinition}
\Definame{computer?hardware} consists of
{columns}\quad
{column}{5.2cm}
{itemize}
a \definiendum{cpu?CPU}{central processing unit} (\definame{cpu?CPU})
\definame{memory?memory}: e.g. RAM, ROM, \ldots
\definame[post=s]{storage-device?storage device}: e.g. Disks, SSD, tape, \ldots
\definiendum{information-processing-system?input subsystem}{input}:
e.g. keyboard, mouse, touchscreen, \ldots
\definiendum{information-processing-system?output subsystem}{output}:
e.g. screen, earphone, printer, \ldots
{itemize}
{column}
{column}{5cm}
\cmhgraphics[width=5cm]{programming/PIC/computer_components}
{column}
{columns}
{sdefinition}

{sdefinition}
\Definame{computer?software} consists of
{columns}\quad
{column}{6cm}
{itemize}
\definame{data?data} that represents objects and their relationships in the world
\definame[post=s]{program?program} that inputs, manipulates, outputs
\definame{data?data}
{itemize}
{column}\quad
{column}{4cm}
\def\myxscale{1.2}\def\myyscale{1.1}
\mhtikzinput[archive=courses/Jacobs/GenCS/course]{course/tikz/data-alg-machine}
{column}
{columns}
{sdefinition}

{sparagraph}[title=Remark]
\Sn{computer?hardware} stores \sn{data?data} and runs
\sns{program?program}.
{sparagraph}
{itemize}
{frame}

```

{nparagraph}
 \usemodule[smglom/computing]{mod?general-purpose-computer}
 \usemodule[smglom/computing]{mod?embedded-system}
 A universal machine has to have -- so experience in
 \sr{computer-science?CS}{computer science} shows certain distinctive parts.
 {itemize}
 A \sn{cpu?CPU} that consists of a
 {itemize}
 \inlinedef{\definame{cpu?control unit} that interprets the \sn{program} and controls the
 flow of instructions} and
 a \inlinedef{\definiendum{cpu?ALU}{arithmetic/logic unit} (\definame{cpu?ALU}) that does the
 actual computations internally}.
 {itemize}
 \Sn{memory?memory} that allows the system to store data during runtime
 (volatile storage; usually RAM) and between runs of the system (persistant storage;
 usually hard disks, solid state disks, magnetic tapes, or optical media).
 I/O devices for the communication with the user and other \sns{computer?computer}.
 {itemize}
 With these components we can build various kinds of universal machines; these range from
 thought experiments like \sns{Turing machine}, to today's
 \sns{general-purpose-computer?general purpose computer} like your laptop with
 various \sns{embedded-system?embedded system} (wristwatches, Internet routers,
 airbag controllers, \ldots) in-between.
 {nparagraph}

{nparagraph}
 \usemodule[courses/FAU/AI/course]{intro/slides?whatisai} Note that -- given enough fantasy -- the
 human brain has the same components. Indeed the human mind is a universal machine -- we
 can think whatever we want, react to the environment, and are not limited to particular
 behaviors. There is a sub-field of \sr{computer-science?CS}{computer science} that
 studies this: \sr{AI}{Artificial Intelligence} (\sn{AI}). In
 this analogy, the brain is the "hardware" --sometimes called "wetware" because it is
 not made of hard silicon or "meat machine"\footnote{Marvin Minsky; one of the founding
 fathers of the field of \sr{AI}{Artificial Intelligence}}. It is instructional to
 think about what the \sn{program?program} and the data might be in this analogy.
 {nparagraph}

{smodule}
 {document}

File: [courses/Jacobs/GenICT/course]{programming/slides/programming-languages.en}
 {document}
 {smodule}{programming-languages}
 \usemodule[smglom/computing]{mod?programming-paradigm}

{frame}
 {Programming Languages}
 {itemize}
 <1-> \Sn{programming} \hateq writing
 \sns{program?program}\lec{Telling the \sn{computer} what to do}
 <2->
 {sassertion}[style=remark]
 The \sn{computer} does exactly as told

{itemize}
extremely fast extremely reliable
completely stupid: will not do what you mean unless you tell it exactly
{itemize}
{sassertion}
<2-> \Sn{programming} can be extremely fun/frustrating/addictive\lec{try it}
<3->
{sdefinition}

A \define{program?programming language} is the \sn{formal language} in which we write \define[post=s]{program?program} \lec{express an \sn{algorithm} concretely}

{sdefinition}
{itemize}
formal, symbolic, precise \sn{meaning} \lec{a machine must understand it}
{itemize}
<4-> There are lots of \sns{program?programming language}

{itemize}
design huge effort in \sr{computer-science?CS}{computer science}
all \sns{program?programming language} equally strong
each is more or less appropriate for a specific task depending on the circumstances

{itemize}
<4-> Lots of \sns{programming paradigm}:
\sr{imperative programming}{imperative},
\sr{functional programming}{functional}, \sr{logic programming}{logic},
\sn{object oriented programming}.
{itemize}
{frame}

{nparagraph}
\usemodule[courses/FAU/AI/course]{intro/slides?whatisai}
\usemodule[smglom/computing]{mod?efficient}
\sn{AI} studies \sr{HI}{human intelligence} with the premise that the brain is a computational machine and that \sn{intelligence} is a “\sn{program?program}” running on it. In particular, the working hypothesis is that we can “program” \sn{intelligence}. Even though \sn{AI} has many successful applications, it has not succeeded in creating a machine that exhibits the equivalent to general human \sn{intelligence}, so the jury is still out whether the \sn{AI} hypothesis is true or not. In any case it is a fascinating area of scientific inquiry.

{nparagraph}

{nparagraph}[title=Note]
\usemodule[smglom/computing]{mod?efficient}
This has an immediate consequence for the discussion in our course. Even though \sns{computer} can execute \sns{program?program} very \sn[post=ly]{efficient}, you should not expect them to “think” like a human. In particular, they will execute \sns{program?program} exactly as you have written them. This has two consequences:

{itemize}
the behavior of \sns{program?program} is -- in principle -- predictable
all errors of \sn{program?program} behavior are your own (the \sn{programmer}'s)

{itemize}
{nparagraph}
{smodule}
{document}

File: [courses/Jacobs/GenICT/course]{programming/slides/program-execution.en}]

{document}

{smodule}{program-execution}

{nparagraph}

In \sr{computer-science?CS}{computer science}, we distinguish two levels on which we can talk about \sns{program?program}. The more general is the level of \sns{algorithm?algorithm}, which is independent of the concrete \sn{program?programming language}. \Sns{algorithm} express the general ideas and flow of computation and can be realized in various languages, but are all equivalent -- in terms of the \sns{algorithm?algorithm} they \sn{implement}.

{nparagraph}

{nparagraph}

\usemodule[smglom/epistemology]{mod?ambiguity}

As they are not bound to \sns{program?programming language} \sns{algorithm?algorithm} transcend them, and we can find them in our daily lives, e.g. as sequences of instructions like recipes, game instructions, and the like. This should make \sns{algorithm} quite familiar; the only difference of \sns{program?program} is that they are written down in an \sn[pre=un]{ambiguous} syntax that a \sn{computer} can understand.

{nparagraph}

{frame}

{Program Execution}

{itemize}

{sdefinition}[id=algorithm.def]

\Definame{algorithm?algorithm}: informal description of what to do (good enough for humans)

{sdefinition}

{sexample}[id=recipe-alg,for=algorithm?algorithm]

{sexample}

{sexample}

\usemodule{python/slides/forloop?python-forloop}

\Sn{program?program}: \sn{computer} processable version, e.g. in \python.

\lstinputmhlisting{python/code/forloop3.py}

{sexample}

{sdefinition}

\Definame{interpreter?interpreter}: reads a \sn{program?program} and executes it directly

{itemize}

special case: \sn{interactive} interpretation\lec{lets you experiment easily}

{itemize}

{sdefinition}

{sdefinition}

\Definame{compiler?compiler}: translates a \sn{program?program} (the \define{source}) into another \sn{program?program} (the \define{program-execution?binary}) in a much simpler \sn{program?programming language} for optimized execution on hardware directly.

{sassertion}[style=remark]
\Sns{compiler?compiler} are \sn{efficient}, but more cumbersome for development.
{sassertion}
{itemize}
{frame}

{nparagraph}
We have two kinds of \sns{program?programming language}: one which the \sn{cpu?CPU} can execute directly -- these are very very difficult for humans to understand and maintain -- and higher-level ones that are understandable by humans. If we want to use high-level languages -- and we do, then we need to have some way bridging the language gap: this is what \sns{compiler?compiler} and \sns{interpreter?interpreter} do.

{nparagraph}
{smodule}
{document}

{sfragment}
{document}

File: [courses/FAU/IWGS/course]{progintro/sec/iwgs.en}]
{document}
{sfragment}[id=sec.programming-iwgs]{Programming in \useSGvar{courseacronym}}
File: [courses/FAU/IWGS/course]{progintro/snip/iwgs-intro.en}]
{document}
{sparagraph}
\usemodule[courses/Jacobs/GenICT/course]{python/slides/nutshell?python-nutshell}

After the general introduction to \sn{programming} in \sref[fallback=the last Section,file=progintro/sec/progintro.en]{sec.progintro}, we now instantiate the situation to the \useSGvar{courseacronym} course, where we use \python as the primary \sn{programming language}.

{sparagraph}
{document}

File: [courses/Jacobs/GenICT/course]{python/slides/python-genict.en}]
{document}
{smodule}{python-genict}

{frame}
{Programming in \useSGvar{courseacronym}: \python}
{itemize}
We will use \python as the \sn{program?programming language} in this course
We cover just enough \python, so that you
{itemize}

understand the joy and principle of \sn{programming}
 can play with objects we present in \useSGvar{courseacronym}.

{itemize}

After a general introduction we will introduce language features as we go along

For more information on \python\lec{homework/preparation}

{center}\huge

{sparagraph}

\inlinedef{\definame{RTFM}} (\hateq “read those \green{fine} manuals”)

{sparagraph}

{center}

{sparagraph}[title=RTFM Resources]

There are also lots of good tutorials on the web,

{itemize}

I like~\cite{LP:on,sthurlow:abpt:url,Sweigart:iwp14};

but also see the language

documentation~\cite{python3doc:on}.

\ifSGvar{courseacronym}{IWGS}{ \cite{Karsdorp:pph}}

is an introduction geared to the (digital) humanities}

{itemize}

{sparagraph}

{itemize}

{frame}

{smodule}

{document}

File: [courses/FAU/IWGS/course]{progintr/snip/iwgs-not.en}]

{document}

{sparagraph}[style=start,title=Note]

\usemodule[smglom/computing]{mod?programming}

that \useSGvar{courseacronym} is not a \sn{programming} course, which concentrates

on teaching a \sn{programming language} in all it gory detail. Instead we want to

use the \useSGvar{courseacronym} lectures to introduce the necessary concepts and use

the tutorials to introduce additional language features based on these.

{sparagraph}

{document}

File: [courses/FAU/IWGS/course]{progintr/slides/lets-hack.en}]

{document}

{frame}

\usemodule[courses/Jacobs/GenICT/course]{python/slides/nutshell?python-nutshell}

{But Seriously\ldots Learning programming in \useSGvar{courseacronym}}

{itemize}

The \useSGvar{courseacronym} lecture teaches you

{itemize}

a general introduction to \sn{programming} and \python\lec{next}

various useful concepts and how they can be done in \python \lec{in

principle}

{itemize}

The \useSGvar{courseacronym} tutorials

{itemize}

teach the actual skill and joy of \sn{programming} \lec{hacking \$ \not=\$

security breach}

supply you with problems so you can practice that.

{sparagraph}[title=Richard Stallman (MIT) on Hacking]
“What they had in common was mainly love of excellence and
\sn{programming}. They wanted to make their programs that they used be as good
as they could. They also wanted to make them do neat things. They wanted to be able
to do something in a more exciting way than anyone believed possible and show “Look
how wonderful this is. I bet you didn’t believe this could be done.”
{sparagraph}

{sparagraph}[style=start,title={So, ...}]
Let’s hack
{sparagraph}
{itemize}
{frame}

{nparagraph}
However, the result would probably be the following:
{nparagraph}

{frame}
{\textwarning} 2am in the Kollegienhaus CIP Pool {\textwarning}}
\cmhgraphics[width=11cm,archive=courses/Jacobs/GenCS/course]{course/PIC/monkeys}
{frame}

{nparagraph}
If we just start hacking before we fully understand the problem, chances are very good
that we will waste time going down blind alleys, and garden paths, instead of attacking
problems. So the main motto of this course is:
{nparagraph}

{frame}
{\textwarning no, let’s think \textwarning}
{itemize}
We have to fully understand the problem, our tools, and the solution space
first\lec{That is what the \useSGvar{courseacronym} lecture is for}
{itemize}
read Richard Stallman’s quote carefully \ergo problem understanding is a crucial
prerequisite for hacking.
{itemize}
\nlex{The GIGO Principle: Garbage In, Garbage Out}\lec{-- ca. 1967}
\nlex{Applets, Not Crapletstm}\lec{-- ca. 1997}
{itemize}
{frame}
{document}

{sfragment}
{document}

File: [courses/FAU/IWGS/course]{progintro/sec/python.en}]
{document}
{sfragment}[id=sec.python]{Programming in Python}

File: [courses/Jacobs/GenICT/course]{python/snip/intro-intro.en}]
{document}
{sparagraph}
\usemodule{python/slides/nutshell?python-nutshell}

In this \currentsectionlevel we will introduce the basics of the \python language. \python will be used as our means to express \sns{algorithm} and to explore the computational properties of the objects we introduce in \useSGvar{courseacronym}.
{sparagraph}
{document}

File: [courses/FAU/IWGS/course]{progintro/sec/hello.en}]
{document}
{sfragment}[id=sec.hello]{Hello \useSGvar{courseacronym}}
File: [courses/Jacobs/GenICT/course]{python/slides/nutshell.en}]
{document}
{smodule}{python-nutshell}
\symdef{idlethree}{\comp{\mathsf{IDLE3}}}

{nparagraph}
Before we get into the \sn{syntax} and \sn{meaning} of \python, let us recap why we chose this particular language for \useSGvar{courseacronym}.
{nparagraph}

{frame}
{python in a Nutshell}
{itemize}

{sparagraph}[title=Why \python?]
{columns}
{column}{7cm}
{itemize}
general purpose \sn{program?programming language}
\sr{imperative programming}{imperative}, \sn{interactive}
\sn{interpreter?interpreter}
{itemize}
{column}
{column}{4cm}
\cmhgraphics[width=4cm]{python/PIC/python-logo}
{column}
{columns}
{itemize}
syntax very easy to learn\lec{spend more time on problem solving}
scales well:
{itemize}
easy for beginners to write simple \sns{program?program},
but advanced software can be written with it as well.
{itemize}
{itemize}
{sparagraph}

{sparagraph}[title=Interactive mode]
The \inlinedeff[for=idlethree]{\python \sn{shell} \$idlethree\$}

{sparagraph}

{sparagraph}[title=For the eager (optional)]
\usemodule[smglom/computing]{mod?installation}

Establish a \python \sn{interpreter?interpreter} (version 3.7) \lec{not 2.?.?, that has different syntax}
{itemize}
 \sn{install} \python from \url{http://python.org}\lec{for offline use}
 make sure (tick box) that the \lstinline|python| executable is added to the path.\lec{makes \sn{shell} \sn[post=ion]{interact} much easier}
{itemize}
{sparagraph}
{itemize}
{frame}

{nparagraph}[title=Installing \python]
\usemodule[smglom/computing]{mod?operating-systems}
\usemodule[smglom/computing]{mod?installation}
\python can be \sn[post=ed]{install} from \url{http://python.org} \ergo
“Downloads”, as a \$\windowsOS\$ \sn[post=er]{install} or a \$\macosxOS\$ disk
image. For \$\linuxOS\$ it is best \sn[post=ed]{install} via the package manager,
e.g. using \lstinputmhlising[language=bash]{python/code/apt-get.sh}

The download will \sn{install} the \python \sn{interpreter?interpreter} and
the \python \sn{shell} \$\idlethree\$ that can be used for
\sn[post=ing]{interact} with the \sn{interpreter?interpreter} directly.
{nparagraph}

{nparagraph}

It is important that you make sure (tick the box in the Windows
\sn[post=er]{install}) that the \lstinline|python| executable is added to the
path. In the \sn{shell}\Ednote{fully introduce the concept of a shell in the next
round}, you can then use the \sr{instruction}{command}
\lstinputmhlising[language=bash,mathescape]{python/code/python-call.sh} to run the
python file \$\pmetavar{filename}\$. This is better than using the windows-specific
\lstinputmhlising[language=bash,mathescape]{python/code/py-call.sh} which does not need
the \lstinline|python| \sn{interpreter} on the path as we will see later.
{nparagraph}
{smodule}
{document}

File: [courses/Jacobs/GenICT/course]{python/slides/expressions.en}]
{document}
{smodule}{python-expressions}
\lstset{language=python}

{frame}[fragile]
 {Arithmetic Expressions in \python}
{itemize}
 Expressions are “\sns{program?program}” that compute
values\lec{here: numbers}
{columns}

```
{column}{8.5cm}
{itemize}
  \blue{Integers} \lec{numbers without a decimal point}
{itemize}
  \sns{operator?operator}: addition (\lstinline|+|), subtraction
(\lstinline| |), multiplication (\lstinline|*|), division (\lstinline|/|),
integer division (\lstinline|//|), remainder/modulo (\lstinline|%|), \ldots
  Division yields a float
  {itemize}
  \blue{Floats} \lec{numbers with a decimal point}
{itemize}
  \Sns{operator?operator}: integer below (\lstinline|floor|),
integer above (\lstinline|ceil|), exponential (\lstinline|exp|), square root
(\lstinline|sqrt|), \ldots
  {itemize}
```

```
{sparagraph}
Numbers are \inlinedef{\definame[post=s]{value?value}, i.e. data objects that
can be computed with.}\lec{reference the last computed one with
\lstinline|_|}
{sparagraph}
```

```
{sdefinition}
\Definame[post=s]{program-expression?expression} are created from
\sns{value?value} (and other
\sns{program-expression?expression}) via
\definame[post=s]{operator?operator}.
{sdefinition}
```

```
{sparagraph}[title=Observation]
The \python \sn{interpreter?interpreter} simplifies
\sns{program-expression?expression} to
\sns{functions?value} by computation.
```

```
{sparagraph}
{itemize}
{column}
{column}{2.5cm}
\cmhgraphics[width=2.5cm]{python/PIC/idle3-expressions}
{column}
{columns}
{itemize}
{frame}
{smodule}
{document}
```

```
File: [courses/Jacobs/GenICT/course]{python/slides/comments.en}]
{document}
{smodule}{python-comments}
\lstset{language=python}
```

```
{nparagraph}
Before we go on to learn more basic \python operators and
\sns{information-processing-system?instruction}, we address an important general topic:
\sns{comment} in \sn{program?program} code.
```

```

{npargraph}

{frame}[fragile,label=slide.comments]
  {\Sns{comment} in \python}
{itemize}
  <1->
  {sparagraph}[title=Generally]
  It is highly advisable to insert \sns{comment} into your \sns{program?program},
  {itemize}
    especially, if others are going to read your code, \lec{TAs/graders}
    you may very well be one of the “others” yourself, \lec{in a year’s time}
    writing \sns{comment} first helps you organize your thoughts.
  {itemize}
  {sparagraph}
  \Sns{comment} are ignored by the \python \sn{interpreter?interpreter} but are
  useful information for the \sn{programmer}.
  <2->
  {sparagraph}[title=In \python]
  there are two kinds of \sns{comment}
  {itemize}
    Single \sn{file-type?line} \sns{comment} start with a \lstinline|#|
    Multiline \sns{comment} start and end with three quotes\lec{single or double:
  \lstinline|""| or \lstinline|"}|}%|
  {itemize}
  {sparagraph}
  <3->
  {sparagraph}[title=Idea]
  Use \sns{comment} to
  {itemize}
    specify what the intended input/output behavior of the \sn{program?program} or fragment
    give the idea of the \sn{algorithm} achieves this behavior.
    specify any assumptions about the context \lec{do we need some file to exist}
    document whether the \sn{program?program} changes the context.
    document any known limitations or errors in your code.
  {itemize}
  {sparagraph}
  {itemize}
  {frame}
  {smodule}
  {document}

  {sfragment}
  {document}

```

```

File: [courses/FAU/IWGS/course]{progintro/sec/jupyterlab.en}]
{document}
{sfragment}[id=sec.jupyterlab]{JupyterLab, a Python Web IDE for IWGS}
File: [courses/FAU/IWGS/course]{progintro/slides/jupyterLab.en}]
{document}
{smodule}{jupyterLab}

```

```

{npargraph}
In \useSGvar{courseacronym}, we want to use the \sn{jupyterLab} cloud service. This runs

```


the \python \sn{interpreter?interpreter} on a cloud server and gives you a \sr{webbrowser?web browser}{browser} \sn{window} with a \sn{web IDE}, which you can use for \sn[post=ing]{interact} with the \sn{interpreter?interpreter}. You will have to make an account there; details to follow.

{nparagraph}

{frame}

{\sn{jupyterLab} A Cloud IDE for \python}

{itemize}

<1->

{sparagraph}[style=start,title=For helping you]

it would be good if the TAs could access to your code

{sparagraph}

<1->

{sparagraph}[title=Idea]

Use a \inlinedef[for=jupyterLab]{\definame{web-IDE?web IDE} (a web based integrated development environment): \sn{jupyterLab}}, which you can use for

\sn[post=ing]{interact} with the \sn{interpreter?interpreter}.

{sparagraph}

<2-> We will use \sn{jupyterLab} for \useSGvar{courseacronym}.\lec{but you can also use \python locally}

<2->

{sparagraph}[title=Homework]

Set up \sn{jupyterLab}

{itemize}

make an account at \url{http://jupyter.kwarc.info}

{itemize}

{sparagraph}

{itemize}

{frame}

{nparagraph}

The advantage of a cloud IDE like \sn{jupyterLab} for a course like

\useSGvar{courseacronym} is that you do not need any \sn[post=ation]{install}, cannot lose your files, and your teachers (the course instructor and the teaching assistants) can see (and even directly \sn{interact} with) the your run time environment. This gives us a much more controlled setting and we can help you better.

{nparagraph}

{nparagraph}

{nparagraph}

Both \$idlethree\$ as well as \sn{jupyterLab} come with an integrated editor for writing

\python programs. These editors gives you \python syntax

highlighting, and \sn{interpreter?interpreter} and debugger integration. In

short, \$idlethree\$ and \sn{jupyterLab} are integrated development environments for

\python. Let us now go through the interface of the \sn{jupyterLab} IDE.

{nparagraph}

{frame}

{\sn{jupyterLab} Components}

{itemize}

<1->

{sdefinition}

The \sn{jupyterLab} \definame{dashboard?dashboard} gives you access to all components.

\only<1>{\cmhgraphics[width=8.5cm]{progintro/PIC/jupyterLab}}

```
{sdefinition}
<2->
{sdefinition}
The \sn{jupyterLab} \define{python console}, i.e. a \python
\sn{interpreter?interpreter} in your
\sr{webbrowser?web browser}{browser}.\lec{use this for \python
\sn[post=ion]{interact} and testing.}
\only<2>{\cmhgraphics[width=11cm]{progintro/PIC/jupyterLab-python-console}}
{sdefinition}
```

```
<3->
{sdefinition}
The \sn{jupyterLab} \define{jupyterLab?terminal}, i.e. a $\unixOS$
\sn{shell?shell} in your browser.\lec{use this for managing files}
\only<3>{\cmhgraphics[width=11cm]{progintro/PIC/jupyterLab-bash-console}}
{sdefinition}
```

```
<4->
File: [smglom/computing]{mod/shell.en}
{document}
{smodule}[creators=miko,title={Shell}]{shell}
```

```
{sdefinition}
A \define{shell} is a \sr{CLI?CLI}{command line interface} for accessing the
\sns{client-server?service} of a \sn{computer?computer}'s
\sr{operating-system?OS}{operating system}.
```

```
There are multiple \sn{shell} \sns{implementation}: \define{sh}, \define{csh},
\define{bash}, \define{zsh}; they differ in advanced features.
{sdefinition}
{smodule}
{document}
```

```
<4->
{sparagraph}[title=Useful \sn{shell} \sr{instruction}{commands}]
See e.g.~\cite{Allen:nutbsc18} for a basic tutorial
{itemize}
\stinline|ls|: “list” the \sns{file} in this \sn{directory}
\stinline|mkdir|: “make” \sr{directory}{folder} (called
“\sn{directory}”)
\stinline|pwd|: “print \sn{working directory}”\lec{where am I}
\stinline|cd| $\pmetavar{dirname}$: “change \sn{directory}”
{itemize}
if $\pmetavar{dirname} = \stinline|..|$ : one up in the \sn{directory}
\sn{tree}
empty \pmetavar{dirname}: go to your \sn{home directory}.
{itemize}
\stinline|rm| $\pmetavar{name}$: remove \sn{file}/\sn{directory}
\stinline|cp|/\stinline|mv| $\pmetavar{filename}$ $\pmetavar{newname}$: copy
to or rename
\stinline|cp|/\stinline|mv| $\pmetavar{filename}$ $\pmetavar{dirname}$: copy
or move to
\ldots see \cite{Allen:nutbsc18} for more \ldots
{itemize}
{sparagraph}
{itemize}
```

```
{frame}
{smodule}
{document}
```

```
File: [courses/Jacobs/GenICT/course]{python/snip/hello-intro.en}}
{document}
{sparagraph}
\usemodule{python/slides/nutshell?python-nutshell}
```

Now that we understand our tools, we can write our first program: Traditionally, this is a “hello-world program” (see~\cite{HWC:on} for a description and a list of hello world programs in hundreds of languages) which just prints the string “Hello World” to the console. For \python, this is very simple as we can see below. We use this program to explain the concept of a program as a (text) file, which can be started from the console.

```
{sparagraph}
{document}
```

```
File: [courses/FAU/IWGS/course]{progintro/slides/helloworld.en}}
{document}
{smodule}{jupyterlab-helloworld}
\lstset{language=python}
```

```
{frame}[label=slide.helloworld]
{A first program in \python}
{itemize}
<1->
{sparagraph}[title=A classic “Hello World” program]
start your \sn{jupyterLab?python console}, type \lstinline|print("Hello IWGS")|.
\lec{print a string}
\only<1>{\cmhgraphics[width=10cm]{progintro/PIC/jupyterLab-helloworld}}
{sparagraph}
<2->
{sparagraph}[title=Alternatively]
{enumerate}
got to the \sn{jupyterLab} \sn{dashboard?dashboard} select “Text File”,
Type your program,
\only<2>{\cmhgraphics[width=10cm]{progintro/PIC/jupyterLab-editor}}
Save the file as \lstinline|hello.py|
Go to your \sn{jupyterLab?terminal} and type \lstinline|python3 hello.py|
[3'] \titleemph{Alternatively}: go to your \sn{jupyterLab?python console} and
type\lec{in the same \sn{directory}}
\lstinputmhlisting[language=bash]{progintro/code/import-hello.sh}
{enumerate}
{sparagraph}
{itemize}
{frame}
```

```
{nparagraph}
We have seen that we can just call a program from the
\sn{jupyterLab?terminal}, if we stored it in a file. In fact, we can do
better: we can make our program behave like a native \sn{shell?shell} command.
```

{enumerate}

The `\sn{file} \sn{file-system?extension} \stinline|py|` is only used by convention, we can leave it out and simply call the file `\stinline|hello|`.

Then we can add a special `\python \sn{commenting?comment}` in the first `\sn{file-type?line} \stinputmhlisting[language=bash,archive=courses/Jacobs/GenICT/course,mathescape]{python/code/pyth on-call.sh}`

which the `\sn{jupyterLab?terminal}` interprets as “call the program `\stinline|python3|` on me”.

Finally, we make the file `\stinline|hello|` executable, i.e. tell the `\sn{jupyterLab?terminal}` the `\sn{file}` should behave like a `\sn{shell} \sr{instruction}{command}` by issuing `\stinputmhlisting[language=bash,archive=courses/Jacobs/GenICT/course]{python/code/chmod.sh}` in the `\sn{directory}` where the file `\stinline|hello|` is stored.

We add the `\sn{file-type?line} \stinputmhlisting[language=bash,mathescape=false,archive=courses/Jacobs/GenICT/course]{python/cod e/path.sh}`

to the file `\stinline|.bashrc|`. This tells the `\sn{jupyterLab?terminal}` where to look for programs (here the respective `\sr{working directory}{current directory}` called `\stinline|.|`)

{enumerate}

With this simple recipe we could in principle extend the repertoire of instructions of the `\sn{jupyterLab?terminal}` and automate repetitive tasks.

{nparagraph}

{smodule}

{document}

File: `[courses/FAU/IWGS/course]{progintr o/slides/jupyterNotebooks.en}`

{document}

{smodule}{jupyterNotebooks}

{nparagraph}

We now come to the signature component of `\sn{jupyterLab}`: `\sns{jupyter notebook}`. They take the important practice of documenting `\sn{computer-code?code}` to a whole new level. Instead of just allowing `\sns{commenting?comment}` in `\sn{program} \sns{file}`, they provide rich text cells, in which we can write elaborate text.

{nparagraph}

{frame}[label=slide.jupyterNotebooks]

{Jupyter Notebooks}

{itemize}

{sdefinition}

`\Definame[post=s]{jupyter notebook}` are documents that combine live runnable code with rich, narrative text (for comments and explanations).

{sdefinition}

{sdefinition}

`\Sns{jupyter notebook}` consist of `\definame[post=s]{cell}` which come in three forms:

{itemize}

a `\definame{raw cell}` shows text as is,

a `\definame{markdown cell}` interprets the contents as markdown text,\lec{later more}

a `\definame{code cell}` interprets the contents as (e.g. `\python`) code.

{itemize}

{sdefinition}

\Sns{cell} can be executed by pressing “shift enter”.\lec{Just “enter” gives a new line}

{sparagraph}[title=Idea]
\Sns{jupyter notebook} act as a \sn{REPL?REPL}, just as \$\\idlethree\$, but allows

{itemize}
documentation in \sr{raw cell}{raw} and \sns{markdown cell}
and
changing and re-executing existing \sns{cell}.

{itemize}
{sparagraph}
{itemize}
{frame}

{frame}[label=slide.jupyterNotebooks2]
{Jupyter Notebooks}
{itemize}

{sexample}[title=Showing off Cells in a Notebook,for=jupyter notebook]
\cmhgraphics[width=8cm]{progintro/PIC/cells-nb}
{sexample}
{itemize}
{frame}
{smodule}
{document}

File: [courses/Jacobs/TDM/course]{digdocs/slides/markdown.en}]
{document}
{smodule}{markdown}
\usemodule[courses/Jacobs/GenCS/course]{www/slides?wikis}
\usemodule[courses/Jacobs/GenICT/course]{maintenance/slides?bugtracker}

{frame}
{Markdown a simple Markup Format Generating \sn{html?HTML}}.
{itemize}

{sparagraph}[title=Idea]
We can translate between \sns{document-type?markup format}.
{sparagraph}

{sdefinition}[id=markdown.def]
\Definame{markdown} is a family of \sns{document-type?markup format} whose
\sns{markup?control word} are unobtrusive and easy to write in a
\sn{text editor}. It is intended to be converted to \sn{html?HTML} and other
formats for display.
{sdefinition}

{sexample}[for=markdown]
\Sn{markdown} is used in applications that want to make user input easy and
\sn{efficient}, e.g. \sns{wikis?wiki} and \sr{bugtracker?bugtracker}{issue tracking
systems}.
{sexample}

{sparagraph}[title=Workflow]

Users write \sn{markdown}, which is formatted to \sn{html?HTML} and then served for display.

{sparagraph}

A good cheat-sheet for \sn{markdown} \sns{markup?control word} can be found at \url{https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet}.

{itemize}

{frame}

{smodule}

{document}

{sfragment}

{document}

File: [courses/FAU/IWGS/course]{progintro/sec/vartypes.en}]

{document}

{sfragment}[id=sec.vartypes]{Variables and Types}

File: [courses/Jacobs/GenICT/course]{python/slides/variables.en}]

{document}

{smodule}{python-variables}

\lstset{language=python}

{nparagraph}

And we start with a general feature of \sns{program?programming language}: we can give names to \sns{value?value} and use them multiple times. Conceptually, we are introducing shortcuts, and in reality, we are giving ourselves a way of storing \sns{value?value} in \sn{memory?memory} so that we can reference them later.

{nparagraph}

{frame}[fragile]

{Variables in \python}

{itemize}

{sparagraph}[title=Idea]

\Sns{value?value} (of \sns{program-expression?expression}) can be given a name for later reference.

{sparagraph}

\inputref[smglom/computing]{mod/program-variable.en}

{sparagraph}[title=Note]

In \python a \sn{variable name}

{itemize}

must start with letter or \lstinline|_|,

cannot be a \python \sn{keyword}

is case-sensitive\lec{\lstinline|foobar|, \lstinline|FooBar|, and \lstinline|fooBar| are different variables}

{itemize}

{sparagraph}

A \sn{variable name} can be used in \sns{program-expression?expression} everywhere its \sn{value?value} could be.

{sdefinition}[title=in \python]

A \define{variable assignment}
\stinline[mathescape]{\$\pmetavar{var}\$=\$\pmetavar{val}\$} \define[post=s]{assignment?assign}
a new \sn{value?value} to a \sn{program-variable?variable}.
{sdefinition}

{sexample}[title=Playing with \python
Variables,id=ex.swap-variables]\strut
\cmhgraphics[width=8cm]{python/PIC/variables}
{sexample}
{itemize}
{frame}

{nparagraph}
Let us fortify our intuition about \sns{program-variable?variable} with some examples. The first shows that we sometimes need \sns{program-variable?variable} to store objects out of the way and the second one that we can use \sns{program-variable?variable} to assemble intermediate results.
{nparagraph}

{frame}[label=slide.variables-ex]
{Variables in \python: Extended Example}
{itemize}

{sexample}[title=Swapping Variables]
To exchange the values of two \sns{program-variable?variable}, we have to cache the first in an auxiliary variable.
\stinputmhlisting{python/code/swap.py}
Here we see the first example of a \inlinedef{\python script, i.e. a series of \python commands, that jointly perform an action (and communicates it to the user)).
{sexample}

{sexample}[title=Variables for Storing Intermediate Variables,id=ex.var-intermediate]
\stinputmhlisting{python/code/ohGott.py}
{sexample}
{itemize}
{frame}

{nparagraph}
If we use \sns{program-variable?variable} to assemble intermediate results, we can use telling names to document what these intermediate objects are -- something we did not do well in \sref[file=python/slides/variables.en]{ex.var-intermediate}; but admittedly, the \sn{meaning} of the objects in this contrived example is questionable.
{nparagraph}
{smodule}
{document}

File: [courses/Jacobs/GenICT/course]{python/slides/datatypes.en}
{document}
{smodule}{python-datatypes}
\lstset{language=python}

{nparagraph}
The next phenomenon in \python is also common to many (but not all) \sns{program?programming language}:

\sr{program-expression?expression}{expressions} are classified by the kind of objects their \sns{value?value} are. Objects can be simple (i.e. of a basic \sn{type?type}); \python has five of these) or complex, i.e. composed of other objects; we will go into that below.

{nparagraph}

{frame}[label=slide.datatypes1]

{Data Types in \python}

{itemize}

{sparagraph}[title=Recall]

\python \sns{program?program} process data (\sns{value?value}), which can be combined by \sns{operator?operator} and \sn{program-variable?variable} into \sns{program-expression?expression}.

{sparagraph}

\sr{type?type}{Data types} group data and tell the \sn{interpreter} what to expect {itemize}

\stinline|1|, \stinline|2|, \stinline|3|, etc. are \sn{data?data} of \sn{type?type} “integer”

\stinline|"hello"| is \sn{data?data} of \sn{type?type} “string”

{itemize}

\sr{type?type}{Data types} determine which operators can be applied

In \python, every \sns{value?value} has a \sn{type?type}, variables can have any \sn{type?type}, but can only be assigned \sns{value?value} of their \sn{type?type}.

{sdefinition}[id=python-basic-datatypes.def]

\python has the following five basic \definame[post=s]{type?type}

{center}\footnotesize

{tabular}{|c|c|l|l|}\hline

\sr{type?type}{Data type} & Keyword & contains& Examples\\\hline\hline

\definame[post=s]{python-datatypes?integer} & \stinline|int|

& bounded \sns{integernumbers?integer}

& \stinline|1|, \stinline|-5|, \stinline|0|, \ldots\\\hline

\definame[post=s]{python-datatypes?float} & \stinline|float|

& \sr{floating-point-number?float}{floating point numbers}

& \stinline|1.2|, \stinline|.125|, \stinline|-1.0|, \ldots\\\hline

\definame[post=s]{string} & \stinline|str|

& \sr{words?word}{strings}

& \stinline|"Hello"|, \stinline|'Hello'|, \stinline|"123"|, \stinline|'a'|, \ldots\\\hline

\definame[post=s]{Boolean} & \stinline|bool|

& \sns{truthvalues?truth value}

& \stinline|True|, \stinline|False|\\\hline

\definame[post=es]{python-datatypes?complex} & \stinline|complex|

& \sns{complexnumbers?complex number}

& \stinline|2+3j|, \ldots\\\hline

{tabular}

{center}

{sdefinition}

We will encounter more \sns{type?type} later.

{itemize}

{frame}

{nparagraph}
We will now see what we can -- and cannot -- do with \sr{type?type}{data types}, this becomes most noticable in \sns{variable assignment} which establishes a \sn{type?type} for the variable (this cannot be change any more) and in the application of \sns{operator?operator} to \sns{subroutine?argument} (which have to be of the correct \sn{type?type}).
{nparagraph}

{frame}[label=slide.datatypes2]
{Data Types in \python (continued)}
{itemize}

The type of a \sn{program-variable?variable} is automatically determined in the first \sn{variable assignment}\lec{before that the variable is unbound} \lstinputmhlisting[linerange=1-5]{python/code/type.py}

{sparagraph}[title=Hint]
The \python \sn{subroutine?function} \lstinline|type| to computes the \sn{type}\lec{don't worry about the \lstinline|class| bit}

{sparagraph}
{itemize}
{frame}

{frame}[label=slide.datatypes3]
{Data Types in \python (continued)}
{itemize}

{sassertion}[style=observation]
\python is strongly typed, i.e. types have to match
{sassertion}

Use data type conversion \sns{subroutine?function} \lstinline|int()|, \lstinline|float()|, \lstinline|complex()|, \lstinline|bool()|, and \lstinline|str()| to adjust types

{sexample}[title=Type Errors and Type Coersion]
\lstinputmhlisting[linerange=6-12]{python/code/type.py}

{sexample}
{itemize}
{frame}
{smodule}
{document}

{sfragment}
{document}

File: [courses/FAU/IWGS/course]{progintro/sec/control.en}]
{document}

{sfragment}[id=sec.control]{Python Control Structures}
File: [courses/Jacobs/GenICT/course]{python/slides/branchloop.en}]
{document}
{smodule}{branchloop}

{nparagraph}
So far, we only know how to make \sns{program?program} that are a simple

sequence of \sns{information-processing-system?instruction} no repetitions, no alternative pathways. \sref[fallback=Swapping values of variables,file=python/slides/variables.en]{ex.swap-variables} is a perfect example. We will now change that by introducing \sns{control structure}, i.e complex \sn{program?program} \sns{information-processing-system?instruction} that change the \sn{control flow} of the \sn{program?program}.

{nparagraph}

{frame}

{Conditionals and Loops}

{itemize}

{sparagraph}[title=Problem]

Up to now \sn{program?program} seem to execute all the \sns{information-processing-system?instruction} in sequence, from the first to the last.\lec{a linear \sn{program?program}}

{sparagraph}

\inputref[smglom/computing]{mod/control-flow.en}

\inputref[smglom/computing]{mod/conditional-execution.en}

{sdefinition}

A \define{condition} is a \sn{python-datatypes?Boolean} \sn{program-expression?expression} in a \sn{control structure}.

{sdefinition}

\inputref[smglom/computing]{mod/loop.en}

{sexample}

In \python, \sns{condition} are constructed by applying a \sn{Boolean} operator to arguments, e.g.

\stinline|3>5|, \stinline|x==3|, \stinline|x!=3|, \ldots\

or by combining simpler conditions by Boolean connectives \stinline|or|, \stinline|and|, and \stinline|not| (using brackets if necessary), e.g.

\stinline|x>5 or x<3|

{sexample}

{itemize}

{frame}

{smodule}

{document}

File: [courses/Jacobs/GenICT/course]{python/snip/python-branchloop.en}]

{document}

{sparagraph}

\usemodule{python/slides?branchloop}

After this general introduction -- \sn{conditional execution} and \sns{loop}) are supported by all \sns{programming language} in some form -- we will see how this is realized in \python

{sparagraph}

{document}

File: [courses/Jacobs/GenICT/course]{python/slides/branching.en}]

{document}

{smodule}{python-branching}

\lstset{language=python,aboveskip=2pt,belowskip=0pt}

{frame}[label=slide.branching1]
{Conditionals in \python}
{itemize}

{sdefinition}
\Sn{conditional execution} via \lstinline|if|/\lstinline|else|
statements
{columns}
{column}{5cm}
\lstinputmhlisting[mathescape]{python/code/branch_schema.py}
\cmhtikzinput[width=5cm]{python/tikz/indenting}
{column}
{column}{6cm}
\def\myxscale{2}\def\myyscale{1.2}
\mhtikzinput[width=6cm]{python/tikz/ifthenelse}
{column}
{columns}
{sdefinition}
\pmetavar{then-part} and \pmetavar{else-part} have to be indented
equally.\lec{e.g. 4 blanks}
If \sns{control structure} are nested they need to be further indented
consistently.
{itemize}
{frame}

{nparagraph}
\python uses indenting to signify nesting of body parts in control structures
-- and other structures as we will see later. This is a very un-typical syntactic choice
in \sns{program?programming language}, which typically use brackets, braces,
or other paired delimiters to indicate nesting and give the freedom of choice in
indenting to \sns{programmer}. This freedom is so ingrained in
\sn{programming} practice, that we emphasize the difference here. The following
example shows \sn{conditional execution} in action.
{nparagraph}

{frame}[label=slide.branching2]
{Conditional Execution Example}
{itemize}

{sexample}[title=Empathy in \python,id=ex.python-empathy]\strut
\lstinputmhlisting{python/code/branching.py}
Note the indenting of the body parts.
{sexample}

{sparagraph}[title=BTW]
\lstinline|input| is an operator that prints its argument string, waits for user
input, and returns that.
{sparagraph}
{itemize}
{frame}

{nparagraph}
But \sn{conditional execution} in \python has one more trick

up its sleeve: what we can do with two branches, we can do with more as well.

{nparagraph}

{frame}

{Variant: Multiple Branches}

{itemize}

{sparagraph}

Making multiple \sn[post=es]{branch} is similar

\stinputmhlisting[mathescape]{python/code/mbranch_schema.py}

{itemize}

The there can be more than one \stinline|elif| clause.

The \pmetavar{condition}s are evaluated from top to bottom and the \pmetavar{then-part} of the first one that comes out \stinline|true| is executed. Then the whole \sn{control structure} is exited.

multiple \sn[post=es]{branch} could achieved by nested \stinline|if|/\stinline|else| structures.

{itemize}

{sparagraph}

{sexample}[title=Better Empathy in \python]

In \sref{ex.python-empathy} we print \stinline|Good!| even if the input is e.g.

\stinline|I feel terrible|, so extend \stinline|if|/\stinline|else| by

\stinputmhlisting{python/code/empathy2.py}

{sexample}

{itemize}

{frame}

{nparagraph}

Note that the \stinline|elif| is just “syntactic sugar” that does not add anything new to the language: we could have expressed the same functionality as two nested if/else statements

\stinputmhlisting[mathescape]{python/code/elif_schema.py}

But this would have introduced an additional layer of nesting (per \stinline|elif| clause in the original). The nested syntax also obscures the fact that all branches are essentially equal.

{nparagraph}

{smodule}

{document}

File: [courses/Jacobs/GenICT/course]{python/slides/looping.en}]

{document}

{smodule}{python-looping}

\stset{language=python}

{nparagraph}

Now let us see the syntax for \sns{loop} in \python.

{nparagraph}

{frame}[label=slide.looping1]

{\Sns{loop} in \python}

{itemize}

{sdefinition}

\python makes \sns{loop} via \lstinline|while| blocks

```
{columns}
{column}{5cm}
{itemize}
```

syntax of the \lstinline|while| \sn{loop}

\lstinputmhlisting[mathescape]{python/code/loop_schema.py}

breaking out of \sns{loop} with \lstinline|break|

skipping the current \sn{loop?body} with \lstinline|continue|

\pmetavar{body} must be indented!

```
{itemize}
{column}\quad
{column}{6cm}
```

\mhtikzinput{python/tikz/while}

```
{column}
{columns}
{sdefinition}
{itemize}
{frame}
```

{nparagraph}

As always we will fortify our intuition with a couple of small examples.

{nparagraph}

{frame}[label=slide.looping2]

{Examples of Loops}

{itemize}

{sexample}[title=Counting in python,id=ex.python-count]\strut

\lstinputmhlisting{python/code/while.py}

This is the standard pattern for using \lstinline|while|: using a loop variable (here \lstinline|count|) and incrementing it in every pass through the loop.

{sexample}

{sexample}[title=Breaking an unbounded Loop,id=ex.python-break]\strut

\lstinputmhlisting{python/code/break.py}

{sexample}

{itemize}

{frame}

{nparagraph}

\sref{ex.python-count} and \sref{ex.python-break} do the same thing: counting from zero to four, but using different mechanisms. This is normal in \sn{programming} there is not “one correct solution”. But the first solution is the “standard one”, and is preferred, since it is shorter and more readable. The \lstinline|break| functionality shown off in the second one is still very useful. Take for instance the problem of computing the product of the numbers -10 to 1.000.000. The naive \sn{implementation} of this is on the left below which does a lot of unnecessary work, because as soon as we passed 0, then the whole product must be zero. A more \sn{efficient} \sn{implementation} is on the right which breaks after seeing the first zero.

{center}

{tabular}{p{3.8cm}@{\quad}p{4cm}}

Direct \Sn{implementation} & More \Sn{efficient} \\\

\lstinputmhlisting{python/code/product.py} &

\lstinputmhlisting{python/code/product-break.py}

{tabular}
{center}
{nparagraph}

{frame}[label=slide.looping3]
{Examples of Loops}
{itemize}

{sexample}[title=Exceptions in the Loop]\strut
\stininputmhlisting{python/code/continue.py}
{sexample}
{itemize}
{frame}

{smodule}
{document}

{sfragment}
{document}

{sfragment}
{document}

File: [courses/FAU/IWGS/course]{progintro/sec/tcs.en}]
{document}
{sfragment}[id=sec.iwgs-tcs]{Some Thoughts about Computers and Programs}
File: [courses/Jacobs/GenICT/course]{programming/slides/tcs-nutshell.en}]
{document}
{smodule}{tcs-nutshell}

{nparagraph}
Finally, we want to go over a couple of general issues pertaining to
\sns{program?program} and (universal) machines. We will just go over them to get the
intuitions -- which are central for understanding
\sr{computer-science?CS}{computer science} and let the lecture “Theoretical
Computer Science” fill in the details and justifications.
{nparagraph}

{frame}[label=slide.tcs-nutshell]
{Computers as Universal Machines (a taste of theoretical CS)}
{itemize}
<1->
{sparagraph}[title=Observation]
\Sns{computer?computer} are \inlinedef{\definame{universal} tools: their
behavior is determined by a \sn{program?program}; they can do anything, the
\sn{program?program} specifies.)
{sparagraph}
<1->
{sparagraph}[title=Context]
Tools in most other disciplines are specific to particular tasks.\lec{except in
e.g. ribosomes in cell biology}

{sparagraph}

<2->

{sassertion}[title=Deep Fundamental Result,style=remark,name=non-computability]

There are things no \sn{computer?computer} can compute.

{sassertion}

<2->

{sexample}[id=ex.halting-problem,for=non-computability]

There cannot be a \sn{program} that decides whether another \sn{program?program} will terminate in \sn{finite} time.

{sexample}

<3->

{sassertion}[style=remark,title=Church-Turing Hypothesis]

There are two classes of languages\inlinedef{

{itemize}

\definame{Turing-complete?Turing complete} (or

\definiendum{Turing-complete?Turing complete}{computationally universal}) ones

that can compute what is theoretically possible.

\definame[post=s]{data-language?data language} that cannot. \lec{but describe data sets}

{itemize}}

{sassertion}

<3->

{sassertion}[style=observation,title=Turing Equivalence]

All \sns{program?programming language} are (made to be)

\sr{Turing complete}{universal}, so they can compute exactly the

same.\lec{\sns{compiler?compiler}\sns{interpreter?interpreter}}

exist}

{sassertion}

<3->

{sparagraph}[title=\ldots in particular \ldots]

Everybody who tells you that one \sns{program?programming language} is the best has

no idea what they're talking about\lec{though differences in \sn{efficiency},

convenience, and beauty exist}

{sparagraph}

{itemize}

{frame}

{frame}[label=slide.ai-nutshell]

{Artificial Intelligence}

{itemize}

{sparagraph}[title=Another Universal Tool]

The human mind. \lec{We can understand/learn anything.}

{sparagraph}

{sparagraph}[title=Strong Artificial Intelligence]

claims that the brain is just another \sn{computer?computer}.

{sparagraph}

{sparagraph}[style=start,title=If that is true]

then

{itemize}

the human mind underlies the same restrictions as computational machines

we may be able to find the “mind-program”.

{itemize}

{sparagraph}

{itemize}
{frame}
{smodule}
{document}

File: [courses/FAU/IWGS/course]{progintro/slides/proglang-synsem.en}]
{document}
{smodule}{proglang-synsem}
\usemodule[courses/Jacobs/ComSem]{\nintro/slides?compositionality}

{nparagraph}
We now come to one of the most important, but maybe least acknowledged principles of
\sns{program?programming language}: The \sr{compositional}{principle of
compositionality}. To fully understand it, we need to fix some fundamental vocabulary.
{nparagraph}

{frame}
{Top Principle of Programming: \Sn[post=ity]{compositional}}
{itemize}

{sassertion}[style=observation]
Modern \sns{program?programming language} compose various
\sns{primitive} and give them a pleasing, concise, and uniform \sn{syntax}.
{sassertion}

{sparagraph}[title=Question]
What does all of this even mean?
{sparagraph}

{sdefinition}
In a \sn{program?programming language}, a \definame{primitive} is a
“basic unit of processing”, i.e. the simplest element that can be given a
procedural meaning (its \definame{semantics}) of its own.
{sdefinition}

{sdefinition}[title={\Sn[post=ity]{compositional}}]
All \sns{program?programming language} provide
\definame[post=s]{composition principle} that allow to \definame{proglang-synsem?compose} smaller
program fragments into larger ones in such a way, that the \sn{semantics} of
the larger is determined by the \sn{semantics} of the smaller ones and that of
the \sn{composition principle} employed.
{sdefinition}

{sassertion}[style=observation]
The \sn{semantics} of a \sn{program?programming language}, is determined
by the meaning of its \sns{primitive} and
\sns{composition principle}.
{sassertion}

{sdefinition}
\Sn{program?programming language} \definame{syntax} describes the surface form
of the program: the admissible character sequences. It is also a composition of the
\sn{syntax} for the \sns{primitive}.
{sdefinition}

{itemize}
{frame}

{nparagraph}

All of this is very abstract -- it has to be as we have not fixed a \sn{programming language} yet and you will only understand the true impact of the \sr{compositional}{compositionality principle} over time and with \sn{programming} experience. Let us now see what this means concretely for our course.

{nparagraph}

{frame}

{Consequences of \Sn[post=ity]{compositional}}

{itemize}

{sassertion}[style=observation]

To understand a \sn{program?programming language}, we (only) have to understand its \sns{primitive}, \sns{composition principle}, and their \sn{syntax}.

{sassertion}

{sdefinition}

The “art of \define{programming?programming}” consists of \sr{proglang-synsem?compose}{composing} the \sns{primitive} of a \sn{program?programming language}.

{sdefinition}

{sassertion}[style=observation]

We only need very few -- about half a dozen -- \sns{primitive} to obtain a \sn{Turing-complete?Turing complete} \sn{program?programming language}.

{sassertion}

{sassertion}[style=observation]

The space of program behaviors we can achieve by \sn{programming} is \sns{infinite} large nonetheless.

{sassertion}

{sassertion}[style=remark]

More \sns{primitive} make \sn{programming} more convenient.

{sassertion}

{sassertion}[style=remark]

\Sns{primitive} in one language can be composed in others.

{sassertion}

{itemize}

{frame}

{smodule}

{document}

File: [courses/FAU/IWGS/course]{progintro/slides/little-large-proglang.en}]

{document}

{smodule}{little-large-proglang}

{frame}

{A note on Programming: Little vs. Large Languages}

{itemize}

{sassertion}[style=observation]

Most such concepts can be studied in isolations, and some can be given a syntax on their own.\lec{standardization}

{sassertion}

{sparagraph}[title=Consequence]

If we understand the concepts and syntax of the sublanguages, then learning another \sn{program?programming language} is relatively easy.

{sparagraph}

{itemize}

{frame}

{smodule}

{document}

{sfragment}

{document}

File: [courses/FAU/IWGS/course]{progintro/sec/more-python.en}]

{document}

{sfragment}[id=sec.more-python]{More about Python}

File: [courses/FAU/IWGS/course]{progintro/snip/more-intro.en}]

{document}

{sparagraph}

\usemodule[courses/Jacobs/GenICT/course]{python/slides/nutshell?python-nutshell}

After we have had some general thoughts about programming in general, we can get back to concrete \python facilities and idioms. We will concentrate on those -- there are lots and lots more -- that are useful in \useSGvar{courseacronym}.

{sparagraph}

{document}

File: [courses/FAU/IWGS/course]{progintro/sec/sequences.en}]

{document}

{sfragment}[id=sec.python-sequences]{Sequences and Iteration}

File: [courses/Jacobs/GenICT/course]{python/snip/sequences-intro.en}]

{document}

{sparagraph}

\usemodule{python/slides/dictionaries?python-dictionaries}

We now come to a commonly used class of objects in \python: sequences, such as \sns{list?list}, sets, tuples, \sns{python-sequences?range}, and \sr{dictionary?dictionary}{dictionaries}.

They are used for storing, accumulating, and accessing objects in various ways in programs. They all have in common, that they can be used for \sr{python-forloop?iteration}{iteration}, thus creating a uniform interface to similar functionality.

{sparagraph}

{document}

File: [courses/Jacobs/GenICT/course]{python/slides/lists.en}]

{document}

{smodule}{python-lists}

\lstset{language=python}

{frame}[fragile]

{Lists in \python}

{itemize}

{sdefinition}

A \define{list?list} is a \sn{finite-cardinality?finite}

\sn{sequences?sequence} of objects, its \define{container?element}.

{sdefinition}

In \sns{program?programming language}, \sns{list} are used for locally storing and passing around collections of objects.

In \python \sns{list} can be written as a sequence of comma separated expressions between square brackets.

{sdefinition}

We call \stininline[mathescape][[\$\pmetavar{seq}\$]] the \define{list constructor}.

{sdefinition}

{sexample}[title=Three lists]

\Sns{container?element} can be of different \sns{type?type}

in \python \stininputmhlisting[linrange=1-3]{python/code/lists.py}

{sexample}

{sexample}[id=python-lists.ex]

\Sn{list} \sns{container?element} can be accessed by specifying ranges

{center}\vspace*{-1.2em}

{tabular}{p{2cm}@{\qqquad}p{2.5cm}@{\qqquad}p{2.5cm}}

\stininputmhlisting[linrange=5-6]{python/code/lists.py} &

\stininputmhlisting[linrange=8-9]{python/code/lists.py} &

\stininputmhlisting[linrange=11-12]{python/code/lists.py}

{tabular}\vspace*{-1.5em}

{center}

{sexample}

{itemize}

{frame}

{nparagraph}

As \sref{python-lists.ex} shows, \python treats counting in list

\sns{accessor} somewhat peculiarly. It starts counting with zero when counting from the front and with one when counting from the back.

{nparagraph}

{smodule}

{document}

File: [courses/Jacobs/GenICT/course]{python/slides/sequences.en}]

{document}

{smodule}{python-sequences}

\usemodule{python/slides/forloop?python-forloop}
\lstset{language=python}

{nparagraph}

But \sns{list?list} are not the only things in \python that can be accessed in the way shown in
\sref[file=python/slides/lists.en]{python-lists.ex}. \python introduces a special class of types the \sn{python-sequences?sequence} types.

{nparagraph}

{frame}[fragile]

{Sequences in \python}

{itemize}

{sdefinition}

\python has more \sns{type?type} that behave just like
\sns{list?list}, they are called \define{python-sequences?sequence}
\sns{type?type}.

{sdefinition}

The most important \sn{python-sequences?sequence} \sns{type?type}
for \useSGvar{courseacronym} are \sns{list?list},
\sns{python-datatypes?string} and
\sns{python-sequences?range}.

{sdefinition}

A \define{python-sequences?range} is a \sn{finite-cardinality?finite}
\sn{python-sequences?sequence} of numbers it can conveniently be constructed by
the \lstinline|range| \sn{subroutine?function}:
\lstinline[mathescape]|range(\$\pmetavar{start}\$,\$\pmetavar{stop}\$,\$\pmetavar{step}\$)|
constructs a \sn{python-sequences?range} from \$\pmetavar{start}\$ (inclusive) to
\$\pmetavar{stop}\$ (exclusive) with step size \$\pmetavar{step}\$.

{sdefinition}

{sexample}

Lists can be constructed from \sns{python-sequences?range}:
\lstinputmhlisting[linenumber=14-15]{python/code/lists.py}
\lstinline|range(1,6,2)| makes a “range” from 1 to 6 with step 2, \lstinline|list|
makes it a list.

{sexample}

{itemize}

{frame}

{nparagraph}

\Sns{python-sequences?range} are useful, because
they are easily and flexibly constructed for \sn{python-forloop?iteration} (up
next).

{nparagraph}

{smodule}

{document}

File: [courses/Jacobs/GenICT/course]{python/slides/forloop.en}]

{document}

{smodule}{python-forloop}

\lstset{language=python}

```
{frame}
  {Iterating over Sequences in \python}
{itemize}
```

```
{sdefinition}
A \define{for loop} \define[post=s]{iterate} a \sn{program?program} fragment over a
\sn{python-sequences?sequence}; we call the process
\define{iteration}. \python uses the following general syntax:
\lstinputmhlisting[mathescape]{python/code/forloop_schema.py}
{sdefinition}
```

```
{sexample}
A \lstinline|range| \sn{subroutine?function} makes an \sn{python-sequences?sequence} over which we
can iterate. \lstinputmhlisting{python/code/forloop3.py}
{sexample}
```

```
{sexample}
\Sns{list?list} and \sns{python-datatypes?string} can also
act as \sns{python-sequences?sequence}. \lec{try it}
\lstinputmhlisting{python/code/for-reverse.py}
{sexample}
{itemize}
{frame}
{smodule}
{document}
```

```
File: [courses/Jacobs/GenICT/course]{python/slides/dictionaries.en}]
{document}
{smodule}{python-dictionaries}
\lstset{language=python,aboveskip=0pt,belowskip=0pt,mathescape}
```

```
{nparagraph}
But \sns{list?list} are not the only \sn{data structure} for
collections of objects. \python provides others that are organized slightly
differently for different applications. We give a particularly useful example here:
\sr{dictionary?dictionary}{dictionaries}.
{nparagraph}
```

```
{frame}[label=slide.dictionaries1]
{ \python \sr{dictionary?dictionary}{Dictionaries}}
{itemize}
```

```
{sdefinition}
A \define{dictionary?dictionary} is an unordered collection of
\sr{pair?pair}{ordered pairs}  $\text{\pair{k}v}$ , where we call  $k$  the
\define{dictionary?key} and  $v$  the \define{dictionary?value}.
{sdefinition}
```

In \python \sr{dictionary}{dictionaries} are written with curly brackets, pairs are separated by commas, and the \sn{dictionary?value} is separated from the \sn{key} by a colon.

```
{sexample}
\sr{dictionary}{Dictionaries} can be used for various purposes,
```

```
{center}\vspace*{-.5em}
{tabular}{p{4.9cm}p{3.1cm}p{2cm}}
\lstinputmhlisting{python/code/painting.py}&
\lstinputmhlisting{python/code/dict_de_en.py}&
\lstinputmhlisting{python/code/enum.py}
{tabular}\vspace*{-1em}
{center}
{sexample}
\sr{dictionary}{Dictionaries} and \sns{python-sequences?sequence}
can be nested, e.g. for a \sn{list?list} of paintings.
{itemize}
{frame}
```

```
{nparagraph}
\sr{dictionary}{Dictionaries} give “keyed access” to collections of data: we can
access a \sn{dictionary?value} via its \sn{key}. In particular, we do not have to remember
the position of a \sn{dictionary?value} in the collection.
{nparagraph}
```

```
{frame}[fragile,label=slide.dictionaries2]
{Interacting with Dictionaries}
{itemize}
```

```
{sexample}[title=Dictionary operations]
{itemize}
\lstinline|painting["title"]| returns the \sn{functions?value} for the \sn{key}
\lstinline|"title"| in the dictionary \lstinline|painting|.
\lstinline|painting["title"]="De Nachtwacht"| changes the \sn{dictionary?value} for
the \sn{key} \lstinline|"title"| to its original Dutch \lec{or adds item}
\lstinline|"title": "De Nachtwacht"|}
{itemize}
{sexample}
```

```
{sexample}[title=Printing Keys and Values]
{center}\lstset{basicstyle=\small\sf}
{tabular}{p{3.1cm}@{\qqquad}p{3.4cm}@{\qqquad}p{3.7cm}}
\sns{key} & \sns{dictionary?value} & \sn{key}/\sn{dictionary?value} \sns{pair?pair}\\
\lstinputmhlisting[linerange=1-2]{python/code/fordict.py} &
\lstinputmhlisting[linerange=3-4]{python/code/fordict.py} &
\lstinputmhlisting[linerange=5-6]{python/code/fordict.py}
{tabular}\vspace*{-1.5em}
{center}
{sexample}
More \sn{dictionary} commands:
{itemize}
\lstinline|if $\pmetavar{key}$ in $\pmetavar{dict}$| checks whether
$\pmetavar{key}$ is a \sn{key} in $\pmetavar{dict}$.
\lstinline|painting.pop("title")| removes the \lstinline|"title"| item from
\lstinline|painting|.
{itemize}
{itemize}
{frame}
```

```
{nparagraph}
Note that \lstinline|thisdict.keys| has a short form: we can just iterate over the keys
```

using \lstinline|for x in thisdict:|.

{nparagraph}
{smodule}
{document}

{sfragment}
{document}

File: [courses/FAU/IWGS/course]{progintro/sec/inout.en}]

{document}

{sfragment}[id=sec.inout]{Input and Output}

File: [courses/Jacobs/GenICT/course]{python/snip/fileio-intro.en}]

{document}

{sparagraph}

\usemodule{python/slides/regexp?python-regexp}

The next topic of our stroll through \python is one that is more practically useful than intrinsically interesting: file input/output. Together with the \sr{regex}{regular expressions} this allows us to write programs that transform files.

{sparagraph}

{document}

File: [courses/Jacobs/GenICT/course]{python/slides/input-print.en}]

{document}

{smodule}{input-print}

\lstset{language=python,mathescape}

{frame}[label=slide.input-print,fragile]

{Input/Output in \python}

{itemize}

{sparagraph}[title=Recall]

The \sn{cpu?CPU} communicates with the user through

\sr{information-processing-system?input subsystem}{input} devices like keyboards and \sr{information-processing-system?output subsystem}{output} devices like the screen.

{sparagraph}

\Sns{program?programming language} provide special \sns{information-processing-system?instruction} for this.

In \python we have already seen

{itemize}

\lstinline|input(\$\pmetavar{prompt}\$)| for

\sr{information-processing-system?input subsystem}{input} from the keyboard, it returns a \sn{python-datatypes?string}.

\lstinline|print(\$\pmetavar{objects}\$,sep=\$\pmetavar{separator}\$,end=\$\pmetavar{endchar}\$)| for \sr{information-processing-system?output subsystem}{output} to the screen.

{itemize}

But \sns{computer} also supply another object to

\sr{information-processing-system?input subsystem}{input} from and

\sr{information-processing-system?output subsystem}{output} to\lec{up next}

{itemize}

{frame}

```
{smodule}  
{document}
```

File: [courses/Jacobs/GenICT/course]{programming/slides/files.en}
{document}
{smodule}{files}

{nparagraph}
\usemodule[smglom/computing]{mod?operating-system}
We now fix some of the nomenclature surrounding \sns{file?file} and
\sns{file-system?file system} provided by most
\sr{OS}{operating system}. Most \sns{programming language} provide
their own bindings that allow to manipulate \sns{file?file}.
{nparagraph}

{frame}[label=slide.files]
{Secondary (Disk) Storage; Files, Folders, etc.}
{itemize}
\inputref[smglom/computing]{mod/file.en}

{sdefinition}
\Sns{file?file} are identified by a \definame{file-system?file name}
which usually consists of a \definame{file-system?base name} and an
\definame{file-system?extension} separated by a dot character.

\Sns{file} are managed by a \definame{file-system?file system} which organize them
hierarchically into named \definiendum{hfs?directory}{folders} and locate them by a
\definame{path}; a sequence of \sr{hfs?directory}{folder} names. The
\sn{file-system?file name} and the \sn{path} together fully identify a
\sn{file?file}.

{sdefinition}
Some \sns{file-system?file system} restrict the characters allowed in
the \sn{file-system?file name} and/or lengths of the
\sn{file-system?base name} or \sn{file-system?extension}.

{sdefinition}
Once a \sn{file?file} has been \definame[post=ed]{files?open}, the
\sn{cpu?CPU} can \definame{write} to it and \definame{read} from it. After use
a \sn{file} should be \definiendum{close}{closed} to protect it from accidental
\sns{read} and \sns{write}.

{sdefinition}
{itemize}
{frame}

{nparagraph}
\usemodule[smglom/computing]{mod?stream}
\usemodule[smglom/computing]{mod?operating-system}
Many \sr{OS}{operating systems} use \sns{file?file} as a primary computational metaphor,
also treating other resources like \sns{file?file}. This leads to an abstraction of
\sns{file?file} called \sns{stream?stream}, which encompass \sns{file?file} as well as
e.g. keyboards, printers, and the screen, which are seen as objects that can be read
from (keyboards) and written to (e.g. screens). This practice allows flexible use of
\sns{program?program}, e.g. re-directing a the (screen) output of a \sn{program?program}
to a \sn{file?file} by simply changing the output \sn{stream?stream}.

{nparagraph}
{smodule}
{document}

File: [courses/Jacobs/GenICT/course]{python/slides/fileio.en}]

{document}

{smodule}{python-fileio}

\lstset{language=python,mathescape}

{nparagraph}

Now we can come to the \python bindings for the \sn{file?file} input/output operations. They are rather straightforward.

{nparagraph}

{frame}[label=slide.python-fileio1,fragile]

{Disk Input/Output in \python}

{itemize}

{sdefinition}

\python uses \definame[post=s]{file object} to encapsulate all file input/output functionality.

{sdefinition}

In \python we have special

\sns{information-processing-system?instruction} for dealing with

\sns{file?file}:

{itemize}

\lstinline|open(\$\pmetavar{path}\$,\$\pmetavar{iospec}\$)| returns a \sn{file object} \$f\$; \$\pmetavar{iospec}\$ is one of \lstinline|r| (\sn{files?read} only; the default), \lstinline|a| (append \hateq \sn{files?write} to the end), and \lstinline|r+| (\sn{files?read}/\sn{files?write}).

\$f\$\lstinline|.read()| \sns{files?read} the \sn{file?file} represented by \sn{file object} \$f\$ into a \sn{python-datatypes?string}.

\$f\$\lstinline|.readline()| reads a single \sn{file-type?line} from the \sn{file?file} (including the newline \sn{character?character} (\lstinline|\n|) otherwise returns the empty string \lstinline|''|.

\$f\$\lstinline|.write(\$\pmetavar{str}\$)| appends the \sn{python-datatypes?string} \$\pmetavar{str}\$ to the end of \$f\$, returns the number of \sns{character?character} written.

\$f\$\lstinline|.close()| closes \$f\$ to protect it from accidental \sns{files?read} and \sns{files?write}.

{itemize}

{sexample}[title=Duplicating the contents of a file]

\lstinputmhlisting[mathescape,linerange=1-3]{python/code/fileio.py}

{sexample}

{itemize}

{frame}

{nparagraph}

The only interesting thing is that we have to declare our intentions when we \sr{files?open}{opening} a \sn{file?file}. This allows the \sn{file-system?file system} to protect the \sns{file?file} against unintended actions and also optimize the data transfer to the

\sns{storage-device?storage device} involved.
{nparagraph}

{nparagraph}
Let us now look at some examples to fortify our intuition about what we can do with
\sns{file?file} in practice.
{nparagraph}

{frame}[label=slide.python-fileio2,fragile]
{Disk Input/Output in \python (continued)}
{itemize}

{sexample}[title=Reading a file linewise,id=ex.read-linewise]\strut
{center}\vspace*{-1.5em}
{tabular}{p{5.2cm}@{\qquad}p{4.8cm}}
\lstinputmhlisting[linrange=5-10]{python/code/fileio.py} &
\lstinputmhlisting[linrange=12-16]{python/code/fileio.py}
{tabular}\vspace*{-1em}
{center}
{sexample}

If you want to read all the lines of a \sn{file?file} in a list you can also
use \lstinline|list(f)| or \lstinline|f.readlines()|.

{sparagraph}
\usemodule{python/slides/libraries?python-libraries}
For \sn[post=ing]{files?read} a \python file we use the
\lstinline|import(\$\pmetavar{basename}\$)|
\sn{information-processing-system?instruction}
{itemize}
it searches for the \sn{file?file} \lstinline|\$\pmetavar{basename}\$.py|, loads it,
interprets it as \python code, and directly executes it.
primarily used for loading \python
\sr{library}{libraries}\lec{additional functionality}
also useful for loading \python-encoded data\lec{e.g. dictionaries}
{itemize}
{sparagraph}
{itemize}
{frame}

{nparagraph}
The code snippet on the right of \sref{ex.read-linewise} show that \sns{file?file}
can be \sr{python-forloop?iterate}{iterated} over using a
\sn{python-forloop?for loop}: the \sn{file object} is implicitly converted
into a sequences of strings via the \lstinline|readline| method.

{nparagraph}
{smodule}
{document}

{sfragment}
{document}

{sfragment}[id=sec.functions]{Functions and Libraries in Python}
File: [courses/Jacobs/GenICT/course]{python/slides/functions-intro.en}
{document}
{smodule}{python-functions-intro}
\usemodule{python/slides/functions?python-functions}

{nparagraph}
We now come to a general device for organizing and modularizing code provided by most
\sns{program?programming language}, including \python. Like
\sns{program-variable?variable}, \sns{subroutine?function} give
names to \python objects -- here fragments of code -- and thus make them
reusable in other contexts.

{nparagraph}

{frame}
{Functions in \python (Introduction)}
{itemize}

{sparagraph}[title=Observation]
Sometimes \sn{programming} tasks are repetitive
\stininputmhlisting[linerange=1-4,mathescape]{python/code/functions.py}
{sparagraph}

{sparagraph}[title=Idea]
We can automate the repetitive part by \sns{subroutine?function}.
{sparagraph}

{sexample}[for=subroutine?function,id=python function.ex]\strut
We encapsulate the greeting functionality in a \sn{subroutine?function}:
\stininputmhlisting[linerange=6-12,mathescape]{python/code/functions.py}
and use it repeatedly.
{sexample}
\sr{subroutine?function}{Functions} can be a very powerful tool for
structuring and documenting \sns{program?program}\lec{if used correctly}
{itemize}
{frame}

{frame}
{Functions in \python (Example)}
{itemize}

{sexample}[title=Multilingual Greeting]
\usemodule[smglom/computing]{mod?localization}
Given a value for \stinline|lang|
\stininputmhlisting[linerange=14-18,mathescape,basicstyle=\small\sf]{python/code/functions.py}
we can even \sr{localization}{localize} (i.e. adapt to the language specified in
\stinline|lang|) the greeting.
{sexample}
{itemize}
{frame}
{smodule}
{document}

```
{document}
{smodule}{python-functions}
\lstset{language=python}
```

```
{nparagraph}
```

We can now make the intuitions above formal and give the exact \python syntax of \sns{subroutine?function}.

```
{nparagraph}
```

```
{frame}[label=slide.functions,fragile]
{Functions in \python (Definition)}
{itemize}
```

```
{sdefinition}[id=python-function.def]
\varseq{aseq}{1,\ellipses,n}{\comp{a}_{#1}}
\varseq{pseq}{1,\ellipses,n}{\comp{p}_{#1}}
A \python \definame{subroutine?function} is defined by a code snippet of
the form \lstinputmhlisting[mathescape]{python/code/function_schema.py}
{itemize}
```

the indented part is called the \definame{subroutine?body} of
 $\$f\$, \text{lec}\{\text{textwarning: whitespace matters in \python}\}$

the $\$pseq\{i\}\$$ are called \definame[post=s]{subroutine?parameter}, and $\$n\$$ the
\definame{subroutine?arity} of $\$f\$$.

```
{itemize}
```

A \sn{subroutine?function} $\$f\$$ can be \definame[post=ed]{subroutine?call} on
\definame[post=s]{subroutine?argument} $\$aseq!\$$ by writing the
\sn{program-expression?expression} \lstinline[mathescape][$\$f\$(\$aseq!\$)$]. This
executes the \sn{subroutine?body} of $\$f\$$ where the (formal)
\sns{parameter} $\$pseq\{i\}\$$ are replaced by the
\sns{subroutine?argument} $\$aseq\{i\}\$$.

```
{sdefinition}
{itemize}
{frame}
{smodule}
{document}
```

File: [courses/FAU/IWGS/course]{progintro/snip/methods-intro.en}]

```
{document}
{sparagraph}
\usemodule[courses/Jacobs/GenICT/course]{python/slides/functions?python-functions}
\usemodule[smglom/computing]{mod?oop-class}
```

We now come to a peculiarity of an object-oriented language like \python: it
treats types as first-class entities, which can be defined by the user -- they are
called \sr{oop-class?class}{classes} then. We will not go into that here, since we
will not need \sr{oop-class?class}{classes} in \useSGvar{courseacronym}, but have
have to briefly talk about \sns{oop?method}, which are essentially functions, but
have a special notation.

```
{sparagraph}
{document}
```

File: [courses/Jacobs/GenICT/course]{python/slides/methods.en}]

{document}
{smodule}{python-methods}
\lstset{language=python}
\makeatletter\providecommand\lst@tagresetfirst{}\makeatletter

{nparagraph}
\python provides two kinds of \sn{subroutine?function}-like facilities: regular
\sns{subroutine?function} as discussed above and \sns{method}, which come with \python
\sn[post=es]{oop-class?class}. We will not attempt a presentation of \sr{OOP}{object
oriented programming} and its particular \sn{implementation} in \python this would be
beyond the mandate of the \useSGvar{courseacronym} course -- but give a brief
introduction that is sufficient to use \sns{method}.
{nparagraph}

{frame}[fragile]
{Functions vs. Methods in \python}
{itemize}
There is another mechanism that is similar to \sns{subroutine?function}
in \python. \lec{we briefly introduce it here to delineate}

{sparagraph}[title=Background]
Actually, the \sns{type?type} from
\sref[fallback=above,file=python/slides/datatypes.en]{python-basic-datatypes.def}
are \sn[post=es]{oop-class?class}, \ldots
{sparagraph}

{sdefinition}
In \python all \sns{value?value} belong to a
\define{oop-class?class}, which provide special
\sns{subroutine?function} we call
\define[post=s]{oop?method}. \Sns{value?value} are also called
\define[post=s]{oop?object}, to emphasise \sn{oop-class?class}
aspects. \Sn{method} application is written with \define{dot notation}:
\stinline[mathescape]{\$\pmetavar{obj}\$. \$\pmetavar{meth}\$(\$\pmetavar{args}\$)|
corresponds to
\stinline[mathescape]{\$\pmetavar{meth}\$(\$\pmetavar{obj}\$,\$\pmetavar{args}\$)|.
{sdefinition}

{sexample}
\usemodule[smglom/sets]{mod?subword}
Finding the position of a \sr{subword}{substring}
\stinputmhlisting[linerange=1-5]{python/code/methods.py}
{sexample}
{itemize}
{frame}

{frame}[fragile,label=slide.methods2]
{Functions vs. Methods in \python}
{itemize}

{sexample}[title=Functions vs. Methods]
{center}\vspace*{-1em}
{tabular}{p{5.4cm}@{\qqquad}p{5.2cm}}
\stinputmhlisting[linerange=7-8]{python/code/methods.py} &
\stinputmhlisting[linerange=10-13]{python/code/methods.py}

```
{tabular}\vspace*{-1em}
{center}
{sexample}
```

```
{sparagraph}[title=Intuition]
```

Only \sns{method} can change \sns{oop?object},
\sns{subroutine?function} return changed copies (of the \sns{oop?object}
they act on).

```
{sparagraph}
{itemize}
{frame}
```

```
{nparagraph}
```

For the purposes of \useSGvar{courseacronym}, it is sufficient to remember that
\sns{method} are a special kind of \sns{subroutine?function} that employ
the \sn{dot notation}. They are provided by the \sn{oop-class?class} of an
\sn{oop?object}.

```
{nparagraph}
{smodule}
{document}
```

File: [courses/Jacobs/GenICT/course][python/slides/libraries.en]

```
{document}
{smodule}{python-libraries}
\lstset{language=python}
```

```
{nparagraph}
```

It is very natural to want to share successful and useful code with others, be it
collaborators in a larger project or company, or the respective community at
large. Given what we have learned so far this is easy to do: we write up the code in a
(collection of) \python files, and make them available for download. Then
others can simply load them via the \lstinline|import| command.

```
{nparagraph}
```

```
{frame}[label=slide.libraries,fragile]
```

```
{\python Libraries}
{itemize}
```

```
{sparagraph}[title=Idea]
```

\Sns{subroutine?function}, \sn[post=es]{oop-class?class}, and
\sns{oop?method} are re usable, so why not package them up for others to
use.

```
{sparagraph}
```

```
{sdefinition}
```

A \python \definame{library} is a \python \sn{file} with a collection of
\sns{subroutine?function}, \sr{oop-class?class}{classes}, and \sns{oop?method}. It
can be \definame[post=ed]{import} (i.e. loaded and interpreted as a \python program
fragment) via the \lstinline|import| command.

```
{sdefinition}
```

There are \$\geq 150.000\$ libraries for \python \lec{\hateq packages on
\url{http://pypi.org}}

```
{itemize}
```

search for them at \url{http://pypi.org}\lec{e.g. 815 packages for “music”}

`\sn{install}` them with `\stinline[mathescape]|pip install $\pmetavar{package name}$|`
 look at how they were done`\lec{all have links to source code}`
 maybe even contribute back (report issues, improve code,
`\dots)\lec{\sr{floss-intro?FLOSS}{open source}}`
`{itemize}`
`{itemize}`
`{frame}`

`{nparagraph}`
 The `\python` community is an `\sr{floss-intro?FLOSS}{open source}` community,
 therefore many developers organize their code into libraries and license them under
`\sr{floss-intro?FLOSS}{open source} \sns{licensing?license}`.

Software repositories like PyPI (the `\python` Package Index) collect
 (references to) and make them for the package manager `\stinline|pip|`, a
`\sn{program?program}` that downloads `\python` libraries and
`\sns{install}` them on the local machine where the `\stinline|import|` command
 can find them.

`{nparagraph}`
`{smodule}`
`{document}`

`{sfragment}`
`{document}`

File: `[courses/FAU/IWGS/course]{progintro/sec/rtfm.en}`
`{document}`
`{sfragment}[id=sec.rtfm]{A Final word on Programming in \useSGvar{courseacronym}}`

File: `[courses/FAU/IWGS/course]{progintro/snip/rtfm.en}`
`{document}`
`{nparagraph}`

`\usemodule[courses/Jacobs/GenICT/course]{python/slides/nutshell?python-nutshell}`

This leaves us with a final word on the way we will handle programming in this course:
`\useSGvar{courseacronym}` is not a `\sn{programming}` course, and we expect you to
 pick up `\python` from the `\useSGvar{courseacronym}` and web/book resources. So,
 recall:

`{nparagraph}`
`{document}`

File: `[courses/Jacobs/GenICT/course]{python/slides/rtfm.en}`
`{document}`
`{smodule}{python-rtfm}`

`{frame}[label=slide.python-rtfm]`
`{For more information on \python}`
`{center}\huge`
`{sparagraph}`
`\inlinedef{\definame{RTFM} (\hateq “read the fine manuals”)}`
`{sparagraph}`
`{center}`
`{frame}`

{smodule}
{document}

File: [courses/FAU/IWGS/course]{progintro/snip/rtfm-more.en}]

{document}

{nparagraph}

\usemodule[courses/Jacobs/GenICT/course]{python/slides/nutshell?python-nutshell}

Our very quick introduction to \python is intended to present the very basics of \sn{programming} and get \useSGvar{courseacronym} students off the ground, so that they can start using programs as tools for the humanities and social sciences.

But there is a lot more to the core functionality \python than our very quick introduction showed, and on top of that there is a wealth of specialized packages and libraries for almost all computational and practical needs.

{nparagraph}

{document}

{sfragment}

{document}

{sfragment}

{document}

File: [courses/FAU/IWGS/course]{progintro/sec/exercises.en}]

{document}

{nfragment}[id=sec.python-exercises]{Exercises}

\includeproblem{progintro/prob/helloworld.en}

\includeproblem{progintro/prob/yearseconds.en}

\includeproblem{progintro/prob/supercalifragilisticexpialidocious.en}

\includeproblem{progintro/prob/humanreadabletime.en}

\includeproblem{progintro/prob/haiku.en}

\includeproblem{progintro/prob/whatsyourname.en}

\includeproblem{progintro/prob/simplebranching.en}

\includeproblem{progintro/prob/looping.en}

\includeproblem{progintro/prob/temperatures.en}

{nfragment}

{document}

{sfragment}

{document}

File: [courses/FAU/IWGS/course]{progintro/sec/datastructures.en}]

{document}

{sfragment}[id=sec.datastructures]{Numbers, Characters, and Strings}

File: [courses/Jacobs/GenICT/course]{datastructures/snip/intro.en}]

{document}

{sparagraph}

\usemodule[smglom/computing]{mod?programming}

\usemodule[smglom/computing]{mod?computer}

In our basic introduction to \sn{programming} above we have convinced ourselves that we need some basic objects to compute with, e.g. Boolean values for conditionals, numbers to calculate with, and characters to form strings for input and output. In this \currentsectionlevel we will look at how these are represented in the \sn{computer}, which in principle can only store binary digits voltage or no voltage on a wire -- which we think of as 1 and 0.

{sparagraph}

{sparagraph}

\usemodule[smglom/computing]{mod?programming}

\usemodule[smglom/computing]{mod?computer}

In this \currentsectionlevel we look at the representation of the basic data types of \sns{programming language} (numbers and characters) in the \sn{computer}; Boolean values (“True” and “False”) can directly be encoded as binary digits.

{sparagraph}

{document}

File: [courses/Jacobs/TDM/course]{digdocs/slides/digitalobjects.en}}

{document}

{frame}

\usemodule[smglom/computing]{mod?computer}

\usemodule[smglom/arithmetics]{mod?pns-common}

{Documents as Digital Objects}

{itemize}

{sparagraph}[title=Question]

how do texts get onto the \sn{computer}? \lec{after all,

\sns{computer} can only do 0/1}

{sparagraph}

{sparagraph}[title=Hint]

At the most basic level, texts are just \sns{sequence} of

\sns{character}.

{sparagraph}

{sparagraph}[title=Answer]

We have to encode \sns{character} as \sns{sequence} of

\sns{bit}.

{sparagraph}

{sparagraph}[title=We will go into how]

\usemodule[smglom/cs]{mod?bits}

{itemize}

documents are represented as \sns{sequence} of

\sns{character},

\sns{character} are represented as \sns{number},

\sns{number} are represented as \sns{pns-common?bit} (0/1).

{itemize}

{sparagraph}

{itemize}

In addition to manipulating normal objects directly linked to their daily survival, humans also invented the manipulation of place-holders or symbols. A \emph{symbol} represents an object or a set of objects in an abstract way. The earliest examples for symbols are the cave paintings showing iconic silhouettes of animals like the famous ones of Cro-Magnon. The invention of symbols is not only an artistic, pleasurable “waste of time” for humans, but it had tremendous consequences. There is archaeological evidence that in ancient times, namely at least some 8000 to 10000 years ago, humans started to use tally bones for counting. This means that the symbol “bone with marks” was used to represent numbers. The important aspect is that this bone is a

```
{nparagraph}  
{smodule}  
{document}
```

```
{document}
```

```
\usemodule[snglom/computing]{mod?computer}
```

$$\backslash usestructure{pns}$$

{nparagraph}

{nparagraph}

{frame}

- {itemize}

{sdefinition}

{sparagraph}

{sparagraph}

[illegible]

{tabular}

{center}

{sparagraph}

```
{itemize}
```

```
{frame}
```

{\nparagraph}[style=clarification,for=positional number system]

The problem with the \sr{unary natural numbers}{unary number system} is that it uses enormous amounts of space, when writing down large numbers. We obviously need a better representation.

{\nparagraph}

{\smodule}

{\document}

File: [courses/Jacobs/GenICT/course]{datastructures/snip/pns.en}

{\document}

{\sparagraph}

\usemodule{datastructures/slides?natural-numbers} The

\sn{unary natural numbers} are very simple and direct, but they

are neither space-efficient, nor easy to manipulate. Therefore we will use different ways of representing numbers in practice.

{\sparagraph}

{\document}

File: [courses/Jacobs/GenICT/course]{datastructures/slides/pns.en}

{\document}

{\smodule}{pns-intro}

\usestructure{pns}

{\frame}{\fragile}

{\Positional Number Systems}

{\itemize}

{\sparagraph}[title=Problem]

Find a better representation system for \sns{natural number}.

{\sparagraph}

{\sparagraph}[title=Idea]

Build a clever code on the \sn{unary natural numbers}, use position information and

\sn{natarith?addition}, \sn{natarith?multiplication}, and

\sn{natarith?exponentiation}.

{\sparagraph}

\inputref[smglom/arithmetics]{mod/positional-number-system.en}

{\sexample}[for=positional number system]

$\mathstrut\{\set{a,b,c},\phi\}$ with with $\mathstrut\{\mathop{\mathrm{defeq}}\{\mathop{\mathrm{apply}}\phi{a}\}0\}$,

$\mathstrut\{\mathop{\mathrm{fundefeq}}{b}\{\mathop{\mathrm{apply}}\phi{b}\}1\}$, and $\mathstrut\{\mathop{\mathrm{defeq}}\{\mathop{\mathrm{apply}}\phi{c}\}2\}$ is a

\sn{positional number system} for \sn{positional-number-system?base} three. We have

$\mathstrut\{\mathop{\mathrm{apply}}\phi{c,a,b}\} =$

$\mathstrut\{\mathop{\mathrm{natplus}}\{\mathop{\mathrm{nattimes}}[\mathop{\mathrm{cdot}}]\{2,\mathop{\mathrm{natpower}}32\},\mathop{\mathrm{nattimes}}[\mathop{\mathrm{cdot}}]\{0,\mathop{\mathrm{natpower}}31\},\mathop{\mathrm{nattimes}}[\mathop{\mathrm{cdot}}]\{1,\mathop{\mathrm{natpower}}30\}\}$

$= \mathstrut\{\mathop{\mathrm{natplus}}\{18,0,1\} = 19$

$\mathstrut\}$

{\sexample}

{\itemize}

{\frame}

{\nparagraph}[style=transition]%,to=PNS-arithm,from=PNS-intro]

If we look at the unary number system from a greater distance, we see that we are not

using a very important feature of strings here: position. As we only have one letter in our alphabet, we cannot, so we should use a larger alphabet. The main idea behind a positional number system \mathcal{D}_b, ϕ_b is that we encode numbers as strings of digit in \mathcal{D}_b , such that the position matters, and to give these encodings a meaning by mapping them into the unary natural numbers via a mapping ϕ_b .

{nparagraph}
{smodule}
{document}

File: [courses/Jacobs/GenCS/course]{pns/slides/PNS-common.en}]
{document}
{smodule}{PNS-common}

\usestructure{pns}
{frame}
{Commonly Used Positional Number Systems}
{itemize}
\inputref[smglom/arithmetics]{mod/pns-common.en}

{sparagraph}[title=Notation]
Attach the base of \mathcal{D}_b to every number from \mathcal{D}_b . \lec{default: decimal}
{sparagraph}

{sparagraph}[title=Trick]
Group triples or quadruples of pns-common?binary digit into recognizable chunks \lec{add leading zeros as needed}
{footnotesize}
{itemize}

$\text{pnsinbase}\{110\ 0011\ 0101\ 1100\}_2 =$
 $\underbrace{\text{pnsinbase}\{0110\}_2}_{\text{pnsinbase}\{6\}_{16}} \underbrace{\text{pnsinbase}\{0011\}_2}_{\text{pnsinbase}\{3\}_{16}}$
 $\underbrace{\text{pnsinbase}\{0101\}_2}_{\text{pnsinbase}\{5\}_{16}} \underbrace{\text{pnsinbase}\{1100\}_2}_{\text{pnsinbase}\{C\}_{16}}$
 $= \text{pnsinbase}\{635C\}_{16}$
 $\text{pnsinbase}\{110\ 001\ 101\ 011\ 100\}_2 =$
 $\underbrace{\text{pnsinbase}\{110\}_2}_{\text{pnsinbase}\{6\}_8} \underbrace{\text{pnsinbase}\{001\}_2}_{\text{pnsinbase}\{1\}_8} \underbrace{\text{pnsinbase}\{101\}_2}_{\text{pnsinbase}\{5\}_8} \underbrace{\text{pnsinbase}\{011\}_2}_{\text{pnsinbase}\{3\}_8} \underbrace{\text{pnsinbase}\{100\}_2}_{\text{pnsinbase}\{4\}_8}$
 $= 61534_{\{8\}}$

$\text{pnsinbase}\{F3A\}_{16} =$
 $\underbrace{\text{pnsinbase}\{F\}_{16}}_{\text{pnsinbase}\{1111\}_2} \underbrace{\text{pnsinbase}\{3\}_{16}}_{\text{pnsinbase}\{0011\}_2}$
 $\underbrace{\text{pnsinbase}\{A\}_{16}}_{\text{pnsinbase}\{1010\}_2}$
 $= \text{pnsinbase}\{111100111010\}_2, \text{quad } \text{pnsinbase}\{4721\}_8 =$
 $\underbrace{\text{pnsinbase}\{4\}_8}_{\text{pnsinbase}\{100\}_2} \underbrace{\text{pnsinbase}\{7\}_8}_{\text{pnsinbase}\{111\}_2} \underbrace{\text{pnsinbase}\{2\}_8}_{\text{pnsinbase}\{010\}_2} \underbrace{\text{pnsinbase}\{1\}_8}_{\text{pnsinbase}\{001\}_2}$
 $= \text{pnsinbase}\{100111010001\}_2$
{scriptsize}
{itemize}
{footnotesize}
{sparagraph}
{itemize}
{frame}

{nparagraph}[style=introduction]%,to=PNS-intro]

We have all seen \sns{positional number system}: our \sn{decimal} system is one (for the \sn{positional-number-system?base} 10). Other systems that important for us are the \sn{pns-common?binary} system (it is the smallest non degenerate one) and the \sn{octal} (base 8) and \sn{hexadecimal} (base 16) systems. These come from the fact that \sn{pns-common?binary} numbers are very hard for humans to scan. Therefore it became customary to group three or four \sns{digit} together and introduce (compound) \sns{digit} for these groups. The \sn{octal} system is mostly relevant for historic reasons, the \sn{hexadecimal} system is in widespread use as syntactic sugar for \sn{pns-common?binary} \sns{number}, which form the basis for electronic circuits, since \sn{pns-common?binary} \sns{digit} can be represented physically by voltage/no voltage.

{nparagraph}

{smodule}

{document}

File: [courses/Jacobs/GenICT/course][datastructures/slides/pns-arith.en]]

{document}

{smodule}{pns-arith}

{frame}

{Arithmetics in Positional Number Systems}

{itemize}

For \sn{natarith?arithmetic} just follow the rules from elementary school\lec{for the right base}

Tom Lehrer's "New Math": \url{https://www.youtube.com/watch?v=DfCJgC2zezW}

{sexample}

{center}

{tabular}{l@{\qquad\qquad\qquad}l}

\Sn{natarith?addition} \sn{positional-number-system?base} 4 &

\sn{pns-common?binary} \sn{natarith?multiplication}\\

\$ {array}{llll}

& 1 & 2 & 3 \\

+ & 1_1 & 2_1 & 3 \\ \hline

& 3 & 1 & 2

{array}\$ &

\$ {array}{llllll}

&&1&0&1&0\\

&*&1&1&0\\ \hline

&&0&0&0&0\\

&1&0&1&0&\\

1&0&1&0&&\\ \hline

1&1&1&1&0&0

{array}

\$

{tabular}

{center}

{sexample}

{itemize}

{frame}

{smodule}

{document}

File: [courses/Jacobs/GenICT/course]{datastructures/slides/pns-back.en}]

{document}

{smodule}{pns-back}

{frame}

{How to get back to Decimal (or any other system)}

{itemize}

{sparagraph}[title=Observation]

\sref[archive=smglom/arithmetics,file=mod/positional-number-system.en]{positional-number-system.def} specifies how we can get from \sn{positional-number-system?base} b representations to \sn{decimal}. We can always go back to the \sn{positional-number-system?base} b \sr{natural number}{numbers}.

{sparagraph}

{sassertion}[style=observation,name=convert-base.obs]

To convert a \sn{decimal} \sn{number} n to base b , use successive \sn{integer division} (\sn{natarith?division} with \sn{natarith?remainder}) by b :

\stinputmhlisting[mathescape,language=pseudocode]{datastructures/code/pns-convert.pcode}

{sassertion}

{sexample}[for=convert-base.obs,title=Convert 456 to base 8]

Result: $\text{pnsinbase}(710)8$

{center}

{tabular}{ll}

$\text{\natdiv}\{456\}8=57$ & $\text{\natmod}\{456\}8=0$

$\text{\natdiv}\{57\}8=7$ & $\text{\natmod}\{57\}8=1$

$\text{\natdiv}\{7\}8=0$ & $\text{\natmod}\{7\}8=7$

{tabular}

{center}

{sexample}

{itemize}

{frame}

{smodule}

{document}

{sfragment}

{document}

File: [courses/FAU/IWGS/course]{progintro/sec/unicode.en}]

{document}

{sfragment}[id=sec.character-encodings]{Characters and their Encodings: ASCII and UniCode}

File: [courses/Jacobs/GenICT/course]{datastructures/snip/encoding-intro.en}]

{document}

{sparagraph}

\usemodule[smglom/computing]{mod?computer}

\usemodule[smglom/www]{mod?utfcodes}

IT systems need to encode \sns{character} from our \sns{alphabet} as \sn{bit} strings (sequences of \sr{bit}{binary digits} (\sns{bit}) 0 and 1) for representation in \sns{computer}. To understand the current state -- the \sn{unicode} standard -- we will take a historical perspective.

{sparagraph}

{document}

File: [courses/Jacobs/GenCS/course]{codes/snip/encoding-standardization.en}]

{document}

{sparagraph}

It is important to understand that encoding and decoding of characters is an activity that requires standardization in multi-device settings -- be it sending a file to the printer or sending an e-mail to a friend on another continent. Concretely, the recipient wants to use the same character mapping for decoding the sequence of bits as the sender used for encoding them -- otherwise the message is garbled.

We observe that we cannot just specify the encoding table in the transmitted document itself, (that information would have to be en/decoded with the other content), so we need to rely document-external external methods like standardization or encoding negotiation at the meta-level. In this \currentsectionlevel we will focus on the former.

{sparagraph}

{document}

File: [courses/Jacobs/GenCS/course]{codes/slides/ASCII.en}]

{document}

{smodule}{ASCII}

\symdef{ASCIIINUL}{\comp{\mathtt{NUL}}}

\symdef{ASCIIISOH}{\comp{\mathtt{SOH}}}

\symdef{ASCIISTX}{\comp{\mathtt{STX}}}

\symdef{ASCIIETX}{\comp{\mathtt{ETX}}}

\symdef{ASCIIEOT}{\comp{\mathtt{EOT}}}

\symdef{ASCIIENQ}{\comp{\mathtt{ENQ}}}

\symdef{ASCIIACK}{\comp{\mathtt{ACK}}}

\symdef{ASCIIBEL}{\comp{\mathtt{BEL}}}

\symdef{ASCIIBS}{\comp{\mathtt{BS}}}

\symdef{ASCIIHT}{\comp{\mathtt{HT}}}

\symdef{ASCIILF}{\comp{\mathtt{LF}}}

\symdef{ASCIIVT}{\comp{\mathtt{VT}}}

\symdef{ASCIIFX}{\comp{\mathtt{FX}}}

\symdef{ASCIICR}{\comp{\mathtt{CR}}}

\symdef{ASCIIISO}{\comp{\mathtt{SO}}}

\symdef{ASCIISI}{\comp{\mathtt{SI}}}

\symdef{ASCIIDLE}{\comp{\mathtt{DLE}}}

\symdef{ASCIIDCone}{\comp{\mathtt{DC1}}}

\symdef{ASCIIDCtwo}{\comp{\mathtt{DC2}}}

\symdef{ASCIIDCthree}{\comp{\mathtt{DC3}}}

\symdef{ASCIIDCfour}{\comp{\mathtt{DC4}}}

\symdef{ASCIIINAK}{\comp{\mathtt{NAK}}}

\symdef{ASCIISYN}{\comp{\mathtt{SYN}}}

\symdef{ASCIIETB}{\comp{\mathtt{ETB}}}

\symdef{ASCIICAN}{\comp{\mathtt{CAN}}}

\symdef{ASCIIEM}{\comp{\mathtt{EM}}}

\symdef{ASCIISUB}{\comp{\mathtt{SUB}}}

\symdef{ASCIIESC}{\comp{\mathtt{ESC}}}

\symdef{ASCIIFS}{\comp{\mathtt{FS}}}

\symdef{ASCIIGS}{\comp{\mathtt{GS}}}

\symdef{ASCIIRS}{\comp{\mathtt{RS}}}


```

\symdef{ASCIIUS}{\comp{\mathtt{US}}}
\symdef{ASCIIspace}{\operatorname{\textvisiblespace}}
\symdef{ASCIIexclamation}{\comp{\mathtt{!}}}
\symdef{ASCIIldquot}{\comp{\mathtt{\char34}}}
\symdef{ASCIIhash}{\comp{\mathtt{\#}}}
\symdef{ASCIIldollar}{\comp{\mathtt{\$}}}
\symdef{ASCIIpercent}{\comp{\mathtt{\char37}}}
\symdef{ASCIIampersand}{\comp{\mathtt{\&}}}
\symdef{ASCIIquote}{\comp{\mathtt{'}}}
\symdef{ASCIIobrack}{\comp{\mathtt{()}}}
\symdef{ASCIIcbrack}{\comp{\mathtt{()}}}
\symdef{ASCIIstar}{\comp{\mathtt{*}}}
\symdef{ASCIIplus}{\comp{\mathtt{+}}}
\symdef{ASCIIcomma}{\comp{\mathtt{,}}}
\symdef{ASCIIldash}{\comp{\mathtt{-}}}
\symdef{ASCIIslash}{\comp{\mathtt{/}}}
\symdef{ASCIIzero}{\comp{\mathtt{0}}}
\symdef{ASCIIone}{\comp{\mathtt{1}}}
\symdef{ASCIItwo}{\comp{\mathtt{2}}}
\symdef{ASCIIthree}{\comp{\mathtt{3}}}
\symdef{ASCIIfour}{\comp{\mathtt{4}}}
\symdef{ASCIIfive}{\comp{\mathtt{5}}}
\symdef{ASCIIsix}{\comp{\mathtt{6}}}
\symdef{ASCIIseven}{\comp{\mathtt{7}}}
\symdef{ASCIIeight}{\comp{\mathtt{8}}}
\symdef{ASCIInine}{\comp{\mathtt{9}}}
\symdef{ASCIIcolon}{\comp{\mathtt{:}}}
\symdef{ASCIIsemicolon}{\comp{\mathtt{;}}}
\symdef{ASCIIless}{\comp{\mathtt{<}}}
\symdef{ASCIIleq}{\comp{\mathtt{=}}}
\symdef{ASCIIgreater}{\comp{\mathtt{>}}}
\symdef{ASCIIquestionmark}{\comp{\mathtt{?}}}
\symdef{ASCIIat}{\comp{\mathtt{@}}}
\symdef{ASCIIA}{\comp{\mathtt{A}}}
\symdef{ASCIIB}{\comp{\mathtt{B}}}
\symdef{ASCIIC}{\comp{\mathtt{C}}}
\symdef{ASCIID}{\comp{\mathtt{D}}}
\symdef{ASCIIE}{\comp{\mathtt{E}}}
\symdef{ASCIIF}{\comp{\mathtt{F}}}
\symdef{ASCIIIG}{\comp{\mathtt{G}}}
\symdef{ASCIIH}{\comp{\mathtt{H}}}
\symdef{ASCIII}{\comp{\mathtt{I}}}
\symdef{ASCIIJ}{\comp{\mathtt{J}}}
\symdef{ASCIIK}{\comp{\mathtt{K}}}
\symdef{ASCIIL}{\comp{\mathtt{L}}}
\symdef{ASCIIM}{\comp{\mathtt{M}}}
\symdef{ASCIIIN}{\comp{\mathtt{N}}}
\symdef{ASCIIIO}{\comp{\mathtt{O}}}
\symdef{ASCIIIP}{\comp{\mathtt{P}}}
\symdef{ASCIIQ}{\comp{\mathtt{Q}}}
\symdef{ASCIIIR}{\comp{\mathtt{R}}}
\symdef{ASCIIS}{\comp{\mathtt{S}}}
\symdef{ASCIIT}{\comp{\mathtt{T}}}
\symdef{ASCIIU}{\comp{\mathtt{U}}}
\symdef{ASCIIIV}{\comp{\mathtt{V}}}

```

```

\symdef{ASCIIW}{\comp{\mathtt{W}}}
\symdef{ASCIIX}{\comp{\mathtt{X}}}
\symdef{ASCIIY}{\comp{\mathtt{Y}}}
\symdef{ASCIIZ}{\comp{\mathtt{Z}}}
\symdef{ASCII\dot}{\comp{\mathtt{.}}}
\symdef{ASCII\osqbrack}{\comp{\mathtt{[]}}}
\symdef{ASCII\backslash}{\comp{\mathtt{\textbackslash}}}
\symdef{ASCII\csqbrack}{\comp{\mathtt{[]}}}
\symdef{ASCII\caret}{\comp{\mathtt{\textasciicircum}}}%
\symdef{ASCII\underscore}{\comp{\mathtt{\_}}}
\symdef{ASCII\backquote}{\comp{\mathtt{'}}}
\symdef{ASCIIa}{\comp{\mathtt{a}}}
\symdef{ASCIIb}{\comp{\mathtt{b}}}
\symdef{ASCIIc}{\comp{\mathtt{c}}}
\symdef{ASCIId}{\comp{\mathtt{d}}}
\symdef{ASCIIe}{\comp{\mathtt{e}}}
\symdef{ASCII\f}{\comp{\mathtt{f}}}
\symdef{ASCIIg}{\comp{\mathtt{g}}}
\symdef{ASCIIh}{\comp{\mathtt{h}}}
\symdef{ASCIIi}{\comp{\mathtt{i}}}
\symdef{ASCIIj}{\comp{\mathtt{j}}}
\symdef{ASCIIk}{\comp{\mathtt{k}}}
\symdef{ASCIIl}{\comp{\mathtt{l}}}
\symdef{ASCIIm}{\comp{\mathtt{m}}}
\symdef{ASCII\m}{\comp{\mathtt{n}}}
\symdef{ASCIIo}{\comp{\mathtt{o}}}
\symdef{ASCIIp}{\comp{\mathtt{p}}}
\symdef{ASCIIq}{\comp{\mathtt{q}}}
\symdef{ASCIIr}{\comp{\mathtt{r}}}
\symdef{ASCII\ss}{\comp{\mathtt{s}}}
\symdef{ASCII\it}{\comp{\mathtt{t}}}
\symdef{ASCIIu}{\comp{\mathtt{u}}}
\symdef{ASCIIv}{\comp{\mathtt{v}}}
\symdef{ASCIIw}{\comp{\mathtt{w}}}
\symdef{ASCIIx}{\comp{\mathtt{x}}}
\symdef{ASCIIy}{\comp{\mathtt{y}}}
\symdef{ASCIIz}{\comp{\mathtt{z}}}
\symdef{ASCII\lbrace}{\comp{\mathtt{\{}}}}
\symdef{ASCII\lvert}{\comp{\mathtt{|}}}
\symdef{ASCII\cbrace}{\comp{\mathtt{\}}}
\symdef{ASCII\tilde}{\comp{\mathtt{\textasciitilde}}}
\symdef{ASCII\DEL}{\comp{\mathtt{\DEL}}}

\symdef{bytesendingwith}[args=1]{\scriptscriptstyle\comp\cdots{#1}}
\symdef{bytesstartingwith}[args=1]{\scriptscriptstyle{#1}\comp\cdots}

```

{nparagraph}

The \sn{ASCII} code we will introduce here is one of the first standardized and widely used character encodings for a complete alphabet. It is still widely used today. The code tries to strike a balance between being able to encode a large set of \sns{character?character} and the representational capabilities in the time of punch cards (see below).

{nparagraph}

{frame}

\\$\ASCIIV\$
 & \\$\ASCIIW\$ & \\$\ASCIIx\$ & \\$\ASCIIY\$ & \\$\ASCIIz\$ & \\$\ASCIIosqbrack\$ & \\$\ASCIIbackslash\$
 & \\$\ASCIIcsqbrack\$ & \\$\ASCIIcaret\$ & \\$\ASCIIunderscore\$\\hline
 \\$\bytesstartingwith6\$ & \\$\ASCIIbackquote\$ & \\$\ASCIIa\$ & \\$\ASCIIb\$ & \\$\ASCIIc\$ & \\$\ASCII d\$ &
 \\$\ASCIIe\$ & \\$\ASCII f\$
 & \\$\ASCIIg\$ & \\$\ASCIIh\$ & \\$\ASCIIi\$ & \\$\ASCIIj\$ & \\$\ASCIIk\$ & \\$\ASCIIl\$ & \\$\ASCII m\$
 & \\$\ASCII n\$ & \\$\ASCIIo\$\\hline
 \\$\bytesstartingwith7\$ & \\$\ASCIIp\$ & \\$\ASCIIq\$ & \\$\ASCIIr\$ & \\$\ASCIIs\$ & \\$\ASCII t\$ & \\$\ASCIIu\$ &
 \\$\ASCIIv\$
 & \\$\ASCIIw\$ & \\$\ASCIIx\$ & \\$\ASCIIy\$ & \\$\ASCIIz\$ & \\$\ASCIIlobrace\$ & \\$\ASCIIvbar\$
 & \\$\ASCIIcbrace\$ & \\$\ASCIItilde\$ & \\$\ASCIIDEL\$\\hline

{tabular}
 {scriptsize}
 {center}
 {sdefinition}

{sparagraph}
 The first 32 \sns{character?character} are control \sns{character?character} for \sn{ASCII} devices like
 printers.
 {sparagraph}

{sparagraph}[title=Motivated by punch cards]
 The \sn{character?character} 0 (\$\pnsinbase{0000000}2\$ in
 \sn{pns-common?binary}) carries no information
 \\$\ASCIIINUL\$, \lec{used as dividers}\\
 \Sn{character?character} 127 (\hateq \$\pnsinbase{1111111}2\$) can be used for
 deleting (overwriting) last value\lec{cannot delete holes}
 {sparagraph}

{sparagraph}
 The \sn{ASCII} code was standardized in 1963 and is still prevalent in
 \sns{computer} today.\lec{but seen as US centric}
 {sparagraph}
 {itemize}
 {frame}
 {smodule}
 {document}

File: [courses/Jacobs/GenCS/course]{codes/slides/punchcard.en}]
 {document}
 {smodule}{punchcard}

{nparagraph}
 Punch cards were the preferred medium for long-term storage of programs up to the
 late 1970s, since they could directly be produced by card punchers and automatically
 read by \sns{computer}.
 {nparagraph}

{frame}[label=slide.punchcard]
 {A Punchcard}
 {itemize}

{sdefinition}[id=punchcard.def]
 A \definame{punch card} is a piece of stiff paper that contains digital

information represented by the presence or absence of holes in predefined positions.

{sdefinition}

{sexample}[for=punch card]

This \sn{punch card} encodes the \$\FortranLanguage\$ statement

\stinline[language=Fortran]|Z(1) = Y + W(1)|

\cmhgraphics[width=12cm]{codes/PIC/punchcard}

{sexample}

{itemize}

{frame}

{nparagraph}

\usemodule[smglom/computing]{mod?interactive}

Up to the 1970s, \sns{computer} were batch machines, where the \sn{programmer} delivered the \sn{program} to the operator (a person behind a counter who fed the programs to the \sn{computer}) and collected the printouts the next morning. Essentially, each punch card represented a single \sn{file-type?line} (80 \sns{character?character}) of \sn{program} code. Direct \sn[post=ion]{interact} with a \sn{computer} is a relatively young mode of operation.

{nparagraph}

{smodule}

{document}

File: [courses/Jacobs/GenCS/course]{codes/slides/ASCII-problems.en}]

{document}

{smodule}{ASCII-problems}

{nparagraph}

The \sn{ASCII} code as above has a variety of problems, for instance that the \sns{control character} are mostly no longer in use, the code is lacking many \sns{character?character} of languages other than the English language it was developed for, and finally, it only uses seven \sns{bit}, where an \sn{octet} (eight \sns{bit}) is the preferred unit in information technology. Therefore a whole zoo of extensions were introduced, which --- due to the fact that there were so many of them --- never quite solved the encoding problem.

{nparagraph}

{frame}

{Problems with \sn{ASCII} encoding}

{itemize}

{sparagraph}[title=Problem]

Many of the control \sns{character?character} are obsolete by now/

\lec{e.g. \$\ASCIIINUL\$, \$\ASCIIIBEL\$, or \$\ASCIIIDEL\$}

{sparagraph}

{sparagraph}[title=Problem]

Many European \sns{character?character} are not represented.\lec{e.g. \e,\~n, ,ss,\ldots}

{sparagraph}

{sparagraph}[title=European ASCII Variants]

Exchange less-used \sns{character?character} for national ones.

{sparagraph}

```
{sexample}[title=German \sn{ASCII},for=character encoding]
Remap e.g. $\map\ASCII\osqbrack{\text{ }}$, $\map\ASCII\csqbrack{\text{ }}$ in German
\sn{ASCII}\lec{"{\!inline!Apple }[!]" comes out as "\texttt{Apple  }}"
{sexample}
```

```
{sdefinition}[title=ISO-Latin (ISO/IEC 8859)]
16 Extensions of \sn{ASCII} to 8-bit (256 \sns{character?character})
{footnotesize}
\definame{ISO Latin} 1 \hateq "Western European", \definame{ISO Latin} 6 \hateq
"Arabic", \definame{ISO Latin} 7
\hateq "Greek"\ldots
{footnotesize}
{sdefinition}
```

```
{sparagraph}[title=Problem]
No cursive Arabic, Asian, African, Old Icelandic Runes, Math,\ldots
{sparagraph}
```

```
{sparagraph}[title=Idea]
\usemodule{codes/slides?utfcodes}
Do something totally different to include all the world's scripts: For a scalable
architecture, separate
{itemize}
  what \sns{character?character} are available, and \lec{\sn{character set}}
  a \sr{function}{mapping} from \sn{pns-common?bit} \sr{word}{strings} to
\sns{character?character}.\lec{\sn{character encoding}}
{itemize}
{sparagraph}
{itemize}
{frame}
{smodule}
{document}
```

```
File: [courses/Jacobs/GenCS/course]{codes/slides/unicode-ucs.en}]
{document}
{smodule}{unicode-ucs}
```

```
{nparagraph}
The goal of the $\unicode$ standard is to cover all the worlds scripts (past, present,
and future) and provide \sn{efficient} encodings for them. The only scripts in regular
use that are currently excluded are fictional scripts like the elvish scripts from the
Lord of the Rings or Klingon scripts from the Star Trek series.
{nparagraph}
```

```
{nparagraph}
An important idea behind $\unicode$ is to separate concerns between standardizing the
\sn{character set} --- i.e. the set of encodable \sns{character?character} and the encoding itself.
{nparagraph}
```

```
{frame}[label=slide.unicode-ucs]
{Unicode and the Universal Character Set}
{itemize}
```

{sdefinition}[title=Twin Standards]

A scalable architecture for representing all the worlds writing systems:

{itemize}

The \definiendum{UCS}{universal character set} (\define{UCS}) defined by the ISO/IEC 10646 International Standard, is a standard set of \define[post=s]{character?character} upon which many \sns{character encoding} are based.

The \define{unicode} standard defines a set of standard \sns{character encoding}, rules for normalization, decomposition, collation, rendering and bidirectional display order.

{itemize}

{sdefinition}\vspace*{-1em}\strut

{sdefinition}

Each \sn{UCS} \sn{character?character} is identified by an \sn[pre=un]{ambiguous} name and an \sn{natural number} called its \define{code point}.

{sdefinition}

{sparagraph}

The \sn{UCS} has 1.1 million \sns{code point} and nearly 100\,000 \sns{character?character}.

{sparagraph}

{sdefinition}

Most (non-Chinese) \sns{character?character} have \sns{code point} in $\text{\integerinterval}{1}{65536}$: the \definiendum{BMP}{basic multilingual plane} (\define{BMP}).

{sdefinition}

{sdefinition}[title=Notation,for={unicodepoint,ucsname}]

For \sns{code point} in the (\sn{BMP}), four \sn{hexadecimal} digits are used, e.g. $\text{\unicodepoint}{0058}$ for the \sn{character?character} $\text{\ucsname}{LATIN CAPITAL LETTER X}$;

{sdefinition}

{itemize}

{frame}

{smodule}

{document}

File: [courses/Jacobs/GenCS/course]{codes/slides/utfcodes.en}]

{document}

{smodule}{utfcodes}

{nparagraph}

Note that there is indeed an issue with space-efficient \sns{character encoding} here. \unicode reserves space for $\text{\natpower}{2}{32}$ (more than a million) \sns{character?character} to be able to handle future scripts. But just simply using 32 bits for every \unicode \sn{character?character} would be extremely wasteful: \unicode - $\text{\sn[post=d]}{code?encode}$ versions of \sn{ASCII} files would be four times as large.

{nparagraph}

{nparagraph}

Therefore \unicode allows multiple \sns{character encoding}. \UTFthirtytwo is a simple 32-bit code that directly uses the \sns{code point} in \sn{pns-common?binary}

form. \textasciitilde is optimized for western languages and coincides with the \textasciitilde where they overlap. As a consequence, \textasciitilde encoded texts can be decoded in \textasciitilde without changes --- but in the \textasciitilde encoding, we can also address all other \textasciitilde \textasciitilde (using multi- \textasciitilde \textasciitilde).

{\npagelabel}

{frame}[label=slide.utfcodes]
 {Character Encodings in Unicode}
 {itemize}

{sdefinition}

A \textasciitilde is a mapping from \textasciitilde to \textasciitilde \textasciitilde to \textasciitilde \textasciitilde .

{sdefinition}

{sparagraph}[title=Idea]

\textasciitilde supports multiple \textasciitilde (but not \textasciitilde set) for \textasciitilde .

{sparagraph}

{sdefinition}[title=Unicode Transformation Format,for={UTFeight,UTFsixteen,UTFthirtytwo}]
 {itemize}

\textasciitilde , 8- \textasciitilde , variable width \textasciitilde , which maximizes compatibility with \textasciitilde .

\textasciitilde , 16- \textasciitilde , variable width \textasciitilde \textasciitilde in Asia

\textasciitilde , a 32- \textasciitilde , fixed width \textasciitilde \textasciitilde as a fallback

{itemize}

{sdefinition}

{sdefinition}

The \textasciitilde \textasciitilde follows the following schema:

{small}

{center}

{tabular}{|l|l|l|l|l|}\hline

\textasciitilde & \textasciitilde 1 & \textasciitilde 2 & \textasciitilde 3 & \textasciitilde 4\\hline

\textasciitilde \textasciitilde

& 0xxxxxxx & & \\hline

\textasciitilde \textasciitilde

& 110xxxxx & 10xxxxxx & & \\hline

\textasciitilde \textasciitilde

& 1110xxxx & 10xxxxxx & 10xxxxxx & & \\hline

\textasciitilde \textasciitilde

& 11110xxx & 10xxxxxx & 10xxxxxx & 10xxxxxx \\hline

{tabular}

{center}

{small}

{sdefinition}

{sexample}[for=unicodepoint]

\textasciitilde is encoded as 00100100\lec{1 byte}\\

\textasciitilde is encoded as 11000010,10100010\lec{two bytes}\\

\textasciitilde is encoded as 11100010,10000010,10101100\lec{three bytes}


```
{sexample}
{itemize}
{frame}
```

```
{nparagraph}
```

Note how the fixed `\sn{pns-common?bit}` `\sn[post=es]{subword?prefix}` in the `\UTFeight` `\sr{character encoding}{encoding}` are engineered to determine which of the four cases apply, so that `\UTFeight` encoded documents can be safely decoded.

```
{nparagraph}
{smodule}
{document}
```

File: `[courses/Jacobs/GenCS/course]{codes/slides/unicode-emojis.en}`

```
{document}
```

```
{frame}
```

```
\usemodule[smglom/computing]{mod?character}
```

```
\usemodule[courses/Jacobs/GenCS/course]{codes/slides?unicode-ucs}
```

```
{XKCD's Take on Recent Unicode Extensions}
```

```
{itemize}
```

```
$\unicode$ 6.0 adopted hundreds of emoji \sns{character?character} in 2010
```

```
\lec{2666 in July 2017}
```

```
Modifying \sns{character?character}\lec{\url{https://xkcd.com/1813/}}
```

```
\cmhgraphics[width=11cm]{codes/PIC/xkcd_vomiting_emoji}
```

```
{itemize}
```

```
{frame}
```

```
{frame}
```

```
\usemodule[courses/Jacobs/GenCS/course]{codes/slides?unicode-ucs}
```

```
{XKCD's Take on Recent Unicode Extensions (cont.)}
```

```
{itemize}
```

```
Recent $\unicode$ extensions\lec{\url{https://xkcd.com/1953/}}
```

```
\cmhgraphics[width=11cm]{codes/PIC/xkcd_unicode}
```

```
{itemize}
```

```
{frame}
```

```
{document}
```

```
{sfragment}
```

```
{document}
```

File: `[courses/FAU/IWGS/course]{progintro/sec/stringcomp.en}`

```
{document}
```

```
{sfragment}[id=sec.stringcomp]{More on Computing with Strings}
```

File: `[courses/FAU/IWGS/course]{progintro/snip/literals.en}`

```
{document}
```

```
{sparagraph}
```

```
\usemodule[courses/Jacobs/GenICT/course]{python/slides/string-literals?python-string-literals}
```

We now extend our repertoire on handling and formatting strings in `\python`: we will introduce `\sn{string}` `\sns{literal}`, which allow writing complex strings.

```
{sparagraph}
```

```
{document}
```

File: [courses/Jacobs/GenICT/course]{python/slides/strings.en}]

{document}

{smodule}{python-strings}

\lstset{language=python}

{frame}

{Playing with Strings and Characters in \python}

{itemize}

{sdefinition}[id=pystring.def]

\python \definame[post=s]{string} are sequences of \$\unicode\$

\sns{character?character}.

{sdefinition}

{sparagraph}[style=warning]

In \python, \sns{character?character} are just strings of length 1.

{sparagraph}

\lstinline|ord| gives the \sn{UCS} \sn{code point} of the

\sn{character?character}, \lstinline|chr| \sn{character?character} for a number.

{sexample}[title=Playing with Characters,id=lcuc-chars.ex]\strut

\lstinputmhlisting{python/code/chars.py}

{sexample}

Strings can be accessed by \sns{range}

\lstinline[mathescape][[\$i\$:\$j\$]] \lec{\lstinline[mathescape][[\$i\$]] \hateq

\lstinline[mathescape][[\$i\$:\$j\$]]}

{sexample}[id=cap.ex]

Taking strings apart and re-assembling them.

\lstinputmhlisting{python/code/cap.py}

{sexample}

{itemize}

{frame}

{nparagraph}

\sref{cap.ex} may be difficult to understand at first. It is a \sn{programming} technique called \sn{recursion?recursion},

i.e. \sns{subroutine?function} that call themselves from within their

\sn{subroutine?body} to solve problems by utilizing solutions to smaller instances of the same problem. \Sn{recursion?recursion} can lead to very concise code, but

requires some getting-used-to.

In \sref{cap.ex} we define a \sn{subroutine?function} \lstinline|cap| that given a

string \lstinline|s| returns a string that is constructed by combining the first

\sn{character?character} uppercased by the \lstinline|uc| \sn{subroutine?function} with

the result of calling the \lstinline|cap| \sn{subroutine?function} on the rest

string -- \lstinline|s| without the first \sn{character?character}. The base case for

the recursion is the empty string, where \lstinline|uc| also returns the empty string.

So let us see what happens in our test \lstinline|cap('iwgs')|:

\noindent\lstinline|cap('iwgs')| \vergo

\lstinline|uc('i')+cap('wgs')| \vergo

```

\stinline|'l'+uc('w')+cap('gs')| \ergo
\stinline|'l'+'W'+uc('g')+cap('s')| \ergo\
\stinline|'IW'+'G'+cap('s')| \ergo
\stinline|'IWG'+uc('s')+cap('')| \ergo
\stinline|'IWG'+'S'+cap('')| \ergo
\stinline|'IWGS'+''| \ergo
\stinline|'IWGS'|
{npargraph}

```

{npargraph}[style=warning]

\sref{lcuc-chars.ex} and \sref{cap.ex} (or any other examples in this lecture) are not production code, but didactically motivated -- to show you what you can do with the objects we are presenting in \python.

In particular, if we “lowercase” a character that is already lowercase -- e.g. by \stinline|lc('c')|, then we get out of the range of the \sn{UCS} code: the answer is \stinline|\x83|, which is the \sn{character?character} with the \sn{hexadecimal} code \stinline|83| (\sn{decimal} \stinline|131|), i.e. the \sn{character?character} \stinline|No Break Here|.

In production code (as used e.g. in the \python \stinline|lower| method), we would have some range checks, etc.

```

{npargraph}
{smodule}
{document}

```

File: [courses/Jacobs/GenICT/course]{python/slides/string-literals.en}]

```

{document}
{smodule}{python-string-literals}
\stset{language=python,aboveskip=2pt,belowskip=0pt}

```

```

{frame}[fragile]
{String Literals in \python}
{itemize}
<1->
{sparagraph}[title=Problem]
How to write \sns{string} including special \sns{character?character}?

```

```

{sparagraph}
<1->\inputref[smglom/computing]{mod/literal.en}
<1->
{sdefinition}
\python uses \sn{string} \sns{literal}, i.e
\sn{character?character} sequences surrounded by one, two, or three sets of matched
single or double quotes for string input. The content can contain
\definame[post=s]{escape sequence}, i.e. the \definame{escape character} backslash
followed by a code \sn{character?character} for problematic \sns{character?character}:
{center}
{tabular}{|l|l|l|l|l|}\hline
Seq & \Sn{meaning} & Seq& \Sn{meaning}\\\hline\hline{}
\stinline|\\| & Backslash (\textbackslash) & \stinline|'| & Single quote (') \\\hline
\stinline|\"| & Double quote (") & \stinline|\a| & Bell (BEL) \\\hline
\stinline|\b| & Backspace (BS) & \stinline|\f| & Form-feed (FF)\\\hline
\stinline|\n| & Linefeed (LF) & \stinline|\r| & Carriage Return (CR)\\\hline

```

`\lstinline|t|` & Horizontal Tab (TAB) & `\lstinline|v|` & Vertical Tab (VT) `\\hline`
`{tabular}`
`{center}`
 In triple-quoted `\sn{string}` `\sns{literal}`, unescaped newlines and quotes are honored, except that three unescaped quotes in a row terminate the `\sn{literal}`.
`{sdefinition}`
`{itemize}`
`{frame}`
`{smodule}`
`{document}`

File: `[courses/Jacobs/GenICT/course]{python/slides/raw-strings.en}`
`{document}`
`\lstset{language=python,aboveskip=2pt,belowskip=0pt}`
`{smodule}{raw-strings}`

`{frame}[fragile]`
`{Raw String Literals in \python}`
`{itemize}`

`{sdefinition}`
`\Sn[post=ing]{prefix}` a `\sn{string}` `\sn{literal}` with a `\lstinline|r|` or `\lstinline|R|` turns it into a `\definame{raw string}` `\sn{literal}`, in which backslashes have no special `\sn{meaning}`.
`{sdefinition}`

`{sparagraph}[title=Note]`
 Using the backslash as an `\sn{escape character}` forces us to escape it as well.
`{sparagraph}`

`{sexample}`
 The string `\lstinline|"a\nb\nc"|` has length five and three lines, but the string `\lstinline|r"a\nb\nc"|` only has length seven and only one line.
`{sexample}`
`{itemize}`
`{frame}`
`{smodule}`
`{document}`

File: `[courses/Jacobs/GenICT/course]{datastructures/snip/encoding-trans.en}`
`{document}`
`{sparagraph}`
`\usemodule{python/slides/nutshell?python-nutshell}`

Now that we understand the “theory” of encodings, let us work out how to program with them in `\python`:
`{sparagraph}`
`{document}`

File: [courses/Jacobs/GenICT/course]{python/slides/unicode.en}]

{document}

{smodule}{python-unicode}

\lstset{language=python}

{nparagraph}

\usemodule{python/slides/libraries?python-libraries}

\Sn{programming} with $\$unicode\$$ strings is particularly simple, strings in
\python are $\$UTFeight\$$ -encoded $\$unicode\$$ strings and all operations on them
are $\$unicode\$$ -based\footnote{Older \sns{program?programming language} have
\sn{ASCII} strings only, and $\$unicode\$$ strings are supplied by external
\sr{library}{libraries}.}. This makes the introduction to $\$unicode\$$ in
\python very short, we only have to know how to produce non-\sn{ASCII}
\sns{character?character}, i.e. the \sns{character?character}
that are not on regular keyboards.

If we know the \sn{code point}, this is very simple: we just use $\$unicode\$$
\sns{python-string-literals?escape sequence}.

{nparagraph}

{frame}

{Unicode in \python}

{itemize}

{sassertion}[style=remark]

The \python \sn{python-datatypes?string} \sr{type?type}{data
type} is $\$unicode\$$, \sr{code?code}{encoded} as $\$UTFeight\$$.

{sassertion}

{sparagraph}[title=How to write $\$unicode\$$ characters?]

there are five ways

{itemize}

write them in your editor\lec{make sure that it uses $\$UTFeight\$$ }

otherwise use \python escape sequences\lec{try it!}

\stininputmhlisting{python/code/unicode.py}

{itemize}

{sparagraph}

{itemize}

{frame}

{nparagraph}

\usemodule[smglom/computing]{mod?text-editor}

\usemodule[smglom/computing]{mod?glyph}

Note that the discussion about entry methods for \sn{unicode} \sns{character?character} applies to
the bare \python \sn{interpreter}, not \python-specific \sn{text editor} modes or \sr{UI}{user
interfaces}, which are often helpful by automatically replacing the input by the
respective \sns{glyph} themselves.

{nparagraph}

{smodule}

{document}

File: [courses/Jacobs/GenICT/course]{python/slides/fstring-literals.en}]

{document}

{smodule}{python-fstring-literals}
\lstset{language=python,aboveskip=2pt,belowskip=0pt}

{nparagraph}
\Sn{string} \sns{literal} are convenient for creating simple
\sn{string} \sns{object}. For more complex ones, we usually want to
build them from pieces, usually using the \sns{variable?value} of
\sns{program-variable?variable} or the results of \sns{subroutine?function}. This
is what \sns{f string} are for in \python; we will cover that now.
{nparagraph}

{frame}
{Formatted String Literals (aka. f-strings)}
{itemize}

{sparagraph}[title=Problem]
In a \sn{program} we often want to build \sns{string} from pieces
that we already have lying around interspersed by other \sns{string}.
{sparagraph}

{sparagraph}[title=Solution]
Use \sn{string} \sn{words?concatenation}:
\lstinputmhlisting{python/code/string-concat.py}
{sparagraph}
We can do better! \lec{mixing blanks and quotes is error-prone}

{sdefinition}
\definiendum{f string}{Formatted string} \sns{literal}
(aka. \definame[post=s]{f string}) are \sn{string} \sns{literal}
can contain \python \sns{program-expression?expression} that will be
\sr{evaluation}{evaluated} -- i.e. replaced with their \sns{value?value}
at runtime.

\Definame[post=s]{f string} are \sn[post=ed]{prefix} by \lstinline|f| or
\lstinline|F|, the \sns{program-expression?expression} are delimited by
curly braces, and the \sns{character?character} \{ and \} themselves are
represented by \{\{ and \}\}.
{sdefinition}

{sexample}[title=An f-String for IWGS]
\lstinputmhlisting{python/code/f-string.py}
{sexample}
{itemize}
{frame}

{frame}
{F-String Example with a Dictionary}
{itemize}

{sexample}[title=An F-String with a Dictionary]
\lstinputmhlisting[basicstyle=\small\sf]{python/code/f-dict.py}
Note that we alternated the quotes here to avoid the following problems:
\lstinputmhlisting[basicstyle=\small\sf]{python/code/f-dict-error.py}
{sexample}
{itemize}

{frame}
{smodule}
{document}

{sfragment}
{document}

File: [courses/FAU/IWGS/course]{progintr/sec/more-functions.en}]

{document}

{sfragment}[id=sec.more-functions]{More on Functions in Python}

File: [courses/FAU/IWGS/course]{progintr/snip/more-functions.en}]

{document}

{sparagraph}

\usemodule[courses/Jacobs/GenICT/course]{python/slides/functions?python-functions} We now extend our repertoire of dealing with functions in \python.

In a sense, we now know all we have to about \python function: we can define them and apply them to arguments. But \python offers us much more: \python

{itemize}

treats functions as “first-class objects”, i.e. entities that

can be given to other functions as arguments, and can be returned as results.

provides more ways of passing arguments to a function than the rather rigid way we have seen above. This can be very convenient and make code more readable.

{itemize}

We will cover these features now. The main motivation for this is that they are widely used in \sn{programming} and being able to read them is important for collaborating with experienced \sns{programmer} and reading existing code.

{sparagraph}

{document}

File: [courses/Jacobs/GenICT/course]{python/slides/lambda.en}]

{document}

{smodule}{python-lambda}

{nparagraph}

We digress to the internals of \sns{subroutine?function} that make them even

more powerful. It turns out that we do not have to give a \sn{subroutine?function} a name at all.

{nparagraph}

{frame}

{Anonymous Functions (\linline|lambda|)}

{itemize}

{sassertion}[style=observation,name=namedfunction]

A \python \sn{subroutine?function} definition combines making a

\sn{function object} with giving it a name.

{sassertion}

{sdefinition}

\python also allows to make \definame[post=s]{anonymous function} via the

\sn{subroutine?function} \sn{literal} \linline|lambda| for

\definame[post=s]{function object}:

`\lstinputmhlisting[mathescape]{python/code/lambda_schema.py}`
{sdefinition}

{sexample}

The following two `\python` fragments are equivalent:

{center}

`{tabular}{p{2cm}@{\qquad}p{4cm}}`

`\lstinputmhlisting[linenrange=1-2]{python/code/lambda.py}` &

`\lstinputmhlisting[linenrange=3]{python/code/lambda.py}`

`{tabular}`

`{center}`

The right one is just a `\sn{variable assignment}` that assigns a
`\sn{function object}` to the `\sn{program-variable?variable}`

`\stinline|cube|`.
`\lec{In fact \python uses the right one internally}`

{sexample}

{sparagraph}[title=Question]

Why use `\sns{anonymous function}`?

{sparagraph}

{sparagraph}[title=Answer]

We may not want to invent (i.e. waste) a name if the `\sn{subroutine?function}` is only used once.
`\lec{examples on the next slide}`

{sparagraph}

{itemize}

{frame}

{smodule}

{document}

File: `[courses/Jacobs/GenICT/course]{python/slides/ho-functions.en}`

{document}

{smodule}{python-ho-functions}

{nparagraph}

`\Sns{anonymous function}` do not seem like a big deal at first, but having a
way to construct a `\sn{subroutine?function}` that can be used in any expression, is very powerful as we
will see now.

{nparagraph}

{frame}

{Higher-Order Functions in `\python`}

{itemize}

{sdefinition}[id=def.ho-function]

We call a `\sn{subroutine?function}` a `\definame{higher order function}`, `\sn{iff}` it
takes a `\sn{subroutine?function}` as `\sn{subroutine?argument}`.

{sdefinition}

{sdefinition}[id=def.map-filter]

`\stinline|map|` and `\stinline|filter|` are built-in

`\sns{higher order function}` in `\python`. They take a

`\sn{subroutine?function}` and a `\sn{list?list}` as arguments.

{itemize}

`\stinline[mathescape]|map($f,$L$)|` returns the `\sn{list}` of

`f-\sns{functions?value}` of the `\sns{container?element}` of `L`.
`\stinline[mathescape]{filter(p, L)}` returns the `\sn[pre=sub-]{list}`
`L'` of those `I` in `L`, such that `\stinline[mathescape]{ p(I)=True}`.
{itemize}
{sdefinition}

{sexample}[id=map-filter.ex]
Mapping over and filtering a `\sn{list}`
`\stinputmhlisting{python/code/map-filter.py}`
{sexample}
{itemize}
{frame}

{nparagraph}
Admittedly, in our example, we could also have defined a named `\sn{subroutine?function}`
`\stinline|twice|` and then mapped that over `\stinline|li|`:

`\stinputmhlisting{python/code/map-twice.py}`

But the code from `\sref{map-filter.ex}` is more compact. Once we get used to the
`\sn{programming}` idiom and understand it, it becomes quite readable.
{nparagraph}
{smodule}
{document}

File: `[courses/Jacobs/GenICT/course]{python/snippet/argpassing.en}`
{document}
{sparagraph}
`\usemodule{python/slides/functions?python-functions}`

Another important feature of `\python \sns{subroutine?function}` is flexible
argument passing. This allows to define `\sns{subroutine?function}` that supply complex
behaviors -- for which we need to set many `\sns{subroutine?parameter}` but
simple calling patterns -- which is good to hide complexity from the
`\sn{programmer}`.
{sparagraph}
{document}

File: `[courses/Jacobs/GenICT/course]{python/slides/kwarg.en}`
{document}
{smodule}{python-kwarg}

{nparagraph}
The first `\sn{subroutine?argument}` passing feature we want to discuss is the
use of `\sns{keyword argument}`, which gets around the problem of having to remember
the position of an argument of a multi-parameter `\sn{subroutine?function}`.
{nparagraph}

{frame}[label=slide.kwarg,fragile]
{Argument Passing in `\python`: Keyword Arguments}
{itemize}

{sdefinition}[id=def.default-argument]

```
\varepsilon{pseq}{1,\ellipses,n}{\comp{p}_{#1}}
\varepsilon{aseq}{1,\ellipses,n}{\comp{a}_{#1}}
\varepsilon{vseq}{1,\ellipses,n}{\comp{pmetar{val}}\comp{pmetar}_{#1}}
```

The last $\text{\natlethan}{k}{n}$ of $\text{\$n}$ parameters of a $\text{\sn{subroutine?function}}$ can be $\text{\definame}[post=s]{\text{keyword argument}}$ of the form

$\text{\inline[mathescape]}{\text{\$pseq}{i}}{\text{\$}}{\text{\$vseq}{i}}{\text{\$}}$: If no argument $\text{\$aseq}{i}{\text{\$}}$ is given in the function call, the $\text{\definame}{\text{default value}}$ $\text{\$vseq}{i}{\text{\$}}$ is taken.

\sdefinition

$\text{\sexample}[id=default-argument.ex,for=default value]$

The head of the $\text{\inline|open| \sn{subroutine?function}}$ is

$\text{\inputmhlisting}[linerange=1-2]{\text{python/code/open.py}}$

Even if we only call it with $\text{\inline|open|("foo")|}$, we can use

$\text{\sns{subroutine?parameter}}$ like \inline|mode| or \inline|opener| in the $\text{\sn{subroutine?body}}$; they have the corresponding $\text{\sn{default value}}$.

We can also give more arguments via keywords, even out of order

$\text{\inputmhlisting}[linerange=3-3]{\text{python/code/open.py}}$

\sexample

\itemize

\frame

$\text{\nparagraph}[title=BTW]$

$\text{\usemodule}{\text{python/slides/lambda?python-lambda}}$ The \inline|opener| argument of \inline|open| is a $\text{\sn{subroutine?function}}$, and often an $\text{\sn{python-lambda?anonymous function}}$ is used if it is specified.

\nparagraph

\smodule

\document

File: $\text{[courses/Jacobs/GenICT/course]}{\text{python/slides/flexary.en}}$

\document

$\text{\smodule}{\text{python-flexary}}$

\nparagraph

The next feature is dual to the last: instead of letting the caller leave out some arguments, we allow the caller more, which is then bound to a $\text{\sn{list?list}}$

$\text{\sn{subroutine?parameter}}$.

\nparagraph

$\text{\frame}[fragile]$

$\text{\{Argument Passing in \python: Flexible Arity}}$

\itemize

$\text{\sdefinition}[id=def.rest-argument]$

```
\varepsilon{pseq}{1,\ellipses,k}{\comp{p}_{#1}}
\varepsilon{aseq}{1,\ellipses,k}{\comp{a}_{#1}}
\varepsilon{acseq}{k+1,\ellipses,n}{\comp{a}_{#1}}
```

$\text{\python \sns{subroutine?function}}$ can take a variable number of $\text{\sns{subroutine?argument}}{\text{\}}$

$\text{\inline[mathescape]}{\text{def }}{\text{\$f}}{\text{(\text{\$pseq!}\text{\$},*\text{\$r}\text{\$})|}}$ allows $\text{\natmethan}{n}{k}$

$\text{\sns{subroutine?argument}}$, e. g.

\lstinline[mathescape]{\$f\$(\$\aseq!\$, \$\acseq!\$)} and binds the \sn{subroutine?parameter} \$r\$ the \definame{rest argument} to the \sn{list?list} \lstinline[mathescape][[\$\acseq!\$]].

{sdefinition}

{sexample}[for=rest argument]

A somewhat construed \sn{subroutine?function} that reports the number of extra arguments

\lstinputmhlisting{python/code/rest.py}

{sexample}

{sdefinition}[id=star-operator]

The \definame{star operator} unpacks a \sn{list?list} into an \sn{subroutine?argument} sequence.

{sdefinition}

{sexample}[title=Passing a starred list]

\lstinputmhlisting{python/code/star-arg.py}

{sexample}

{itemize}

{frame}

{nparagraph}

Actually the \sn{star operator} can be used in other situations as well, consider for instance

\lstinputmhlisting{python/code/star-operator.py}

Here we have used the \sn{star operator} twice: First to pass the list

\lstinline|numbers| as arguments to the \sn{python-lists?list constructor} and a

second time to pass the extended list \lstinline|more_numbers| to the \lstinline|print|

\sn{subroutine?function}.

{nparagraph}

{smodule}

{document}

File: [courses/Jacobs/GenICT/course]{python/slides/flexary-kwargs.en}]

{document}

{smodule}{python-flexary-kwargs}

{nparagraph} Finally, we can combine the ideas from the last two to make \sns{python-kwargs?keyword argument} flexary.

{nparagraph}

{frame}[label=slide.flexary-kwargs1]

{Argument Passing in \python: Flexible Keyword Arguments}

{itemize}

{sdefinition}[id=def.keyword-arguments]

\python \sns{subroutine?function} can take

\definame[post=s]{keyword argument}: \

if \$k\$ is a sequence of key/value pairs then

\lstinline[mathescape]def \$f\$(\$p_1\$, \$\ldots\$, \$p_n\$, **\$k\$)|

binds the keys to values in the body of \$f\$.

{sdefinition}

{sexample}[for=keyword argument]

```
\lstinputmhlisting{python/code/kwargs.py}
{sexample}
{itemize}
{frame}
```

```
{nparagraph}
```

Just as for the flexible arity case above, we have an operator that unpacks argument structures, here a dictionary.

```
{nparagraph}
```

```
{frame}[label=slide.flexary-kwargs2]
{Argument Passing in \python: Flexible Keyword Arguments (cont.)}
{itemize}
```

```
{sdefinition}[id=def.dstar-operator]3
```

The \define{double star operator} unpacks a
\sn{dictionary?dictionary} into a sequence of
\sns{python-kwargs?keyword argument}.

```
{sdefinition}
```

```
{sexample}[title=Passing around dates as dictionaries]
```

```
\lstinputmhlisting{python/code/dstar-operator.py}
{sexample}
```

```
{sexample}[title=Mixing formal and keyword arguments]
```

```
\lstinputmhlisting{python/code/dict-arg.py}
{sexample}
{itemize}
{frame}
{smodule}
{document}
```

File: [courses/FAU/IWGS/course]{progintro/snippet/disclaimer.en}]

```
{document}
```

```
{sparagraph}[title=Disclaimer]
```

```
\usemodule[courses/Jacobs/GenICT/course]{python/slides/functions?python-functions}
```

The last couple of features of \python functions are a bit more advanced than would usually be expected from a \python \sn{programming} introduction in a course such as \useSGvar{courseacronym}. But one of the goals of \useSGvar{courseacronym} is to empower students to be able to read \python code of more experienced authors. And that kind of code may very well contain these features, so we need to cover them in \useSGvar{courseacronym}.

So the last couple of slides should be considered as an “early exposure for understanding” rather than “essential to know for \useSGvar{courseacronym}” content.

```
{sparagraph}
```

```
{document}
```

```
{sfragment}
```

```
{document}
```

File: [courses/FAU/IWGS/course]{progintr/sec/regexp.en}]

{document}

{sfragment}[id=sec.regexp]{Regular Expressions: Patterns in Strings}

File: [courses/Jacobs/TDM/course]{doccomp/snip/regexp.en}]

{document}

{sparagraph}

\usemodule{doccomp/slides?regexp-practical}

Now we can come to the main topic of this \currentsectionlevel: \sr{regex}{regular expressions}, A domain-specific language for describing string patterns. \sr{regex}{Regular expressions} are extremely useful, but also quite cryptical at first. They should be understood as a powerful tool, that relies on a language with a very limited vocabulary. It is more important to understand what this tool can do and how it works in principle than memorizing the vocabulary -- that can be looked up on demand.

{sparagraph}

{document}

File: [courses/Jacobs/TDM/course]{doccomp/slides/regexp-practical.en}]

{document}

{smodule}{regexp-practical}

{frame}

{Problem: Text/Data File Manipulation}

{itemize}

<1->

{sparagraph}[title=Problem 1 (Information Extraction)]

We often want to extract information from large document collections, e.g.

{itemize}

e-mail addresses or dates from collected correspondences

dates and places from newsfeeds

links from web pages

{itemize}

{sparagraph}

<2->

{sparagraph}[title=Problem 2 (Data Cleaning)]

The representation in data files is often too noisy and inconsistent for directly importing into an application; e.g.

{itemize}

standardizing different spellings of e.g. city names, \lec{Nuremberg vs. N rnberg}

eliminating higher \$\unicode\$ \sns{character?character}, when the

application only accepts \sn{ASCII},

separating structured texts into data blocks. \lec{e.g. in \$x\$-separated lists}

{itemize}

{sparagraph}

<3->

{sparagraph}[title=Enabling Technology]

Specifying text/data fragments \ergo \sr{regex}{regular expressions}.

{sparagraph}

{itemize}

{frame}

{nparagraph}

\usemodule[smglom/computing]{mod?pythonLanguage}

There are several dialects of \sr{regex}{regular expression languages} that differ in details, but share the general setup and syntax. Here we introduce the \python variant and recommend~\cite{PyRegex:on} for a cheat-sheet on \python \sr{regex}{regular expressions} (and an integrated \sn{regex} tester).

{nparagraph}

{frame}[label=slide.regexp-practical]

{Regular Expressions, see \cite{python-regex:URL}}

{itemize}

{sdefinition}[id=regexp.def]

A \definiendum{regex}{regular expression} (also called \definame{regex}) is a \sr{formal language}{formal} \sn{wfexp?expression} that specifies a set of \sr{words?word}{strings}.

{sdefinition}

{sdefinition}[id=regexp-metachar.def,title=Meta-Characters for
Regexps]

{center}\footnotesize

{tabular}{|l|l|}\hline%

char & denotes \\\hline\hline

\$\ASCIIldot\$ & any single \sn{character?character} (except a newline)\\\hline

\$\ASCIIlcaret\$ & beginning of a \sr{words?word}{string}\\\hline

\$\ASCIIldollar\$ & end of a \sr{words?word}{string}\\\hline

\$\ASCIIlosqbrack\$\ldots\$\ASCIIcsqbrack\$/\$\ASCIIlosqbrack\ASCIIlcaret\$\ldots\$\ASCIIcsqbrack\$

& any single \sn{character?character} in/not in the brackets\\\hline

\$\ASCIIlosqbrack x\ASCIIldash y\ASCIIcsqbrack\$/\$\ASCIIlosqbrack\ASCIIlcaret x\ASCIIldash

y\ASCIIcsqbrack\$

& any single \sn{character?character} in/not in range \$x\$ to \$y\$\\\hline

\$\ASCIIlobrack\$\ldots\$\ASCIIlcbreck\$ & marks a \sn{capture group}\\\hline

\$\ASCIIbackslash{n}\$ & the \$n\$ th \sr{capture group}{captured group}\\\hline

\$\ASCIIlvar\$ & disjunction\\\hline

\$\ASCIIlstar\$ & matches preceding element zero or more times\\\hline

\$\ASCIIlplus\$ & matches preceding element one or more times\\\hline

\$\ASCIIlquestionmark\$ & matches preceding element zero or one times\\\hline

\$\ASCIIlobrace{n}\ASCIIlcomma{m}\ASCIIlcbreck\$ & matches the preceding element between \$n\$ and \$m\$ times\\\hline

\$\ASCIIbackslash\ASCIIIS\$/\$\ASCIIbackslash\ASCIIIs\$ & non-/whitespace

\sn{character?character}\\\hline

\$\ASCIIbackslash\ASCIIIW\$/\$\ASCIIbackslash\ASCIIlw\$ & non-/word \sn{character?character}\\\hline

\$\ASCIIbackslash\ASCIIID\$/\$\ASCIIbackslash\ASCIIId\$ & non-/digit (not only 0-9,
but also e.g. arabic digits)\\\hline

{tabular}

{center}

All other \sns{character?character} match themselves, to match e.g. a \$\ASCIIlquestionmark\$, escape with a \$\ASCIIbackslash\$: \lstineline|\\?|.

{sdefinition}

{itemize}

{frame}

{smodule}

{document}

File: [courses/Jacobs/TDM/course][doccomp/slides/regexp-ex.en]

{document}

{smodule}{regexp-ex}

{nparagraph}

Let us now fortify our intuition with some (simple) examples and a more complex one.

{nparagraph}

{frame}[fragile]

{Regular Expression Examples}

{itemize}

{sexample}[id=regexp.ex,for=regex,title=Regular Expressions and their Values]

{center}\small

{tabular}{|l|l|}\hline%

regexp & values \\\hline\hline

\lstinline|car| & \lstinline|car|\\\hline

\lstinline|.at| & \lstinline|cat|, \lstinline|hat|, \lstinline|mat|, \ldots\\\hline

\lstinline|[hc]at| & \lstinline|cat|, \lstinline|hat|\\\hline

\lstinline|^[c]at| & \lstinline|hat|, \lstinline|mat|, \ldots (but not \lstinline|cat|)\\\hline

\lstinline|^[hc]at| & \lstinline|hat|, \lstinline|cat|, but only at the beginning of the line\\\hline

\lstinline|[0-9]| & Digits\\\hline

\lstinline|[1-9][0-9]*| & natural numbers\\\hline

\lstinline|(.*)1| & \lstinline|mama|, \lstinline|papa|, \lstinline|wakawaka|\\\hline

\lstinline!cat|dog! & \lstinline|cat|, \lstinline|dog|\\\hline%

{tabular}

{center}

{sexample}

{sparagraph}

\usemodule[smglob/computing]{mod?compiler}

\usemodule{doccomp/slides?yacc}

A \sr{regex}{regular expression} can be interpreted by a

\inlinedef{definame{regular expression processor} (a \sn{program} that

identifies parts that match the provided specification)) or a

\sn[post=d]{compile} by a \sn{parser generator}.

{sparagraph}

{sexample}[title=A more complex example,id=complex-regexp.ex,for=regex]

The following \sn{regex} matches times in a variety of

formats, such as \lstinline|10:22am|, \lstinline|21:10|, \lstinline|08h55|, and

\lstinline|7.15 pm|.

\inputmhlisting[basicstyle=\small\sf]{doccomp/code/dates.pl}

{sexample}

{itemize}

{frame}

{smodule}

{document}

File: [courses/Jacobs/TDM/course][doccomp/slides/regexpal.en]

{document}

{smodule}{regexpal}

{nparagraph}
 As we have seen \sr{regex}{regular expressions} can become quite cryptic and long (cf. e.g. \sref[file=doccomp/slides/regexp-ex.en]{complex-regexp.ex}), so we need help in developing them. One way is to use one of the many regexp testers online
 {nparagraph}

{frame}
 {Playing with Regular Expressions}
 {itemize}
 If you want to play with \sns{regex}, go e.g. to
 \url{http://regex101.com}
 \cmhgraphics[width=10.5cm]{doccomp/PIC/regex101}
 {itemize}
 {frame}
 {smodule}
 {document}

File: [courses/Jacobs/GenICT/course]{python/slides/regexp.en}
 {document}
 {smodule}{python-regexp}
 \lstset{language=python,mathescape}

{nparagraph}
 After covering \sr{regex}{regular expressions} in the abstract, we will see how they are integrated into \sns{programming language} to solve problems. Of course we take \python as an example.
 {nparagraph}

{frame}[fragile]
 {Regular Expressions in \python}
 {itemize}
 We can use \sr{regex}{regular expressions} directly in \python by importing the \lstinline|re| module \lec{just add \lstinline|import re| at the beginning}
 As \python has \$\unicode\$ strings, \sr{regex}{regular expressions} support \$\unicode\$ as well.
 Useful \python \sns{subroutine?function} that use \sr{regex}{regular expressions}.
 {itemize}
 \lstinline|re.findall(\$\pmetavar{pat}\$,\$\pmetavar{str}\$)|: Return a list of non-overlapping matches of \$\pmetavar{pat}\$ in \$\pmetavar{str}\$.
 \lstinputmhlisting[linenr=1-2]{python/code/regexp.py}
 \lstinline|re.sub(\$\pmetavar{pat}\$,\$\pmetavar{sub}\$,\$\pmetavar{str}\$)|: Replace \sr{subword}{substrings} that match \$\pmetavar{pat}\$ in \$\pmetavar{str}\$ by \$\pmetavar{sub}\$.
 \lstinputmhlisting[linenr=4-5]{python/code/regexp.py}
 \lstinline|re.split(\$\pmetavar{pat}\$,\$\pmetavar{str}\$)|: Split \$\pmetavar{str}\$ into \sr{subword}{substrings} that match \$\pmetavar{pat}\$.
 \lstinputmhlisting[linenr=7-10]{python/code/regexp.py}
 {itemize}
 {itemize}
 {frame}

{nparagraph}
 As \sr{regex}{regular expressions} form a special language for

describing sets of strings, it is not surprising that they are used in all kinds of searching, splitting, and `\sr{subword}{substring}` replacement operations. As the language of `\sr{regex}{regular expressions}` is well standardized, these more or less work the same in all `\sns{programming language}`, so what you learn for `\python`, you can re-use in other `\sr{programming language}{languages}`.

{nparagraph}
{smodule}
{document}

File: [courses/Jacobs/GenICT/course]{python/slides/humanities-ex.en}]

{document}
{smodule}{python-humanities-ex}
\lstset{language=python}

{nparagraph}
\usemodule{python/slides/libraries?python-libraries}
We will now see what we can do with `\sr{regex}{regular expressions}` in a practical example. You should consider it as a “code reading/understanding” exercise, not think of it as something you should (easily) be able to do yourself. But `\sref[file=python/slides/humanities-ex.en]{corranon.ex}` could serve as a quarry of ideas for things you can do to texts with `\sr{regex}{regular expressions}`.

{nparagraph}

{frame}[label=slide.humanities-ex]
{Example: Correcting and Anonymizing Documents}
{itemize}

{sexample}[title=Document Cleanup,id=corranon.ex]
\usemodule{python/slides/libraries?python-libraries}

We write a `\sn{subroutine?function}` that makes simple corrections on documents and also crosses out all names to anonymize.

{itemize}
 `\nlex{The worst president of the US,arguably was George W. Bush, right?}`
 `\nlex{However,are you famiLLar with Paul Erd\H{o}s or Henri Poincar\’e?}\lec{Unicode}`
{itemize}

Here is the `\sn{subroutine?function}`

{itemize}
 we import the `\sr{regex}{regular expressions}` `\sn{library}` and start
the `\sn{subroutine?function}`

\stinputmhlisting[linerange=1-2]{python/code/humanities-ex.py}

 we first add blanks after commata

\stinputmhlisting[linerange=3-3]{python/code/humanities-ex.py}

 capitalize the first letter of a new sentence,

\stinputmhlisting[linerange=4-6]{python/code/humanities-ex.py}

{itemize}
{sexample}
{itemize}
{frame}

{nparagraph}
This `\sn{program}` is just a series of stepwise `\sr{regex}{regular expression}`

computations that are assigned to the variable `\stinline|s|`. For the last one, we use the `\stinline|lambda|` operator that constructs a `\sn{subroutine?function}` as an argument (the second) to `\stinline|re.sub|`. We use the `\sns{python-lambda?anonymous function}` because this `\sn{subroutine?function}` is only used once.

{nparagraph}

{nparagraph}

This worked well, so we just continue along these lines.

{nparagraph}

{frame}[label=slide.humanities-ex2]

{Example: Correcting and Anonymizing Documents (cont.)}

{itemize}

{sexample}[title=Document Cleanup (continued),id=corranon.ex2]

{itemize}

next we make abbreviations for `\sr{regex}{regular expressions}` to save space `\stinputmhlisting[linerange=7-8]{python/code/humanities-ex.py}`

remove capital letters in the middle of words

`\stinputmhlisting[linerange=9-11,basicstyle=\small\sff]{python/code/humanities-ex.py}`

and we cross-out for official public versions of government documents,

`\stinputmhlisting[linerange=12-14]{python/code/humanities-ex.py}`

finally, we return the result

`\stinputmhlisting[linerange=15-15]{python/code/humanities-ex.py}`

{itemize}

`\nlex{The worst president of the US,arguably was George W. Bush, right?}\\ becomes\\`

`\nlex{The worst president of the US, arguably was XXXXXX XX XXXX, right?}`

{sexample}

{itemize}

{frame}

{nparagraph}

We show the whole program again, to see that it is relatively small (thanks to the very compact -- if cryptic -- `\sr{regex}{regular expressions}`), when we leave out all the `\sns{comment}`.

{nparagraph}

{frame}[label=slide.humanities-ex3]

{Example: Correcting and Anonymizing Documents (all)}

{itemize}

{sexample}[title=Document Cleanup (overview),id=corranon.ex3]

`\stinputmhlisting{python/code/humanities-ex.py}`

{sexample}

{itemize}

{frame}

{smodule}

{document}

{sfragment}

{document}

File: [courses/FAU/IWGS/course]{proginthro/sec/ds-exercises.en}]
 {document}
 {nfragment}[id=sec.ds-exercises]{Exercises}
 \includeproblem{proginthro/prob/basiclists.en}
 \includeproblem{proginthro/prob/certainInput.en}
 \includeproblem{proginthro/prob/dictionaries.en}
 \includeproblem{proginthro/prob/egyptNumerals.en}
 \includeproblem{proginthro/prob/discuss-encodings.en}
 \includeproblem{proginthro/prob/egyptText1.en}
 \includeproblem{proginthro/prob/egyptText2.en}
 \includeproblem{proginthro/prob/basechange.en}
 \includeproblem{proginthro/prob/regex1.en}
 \includeproblem{proginthro/prob/regex2.en}
 \includeproblem{proginthro/prob/regex3.en}
 {nfragment}
 {document}

{sfragment}
 {document}

File: [courses/FAU/IWGS/course]{digdocs/sec/digdocs.en}]
 {document}
 {sfragment}[id=sec.digdocs]{Documents as Digital Objects}
 File: [courses/Jacobs/TDM/course]{digdocs/snip/intro.en}]
 {document}
 {sparagraph}
 \usemodule[smglom/computing]{mod?computer}

In this \currentsectionlevel we take a first look at documents and how they are represented on the \sn{computer}.
 {sparagraph}
 {document}

File: [courses/FAU/IWGS/course]{digdocs/sec/files.en}]
 {document}
 {sfragment}[id=sec.files]{Representing \& Manipulating Documents on a Computer}
 File: [courses/Jacobs/TDM/course]{digdocs/snip/docrep-intro.en}]
 {document}
 {sparagraph}
 \usemodule[smglom/computing]{mod?character}
 \usemodule[smglom/arithmetics]{mod?pns-common}

Now that we can represent \sns{character} as \sn{bit}
 \sns{sequence}, we can represent text documents. In principle text documents are just \sns{sequence} of \sns{character}; they can be represented by just concatenating them.
 {sparagraph}
 {document}

File: [courses/Jacobs/TDM/course]{digdocs/slides/edocs.en}]

```

{document}
{smodule}{edocs}

{frame}
{Electronic Documents}
{itemize}
\inputref[smglom/computing]{mod/electronic-document.en}

{sexample}[for=electronic document]
\usemodule[smglom/computing]{mod?digital-image}
\usemodule[smglom/computing]{mod?PDF}
\sns{PDF}, \sns{digital image}, videos, audio recordings, web pages, \ldots
{sexample}
\inputref[smglom/computing]{mod/digital-text.en}
\inputref[smglom/computing]{mod/text-types.en}

{sexample}[for={plain text,formatted text}]
\usemodule[courses/FAU/IWGS/course]{progintr/slides?jupyterLab}
\usemodule[smglom/computing]{mod?PDF}
\python \sns{program?program} are \sn{plain text},
\sns{PDF} are \sr{formatted text}{formatted}.
{sexample}
{itemize}
{frame}
{smodule}
{document}

```

File: [courses/Jacobs/TDM/course]{digdocs/slides/markup.en}]

```

{document}
{smodule}{rc*markup}

```

```

{nparagraph}

```

We will now establish a nomenclature for giving instructions to a
\sn{electronic-document?document renderer}. This has originated from movable (lead)
type based typesetting but carries over well to
\sns{electronic-document?electronic document}.

```

{nparagraph}

```

```

{frame}
{Document Markup}
{itemize}
\inputref[smglom/computing]{mod/markup.en}

```

```

{sexample}[id=document-markup.ex,for=markup]
A text with \sr{control word}{markup codes} (for printing)
\cmhgraphics[width=8cm]{digdocs/PIC/markup}
{sexample}
\inputref[smglom/computing]{mod/document-type.en}

```

```

{sassertion}[style=remark]
\Sn{markup} turns \sn{plain text} into \sn{formatted text}.
{sassertion}
{itemize}
{frame}

```

{nparagraph}

There are many systems for \sr{markup}{document markup}, ranging from informal ones as in \sref{document-markup.ex} that specify the intended document appearance to humans -- in this case the printer -- to technical ones which can be understood by machines but serving the same purpose.

{nparagraph}

{nparagraph}

\Sn{markup} is by no means limited to \inlinedef{\definame{visual markup} for documents intended for printing} as \sref{document-markup.ex} may suggest. There are \inlinedef{\definame{aural markup} formats that instruct \sns{electronic-document?document renderer} that transform documents to audio streams of e.g. reading speeds, intonation, and stress.}

{nparagraph}

{smodule}

{document}

File: [courses/Jacobs/TDM/course]{digdocs/slides/file-type.en}]

{document}

{smodule}{file-type}

{nparagraph}

\usemodule[smglom/computing]{mod?interactive}

We now come to another aspect of \sns{electronic-document?electronic document}: We mostly \sn{interact} with them in the form of \sns{file?file}. Again, we fix our nomenclature.

{nparagraph}

{frame}

{File Types}

{itemize}

{sassertion}[style=observation]

We mostly encounter \sns{electronic document} in the form of \sns{file?file} on some \sn{storage medium}.

{sassertion}

\inputref[smglom/computing]{mod/file-type.en}

{sassertion}[style=remark]

\Sns{text file} are usually encoded with \sn{ASCII},

\sn{ASCII-problems?ISO Latin}, or increasingly \$\unicode\$ encodings like \$\UTFeight\$.

{sassertion}

{sexample}[for=text file]

\usemodule[courses/Jacobs/GenICT/course]{python/slides/nutshell?python-nutshell}

\python programs are stored in \sns{text file}.

{sexample}

{sparagraph}

In practice, \sns{text file} are often processed as a

\sn{sequences?sequence} of \inlinedef{\definiendum[post=s]{line}}{text line} (or

just `\define{post=s}{line}}`, i.e. sub strings separated by the
`\define{line feed character} {\unicodepoint{000A}}; \ucsname{LINE FEED`
`(LF)}`. The `\define{line number}` is just the position in the sequence.

`{sparagraph}`
`{itemize}`
`{frame}`

`{nparagraph}[style=remark]`
`\usemodule{digdocs/slides/markup?rc*markup}`
`\Sn{plain text}` is different from `\sn{formatted text}`, which includes `\sr{markup?control`
`word}{markup code}`, and `\sns{binary file}` in which some portions must be interpreted
as binary data (encoded `\sns{integer}`, `\sns{real number}`, `\sns{digital image}`, etc.)

`{nparagraph}`
`{smodule}`
`{document}`

File: `[courses/Jacobs/TDM/course]{digdocs/slides/editors.en}`

`{document}`
`{smodule}{editors}`
`\symdef{emacsEditor}{\comp{\mathtt{emacs}}}`
`\symdef{viEditor}{\comp{\mathtt{vi}}}`
`\symdef{sublimeEditor}{\comp{\mathtt{sublime}}}`
`\symdef{notepadEditor}{\comp{\mathtt{Notepad}}}`
`\symdef{etherpad}{\comp{\mathtt{EtherPad}}}`

`{nparagraph}`
`\usemodule[courses/Jacobs/GenCS/course]{xml/slides?unicode-nutshell}`
`\usemodule[smglom/computing]{mod?shell}`
As we have seen above, it does not take much to `\sr{electronic-document?document`
`renderer}{render}` a `\sn{file-type?text file}`: we only need to guess the right
`\sn{unicode-nutshell?encoding scheme}` so we can decode the file and show the
character sequence to the user. Indeed the `\unixOS$ \stinline|cat|` just prints the
contents of a `\sn{file-type?text file}` to a `\sn{shell?shell}`. But we need much
more, we need tools with which we can compose and edit `\sns{file-type?text`
`file}`; we do this with `\sns{text editor}`, which we will discuss now.

`{nparagraph}`

`{frame}`
`{Text Editors}`
`{itemize}`

`{sdefinition}[id=text-editor.def]`
A `\define{text editor}` is a program used for
`\sr{electronic-document?document renderer}{rendering}` and manipulating
`\sns{file-type?text file}`.
`{sdefinition}`

`{sexample}[for=text editor]`
Popular `\sns{text editor}` include
`{itemize}`
`\inlinedef[for=notepadEditor]{\notepadEditor$ is a simple \sr{text editor}{editor} distributed with`
`\windowsOS$}`
`\inlinedef[for=emacsEditor]{\emacsEditor$ and \viEditor$ are powerful \sr{text editor}{editors}`

originating from `\unixOS` and optimized for `\sn{programming}`.

`\inlinedef[for=sublimeEditor]{\sublimeEditor$ is a sophisticated \sn{programming} \sr{text editor}{editor} for multiple \sr{operating-system?OS}{operating systems}.`

`\inlinedef[for=etherpad]{\etherpad$ is a browser-based real-time collaborative editor.`

`{itemize}`

`{sexample}`

`{sexample}[style=counter,for=text editor]`

`\usemodule{digdocs/slides?wordprocessors}`

Even though it can save documents as `\sns{file-type?text file}`, `\MSWord` is not usually considered a `\sn{text editor}`, since it is optimized towards `\sn{formatted text}`; such “editors” are called `\sns{wordprocessors?word processor}`.

`{sexample}`

`{itemize}`

`{frame}`

`{smodule}`

`{document}`

File: `[courses/Jacobs/TDM/course]{digdocs/slides/wordprocessors.en}`

`{document}`

`{smodule}{wordprocessors}`

`\symdef{MSWord}{\comp{\mathtt{MS Word}}}`

`\symdef{MSOffice}{\comp{\mathtt{MS Office}}}`

`\symdef{LibreOffice}{\comp{\mathtt{LibreOffice}}}`

`\symdef{OpenOffice}{\comp{\mathtt{OpenOffice}}}`

`\symdef{ApplePages}{\comp{\mathtt{Pages}}}`

`\symdef{OfficeOnline}{\comp{\mathtt{Office Online}}}`

`\symdef{GoogleDocs}{\comp{\mathtt{GoogleDocs}}}`

`{nparagraph}`

What `\sns{text editor}` do for `\sns{file-type?text file}`, `\sns{word processor}` do for other `\sns{electronic-document?electronic document}`.

`{nparagraph}`

`{frame}`

`{Word Processors and Formatted Text}`

`{itemize}`

`{sdefinition}[id=word-processor.def]`

A `\define{word processor}` is a software application, that -- apart from being a `\sn{electronic-document?document renderer}` -- also supports the tasks of composition, editing, formatting, printing of `\sns{electronic-document?electronic document}`.

`{sdefinition}`

`{sexample}[for=word processor]`

Popular `\sns{word processor}` include

`{itemize}`

`\inlinedef[for={MSWord,MSOffice,OOXML}]{\defnotation\MSWord$, an elaborated \sn{word processor} for`

`\$windowsOS$, whose native format is \definiendum{OOXML}{Office Open XML}`

{\defname{OOXML}; \sn{file} \sn{file-system?extension} \lstineline|.docx|)}.

\inlinedef[for={OpenOffice,LibreOffice,ODF}]{\OpenOffice\$ and \$\LibreOffice\$ are similar \sns{word processor} using the \defname{ODF} format
(\definiendum{ODF}{Open Office Format}; \sn{file} \sn{file-system?extension} \lstineline|.odf|) natively, but can also import other formats.}.

\inlinedef[for=ApplePages]{\ApplePages\$, a \sns{word processor} for \$\macosxOS\$ it uses a proprietary format}.

\inlinedef[for={OfficeOnline,GoogleDocs}]{\OfficeOnline\$ and \$\GoogleDocs\$ are browser-based real-time collaborative \sns{word processor}}.

{itemize}
{sexample}

{sexample}[style=counter,for=word processor]
\usemodule{digdocs/slides/markup?rc*markup}
\Sn{text editor} are usually not considered to be \sns{word processor}, even though they can sometimes be used to edit \sn{markup?markup} based \sn{formatted text}.

{sexample}
{itemize}
{frame}
{smodule}
{document}

{sfragment}
{document}

File: [courses/Jacobs/TDM/course]{digdocs/snip/measuring-trans.en}]

{document}
{sparagraph}

Before we go on, let us first get into some basics: how do we measure information, and how does this relate to units of information we know.

{sparagraph}
{document}

File: [courses/Jacobs/TDM/course]{digdocs/sec/measuring.en}]

{document}
{sfragment}{Measuring Sizes of Documents/Units of Information}

File: [courses/Jacobs/TDM/course]{digdocs/snip/measuring-intro.en}]

{document}
{sparagraph}

Having represented documents as sequences of characters, we can use that to measure the sizes of documents. In this \currentsectionlevel we will have a look at the underlying units of information and try to get an intuition about what we can store in files.

{sparagraph}

{sparagraph}[style=warning]

We will take a very generous stance towards what a document is, in particular, we will include pictures, audio files, spreadsheets, computer aided designs, \ldots.

{sparagraph}
{document}

File: [courses/Jacobs/GenCS/course]{memory/slides/bitbyte.en}}

{document}

{smodule}{bitbyte}

{frame}

{\Sns{quantities?unit} for Information}

{itemize}

{sparagraph}[title=Observation]

The smallest \sn{quantities?unit} of information is knowing the state of a system with only two states.

{sparagraph}

{sdefinition}[id=bit.def]

A \definame{bits?bit} (a contraction of “binary digit”) is the basic \sn{quantities?unit} of capacity of a data storage device or communication channel. The capacity of a system which can exist in only two states, is one \sn{bit} (written as $\$quantityof1\text{bit}$)

{sdefinition}

{sparagraph}[title=Note]

\usemodule{codes/slides?ASCII}

In the \sr{ASCII}{ASCII encoding}, one \sn{character?character} is encoded as $\$quantityof8\text{bit}$, so we introduce another basic \sn{quantities?unit}:

{sparagraph}

{sdefinition}[id=byte.def]

The \definame{bits?byte} is a derived \sn{quantities?unit} for information capacity: $\$quantityof1\text{byte}=\$quantityof8\text{bit}$.

{sdefinition}

{itemize}

{frame}

{smodule}

{document}

File: [courses/Jacobs/GenCS/course]{memory/slides/units-information.en}}

{document}

{smodule}{units-information}

{nparagraph}

From the basic units of information, we can make prefixed units for prefixed units for larger chunks of \sn{information}. But note that the usual \sr{SIprefix?prefix}{SI unit prefixes} are inconvenient for application to information measures, since powers of two are much more natural to realize.

{nparagraph}

{frame}

{Larger Units of Information via Binary Prefixes}

{itemize}

{sparagraph}

\usemodule{memory/slides?storage-element}

We will see that \sn{memory} comes naturally in \sns{natarith?power} to 2, as we address memory cell by \sn{pns-common?binary} \sns{number}, therefore the derived \sn{information}

\sns{unit} are prefixed by special \sn[post=es]{SIprefix?prefix} that are based on \sns{natarith?power} of 2.

{sparagraph}

{sdefinition}[title=Binary Prefixes]

The following \define[post=es]{bits?binary unit prefix} are used for \sn{information} \sns{unit} because they are similar to the \sr{SIprefix?prefix}{SI unit prefixes}.

{center}

{tabular}{|l|l|l|l|l|l|l|l|}\hline

prefix & symbol & $\text{\natpower{2}n}$ & decimal & \textasciitilde \sr{SIprefix?prefix}{SI prefix} & Symbol \\ \hline

\define{bits?kibi} & $\text{\defnotation\kibi!}$ & $\text{\natpower{2}{10}}$ & 1024 & \sn{kilo} & \kilo! \\

\define{bits?mebi} & $\text{\defnotation\mebi!}$ & $\text{\natpower{2}{20}}$ & 1048576 & \sn{mega} & \mega! \\

\define{bits?gibi} & $\text{\defnotation\gibi!}$ & $\text{\natpower{2}{30}}$ & $\text{\scinotation{1.074}{9}}$ & \sn{giga} & \giga! \\

\define{bits?tebi} & $\text{\defnotation\tebi!}$ & $\text{\natpower{2}{40}}$ & $\text{\scinotation{1.1}{12}}$ & \sn{tera} & \tera! \\

\define{bits?pebi} & $\text{\defnotation\pebi!}$ & $\text{\natpower{2}{50}}$ & $\text{\scinotation{1.125}{15}}$ & \sn{peta} & \peta! \\

\define{bits?exbi} & $\text{\defnotation\exbi!}$ & $\text{\natpower{2}{60}}$ & $\text{\scinotation{1.153}{18}}$ & \sn{exa} & \exa! \\

\define{bits?zebi} & $\text{\defnotation\zebi!}$ & $\text{\natpower{2}{70}}$ & $\text{\scinotation{1.181}{21}}$ & \sn{zetta} & \zetta! \\

\define{bits?yobi} & $\text{\defnotation\yobi!}$ & $\text{\natpower{2}{80}}$ & $\text{\scinotation{1.209}{24}}$ & \sn{yotta} & \yotta! \\ \hline

{tabular}

{center}

{sdefinition}

{sparagraph}[title=Note]

The correspondence works better on the smaller prefixes; for \sn{yobi} vs. \sn{SIprefix?yotta} there is a $\text{\percentage{20}}$ difference in magnitude.

{sparagraph}

{sparagraph}[name=si-bi-note]

The \sr{SIprefix?prefix}{SI unit prefixes} (and their operators) are often used instead of the correct \sn{binary} ones defined here.

{sparagraph}

{sexample}[for=si-bi-note]

You can buy hard-disks that say that their capacity is “one terabyte”, but they actually have a capacity of one \sn{tebi}\sn{byte}.

{sexample}

{itemize}

{frame}

{smodule}

{document}

File: [courses/Jacobs/GenCS/course]{memory/slides/exabyte.en}]

{document}

{smodule}{exabyte}

\usemodule[smglom/computing]{mod?database}

{nparagraph}

Let us now look at some information quantities and their real-world counterparts to get an intuition for the information content.

{nparagraph}

{frame}[label=slide.exabyte1]

\usemodule[courses/Jacobs/GenCS/course]{codes/slides?unicode-ucs}

{How much Information?}

{center}

{tabular}{|l|l|}\hline

\textbf{Bit (\$\textit{bit}\$)} & \emph{binary digit 0/1}\

\textbf{Byte (\$\textit{byte}\$)} & \emph{8 bit}\

2 Bytes & A \$\textit{character}\$ in UTF.\

10 Bytes & your name.\

\textbf{Kilobyte (\$\textit{kilobyte}\$)} & \emph{1,000 bytes OR \$\textit{2}^3\$ bytes}\

2 Kilobytes & A Typewritten page.\

100 Kilobytes & A low-resolution photograph.\

\textbf{Megabyte (\$\textit{megabyte}\$)} & \emph{1,000,000 bytes OR \$\textit{2}^6\$ bytes}\

1 Megabyte & A small novel or a 3.5 inch floppy disk.\

2 Megabytes & A high-resolution photograph.\

5 Megabytes & The complete works of Shakespeare. \

10 Megabytes & A minute of high-fidelity sound.\

100 Megabytes & 1 meter of shelved books. \

500 Megabytes & A CD-ROM.\

\textbf{Gigabyte (\$\textit{gigabyte}\$)} & \emph{1,000,000,000 bytes or \$\textit{2}^9\$ bytes}\

1 Gigabyte & a pickup truck filled with books. \

20 Gigabytes & A good collection of the works of Beethoven. \

100 Gigabytes & A library floor of academic journals.\

{tabular}

{center}

{frame}

{frame}[label=slide.exabyte2]

{How much Information?}

{center}

{tabular}{|l|l|}\hline

\textbf{Terabyte (\$\textit{terabyte}\$)} & \emph{1,000,000,000,000 bytes or \$\textit{2}^{12}\$ bytes}\

1 Terabyte & 50000 trees made into paper and printed. \

2 Terabytes & An academic research library. \

10 Terabytes & The print collections of the U.S. Library of Congress. \

400 Terabytes & National Climate Data Center (NOAA) \$\textit{database}\$.\

\textbf{Petabyte (\$\textit{petabyte}\$)} & \emph{1,000,000,000,000,000 bytes or \$\textit{2}^{15}\$ bytes}\

1 Petabyte & 3 years of EOS data (2001). \

2 Petabytes & All U.S. academic research libraries. \

20 Petabytes & Production of hard-disk drives in 1995. \

200 Petabytes & All printed material (ever).\

\textbf{Exabyte (\$\textit{exabyte}\$)} & \emph{1,000,000,000,000,000,000 bytes or \$\textit{2}^{18}\$ bytes}\

2 Exabytes & Total volume of information generated in 1999.\

5 Exabytes & All words ever spoken by human beings ever.\

300 Exabytes & All data stored digitally in 2007. \

\textbf{Zettabyte (\$\textit{zettabyte}\$)} & \emph{1,000,000,000,000,000,000,000 bytes or \$\textit{2}^{21}\$ bytes}\

2 Zettabytes & Total volume digital data transmitted in 2011\

100 Zettabytes & Data equivalent to the human Genome in one body.\

{tabular}

{center}
{frame}

{nparagraph}

The information in this table is compiled from various studies, most recently
\cite{HilLop:wtcscii11}.

{nparagraph}

{nparagraph}[title=Note]

\usemodule[smglom/computing]{mod?digital-image}

Information content of real-world artifacts can be assessed differently, depending on the view. Consider for instance a text typewritten on a single page. According to our definition, this has ca. $\$ \backslash \text{quantityof} 2 \backslash \text{kilobyte} \$$, but if we fax it, the $\backslash \text{sr} \{ \text{digital image} \} \{ \text{image} \}$ of the page has $\$ \backslash \text{quantityof} 2 \backslash \text{megabyte} \$$ or more, and a recording of a text read out loud is ca. $\$ \backslash \text{quantityof} \{ 50 \} \backslash \text{megabyte} \$$. Whether this is a terrible waste of bandwidth depends on the application. On a fax, we can use the shape of the signature for identification (here we actually care more about the shape of the ink mark than the letters it encodes) or can see the shape of a coffee stain. In the audio recording we can hear the inflections and sentence melodies to gain an impression on the emotions that come with text.

{nparagraph}

{smodule}

{document}

{sfragment}

{document}

File: [courses/FAU/IWGS/course]{digdocs/sec/html.en}]

{document}

{sfragment}[id=sec.html]{Hypertext Markup Language}

File: [courses/Jacobs/GenCS/course]{www/snip/html-trans.en}]

{document}

{sparagraph}

\usemodule{www/slides?html}

$\backslash \text{sn} \{ \text{WWW} \}$ documents have a specialized $\backslash \text{sn} \{ \text{document-type?document type} \}$ that mixes markup for document structure with layout markup, hyper-references, and $\backslash \text{sn} [\text{post=ion}] \{ \text{interact} \}$. The $\backslash \text{sn} \{ \text{html?HTML} \}$ markup elements always concern text fragments, they can be nested but may not otherwise overlap. This essentially turns a text into a document tree.

{sparagraph}

{document}

File: [courses/FAU/IWGS/course]{digdocs/snip/html-iwgs.en}]

{document}

{sparagraph}

\usemodule[courses/Jacobs/GenCS/course]{xml/slides?xml-nutshell}

\usemodule[courses/Jacobs/GenCS/course]{www/slides?html}

\usemodule[smglom/www]{mod?webapp}

\usemodule[courses/Jacobs/GenCS/course]{www/slides?CSS}

In $\backslash \text{useSGvar} \{ \text{courseacronym} \}$, we discuss $\backslash \text{sn} \{ \text{html?HTML} \}$ mostly as a way to build

interfaces of \sns{web application}. Therefore we will prioritize those aspects of \sn{html?HTML} that have to do with “programming documents” over the creation of nice-looking \sns{web page}. Therefore we will pick up the notion of nested text fragments marked up by well-bracketed tags and elements in \sref[file=digdocs/sec/html-trees.en]{sec.html-trees} and generalize these ideas to \sn{xml?XML} as a general representation paradigm for semi-structured data in \sref[file=digdocs/sec/xml-overview.en]{sec.xml-overview}.

We will also postpone the discussion of \sr{CSS}{cascading stylesheets}, which have evolved as the dominant technology for the specification of presentation (layout, colors, and fonts) for marked-up documents, to \sref[fallback=the chapter on web applications,file=webapps/sec/webapps.en]{sec.webapps}.

{sparagraph}
{document}

File: [courses/FAU/IWGS/course]{digdocs/sec/html-intro.en}]

{document}
{sfragment}[id=sec.html-intro]{Introduction}

File: [courses/Jacobs/GenCS/course]{www/snip/html-history.en}]

{document}
{sparagraph}
\usemodule{www/slides/html5?htmlfive}

\sn{html?HTML} was created in 1990 and standardized in version 4 in 1997~\cite{RagHor:html98}. Since then the \sn{WWW} has evolved considerably from a web of static \sns{web-page?web page} to a \sr{WWW}{Web} in which highly dynamic \sns{web-page?web page} become user interfaces for web-based applications and even mobile applets. \sn{htmlfive?HTML5} standardized the necessary infrastructure in 2014~\cite{W3C:html5}.

{sparagraph}
{document}

File: [courses/Jacobs/GenCS/course]{www/slides/html.en}]

{document}
{smodule}{html}

{frame}[label=slide.html]
{\sn{html?HTML}: Hypertext Markup Language}
{itemize}

{sdefinition}[id=html.def]

The \definiendum{HTML}{HyperText Markup Language} (\definame{HTML}), is a representation format for \sns{web-page?web page}~\cite{W3C:html5}.

{sdefinition}

{sdefinition}[id=html-elements.def,title=Main markup elements of HTML]

\sn{HTML} marks up the structure and appearance of text with \definame[post=s]{tag} of the form \linline|<el>| (\definame{begin tag}), \linline|</el>| (\definame{end tag}), and \linline|<el/>| (\definame{empty tag}), where \linline|el| is one of the following

{center}\small
{tabular}{|p{1.6cm}|p{2.5cm}|p{1.5cm}|p{3.5cm}|}\hline
structure & \texttt{html}, \texttt{head}, \texttt{body} &

```

\sn{metadata} & \texttt{title}, \texttt{link}, \texttt{meta}\\\hline
headings & \texttt{h1}, \texttt{h2}, \ldots, \texttt{h6} &
paragraphs & \texttt{p}, \texttt{br} \\\hline
lists & \texttt{ul}, \texttt{ol}, \texttt{dl}, \ldots, \texttt{li} &
\sn{hyperlink} & \texttt{a}\\\hline
\sn{multimedia} & \texttt{img}, \texttt{video}, \texttt{audio} &
tables & \texttt{table}, \texttt{th}, \texttt{tr}, \texttt{td}, \ldots \\\hline
styling & \texttt{style}, \texttt{div}, \texttt{span} &
old style & \texttt{b}, \texttt{u}, \texttt{tt}, \texttt{i}, \ldots \\\hline
\sn[post=ion]{interact} & \texttt{script} &
forms & \texttt{form}, \texttt{input}, \texttt{button}\\\hline
Math & MathML (formulae) &
\sn{interactive} graphics & vector graphics (SVG) and \texttt{canvas} (2D bitmapped)\\\hline
{tabular}
{center}
{sdefinition}

```

```

{sexample}[id=html-simple.ex,for=HTML]
A (very simple) \sn{HTML} file with a single paragraph.
\stinputmhlisting[language=HTML,mathescape,aboveskip=0pt]{www/code/hello.html}
{sexample}
{itemize}
{frame}

```

```

{nparagraph}
The thing to understand here is that \sn{html?HTML} uses the characters \stinline|<,
\stinline|>, and \stinline|/| to delimit the markup. All markup is in the form of
\sn{tag}, so anything that is not between \stinline|<| and \stinline|>| is the
\sn{markup?textual content}.
{nparagraph}
{smodule}
{document}

```

```

File: [courses/Jacobs/GenCS/course]{www/snip/html-ex-trans.en}
{document}
{sparagraph}
\usemodule{www/slides?html} We will not give a complete introduction to the various tags
and elements of the \sn{html?HTML} language here, but refer the reader to the
\sn{html?HTML} recommendation \cite{W3C:html5} and the plethora of excellent web
tutorials. Instead we will introduce the concepts of \sn{html?HTML} markup by way
of examples.
{sparagraph}

{document}

```

```

File: [courses/Jacobs/GenCS/course]{www/slides/html-ex.en}
{document}
{smodule}{html-ex}
\stset{language=HTML}

{nparagraph}

```

The best way to understand `<html>` is via an example. Here we have prepared a simple file that shows off some of the basic functionality of `<html>`.

`<para>`

`<frame>[label=slide.html-ex1]`

`<A very first <html> Example (Source)>`

`<input type="text" value="http://www.code/first-html.html">`

`</frame>`

`</para>`

The thing to understand here is that `<html>` markup is itself a well-balanced structure of `<begin tag>` and `<end tag>`. That wrap other balanced `<html>` structures and -- eventually -- `<textual content>`. The `<html>` recommendation `<W3C:html5>` specifies the visual appearance expectation and `<post=ions>` afforded by the respective `<tag>`, which `<html>`-aware software systems -- e.g. a `<web browser>` -- then execute. In the next slide we see how `<firefox browser>` displays the `<html>` document from the previous.

`</para>`

`<frame>[label=slide.html-ex2]`

`<A very first <html> Example (Result)>`

``

`</frame>`

`<module>`

`<document>`

`<fragment>`

`</document>`

File: `[courses/FAU/IWGS/course]{digdocs/sec/browsers.en}`

`<document>`

`<fragment[id=sec.browsers]{Interacting with HTML in Web Browsers}>`

File: `[courses/FAU/IWGS/course]{digdocs/snip/browser-trans.en}`

`<document>`

`<para>`

`<usemodule[courses/Jacobs/GenCS/course]{www/slides?webbrowser}>`

`<usemodule[courses/Jacobs/GenCS/course]{www/slides?html}>`

In the last slide, we have seen `<firefox browser>` as a

`<electronic-document?document renderer>` for `<html>`. We will now introduce this class of `<program?program>` in general and point out a few others.

`<para>`

`</document>`

File: `[courses/Jacobs/GenCS/course]{www/slides/webbrowser.en}`

`<document>`

`<module>{webbrowser}`

`<symdef{firefox browser}[name=firefox]{\comp{\mathtt{FireFox}}}>`

`<symdef{edge browser}[name=edge]{\comp{\mathtt{Edge}}}>`

```

\symdef{msiebrowser}[name=msie]{\comp{\mathtt{MS Internet Explorer}}}
\symdef{safaribrowser}[name=safari]{\comp{\mathtt{Safari}}}
\symdef{chromebrowser}[name=chrome]{\comp{\mathtt{Chrome}}}
\symdef{webkitbrowser}[name=webkit]{\comp{\mathtt{WebKit}}}

```

```

{frame}
  {\sr{web browser}{Web Browsers}}
{itemize}
  \inputref[smglom/www]{mod/webbrowser.en}

```

```

{sparagraph}[title=Practical Browser Tools]
{itemize}
  Status Bar: security info, page load progress
  Favorites (bookmarks)
  View Source: view the code of a \sn{web-page?web page}
  Tools/Internet Options, history, temporary Internet files, home page, auto
  complete, security settings, programs, etc.
{itemize}
{sparagraph}

```

```

{sexample}[for=web browser,title=Common Browsers]
{itemize}
  \inlinedef[for=msiebrowser]{\msiebrowser$ is an once dominant, now obsolete browser for
  $\windowsOS$.}
  \inlinedef[for=edgebrowser]{\edgebrowser$ is provided by Microsoft for $\windowsOS$.}\lec{replaces
  $\msiebrowser$}
  \inlinedef[for=firefoxbrowser]{\firefoxbrowser$ is an open source \sr{web browser}{browser} for all
  platforms, it is known
  for its standards compliance.}
  \inlinedef[for=safaribrowser]{\safaribrowser$ is provided by Apple for $\macosxOS$ and
  $\windowsOS$.}
  \inlinedef[for=chromebrowser]{\chromebrowser$ is a lean and mean \sr{web browser}{browser}
  provided by Google Inc.}\lec{very common}
  \inlinedef[for=webkitbrowser]{\webkitbrowser$ is a library that forms the open source basis for
  $\safaribrowser$ and $\chromebrowser$.}
{itemize}
{sexample}
{itemize}
{frame}
{smodule}
{document}

```

File: [courses/Jacobs/GenCS/course]{www/slides/browser-tools.en}]

```

{document}
{smodule}{browser-tools}
{nparagraph}

```

Let us now look at a couple of more advanced tools available in most
\sns{webbrowser?web browser} for dealing with the underlying \sn{html?HTML} document.

```

{nparagraph}
{frame}
  {Browser Tools for dealing with \sn{html?HTML}, e.g. in $\firefoxbrowser$}
{itemize}
  <1-> Hit Control-U to see the page source in the \sr{web browser}{browser}
  \only<1>{\cmhgraphics[width=11cm]{www/PIC/first-html-source}}

```



```

<2-> go to an element and right-click \ergo “Inspect element”
\only<2>{\cmhgraphics[width=10cm]{www/PIC/first-html-inspector}}
  {itemize}
  {frame}
  {nparagraph}
\usemodule[smglom/www]{mod?webapp}

```

We have used `\firefoxbrowser` as an example here, but these tools are available in some form in all major `\sr{web browser}{browsers}` the `\sr{web browser}{browser}` vendors want to make their offerings attractive to web developers, so that web pages and `\sns{web application}` get tested and debugged in them and therefore work as expected.

```

  {nparagraph}
  {smodule}
  {document}

```

```

  {sfragment}
  {document}

```

```

File: [courses/FAU/IWGS/course]{digdocs/sec/contact.en}]
  {document}
  {sfragment}[id=sec.html-contact-form]{A Worked Example: The Contact Form}
File: [courses/FAU/IWGS/course]{digdocs/slides/html-worked-example.en}]
  {document}
  {nparagraph}
\usemodule[smglom/www]{mod?web-site}

```

After this simple example, we will come to a more complex one: a little “contact form” as we find on many `\sns{web site}` that can be used for sending a message to the owner of the site. Let us only look at the design of the form document before we go into the `\sn[post=ion]{interact}` facilities afforded it.

```

  {frame}[fragile,t]
\usemodule[courses/Jacobs/GenCS/course]{www/slides?html}
  {HTML in Practice: Worked Example}
\lstset{language=HTML}
  {itemize}
  <1-> Make a design and “paper prototype” of the page:
\only<1>{\cmhgraphics[width=6.5cm]{digdocs/PIC/design}}
  <2-> Put the intended text into a file: \lstinline|contact.html|:
\only<2>{\clstininputmhlisting[linewidth=6cm]{digdocs/code/contact1.html}}
  <3-> Load into your browser to check the state:
\only<3>{\cmhgraphics[width=11cm]{digdocs/PIC/browser1}}
  <4-> Add title, paragraph and button markup:
  {onlyenv}<4>
  {center}
\parbox[c]{6.5cm}{\lstinputmhlisting{digdocs/code/contact2.html}}\qqquad
\parbox[c]{4.5cm}{\mhgraphics[width=4.5cm]{digdocs/PIC/browser2}}
  {center}
  {onlyenv}
  <5-> Add input fields and breaks:

```

```

{onlyenv}<5>
{center}
\parbox[c]{6.5cm}{\lstinputmhlisting{digdocs/code/contact3.html}}\qquad
\parbox[c]{3cm}{\mhgraphics[width=3cm]{digdocs/PIC/browser3}}
{center}
{onlyenv}
<6-> Convert into a \sn{html?HTML} form with action (message receipt):
\lstset{basicstyle=\footnotesize\sf}
{onlyenv}<6>
{center}
\parbox[c]{5.5cm}{\lstinputmhlisting{digdocs/code/contact4.html}}\qquad
\parbox[c]{5cm}{\lstinputmhlisting{digdocs/code/contact-after.html}}
{center}
{onlyenv}
{onlyenv}<7>
{center}
\parbox[c]{2cm}{\mhgraphics[width=2cm]{digdocs/PIC/browser4}}\qquad
\parbox[c]{7cm}{\mhgraphics[width=6cm]{digdocs/PIC/browser4-after}}
{center}
{onlyenv}
<8-> That's as far as we will go, the rest is page layout and
\sn[post=ion]{interact}.\lec{up next}
{itemize}
{frame}
{document}

```

```

File: [courses/FAU/IWGS/course]{digdocs/slides/html-forms.en}}
{document}
{smodule}{html-forms}
\lstset{language=HTML}
\usemodule{webapps/slides?html-form-data}

```

```

{npargraph}
After designing the functional (what are the text blocks) structure of the contact form,
we will need to understand the \sn[post=ion]{interact} with the contact form.
{npargraph}

```

```

{frame}[label=slide.html-forms,fragile]
{\sn{html?HTML} Forms}
{itemize}

```

```

{sparagraph}[title=Question]
But how does the \sn[post=ion]{interact} with the contact form really work?
{sparagraph}

```

```

{sdefinition}
A \define{HTML form} is realized by the \sn{html?HTML} \lstinline|form| \sns{tag},
which groups the layout and \define[post=s]{input element}:
{itemize}
\lstinline[mathescape]<form action="$\pmetavar{URI}$"...>| specifies the
\define{form action} \lec{as a \sn{web page} address}.
the \define{input element} \lstinline|<input type="submit" .../>| triggers the \sn{form action}: it
sends the \define{html-form-data?form data} to \sn{web page} specified
there.

```

{itemize}
{sdefinition}

{sexample}[title=In the Contact Form,for=HTML form]
We send the request
\lstinputmhlisting{digdocs/code/contact-get.url}

We current ignore the \sn{form data} (the part after the \lstinline|?|)

{sexample}
We will come to the full story of processing actions later.
{itemize}
{frame}

{nparagraph}
Unfortunately, we can only see what the browser sends to the server at the current state of play, not what the server does with the information. But we will get to this when we take up the example again.
{nparagraph}

{nparagraph}
For the moment, we made use of the fact that we can just specify the page \lstinline|contact-after.html|, which the browser displays next. That ignores the \sn{query} part and -- via a \lstinline|form| \sns{tag} of its own gets the user back to the original contact form.
{nparagraph}
{smodule}
{document}

File: [courses/FAU/IWGS/course]{digdocs/slides/more-input.en}
{document}
{smodule}{more-input}

{frame}[label=slide.more-input]
{More useful types of Input fields}
{itemize}
<1-> Radio buttons: \lstinline|type="radio"| \lec{grouped by \lstinline|name| \sn{xml-markup?attribute}}
{onlyenv}<1>\centering
\parbox[c]{9cm}{\lstinputmhlisting[linenrange=5-7,linewidth=9cm,basicstyle=\footnotesize\sf]{digdocs/code/more-input.html}}
\quad\fbbox{\parbox[c]{1.5cm}{\mhgraphics[width=1.5cm]{digdocs/PIC/radio}}}
{onlyenv}
<2-> Check boxes: \lstinline|type="checkbox"|
{onlyenv}<2>
\lstinputmhlisting[linenrange=10-13,linewidth=10cm,basicstyle=\footnotesize\sf]{digdocs/code/more-input.html}
\fbbox{\mhgraphics[width=8cm]{digdocs/PIC/checkbox}}
{onlyenv}
<3-> File selector dialogs\lec{\sn[post=ion]{interact} is system specific here for MacOS Mojave}
{onlyenv}<3>
\lstinputmhlisting[linenrange=15-15,linewidth=9cm,basicstyle=\footnotesize\sf]{digdocs/code/more-input.html}
\fbbox{\mhgraphics[width=8cm]{digdocs/PIC/fileselector}}

```

{onlyenv}
<4-> Drop down menus: \stinline|select| and \stinline|option|\\
{onlyenv}<4>\centering
\parbox[c]{6.4cm}{\stinputmhlising[linrange=17-23,linewidth=6.4cm,basicstyle=\footnotesize\sf]{digdocs/
code/more-input.html}}
\quad\fbbox{\parbox[c]{2cm}{\mhgraphics[width=2cm]{digdocs/PIC/select}}}}
{onlyenv}
{itemize}
{frame}
{smodule}
{document}

```

```

{sfragment}
{document}

```

```

{sfragment}
{document}

```

```

File: [courses/FAU/IWGS/course]{digdocs/sec/html-trees.en}]
{document}

```

```

{sfragment}[id=sec.html-trees]{Documents as Trees}
File: [courses/FAU/IWGS/course]{digdocs/snip/trees-intro.en}]
{document}

```

```

{sparagraph}
\usemodule[courses/Jacobs/GenCS/course]{www/slides?html-ex}
\usemodule[smglom/cs]{mod?computer-science}

```

We have concentrated on \sn{html?HTML} as a \sn{document-type?document type} for \sn{interactive} \sn{multimedia} documents. Before we progress, we want to discuss an important feature: all practical \sns{document-type?document type} that employ \sns{markup?control word} are in some sense well-bracketed. Well-bracketed structures are well-understood in \sn{computer-science?CS} and \sn{mathematics}: they are called \sns{tree?tree} and come with a rich and useful collection of descriptive concepts and tools. We will present the concepts in this \currentsectionlevel and the tools they enable in \sref[fallback=the next,file=digdocs/sec/xml-overview.en]{sec.xml-overview}.

```

{sparagraph}
{document}

```

```

File: [courses/FAU/IWGS/course]{digdocs/slides/wbs-cs.en}]
{document}

```

```

{smodule}{wbs-cs}

```

```

{frame}
{Well-Bracketed Structures in Computer Science}
{itemize}

```

```

{sassertion}[style=observation]

```

```

\usemodule[courses/Jacobs/GenCS/course]{www/slides?html}

```

We often deal with well-bracketed structures in \sn{computer-science?CS}, e.g.

```

{itemize}

```

<1-> Expressions: e.g.
$$\frac{3 \cdot (a+5)}{2x+7}$$
 \lec{numerator an denominator

in fractions implicitly bracketed}

<2-> \sr{markup format}{Markup languages} like \sn{html?HTML}:

{onlyenv}<2>

\stinputmhlisting[language=HTML]{digdocs/code/html-generic.html}

{onlyenv}

<3->

\usemodule[courses/Jacobs/GenICT/course]{python/slides/branching?python-branching}

\Sns{programming language} like python:

\only<3>{\stinputmhlisting[archive=courses/Jacobs/GenICT/course,language=python]{python/code/branching.py}}

{itemize}

{sassertion}

<4->

{sparagraph}[title=Idea]

Come up with a common \sn{data structure} that allows to program the same

\sns{algorithm} for all of them. \lec{common approach to scaling in

\sr{computer-science?CS}{computer science}}

{sparagraph}

{itemize}

{frame}

{smodule}

{document}

File: [courses/FAU/IWGS/course]{digdocs/slides/wbs-trees.en}]

{document}

{smodule}{wbs-trees}

{frame}

{A Common \sr{data structure}{Data Structure} for Well Bracketed Structures}

{itemize}

{sassertion}[style=observation]

In well-bracketed strutures, brackets contain two kinds of objects

{itemize}

bracket-less objects

well-bracketed structures themselves

{itemize}

{sassertion}

{sparagraph}[title=Idea,name=wbs]

Write bracket pairs and bracket-less objects as nodes, connect with an arrow when contained.\lec{let arrows point downwards}

{sparagraph}

{sexample}[for=wbs]

\usemodule[courses/Jacobs/GenCS/course]{www/slides?html}

Let's try this for \sn{html?HTML} creating nodes top to bottom

{columns}

{column}{6.5cm}

\stinputmhlisting[language=HTML]{digdocs/code/html-generic2.html}

{column}

\qqquad

```
{column}{3.5cm}
\mhtikzinput{digdocs/tikz/html-tree}
{column}
{columns}
{sexample}
```

```
{sdefinition}
We call such structures \definame{tree?tree}.\lec{more on \sns{tree?tree} next}
{sdefinition}
{itemize}
{frame}
{smodule}
{document}
```

```
File: [courses/FAU/IWGS/course]{digdocs/slides/trees-cs.en}]
{document}
{smodule}{trees-cs}
```

```
{nparagraph}
\Sns{tree?tree} are well understood \sn{mathematical} objects and \sn{tree?tree}
\sns{data-structure?data structure} are very commonly used in
\sr{computer-science?CS}{computer science} and \sn{programming?programming}. As such
they have a well-developed nomenclature, which we will introduce now.
{nparagraph}
```

```
{frame}[label=slide.trees-cs1]
{Well-Bracketed Structures: Tree Nomenclature}
{itemize}
<1->
```

```
{sdefinition}
In \sn{mathematics} and \sn{computer-science?CS}, such well-bracketed structures are called
\definame[post=s]{tree?tree} (with \definame{forest?root},
\definame[post=es]{branch?branch}, \definiendum{leaf}{leaves}, and
\definame{height-depth?height}). \lec{but written upside down}
{sdefinition}
```

```
\only<1>{\vspace*{5cm}}
<2->
```

```
{sexample}[for={path,forest?root,leaf}]
In a \sn{tree}, there is only one \sn{path} from the \sn{forest?root} to the
\sr{leaf}{leaves}
{onlyenv}<2>
{columns}[c]
{column}{5cm}
{column}
```

```
\qqquad
{column}{4cm}
\def\myxscale{.8}\def\myyscale{1}\mhtikzinput{digdocs/tikz/html-tree}
{column}
{columns}
{onlyenv}
{sexample}
```

```
<3->
\usestructure{tree}
```

```

{sdefinition}
We speak of \definame{parent}, \definame{child},
\definame{ancestor-descendant?ancestor}, and
\definame{ancestor-descendant?descendant} \sns{graph?node}
\lec{genealogy nomenclature}.
{onlyenv}<3>
{columns}
{column}{4cm}
{column}
\qqquad
{column}{4cm}
\def\myxscale{.8}\def\myyscale{1}\mhtikzinput{digdocs/tikz/html-tree}
{column}
{columns}
{onlyenv}
{sdefinition}
{itemize}
{frame}

```

```

{nparagraph}[title=Why are trees written upside-down?]
The main answer is that we want to draw \sn{tree?tree} diagrams in text. And we
naturally start drawing a \sn{tree?tree} at the \sn{forest?root}. So, if a
\sn{tree?tree} grows from the \sn{forest?root} and we do not exactly know the \sn{tree}
\sn{height-depth?height}, then we do not know how much space to leave. When we write
trees upside down, we can directly start from the \sn{forest?root} and grow the
\sn{tree?tree} downward as long as we need. We will keep to this tradition in the
\useSGvar{courseacronym} course.
{nparagraph}

```

```

{frame}[label=slide.trees-cs2]
{Upside Down Trees in Nature}
{itemize}
    Actually, upside down trees exist in nature (though rarely):
\cmhgraphics[height=6.5cm]{digdocs/PIC/upsideowntree}
This is a fig tree in Bacoli,
Italy; see \url{https://www.atlasobscura.com/places/upside-down-fig-tree}
{itemize}
{frame}
{smodule}
{document}

```

```

File: [courses/FAU/IWGS/course]{digdocs/slides/tree-computing.en}]
{document}
{smodule}{tree-computing}
\lstset{language=python,aboveskip=0pt,belowskip=0pt}

```

```

{nparagraph}
We will now make use of the \sn{tree?tree} structure for computation. Even if the
computing tasks we pursue here may seem a bit abstract, they show very nicely how
\sn{tree} \sns{algorithm} typically work.
{nparagraph}

```

```

{frame}

```

{Computing with Trees in \python}
{itemize}

{sassertion}[style=observation]
All connected substructures of \sns{tree?tree} are \sns{tree?tree} themselves.

{sassertion}
{columns}
{column}[c]{8cm}
{itemize}

{sparagraph}[title=Idea]
operate on the \sn{tree} by “Divide and Conquer”
{itemize}
operate on the two \sns{subtree}
combine results, taking \sn{forest?root} into account
{itemize}
{sparagraph}
{itemize}
{column}

\quad
{column}[c]{3cm}
\mhtikzinput{digdocs/tikz/ctree}
{column}
{columns}

This approach lends itself very well to
\sr{recursion?recursion}{recursive programming}
\lec{\sns{subroutine?function} that \sn{subroutine?call} themselves}

{sparagraph}[title=Idea]
Represent \sns{tree?tree} as \sns{list?list} of
\sn{tree} labels and \sns{list?list} (of
\sns{subtree?subtree}).
{sparagraph}

{sexample}[title=The tree above,id=tree-height.ex,for=tree]
Represented as \lstinline|[1,[2,[[4],[5]]],[3,[[6],[7]]]]\\
compute the \sr{height-depth?tree height}{tree height} by the following
\python functions:
{center}\lstset{basicstyle=\small\sf}
{tabular}{c>{\quad}c}
\lstinputmhlisting[linenrange=7-11,linewidth=4.5cm]{digdocs/code/height.py}&
\lstinputmhlisting[linenrange=1-5,linewidth=6.1cm]{digdocs/code/height.py}
{tabular}
{center}
{sexample}
{itemize}
{frame}

{nparagraph}
Let us have a closer look at \sref{tree-height.ex}. The \sn{algorithm} consists of two
\sns{subroutine?function}:
\usestructure{tree}
{enumerate}
\lstinline|height|, which computes the \sr{height-depth?tree height}{height}

of an input `\sn{tree?tree}` by delegating the computation of the maximal `\sr{height-depth?tree height}{height}` of its `\sr{child}{children}` to `\stinline|maxh|` and then incrementing the value by 1.

`\stinline|maxh|`, which takes a list of `\sns{tree?tree}` and computes the maximum of their `\sr{height-depth?tree height}{heights}` by calling `\stinline|height|` on the first input `\sn{tree?tree}` and then comparing with the maximal `\sr{height-depth?tree height}{height}` of the remaining `\sns{tree?tree}`.

`{enumerate}`

`\inlinedef{Note that \stinline|maxh| and \stinline|height| each`
`\sn{subroutine?call} the other. We call such`
`\sns{subroutine?function} \definame{recursion?mutually recursive}.}` Here
this behavior poses no problem, since the arguments in the recursive calls are smaller
than the inputs: for `\stinline|maxh|` it is the rest list, and for `\stinline|height|`
the “list of `\sn[post=ren]{child}`” of the input `\sn{tree}`.

`{nparagraph}`

`{nparagraph}`
`\sref{tree-height.ex}` was complex for two reasons: `\sr{mutually recursive}{mutual recursion}`
and the somewhat cryptic encoding of trees as lists of lists of integers. We claim that
`\sr{recursion}{recursive programming}` is “not a `\sn{bug}`, but a feature”, as
it allows to succinctly capture the “divide-and-conquer” approach afforded by
trees. For the cryptic encoding of trees we can do better.

`{nparagraph}`

`{frame}`
`{Computing with Trees in \python (Dictionaries)}`
`{itemize}`

`{sparagraph}[title=That was a bit cryptic]`
i.e. very difficult to read `\sn{debug}`
`{sparagraph}`

`{sparagraph}[title=Idea]`
why not use `\sr{dictionary?dictionary}{dictionaries}`? `\lec{they are more`
`explicit}`
`{sparagraph}`

`{sexample}[for={python,tree}]`
`\Sn{compute} the \sn{tree} weight (the sum of all labels) by`
`{center}\stset{basicstyle=\small\sf}`
`{tabular}{|>\quad|}`
`\stinputmhlisting[linrange=1-15,linewidth=4.2cm]{digdocs/code/dict-weight.py} &`
`\stinputmhlisting[linrange=16-27,linewidth=7cm]{digdocs/code/dict-weight.py}`
`{tabular}`
`{center}`
`{sexample}`
`{itemize}`
`{frame}`

`{nparagraph}`
Again, we have two `\sn{mutually recursive} \sns{subroutine?function}`:
`\stinline|weight|` that takes a tree, and `\stinline|wsum|` that takes a list and the
recursion goes analogously. Only that this time, the list of `\sn[post=ren]{child}` is a dictionary

value and the calls are clearer. The only real difference, is that in \lstinline|wsum| we have to add up the weight of the head of the list an the joint sum of the rest list.

```
{nparagraph}
{smodule}
{document}
```

File: [courses/Jacobs/GenCS/course]{xml/slides/dom.en}]

```
{document}
{smodule}{dom}
```

```
{frame}
{The Document Object Model}
{itemize}
\inputref[smglom/www]{mod/DOM.en}
```

```
{sparagraph}[title=Idea]
\usemodule{www/slides?webbrowser}
\usemodule{www/slides?html}
When a \sn{webbrowser?web browser} loads a \sn{html?HTML} page, it directly
\sns{parse} it into a \sn{DOM} and then works exclusively on that. In particular,
the \sn{html?HTML} document is immediately discarded; documents are rendered from
the \sn{DOM}.
```

```
{sparagraph}
{itemize}
{frame}
{smodule}
{document}
```

```
{sfragment}
{document}
```

File: [courses/FAU/IWGS/course]{digdocs/sec/xml-overview.en}]

```
{document}
{sfragment}[id=sec.xml-overview]{An Overview over XML Technologies}
```

File: [courses/Jacobs/GenCS/course]{xml/snip/intro.en}]

```
{document}
{sparagraph}
\usemodule{www/slides?browser-rendering-pipeline}
```

We have seen that many of the technologies that deal with marked-up documents utilize the tree-like structure of (the \sn{DOM}) of \sn{html?HTML} documents. Indeed, it is possible to abstract from the concrete vocabulary of \sn{html?HTML} that implements the intended layout of hypertexts and the function of its fragments, and build a generic framework for document trees. This is what we will study in this \currentsectionlevel.

```
{sparagraph}
{document}
```

File: [courses/FAU/IWGS/course]{digdocs/sec/xml-intro.en}]

```
{document}
{sfragment}[id=sec.xml-intro]{Introduction to XML}
```

File: [courses/Jacobs/GenCS/course]{xml/slides/xml-nutshell.en}]

{document}

{smodule}{xml-nutshell}

{frame}

{\sn{xml?XML} (E\red{X}tensible \red{M}arkup \red{L}anguage)}

{itemize}

{sdefinition}

\Definame{xml?XML} (short for \definiendum{xml?XML}{Extensible Markup Language}) is a framework for \sns{document-type?markup format} for documents and structured \sn{data?data}.

{itemize}

\Sn{tree} representation language\lec{begin/end brackets}

Restrict instances by \emph{Doc. Type Def. (DTD)} or

\emph{Schema}\lec{Grammar}

Presentation markup by \emph{style files}\lec{XSL: \red{X}ML \red{S}tyle

\red{L}anguage}

{itemize}

{sdefinition}

{sparagraph}[title=Intuition]

\usemodule{www/slides?html} \sn{XML} is extensible \sn{html?HTML}

{sparagraph}

logic annotation (\emph{markup}) instead of presentation!

many tools available: \sns{parser}, \sn{compression}, data bases, \ldots

\titleemph{conceptually}: transfer of \sns{tree?tree} instead of

\sr{words?word}{strings}.

details at \url{http://w3c.org}\lec{\sn{XML} is standardize by the WWW

Consortium}

{itemize}

{frame}

{nparagraph}

The idea of \sn{XML} being an “extensible” \sr{markup format}{markup language} may be a bit of a misnomer. It is made “extensible” by giving language designers ways of specifying their own vocabularies. As such \sn{XML} does not have a vocabulary of its own, so we could have also it an “empty”

\sr{markup format}{markup language} that can be filled with a vocabulary.

{nparagraph}

{smodule}

{document}

File: [courses/Jacobs/GenCS/course]{xml/slides/xml-everywhere-xhtml.en}]

{document}

{smodule}{xml-everywhere-xhtml}

{frame}

{\sn{xml?XML} is Everywhere (E.g. Web Pages)}

{itemize}

{sexample}[id=xml-webpage,for={HTML,web page}]

\usemodule{www/slides?webbrowser}

\usemodule{xml/mod?gui-menu}

Open \sn{web page} file in \$\firefoxbrowser\$, then click on
\$\menupath{\menuitem{View},\menuitem{Page Source}}\$, you get the following
text:\lec{showing only a small part and reformatting}
\stinputmhlisting[language=XML,mathescape,basicstyle=\footnotesize\ttfamily,
morekeywords={\l3html,head,meta,body,p,i,strong,a,title,br}]{xml/code/miko.html}
{sexample}

{sdefinition}
\Definame{XHTML} is the \sn{xml?XML} version of \sn{html?HTML}.\lec{just make it valid
\sn{xml?XML}}
{sdefinition}
{itemize}
{frame}
{smodule}
{document}

File: [courses/Jacobs/GenCS/course]{xml/slides/xml-everywhere-catalogs.en}]
{document}
{smodule}{xml-everywhere-catalogs}

{nparagraph}
Now we see an example of an \sn{xml?XML} file that is used for communicating data in a
machine-readable, but human-understandable way.
{nparagraph}

{frame}
{\sn{xml?XML} is Everywhere (E.g. Catalogs)}
{itemize}

{sexample}[title=The NYC Galleries Catalog,id=ex.mygalleries,for=XML]
A public \sn{xml?XML} file at\ \url{https://data.cityofnewyork.us/download/kcrm_j9hh/application/xml}
\stinputmhlisting[language=XML,linrange=1-18,basicstyle=\small\sf]{xml/code/museums-nyc.xml}
{sexample}
{itemize}
{frame}

\usestructure{tree}
{nparagraph}
\usemodule{xml/slides?xml-trees} This \sn{xml?XML} uses an ad hoc
\sr{markup format}{markup language}: Every \stinline|<museum>|
\sr{XML element}{element} represents one museum in New York City
(NYC). Its \sn[post=ren]{child} convey the detailed information as “key value
pairs”.
{nparagraph}
{smodule}
{document}

File: [courses/Jacobs/GenCS/course]{xml/slides/xml-everywhere-docx.en}}
{document}
{smodule}{xml-everywhere-docx}

{nparagraph}

And now, if you still need proof that `\sn{xml?XML}` is really used almost everywhere, here is the ultimate example.

`{nparagraph}`

`{frame}[label=slide.xml-everywhere-docx]`

`{\sn{xml?XML} is Everywhere (E.g. Office Suites)}`

`{itemize}`

`{sexample}[id=xml-docx,title=MS Office uses \sn{XML},for=XML]`

The `\MSOffice$` suite and `\LibreOffice$` use `\sn[post=ed]{compress} \sn{xml?XML}` as an `\sn{electronic-document?electronic document}` format.

`{enumerate}`

Save a `\MSOffice$` file `\stinline|test.docx|`, add the `\sn{file-system?extension}` `\stinline|.zip|` to obtain `\stinline|test.docx.zip|`.

`\sn[pre=Un]{compress}` with `\stinline|unzip|` (`\unixOS$`) or open File Explorer, right-click `\ergo` “Extract All” (`\windowsOS$`)

You obtain a folder with 15+ files, the content is in

`\stinline|word/contents.xml|`

Other files have packaging information, `\sr{digital image}{images}`, and other objects.

`{enumerate}`

`\textwarning` This is huge and offensively ugly.

`{itemize}`

But you have everything you wanted and more

In particular, you can process the contents via a program now.

`{itemize}`

`{sexample}`

`{itemize}`

`{frame}`

`{smodule}`

`{document}`

File: `[courses/Jacobs/GenCS/course]{xml/slides/xml-trees.en}`

`{document}`

`{smodule}{xml-trees}`

`\usestructure{XML document tree}`

`{frame}[label=slide.xml-trees]`

`{\sn{xml?XML} Documents as Trees}`

`{itemize}`

`{sparagraph}[title=Idea]`

An `\sn{xml?XML}` Document is a Tree

`{columns}`

`{column}{5cm}`

`\stinputmhlisting[numbers=none,mathescape,`

`morekeywords={\omtext,CMP},`

`morekeywords={\om:OMOBJ,om:OMS}]{xml/code/omtext.xml}`

`{column}\qqquad`

`{column}{5cm}\vspace*{2em}`

`\mhtikzinput[width=\textwidth]{xml/tikz/xmltreepicture}`

`{column}`

{columns}
{sparagraph}

{sdefinition}[id=xml-nodes.def]

The \define{XML document tree} is made up of \define[post=s]{XML element},
\define[post=s]{attribute node}, \define[post=s]{text node}\lec{and
\define[post=s]{namespace declaration}, comments,\ldots}

{sdefinition}
{itemize}
{frame}

{frame}[label=slide.xml-trees2]

{XML Documents as Trees (continued)}
{itemize}

\usestructure{tree}
{sdefinition}

For \sn{communication} this \sn{tree} is \sr{serialization}{serialized} into a
balanced bracketing structure, where

{itemize}
an \sr{forest?inner node}{inner} \sn{XML element} \sns{node} is represented by the brackets
\stinline|<el>| (called the \define{opening tag}) and \stinline|</el>| (called
the \define{closing tag}),

the \sr{leaf}{leaves} of the \sn{xml?XML} \sn{tree} are
represented by \define[post=s]{empty element tag}
(\sr{serialization}{serialized} as \stinline|<el></el>|, which can be
abbreviated as \stinline|<el/>|,

and \sn{text node} (\sr{serialization}{serialized} as a sequence of
\$\unicode\$ \sns{character?character}).

An \sn{XML element} \sn{node} can be annotated by further information using
\sns{attribute node} \sr{serialization}{serialized} as an
\define{xml-markup?attribute} in its \sn{opening tag}.

{itemize}
{sdefinition}

{sparagraph}[title=Note]

As a document is a \sn{tree}, the \sn{xml?XML} specification mandates that there
must be a unique \inlinedef{\define{document root}}.

{sparagraph}
{itemize}
{frame}
{smodule}
{document}

{sfragment}
{document}

File: [courses/FAU/IWGS/course]{digdocs/sec/lxml.en}]

{document}

{sfragment}[id=sec.lxml]{Computing with XML in Python}

File: [courses/FAU/IWGS/course]{digdocs/snip/lxml.en}]

```
{document}
{sparagraph}
\usemodule{digdocs/slides?tree-computing}
\usemodule[courses/Jacobs/GenCS/course]{xml/slides?xml-trees}
\usemodule[courses/Jacobs/GenICT/course]{python/slides/strings?python-strings}
```

We have claimed above that the \sn{tree?tree} nature of \sn{xml?XML} documents is one of the main advantages. Let us now see how \python makes good on this promise.

```
{sparagraph}
```

```
{sparagraph}
\usemodule[courses/Jacobs/GenCS/course]{xml/slides?xml-trees}
\usemodule[courses/Jacobs/GenCS/course]{www/slides?html}
\usemodule[courses/Jacobs/GenCS/course]{xml/slides?xpath-nutshell}
\usemodule[courses/Jacobs/GenICT/course]{python/slides/strings?python-strings}
```

We use the external \linline|lxml| library~\cite{lxml:on} in \useSGvar{courseacronym}, even though the \python distribution includes the standard library \linline|ElementTree| library~\cite{ElementTree:on} for dealing with \sn{xml?XML}. \linline|lxml| subsumes \linline|ElementTree| and extends it by functionality for \sn{xpath-nutshell?XPath} and can \sn{parse} a large set of \sn{html?HTML} documents even though they are not valid \sn{xml?XML}. This makes \linline|lxml| a better basis for practical applications in the Digital Humanities.

```
{sparagraph}
```

```
{sparagraph}[title=Acknowledgements]
```

Many of the examples and the flow of exposition in the next slides has been adapted from the \linline|lxml| tutorial~\cite{lxml:tutorial:on}.

```
{sparagraph}
```

```
{document}
```

File: [courses/FAU/IWGS/course]{digdocs/slides/lxml-elements.en}]

```
{document}
{smodule}{lxml-elements}
\lstset{language=python,aboveskip=3pt,belowskip=2pt}

\usestructure{tree}
{frame}
{Computing with \sn{xml?XML} in \python (\sr{XML element}{Elements})}
{itemize}
```

<1-> The \linline|lxml| library~\cite{lxml:on} provides \python bindings for the (low-level) \linline|LibXML2| library.\lec{\sn{install} it with \linline|pip3 install lxml|}

<2-> The \linline|ElementTree| \sn{API?API} is the main way to programmatically \sn{interact} with \sn{xml?XML}. Activate it by importing \linline|etree| from \linline|lxml|:

```
\lstinputmhlisting[linenrange=1-1]{digdocs/code/lxml-element.py}
```

<3-> \sr{XML element}{Elements} are easily created, their properties are accessed with special \sn{accessor} \sns{oop?method}

```
\lstinputmhlisting[linenrange=2-4]{digdocs/code/lxml-element.py}
```

<4-> \sr{XML element}{Elements} are organised in an \sn{xml?XML} \sn{tree?tree} structure. To create \sn{child} \sr{XML element}{element} \sns{graph?node} and add them to a \sn{parent}

\sr{XML element}{element} \sns{graph?node}, you can use the \lstinline|append()| method:
\lstinputmhlisting[linerange=5-5]{digdocs/code/lxml-element.py}

<5->

{sparagraph}[title=Abbreviation]

create a \sn{child}

\sr{XML element}{element} \sn{graph?node} and add it to a \sn{parent}.

\lstinputmhlisting[linerange=6-7]{digdocs/code/lxml-element.py}

{sparagraph}

{itemize}

{frame}

{frame}

{Computing with \sn{xml?XML} in \python (Result)}

{itemize}

Here is the resulting \sn{xml?XML} tree so far; we

\sr{serialization}{serialize} it via \lstinline|etree.tostring|

\lstinputmhlisting[linerange=8-13]{digdocs/code/lxml-element.py}

BTW, the \lstinline|etree.tostring| is highly configurable via default arguments.

\lstinputmhlisting[basicstyle=\small\sf]{digdocs/code/etree.tostring.py}

The \lstinline|lxml| API documentation~\cite{lxml:API:on} has the details.

{itemize}

{frame}

{nparagraph}

This method of “manually” producing \sn{xml?XML} \sns{tree?tree} in memory by applying

\lstinline|etree| methods may seem very clumsy and tedious. But the power of

\lstinline|lxml| lies in the fact that these can be embedded in \python programs. And as

always, \sn{programming} gives us the power to do things very \sn[post=ly]{efficient}.

{nparagraph}

{frame}

{Computing with \sn{xml?XML} in \python (Automation)}

{itemize}

This may seem trivial and/or tedious, but we have \python power now:

\lstinputmhlisting[linerange=2-5]{digdocs/code/kchildren.py}

produces a tree with 1000 \sn[post=ren]{child} without much effort.

\lstinputmhlisting[firstline=6]{digdocs/code/kchildren.py}

We abstain from printing the \sn{xml?XML} tree (too large) and only check the length.

{itemize}

{frame}

{smodule}

{document}

File: [courses/FAU/IWGS/course]{digdocs/slides/lxml-attributes.en}}

{document}

{smodule}{lxml-attributes}

{nparagraph}

But \sn{xml?XML} documents that only have \sr{XML element}{elements}, are boring; let's do

\sn{xml?XML} \sns{xml-markup?attribute} next. Recall that attributes are essentially string-valued

key/value pairs. So what could be more natural than treating them like

\sr{dictionary?dictionary}{dictionaries}.

{nparagraph}

{frame}

{Computing with \sn{xml?XML} in \python (\Sns{xml-markup?attribute})}

{itemize}

\Sns{xml-markup?attribute} can directly be added in the \lstinline|Element| function
\lstinputmhlisting[linerange=2-5]{digdocs/code/lxml-attribute.py}

The \lstinline|.get| method returns \sns{xml-markup?attribute} in a \sn{dictionary}-like object:

\lstinputmhlisting[linerange=6-7]{digdocs/code/lxml-attribute.py}

We can set them with the \lstinline|.set| method:

\lstinputmhlisting[linerange=8-10]{digdocs/code/lxml-attribute.py} This results in a changed \sr{XML element}{element}:

\lstinputmhlisting[linerange=12-13]{digdocs/code/lxml-attribute.py}

{itemize}

{frame}

{nparagraph}

Recall that we could use \python \sr{dictionary?dictionary}{dictionaries} for iterating over in a \lstinline|for| loop. We can do the same for \sns{xml-markup?attribute}:

{nparagraph}

{frame}

{Computing with \sn{xml?XML} in \python (\Sns{xml-markup?attribute}; continued)}

{itemize}

We can access \sns{xml-markup?attribute} by the \lstinline|keys|, \lstinline|values|, and \lstinline|items| methods, known from \sr{dictionary?dictionary}{dictionaries}:

\lstinputmhlisting[linerange=15-21]{digdocs/code/lxml-attribute.py}

{sparagraph}[style=warning]

To get a ‘real’ dictionary, use the \lstinline|attrib| method\lec{e.g. to pass around}

\lstinputmhlisting[linerange=23-23]{digdocs/code/lxml-attribute.py}

Note that \lstinline|attributes| participates in any changes to \lstinline|root| and vice versa.

{sparagraph}

{sparagraph}[style=warning]

To get an independent snapshot of the \sns{xml-markup?attribute} that does not depend on the \sn{xml?XML} tree, copy it into a \lstinline|dict|:

\lstinputmhlisting[linerange=25-27]{digdocs/code/lxml-attribute.py}

{sparagraph}

{itemize}

{frame}

{nparagraph}

The last two items touch a somewhat delicate subject in

\sn{programming}. \inlinedef{Definame{mutable} an \definame{immutable}}

\sns{data-structure?data structure}: the former can be changed in place

as we have above with the \lstinline|.set| method, and the latter cannot.} Both have their justification and respective advantages. \Sn{immutable}

\sns{data-structure?data structure} are “safe” in the sense that they

cannot be changed unexpectedly by another part of the \sn{program?program}, they

have the disadvantage that every time we want to have a variant, we have to copy the

whole object. \Sn{mutable} ones do not -- we can change in place -- but we have to be very careful about who accesses them when.

This is also the reason why we spoke of “dictionary-like interface” to \sn{xml?XML} trees in \stinline|lxml|: \sr{dictionary?dictionary}{dictionaries} are \sn{immutable}, while \sn{xml?XML} trees are not.

{nparagraph}
{smodule}
{document}

File: [courses/FAU/IWGS/course]{digdocs/slides/lxml-text.en}]

{document}
{smodule}{lxml-text}
\usestructure{XML document tree}

{nparagraph}
The main remaining functionality in \sn{xml?XML} is the treatment of text. \inlinedef{\sn{xml?XML} treats text as special kinds of \sn{graph?node} in the \sn{tree?tree}:
\definame[post=s]{text node}}. They can be treated just like any other \sn{graph?node} in the \sn{xml?XML} \sn{tree?tree} in the \stinline|etree| library.

{nparagraph}

{frame}
{Computing with \sn{xml?XML} in \python (Text nodes)}
{itemize}
\sns{XML element} can contain text: we use the \stinline|.text| property to access and set it. \stinputmhlisting[linerange=2-7]{digdocs/code/lxml-text.py}

{itemize}
{frame}

{nparagraph}
To get a real intuition about what is happening, let us see how we can use all the functionality so far: we programmatically construct an \sn{html?HTML} \sn{tree?tree}.

{nparagraph}

{frame}[label=slide.lxml-text]
{Case Study: Creating an \sn{html?HTML} document}
{itemize}
We create nested \stinline|html| and \stinline|body| \sr{XML element}{elements}
\stinputmhlisting[linerange=9-10]{digdocs/code/lxml-text.py}
Then we inject a text node into the latter using the \stinline|.text| property.
\stinputmhlisting[linerange=11-11]{digdocs/code/lxml-text.py}
Let's check the result
\stinputmhlisting[linerange=13-14]{digdocs/code/lxml-text.py}
We add another \sr{XML element}{element}: a line break and check the result
\stinputmhlisting[linerange=16-18]{digdocs/code/lxml-text.py}
Finally, we can add trailing text via the \stinline|.tail| property
\stinputmhlisting[linerange=20-22]{digdocs/code/lxml-text.py}

{itemize}
{frame}

{nparagraph}
Note the use of the \stinline|.tail| property here? While the \stinline|.text|

property can be used to set “all” the text in an `\sn{xml?XML}` `\sr{XML element}{element}`, we have to use the `\stinline|.tail|` property to add trailing text (e.g. after the `\stinline|
|` `\sr{XML element}{element}`).

`{nparagraph}`
`{smodule}`
`{document}`

File: `[courses/FAU/IWGS/course]{digdocs/slides/lxml-literals.en}`

`{document}`
`{smodule}{lxml-literals}`

`{nparagraph}`

Notwithstanding the “python power” argument from above, there are situations, where we just want to write down `\sn{xml?XML}` fragments and insert them into (programmatically created) `\sn{xml?XML}`

`\sns{tree?tree}`. `\stinline|lxml|` as functionality for this:

`\sns{XML literal}`, which we introduce now.

`{nparagraph}`

`{frame}[label=slide.lxml-literals]`

`{Computing with \sn{xml?XML} in python (\sn{xml?XML} Literals) }`

`{itemize}`

`{sdefinition}`

We call any `\sn{string}` that is well-formed

`\sn{xml?XML}` an `\definame{XML literal}`.

`{sdefinition}`

We can use the `\stinline|XML|` `\sn{subroutine?function}` to read

`\sns{XML literal}`.

`\stinputmhlisting[linerange=2-2]{digdocs/code/lxml-parse.py}` The result is a first-class `\sr{XML element}{element}` `\sn{tree}`, which we can use as above

`\stinputmhlisting[linerange=3-6]{digdocs/code/lxml-parse.py}` BTW, the

`\stinline|fromstring|` `\sn{subroutine?function}` does the same.

There is a variant `\stinline|html|` that also supplies the necessary `\sn{html?HTML}` decoration. `\stinputmhlisting[linerange=8-10]{digdocs/code/lxml-parse.py}`

`{sparagraph}[title=BTW]`

If you want to read only the text content of an `\sn{XML element}`, i.e. without any intermediate tags, use the `\stinline|method|` `\sn{keyword}` in `\stinline|tostring|`:

`\stinputmhlisting[linerange=12-13]{digdocs/code/lxml-parse.py}`

`{sparagraph}`
`{itemize}`
`{frame}`
`{smodule}`
`{document}`

`{sfragment}`
`{document}`

File: [courses/FAU/IWGS/course]{digdocs/sec/xmlns.en}]

{document}

{sfragment}[id=sec.xmlns]{XML Namespaces}

File: [courses/Jacobs/GenCS/course]{xml/slides/xml-everywhere.en}]

{document}

{smodule}{xml-everywhere}

{frame}[label=slide.xml-everywhere]

{\sn{xml?XML} is Everywhere (E.g. document \sn{metadata})}

{itemize}

{sexample}[id=xml-metadata,for={metadata,XML}]

\usemodule[smglom/computing]{mod?GUI}\usemodule[smglom/computing]{mod?PDF}

\usemodule{xml/mod?gui-menu}

Open a \sn{PDF?PDF} file in \$\acrobatReader\$, then click on

\[menupath{\menuitem{File},\menuitem{Document Properties},\menuitem{Document Metadata},\menuitem{View Source}}\]

you get the following text:\lec{showing only a small part}

\stinputmhlisting[language=XML,mathescape,basicstyle=\footnotesize\ttfamily,aboveskip=0pt,belowskip=0pt,

morekeywords={{[2]rdf:RDF,rdf:Description},

morekeywords={{[3]pdf:CreationDate,pdf:ModDate,pdf:Producer,pdf:Author,pdf:Creator,pdf:Title},

morekeywords={{[4]dc:creator,dc:title}}]

{xml/code/acrobat.xml}

{sexample}

{sexample}[for={metadata,vocabulary}]

\usemodule[courses/Jacobs/CompLog]{semweb/slides?rdf-nutshell}

\sref{xml-metadata} mixes \sr{XML element}{elements} from three different vocabularies:

{itemize}

\sn{RDF}: \stinline{xmlns:rdf} for the “Resource Description Format”,

PDF: \stinline{xmlns:pdf} for the “Portable Document Format”, and

DC: \stinline{xmlns:dc} for the “Dublin Core” vocabulary

{itemize}

{sexample}

{itemize}

{frame}

{nparagraph}

\usemodule[smglom/computing]{mod?PDF}

\usemodule[courses/Jacobs/TDM/course]{digdocs/slides?wordprocessors}

\usemodule[smglom/computing]{mod?database}

This is an excerpt from the document \sn{metadata} which \$\acrobatDistiller\$ saves along

with each \sn{PDF?PDF} document it creates. It contains various kinds of information

about the creator of the document, its title, the software version used in creating it

and much more. Document \sn{metadata} is useful for libraries, bookselling companies,

all kind of text \sns{database}, book search engines, and generally all institutions or

persons or programs that wish to get an overview of some set of books, documents,

texts. The important thing about this document \sn{metadata} text is that it is not

written in an arbitrary, \sn{PDF?PDF} proprietary format. Document \sn{metadata} only

make sense if these \sn{metadata} are independent of the specific format of the

text. The \sn{metadata} that \$\MSWord\$ saves with each Word document should be in the

same format as the \sn{metadata} that Amazon saves with each of its book records, and

again the same that the British library uses, etc.

```
{nparagraph}
{smodule}
{document}
```

File: [courses/Jacobs/GenCS/course]{xml/slides/xmlns.en}]
{document}
{smodule}{xmlns}

{nparagraph}
We will now reflect what we have seen in \sref[fallback=the example
above,file=xml/slides/xml-everywhere.en]{xml-metadata} and fully define the namespacing
mechanisms involved. Note that these definitions are technically involved, but
conceptually quite natural. As a consequence they should be read more with an eye
towards “what are we trying to achieve” than the technical details.
{nparagraph}

{frame}[label=slide.xmlns,fragile]
{Mixing Vocabularies via \sn{xml?XML} Namespaces}
{itemize}
<1->
{sparagraph}[title=Problem]
We would like to reuse \sr{XML element}{elements} from different \sn{xml?XML} vocabularies\\
What happens if \sr{XML element}{element} names coincide, but have different meanings?

{sparagraph}
<1->
{sparagraph}[title=Idea]
\sr{disambiguation}{Disambiguate} them by vocabulary name.\lec{prefix}
{sparagraph}
<2->

{sparagraph}[title=Problem]
What if vocabulary names are not unique? \lec{e.g. different versions}
{sparagraph}
<2->

{sparagraph}[title=Idea]
Use a long string for identification and a short prefix for referencing
{sparagraph}
<3->

{sdefinition}
An \definiendum{xmlns}{XML namespace} is a string that identifies an \sn{xml?XML}
vocabulary. Every \sr{XML element}{element} and \sn{xml-markup?attribute} name in \sn{xml?XML}
consists of
a \define{local name} and a \sr{xmlns}{namespace}.
{sdefinition}

<3->
{sdefinition}
A \define{namespace declaration} is an \sn{xml-markup?attribute}
\stinline|xmlns:|\pmetavar{prefix}|=| whose value is an \sr{xmlns}{XML namespace}
\$n\$ on an \sn{XML element} \$e\$. The first associates the
\define{namespace prefix} \pmetavar{prefix} with the \sr{xmlns}{namespace} \$n\$ in
\$e\$: Then, any \sn{XML element} in \$e\$ with a \define{prefixed name}
\stinline[mathescape]|\pmetavar{prefix}\$. \$\pmetavar{name}\$| has
\sr{xmlns}{namespace} \$n\$ and \sn{local name} \$\pmetavar{name}\$.

A `\define{default namespace declaration} \linline[mathescape]{xmlns=d}` on an `\sr{XML element}{element} e` gives all `\sr{XML element}{elements}` in `e` whose name is not `\sr{prefixed name}{prefixed}`, the `\sr{xmlns}{namepsace} d`.

`\Sns{namespace declaration}` on `\sns{subtree?subtree}` shadow the ones on `\sns{subtree?supertree}`.

```
{sdefinition}
{itemize}
{frame}
{smodule}
{document}
```

```
{sfragment}
{document}
```

File: `[courses/FAU/IWGS/course]{digdocs/sec/xpath.en}`

```
{document}
{sfragment}[id=sec.xpath]{XPath: Specifying XML Subtrees}
```

File: `[courses/Jacobs/GenCS/course]{xml/slides/xpath-nutshell.en}`

```
{document}
{smodule}{xpath-nutshell}
```

```
\usestructure{tree}
{nparagraph}
```

One of the great advantages of viewing marked-up documents as trees is that we can describe subsets of its nodes.

```
{nparagraph}
```

```
{frame}[label=slide.xpath-nutshell]
{\sn{xpath-nutshell?XPath}, A Language for talking about \sn{xml?XML} Tree Fragments}
{itemize}
```

```
{sdefinition}[id=XPath.def]
```

The `\definiendum{XPath}{XML path language}` (`\define{XPath}`) is a language framework for specifying fragments of `\sn{xml?XML}` trees.

```
{sdefinition}
```

```
{sparagraph}[title=Intuition]
```

`\usemodule[courses/Jacobs/TDM/course]{doccomp/slides?regexp-practical}`

`\sn{xpath-nutshell?XPath}` is for `\sns{tree?tree}` what

`\sr{regex}{regular expressions}` are for

`\sr{words?word}{strings}`.

```
{sparagraph}
```

```
{sexample}[for=XPath]
```

```
{columns}\footnotesize
```

```
{column}{5.5cm}\footnotesize
```

```
\def\myxscale{.7}\def\myscale{1.2}
```

```
\mhtikzinput[width=6cm]{xml/tikz/xmltreepicture}
```

```
{column}\qqquad
```

```

{column}{6cm}\small
{tabular}{|l|p{2.2cm}|}\hline
\sn{xpath-nutshell?XPath} exp. & fragment \\\hline
\stinline|/| & root \\\hline
\stinline|omtext/CMP/*| & all \stinline|<CMP>| \sn[post=ren]{child} \\\hline
\stinline|//@name| & the \stinline|name| \sn{xml-markup?attribute} on the
\stinline|<OMS>| \sr{XML element}{element} \\\hline
\stinline|//CMP/*[1]| & the first child of all \stinline|<CMP>| \sr{XML element}{elements} \\\hline
\stinline|//*[ @cd='nums1']| & all \sr{XML element}{elements} whose \stinline|cd| has value
\stinline|nums1| \\\hline

```

```

{tabular}
{column}
{columns}
{sexample}
{itemize}
{frame}

```

```
{nparagraph}
```

An \sn{xpath-nutshell?XPath} processor is an application or library that reads an \sn{xml?XML} file into a \sn{DOM} and given an \sn{xpath-nutshell?XPath} expression returns (pointers to) the set of nodes in the \sn{DOM} that satisfy the expression.

```

{nparagraph}
{smodule}
{document}

```

File: [courses/FAU/IWGS/course]{digdocs/slides/lxml-xpath.en}]

```
{document}
```

```
{smodule}{lxml-xpath}
```

```
\lstset{basicstyle=\small\sf,language=python,aboveskip=2pt,belowskip=2pt}
```

```
{frame}
```

```
{Computing with \sn{xml?XML} in \python}
```

```
(\sn{xpath-nutshell?XPath})}
```

```
{itemize}
```

```
<1-> Say we have an \sn{xml?XML} tree:
```

```
\stinputmhlisting[linrange=1-2]{digdocs/code/lxml-xpath.py}
```

```
<2-> Then \stinline|xpath()| selects the list of matching \sr{XML element}{elements} for an
```

```
\sn{xpath-nutshell?XPath}:
```

```
\stinputmhlisting[linrange=4-8]{digdocs/code/lxml-xpath.py}
```

```
<3-> And we can do it again, \ldots
```

```
\stinputmhlisting[linrange=10-12]{digdocs/code/lxml-xpath.py}
```

```
<4-> The \stinline|xpath()| method has support for \sn{xpath-nutshell?XPath} variables:
```

```
\stinputmhlisting[linrange=14-18]{digdocs/code/lxml-xpath.py}
```

```

{itemize}
{frame}
{smodule}
{document}

```

File: [courses/FAU/IWGS/course]{digdocs/slides/xpath-leonardo.en}]

```
{document}
```

{smodule}{xpath-leonardo}

{nparagraph}

To see that \sn{xpath-nutshell?XPath} is not just a plaything, we will now look at a typical example where we can identify useful subtrees in a large \sn{html?HTML} document: the Wikipedia page on paintings by Leonardo da Vinci.

{nparagraph}

{frame}[label=slide.xpath-leonardo]

{\sn{xpath-nutshell?XPath} Example: Scraping Wikipedia}

{itemize}

{sexample}[title=Extracting Information from HTML,for=XPath]

{itemize}

<1-> We want a list of all titles of paintings by Leonardo da Vinci.

<2-> open

\url{https://en.wikipedia.org/wiki/List_of_works_by_Leonardo_da_Vinci} in

\$\texttt{firefoxbrowser}\$. \lec{save it into a file \texttt{leo.html}}

<3-> call \sn{DOM} inspector to get an idea of the

\sn{xpath-nutshell?XPath} of titles. \lec{bottom line} \

\only<3>{\cmhgraphics[width=10cm]{digdocs/PIC/monalisa-inspect}}

The path is \texttt{table > tbody > tr > td > dl > dd > i > b > a} \

\titleemph{Alternatively}: right-click on highlighted line, \ergo "copy" \ergo "XPath", gives \

\texttt{/html/body/div[3]/div[3]/div[4]/div/table[4]/tbody/tr[3]/td[2]/dl/dd/i/b/a}.

<4-> \titleemph{Idea}: We want to use the second table cells \texttt{td[2]}.

<4-> Program it in \python using the \texttt{xml} library:

\texttt{titles} is list of title strings.

\texttt{titles} is list of title strings.

{itemize}

{sexample}

{itemize}

{frame}

{nparagraph}

If the task of writing an \sn{xpath-nutshell?XPath} for extracting the \$50+\$ titles from this page does not convince you as worth learning \sn{xpath-nutshell?XPath} for, consider that Wikipedia has ca. 30 such

lists, which apparently have exactly the same tree structure, so the \sn{xpath-nutshell?XPath} developed once for da Vinci, probably works for all the others as well.

{nparagraph}

{smodule}

{document}

{sfragment}

{document}

{sfragment}

{document}

File: [courses/FAU/IWGS/course]{digdocs/sec/exercises.en}}

{document}


```

{nfragment}[id=sec.digdocs-exercises]{Exercises}
\includeproblem{digdocs/prob/simple-table.en}
\includeproblem{digdocs/prob/simple-page.en}
\includeproblem{digdocs/prob/simple-form.en}
\includeproblem{proginintro/prob/regex-pizza.en}
\includeproblem{proginintro/prob/trees.en}
\includeproblem{digdocs/prob/simple-xml.en}
\includeproblem{proginintro/prob/genHTML-1.en}
\includeproblem{proginintro/prob/genHTML-2.en}
\includeproblem{proginintro/prob/genHTML-3.en}
{nfragment}
{document}

```

```

{sfragment}
{document}

```

```

File: [courses/FAU/IWGS/course]{webapps/sec/webapps.en}
{document}
{sfragment}[id=sec.webapps]{Web Applications}
File: [courses/FAU/IWGS/course]{webapps/snippet/intro.en}
{document}
{sparagraph}
\usemodule[courses/Jacobs/GenCS/course]{www/slides?CSS}
\usemodule[smglom/computing]{mod?installation}
\usemodule[smglom/www]{mod?webserver}

```

In this \currentsectionlevel we will see how we can turn \sn{html?HTML} pages into \sr{WWW}{web}-based \sns{appsys-software?application} that can be used without having to \sn{install} additional software.

For that we discuss the basics of the \sr{WWW}{World Wide Web} as the \sn{client server architecture} that enables such \sns{appsys-software?application}. Then we take up the contact form example to get an understanding how information is passed between \sn{client} and \sn{server} in \sn{interactive} \sns{web page}. This motivates a discussion of server-side computation of \sns{web page} that can react to such information. A discussion of \sn{CSS?CSS} styling shows how to make the \sns{web page} that are generated can be made visually appealing. We conclude the \currentsectionlevel by a discussion of client-side computation that allows making \sns{web page} \sn{interactive} without recurring to the \sr{web server}{server}.

```

{sparagraph}
{document}

```

```

\excursion[courses/Jacobs/GenCS/course]{internet-basics}
{internet/sec/basics.en}
{The World Wide Web as we introduce it here is based on the Internet infrastructure
and protocols. In some places it may be useful to read up on this in}
File: [courses/FAU/IWGS/course]{webapps/sec/webapps-intro.en}
{document}
{sfragment}[id=sec.webapps-intro]{Web Applications: The Idea}

```

File: [courses/Jacobs/GenCS/course]{www/slides/webapps.en}]

{document}

{smodule}{webapps}

{frame}[label=slide.webapps]

{Web Applications: Using Applications without Installing}

{itemize}

\inputref[smglom/www]{mod/webapp.en}

{sexample}[id=webapps.ex,for=web application]

Commonly used \sns{web application} include

{itemize}

\url{http://ebay.com}; auction pages are generated from databases.

\url{http://www.weather.com}; weather information generated from weather feeds.

\url{http://slashdot.org}; aggregation of news feeds/discussions.

\url{http://github.com}; source code hosting and project management.

\url{http://studon}; course/exam management from students records.

{itemize}

{sexample}

{sparagraph}[title=Common Traits]

\usemodule[smglom/computing]{mod?database}

Pages generated from \sns{database} and external feeds, content

submission via \sn{html?HTML} forms, file upload, dynamic \sn{html?HTML}.

{sparagraph}

{itemize}

{frame}

{smodule}

{document}

File: [courses/FAU/IWGS/course]{webapps/slides/webapp-anatomy.en}]

{document}

{smodule}{webapp-anatomy}

{nparagraph}

We have seen that \sns{web application} are a common way of building

\sn{appsys-software?application software}. To understand how this works let us now have a look at the components.

{nparagraph}

{frame}[label=slide.webapp-anatomy]

{Anatomy of a Web Application}

{itemize}

{sdefinition}

A \sn{web application} consists of two parts:

{itemize}

A \define{front end} that handles the \sr{interact}{user interaction}.

A \define{back end} that stores, computes and serves the application content.

{itemize}

\cmhtikzinput{webapps/tikz/webapp-anatomy}

Both parts rely on (separate) computational facilities.\

A \sn{database?database} as a \define{persistence layer} is optional.

{sdefinition}

{sparagraph}[title=Note]

The \sn{webbrowser?web browser}, \sn{webserver?web server}, and \sn{database?database} can

{itemize}

be deployed on different \sns{computer}, \lec{high throughput}

all run on your laptop \lec{e.g. for development}

{itemize}

{sparagraph}

{itemize}

{frame}

{smodule}

{document}

{sfragment}

{document}

File: [courses/FAU/IWGS/course]{webapps/snippet/trans.en}]

{document}

{sparagraph}

\usemodule[courses/Jacobs/GenICT/course]{python/slides/nutshell?python-nutshell}

\usemodule[courses/Jacobs/GenCS/course]{www/slides?webapps}

To understand \sns{web application}, we will first need to understand

{enumerate}

how we can express \sns{web page} in \sn{html?HTML} and (see

\sref[fallback=above,file=digdocs/sec/html.en]{sec.html}) \sn{interact} with them

for data input (we recap this in

\sref[file=webapps/sec/forms-recap.en]{sec.forms-recap}),

the basics of how the \sr{WWW}{World Wide Web} works as a distribution

framework (see \sref[fallback=the last

chapter,file=webapps/sec/www-basics.en]{sec.www-basics}),

how we can generate \sn{html?HTML} documents programmatically (in our case in

\python; see \sref[file=webapps/sec/serverside.en]{sec.serverside}) as

answer pages, and finally

how we can make \sn{html?HTML} pages dynamic by client side manipulation

(see~\sref[file=webapps/sec/clientside.en]{sec.clientside}).

{enumerate}

{sparagraph}

{document}

File: [courses/FAU/IWGS/course]{webapps/sec/www-basics.en}]

{document}

{sfragment}[id=sec.www-basics]{Basic Concepts of the World Wide Web}

File: [courses/FAU/IWGS/course]{webapps/snippet/www-intro.en}]

{document}

{sparagraph}

\usemodule[courses/Jacobs/GenCS/course]{www/slides?webapps}

We will now present a very brief introduction into the concepts, mechanisms, and technologies that underlie the \sr{www?WWW}{World Wide Web} and thus \sns{web application}, which are our interest here.

{sparagraph}
{document}

File: [courses/Jacobs/GenCS/course]{www/sec/preliminaries.en}]

{document}

{sfragment}{Preliminaries}

File: [courses/Jacobs/GenCS/course]{www/snip/www-intro.en}]

{document}

{sparagraph}

\usemodule{www/slides?www-concepts}

The \sn{WWW} is the \sn{hypertext}/\sn{multimedia} part of the \sn{internet}. It is \sn{post=ed}{implement} as a service on top of the \sn{internet} (at the application level) based on specific protocols and markup formats for documents.

{sparagraph}

{document}

File: [courses/Jacobs/GenCS/course]{www/slides/interweb-intro.en}]

{document}

{smodule}{internetweb}

{frame}

{The Internet and the Web}

{itemize}

\inputref[smglom/www]{mod/internet.en}

\inputref[smglom/www]{mod/www.en}

{sparagraph}[title=Intuition]

The \sn{www?WWW} is the \sn{multimedia} part of the \sn{internet?internet}, they form critical infrastructure for modern society and commerce.

{sparagraph}

The \sn{internet}/\sn{WWW} is huge:

{center}

{tabular}{|l|l|l|l|l|}\hline

Year & \sr{WWW}{Web} & Deep Web & eMail\\\hline\hline

1999 & 21 TB & 100 TB & 11TB\\\hline

2003 & 167 TB & 92 PB & 447 PB \\\hline

2010 & ???? & ?????? & ??????\hline

{tabular}

{center}

We want to understand how it works.\lec{services and scalability issues}

{itemize}

{frame}

{smodule}

{document}

File: [courses/Jacobs/GenCS/course]{www/snip/www-concepts-trans.en}]

{document}

{sparagraph}

Given this recap we can now introduce some vocabulary to help us discuss the phenomena.

{sparagraph}

{document}

File: [courses/Jacobs/GenCS/course]{www/slides/www-concepts.en}]

{document}

{smodule}{www-concepts}

{frame}

{Concepts of the World Wide Web}

{itemize}

{sdefinition}[id=web-page.def]

A \define{web page} is a document on the \sn{WWW} that can include \sn{multimedia} \sn{data} and \sns{hyperlink?hyperlink}.

{sdefinition}

{sparagraph}[title=Note]\usemodule{www/slides?html}

\Sns{web-page?web page} are usually \sr{markup}{marked up} in in \sn{html?HTML}.

{sparagraph}

{sdefinition}[id=web-site.def]

A \define{web-site?web site} is a collection of related \sns{web-page?web page} usually designed or controlled by the same individual or organization.

{sdefinition}

A \sn{web site} generally shares a common domain name.

{sdefinition}[id=hyperlink.def]

A \define{hyperlink?hyperlink} is a reference to data that can immediately be followed by the user or that is followed automatically by a \sn{user-agent?user agent}.

{sdefinition}

{sdefinition}[id=hypertext.def]

A collection text documents with \sns{hyperlink?hyperlink} that point to text fragments within the collection is called a \define{hypertext?hypertext}. The action of following \sns{hyperlink?hyperlink} in a \sn{hypertext} is called \define{browsing} or \definiendum{browsing}{navigating} the \sn{hypertext}.

{sdefinition}

In this sense, the \sn{WWW} is a \sn{multimedia} \sn{hypertext}.

{itemize}

{frame}

{smodule}

{document}

{sfragment}

{document}

File: [courses/Jacobs/GenCS/course]{www/sec/addressing.en}]

{document}

{sfragment}[id=www-addressing]{Addressing on the World Wide Web}

File: [courses/Jacobs/GenCS/course]{www/snip/addressing-intro.en}]

{document}

{sparagraph}

\usemodule{www/slides?www-concepts}

The essential idea is that the \sr{WWW}{World Wide Web} consists of a set of resources (documents, \sr{digital image}{images}, movies, etc.) that are connected by links (like a spider-web). In the \sn{WWW}, the links consist of pointers to addresses of resources. To realize them, we only need addresses of resources (much as we have IP numbers as addresses to hosts on the \sn{internet}).

{sparagraph}

{document}

File: [courses/Jacobs/GenCS/course]{www/slides/uri-nutshell.en}]

{document}

{smodule}{uri-nutshell}

{frame}[label=slide.uri-nutshell]

{Uniform Resource Identifier (\sn{URI?URI}), Plumbing of the Web}

{itemize}

{sdefinition}[id=URI.def]

A \definiendum{URI?URI}{uniform resource identifier} (\definame{URI?URI}) is a global identifiers of local or network-retrievable documents, or media files (\definame[post=s]{URI?web resource}). \sns{URI} adhere a uniform \sn{syntax} (\sn{grammar}) defined in RFC-3986 \cite{BerFieMas:05}.

A \sn{URI?URI} is made up of the following \definame[post=s]{URI?component}:

{itemize}

a \definame{URI?scheme} that specifies the protocol governing the resource,
an \definame{URI?authority}: the host (authentication there) that provides
the resource,

a \definame{URI?path} in the hierarchically organized resources on the host,
a \definame{URI?query} in the non-hierarchically organized part of the host
data, and

a \definame{URI?fragment identifier} in the resource.

{itemize}

{sdefinition}

{sexample}[for=URI]

The following are two example \sns{URI?URI} and their component parts:

\stinputmhlisting[columns=fixed,basicstyle=\ttfamily,belowskip=0pt,aboveskip=0pt]{www/code/uri.txt}

{sexample}

{sparagraph}[title=Note]

\usemodule[smglom/www]{mod?webbrowser}

\sns{URI} only \red{identify} documents, they do not have to provide access to them (e.g. in a \sr{web browser}{browser}).

{sparagraph}

{itemize}

{frame}

{nparagraph}

\usemodule{www/slides?url-urn}

The definition above only specifies the structure of a \sn{URI} and its functional

parts. It is designed to cover and unify a lot of existing addressing schemes, including \sns{URL?URL} (which we cover next), ISBN numbers (book identifiers), and mail addresses.

{nparagraph}
{smodule}
{document}

File: [courses/Jacobs/GenCS/course]{www/slides/reluri.en}]

{document}
{smodule}{relative-uri}

{nparagraph}

In many situations \sns{URI?URI} still have to be entered by hand, so they can become quite unwieldy. Therefore there is a way to abbreviate them.

{nparagraph}

{frame}[{fragile}]%needed somehow
{Relative URIs}
{itemize}

{sdefinition}[id=relative-URI.def]

\sns{URI?URI} can be abbreviated to \definame[post=s]{relative URI};
missing parts are filled in from the context.

{sdefinition}

{sexample}[id=relative-URI-ex,for=relative URI]

Relative \sns{URI?URI} are more convenient to write

{center}

{tabular}{|l|l|l|}\hline

relative \sn{URI?URI} & abbreviates & in context\\\hline

\stinline|#foo| & \stinline[mathescape]{\$\stringmetavar{current-file}\$#foo|

& curent file\\\hline

\stinline|bar.txt| & \stinline|file:///home/kohlhase/foo/bar.txt|

& file system\\\hline

\stinline|../bar/bar.html| & \stinline|http://example.org/bar/bar.html|

& on the web\\\hline

{tabular}

{center}

{sexample}

{sdefinition}

To distinguish them from \sns{relative URI}, we call

\sns{URI?URI} \definame[post=s]{absolute URI}.

{sdefinition}

{itemize}

{frame}

{nparagraph}

The important concept to grasp for relative \sns{URI?URI} is that the missing parts can be reconstructed from the context they are found in: the document itself and how it was retrieved.

{nparagraph}

{nparagraph}

For the file system example, we are assuming that the document is a file
\stinline|foo.html| that was loaded from the file system -- under the file system \sn{URI?URI}
\stinline|file:///home/kohlhase/foo/foo.html| -- and for the web example via the \sn{URI?URI}
\stinline|//example.org/foo/foo.html|. Note that in the last example, the relative \sn{URI?URI}
\stinline|../bar/| goes up one segment of the path component (that is the meaning of
\stinline|../|), and specifies the file \stinline|bar.html| in the \sn{directory}
\stinline|bar|.

{nparagraph}
\usemodule{www/slides?www-concepts}
But \sns{relative URI} have another advantage over
\sns{absolute URI}: they make a \sn{web-page?web page} or
\sn{web-site?web site} easier to move. If a \sn{web site} only has
\sr{hyperlink}{links} using \sns{relative URI} internally, then those do
not mention e.g. \sn{URI?authority} (this is recovered from context and therefore
variable), so we can freely move the web-site e.g. between domains.

{nparagraph}
{smodule}
{document}

File: [courses/Jacobs/GenCS/course]{www/slides/url-urn.en}]

{nparagraph}
Note that some forms of \sns{URI?URI} can be used for actually locating (or
accessing) the identified resources, e.g. for retrieval, if the resource is a document
or sending to, if the resource is a mailbox. Such \sns{URI?URI} are called
“uniform resource \emph{locators}”, all others “uniform resource \emph{locators}”.

{frame}[label=slide.url-urn]
{Uniform Resource Names and Locators}
{itemize}

{sdefinition}[id=URLN.def]
A \definiendum{URL?URL}{uniform resource locator} (\definame{URL?URL}) is a
\sn{URI?URI} that gives access to a \sn{URI?web resource}, by
specifying an access method or location. All other \sns{URI?URI} are
called \definiendum{URL?URN}{uniform resource name} (\definame{URL?URN}).

{sparagraph}[title=Idea]
A \sn{URN} defines the identity of a resource, a \sn{URL} provides a method
for finding it.

{sexample}[for=URL]
\usemodule[smglom/www]{mod?webbrowser}

The following \sn{URI?URI} is a \sn{URL}\lec{try it in your
\sr{web browser}{browser}} \stinline|http://kwarc.info/kohlhase/index.html|

{sexample}[for=URN]
\stinline|urn:isbn:978-3-540-37897-6| only
identifies~\cite{Kohlhase:OMDoc1.2}\lec{it is in the library}
{sexample}

{sparagraph}
\sns{URN} can be turned into \sns{URL} via a catalog
service, e.g. \url{http://wm-urn.org/urn:isbn:978-3-540-37897-6}
{sparagraph}

{sparagraph}[title=Note]
\usemodule{www/slides/interweb-intro?internetweb}
\sns{URI?URI} are one of the core features of the web infrastructure,
they are considered to be the \red{plumbing of the \sn{WWW}}.\lec{direct the
flow of data}
{sparagraph}
{itemize}
{frame}

{nparagraph}
Historically, started out as \sns{URL} as short strings used for locating
documents on the \sn{internet?internet}. The generalization to identifiers (and the
addition of \sns{URN}) as a concept only came about when the concepts
evolved and the application layer of the \sn{internet?internet} grew and needed more structure.
{nparagraph}

{nparagraph}
Note that there are two ways in \sn{URI?URI} can fail to be resource
locators: first, the scheme does not support direct access (as the ISBN scheme in our
example), or the scheme specifies an access method, but address does not point to an
actual resource that could be accessed. Of course, the problem of “dangling links”
occurs everywhere we have addressing (and change), and so we will neglect it from our
discussion. In practice, the \sn{URL}/\sn{URN} distinction is mainly driven by the
scheme part of a \sn{URI?URI}, which specifies the access/identification
scheme.
{nparagraph}
{smodule}
{document}

File: [courses/Jacobs/GenCS/course]{www/slides/iri-nutshell.en}]
{document}
{smodule}{iri-nutshell}

{frame}[fragile]% needed somehow
{Internationalized Resource Identifiers}
{itemize}

{sassertion}[style=remark]
\sns{URI?URI} are \sn{ASCII} strings.
{sassertion}

{sparagraph}[title=Problem]

This is awkward e.g. for \nlex{France T\’el\’ecom}, worse in Asia.
{sparagraph}

{sparagraph}[title=Solution?]
Use \sn{unicode}! \lec{no, too young/unsafe}
{sparagraph}

{sdefinition}
\definiendum[post=s,root=internationalized resource
identifier]{IRI}{Internationalized resource identifiers} (\definame[post=s]{IRI})
extend the \sn{ASCII}-based \sns{URI?URI} to the
\sr{unicode-ucs?UCS}{universal character set}.
{sdefinition}

{sdefinition}
\definame{URI encoding} maps \sn[pre=non-]{ASCII} characters to \sn{ASCII} strings:
{enumerate}
Map each \sn{character?character} to its \$UTFeight\$ representation.
Represent each \sn{bits?byte} of the \$UTFeight\$ representation by three
characters.
The first \sn{character?character} is the percent sign (\char37),
and the other two \sns{character?character} are the \sn{hexadecimal} representation of
the \sn{byte}.
{enumerate}
\definame{URI decoding} is the dual operation.
{sdefinition}

{sexample}[for=URI encoding]
The letter “\” (\$\unicodepoint{142}\$) would be represented as \texttt{\char37
C5\char37 82}.
{sexample}

{sexample}[for=URI encoding]
\texttt{http://www. bergr \ss en.de} becomes\\
\texttt{http://www.\%C3\%9Cbergr\char37 C3\char37 B6\char37 C3\char37 9Fen.de}
{sexample}

{sassertion}[style=remark]
\usemodule[smglom/www]{mod?webbrowser}
Your \sr{web browser}{browser} can still show the
\sr{URI decoding}{URI decoded} version \lec{so you can read it}
{sassertion}
{itemize}
{frame}
{smodule}
{document}

{sfragment}
{document}

File: [courses/FAU/IWGS/course]{webapps/sec/running.en}]
{document}
{sfragment}[id=sec.www-running]{Running the World Wide Web}

File: [courses/Jacobs/GenCS/course]{www/slides/www-overview.en}]
{document}
{smodule}{www-overview}

{nparagraph}
\usemodule{www/slides?https-protocol}

The infrastructure of the \sn{WWW} relies on a client-server architecture, where the \sr{webserver?web server}{servers} (called \sns{webserver?web server}) provide documents and the clients (usually \sns{webbrowser?web browser}) present the documents to the (human) users. Clients and \sr{webserver?web server}{servers} communicate via the \sns{http-protocol?HTTP} and \sns{HTTPS} protocols. We give an overview via a concrete example before we go into details.

{nparagraph}

{frame}[label=slide.www-overview]
{The \sr{www?WWW}{World Wide Web} as a Client/Server System}
\cmhgraphics[width=12cm]{www/PIC/httpprocess}
{frame}
{smodule}
{document}

File: [courses/Jacobs/GenCS/course]{www/snip/http-trans.en}]
{document}
{sparagraph}
\usemodule[smglom/www]{mod?http-protocol}
\usemodule[smglom/www]{mod?webserver}
\usemodule[smglom/www]{mod?webbrowser}
The \sn{web browser} communicates with the \sn{web server} through a specialized protocol, the \sr{HTTP}{hypertext transfer protocol}, which we cover now.

{sparagraph}
{document}

File: [courses/Jacobs/GenCS/course]{www/slides/http-protocol.en}]
{document}
{smodule}{http-protocol}

{frame}
{\sn{HTTP}: Hypertext Transfer Protocol}
{itemize}

{sdefinition}[id=http.def]
The \definiendum{HTTP}{Hypertext Transfer Protocol} (\definame{HTTP}) is an application layer protocol for distributed, collaborative, hypermedia information systems.

{sdefinition}
June 1999: \sn{HTTP}/1.1 is defined in RFC 2616 \cite{FieGet:http99}.

{sparagraph}[title=Preview/Recap,id=user-agent-web-server]
\usemodule[smglom/www]{mod?user-agent}
\sn{HTTP} is used by a \sn{client} (called \sn{user agent}) to access web \sns{URI?web resource} (addressed by \sr{URL?URL}{uniform resource locators})

(\sns{URL?URL})) via a \sn{HTTP request}. The \sn{web server} answers by supplying the \sn{web resource} (and \sn{metadata}).

{sparagraph}

{sdefinition}

Most important \sn{HTTP} request \define{post=s}{method}.\lec{5 more less prominent}

{center}\small

{tabular}{|l|p{7cm}|l|}\hline

\define{GET} & Requests a representation of the specified resource. & \sn{safe} \\\hline

\define{PUT} & Uploads a representation of the specified resource. & \sn{idempotent} \\\hline

\define{DELETE} & Deletes the specified resource. & \sn{idempotent} \\\hline

\define{POST} & Submits data to be processed (e.g., from a web form) to the identified resource. & \\\hline

{tabular}

{center}

{sdefinition}

{sdefinition}[id=safe.def]

We call a \sn{HTTP} request \define{safe}, \sn{iff} it does not change the state in the \sn{web server}.\lec{except for server logs, counters,\ldots; no side effects}

{sdefinition}

{sdefinition}[id=idempotent.def]

We call a \sn{HTTP} request \define{idempotent}, \sn{iff} executing it twice has the same effect as executing it once.

{sdefinition}

\sn{HTTP} is a stateless protocol.\lec{very memory \sn{efficient} for the server.}

{itemize}

{frame}

{smodule}

{document}

File: [courses/Jacobs/GenCS/course]{www/slides/webserver.en}

{document}

{smodule}{webserver}

\symdef{apachewebserver}{\comp{\mathtt{apache}}}

\symdef{IISwebserver}{\comp{\mathtt{IIS}}}

\symdef{nginxwebserver}{\comp{\mathtt{nginx}}}

{nparagraph}

Finally, we come to the last component, the \sn{web server}, which is responsible for providing the \sn{web-page?web page} requested by the user.

{nparagraph}

{frame}

{Web Servers}

{itemize}

\inputref[smglom/www]{mod/webserver.en}

{sexample}[for=web server,title=Common Web Servers]

{itemize}

\inlinedef[for=apachewebserver]{\$\apachewebserver\$ is an open source \sn{web server} that serves about

$\frac{50}{100}$ of the [WWW](#).
`\inlinedef[for=nginxwebserver]{\nginxwebserver is a lightweight open source
\sn{web server}. \lec{ca. $\frac{35}{100}$
\inlinedef[for=IISwebserver]{\IISwebserver is a proprietary \sn{web server} provided by Microsoft
Inc.}
{itemize}
{sexample}
\inputref[smglom/www]{mod/host.en}
Even though \sns{webserver?web server} are very complex software
systems, they come \sn[pre=pre,post=ed]{install} on most \unixOS systems and
can be downloaded for \windowsOS~\cite{xampp:webpage}.
{itemize}
{frame}
{smodule}
{document}`

File: `[courses/Jacobs/GenCS/course]{www/snip/http-ex-trans.en}`
`{document}`
`{sparagraph}`
`\usemodule{www/slides?http-protocol}`

Now that we have seen all the components we fortify our intuition of what actually goes
down the net by tracing the `\sn{http-protocol?HTTP}` messages.
`{sparagraph}`
`{document}`

File: `[courses/Jacobs/GenCS/course]{www/slides/http-ex.en}`
`{document}`
`{smodule}{http-ex}`

`{frame}[label=slide.http-ex]`
`{Example: An HTTP request in real life}`
`{itemize}`
Send off a `\stinline!GET!` request for `\url{http://www.nowhere123.com/doc/index.html}`
`\stinputmhlisting[basicstyle=\footnotesize\ttfamily]{www/code/http.txt}`
The response from the server
`\stinputmhlisting[basicstyle=\footnotesize\ttfamily]{www/code/http-response.txt}`

`{sparagraph}[title=Note]`
As you can see, these are clear-text messages that go over an unprotected
network. A consequence is that everyone on this network can intercept this
communication and see what you are doing/reading/watching.
`{sparagraph}`
`{itemize}`
`{frame}`
`{smodule}`
`{document}`

`{sfragment}`
`{document}`

```
{sfragment}
{document}
```

```
File: [courses/FAU/IWGS/course]{webapps/sec/forms-recap.en}}
{document}
{sfragment}[id=sec.forms-recap]{Recap: HTML Forms Data Transmission}
File: [courses/FAU/IWGS/course]{webapps/snip/trans-recap.en}}
{document}
{sparagraph}
\usemodule{webapps/slides?forms-recap}
\usemodule[smglom/www]{mod?webapp}
```

The first two requirements for \sns{web application} above are already met by \sn{html?HTML} in terms of \sn{html?HTML} forms (see slide~\ref{slide.html-forms} ff.). Let us recap and extend\ednote{continue}

```
{sparagraph}
{document}
```

```
File: [courses/FAU/IWGS/course]{webapps/slides/forms-recap.en}}
{document}
{smodule}{forms-recap}
\lstset{language=HTML}
```

```
{frame}[fragile,label=slide.forms-recap1]
  {Recap \sn{html?HTML} Forms: Submitting Data to the Web Server}
{itemize}
```

```
{sparagraph}[title=Recall]
\sn{html?HTML} forms collect data via named \lstinline|input| elements, the submit
\sn{event} triggers a \sn{http-protocol?HTTP} request to the \sn{URL?URL} specified
in the \lstinline|action| attribute.
{sparagraph}
```

```
{sexample}[id=html-forms.ex,for=HTML form]
Forms contain input fields and explanations.
\lstinputmhlisting{webapps/code/form.html} yields the following in a
\sn{webbrowser?web browser}:
\cmhgraphics[width=11.5cm]{webapps/PIC/form}
Pressing the submit button activates a \sn{http-protocol?HTTP}
\sn{http-protocol?GET} request to the \sn{URL?URL}
\lstinline[mathescape]login.html?user=$\pmetavar{name}$&pass=$\pmetavar{passwd}$|
{sexample}
```

```
{sparagraph}[style=warning]
Never use the \sn{http-protocol?GET} method for submitting passwords \lec{see below}
{sparagraph}
{itemize}
{frame}
```

```
{nparagraph}
We can now use the tools any modern browser supplies to check up on this claim. In fact,
using the browser tools is essential for advanced web development. Here we use the web
```

console, that monitors any activity, to check upon what really happens when we \sn{interact} with the \sn{web page}.

{nparagraph}

{frame}[label=slide.forms-recap2]

{Checking up on the Transmission}

{itemize}

<1-> Let's verify the claims above using browser tools\lec{here the web console}

<1-> Loading the file and filling in the form:\lec{console logs file \sn{URI?URI}}

\only<1>{\cmhgraphics[width=11cm]{webapps/PIC/form-console1}}

<2-> After submitting the form:\lec{console logs the \sn{http-protocol?HTTP} request}

\only<2>{\cmhgraphics[width=11cm]{webapps/PIC/form-console2}}

{itemize}

{frame}

{nparagraph}

A side effect of re-playing our development in the browser is that we see another type of \sn{html-forms?input element}: A password field, which hides user input from un-authorized eyes. We also see that the \sn{http-protocol?GET} request incorporates the \sn{html-form-data?form data} which contains the password into the \sn{URI?URI} of the request, which is visible to everyone on the web. We will come back to this problem later.

{nparagraph}

{smodule}

{document}

File: [courses/FAU/IWGS/course]{webapps/slides/html-form-data.en}]

{document}

{smodule}{html-form-data}

{nparagraph}

Let us now look at the data transmission mechanism in more detail to see what is actually transmitted and how.

{nparagraph}

{frame}[label=slide.html-form-data,fragile]

{\sn{html?HTML} Forms and Form Data Transmission}

{itemize}

{sparagraph}

We specify the \sn{http-protocol?HTTP} communication of \sn{html?HTML} forms in detail.

{sparagraph}

{sdefinition}

The \sn{html?HTML} \linline{form} element groups the layout and \linline{input} elements:

{itemize}

\linline[mathescape]<form action="\pmetavar{URI}" method="\pmetavar{req}">| specifies the \sn{html-forms?form action} in terms of a \sn{http-protocol?HTTP request} \$\pmetavar{req}\$ to the \sn{URI?URI} \$\pmetavar{URI}\$.

The \define{form data} consists of a string \$\pmetavar{data}\$ of the form \linline[mathescape]\$n_1\$=\$v_1\$&&\cdots&&\$n_k\$=\$v_k\$, where

{itemize}

$\$n_i\$$ are the values of the `\$inline{name|}` attributes of the input fields and $\$v_i\$$ are their values at the time of submission.

{itemize}

`\$inline|<input type="submit" .../>|` triggers the `\$n{html-forms?form action}`: it composes a `\$n{http-protocol?HTTP request}`

{itemize}

If $\$pmetavar{req}\$$ is `\$inline|get|` (the default), then the browser issues a `\$n{http-protocol?GET}` request

$\$pmetavar{URI}\$$ `\$inline[mathescape]|?\$pmetavar{data}\$|`.

If $\$pmetavar{req}\$$ is `\$inline|post|`, then the browser issues a `\$n{http-protocol?POST}` request to

$\$pmetavar{URI}\$$ with document content $\$pmetavar{data}\$$.

{itemize}

{itemize}

{sdefinition}

We now also understand the form action, but should we use `\$n{http-protocol?GET}` or `\$n{http-protocol?POST}`.

{itemize}

{frame}

{smodule}

{document}

File: `[courses/FAU/IWGS/course]{webapps/snippet/getvspost.en}`

{document}

{sparagraph}

`\$usemodule[courses/Jacobs/GenCS/course]{www/slides?http-protocol}`

To understand whether we should use the `\$n{http-protocol?GET}` or

`\$n{http-protocol?POST}` methods, we have to look into the details, which we will now summarize.

{sparagraph}

{document}

File: `[courses/FAU/IWGS/course]{webapps/slides/getvspost.en}`

{document}

{smodule}{getvspost}

{frame}[label=slide.getvspost]

{Practical Differences between `\$n{http-protocol?HTTP}`
`\$n{http-protocol?GET}` and `\$n{http-protocol?POST}`}

{itemize}

{sparagraph}[title=Observation,title=Using GET vs. POST in HTML Forms,id=getvspost.obs]\\$strut

{center}

{tabular}{|I|I|I|}\\$hline

& `\$n{http-protocol?GET}` & `\$n{http-protocol?POST}`\\\\$hline\\$hline

Caching & possible & never\\\\$hline

Browser History & Yes & never \\\\$hline

Bookmarking & Yes & No \\\\$hline

Change Server Data & No & Yes \\\\$hline

Size Restrictions & $\leq 2\text{KB}$ & No \\\\$hline

Encryption & No & HTTPS\\\\$hline

{tabular}

{center}
{sparagraph}

{sparagraph}[title=Upshot]
\sn{http-protocol?HTTP} \sn{http-protocol?GET} is more convenient, but
less potent.
{sparagraph}

{sparagraph}[style=warning]
Always use \sn{http-protocol?POST} for sensitive data! \lec{passwords, personal data, etc.}\
\sn{http-protocol?GET} data is part of the \sn{URI?URI} and thus
unencrypted, \sn{http-protocol?POST} data via HTTPS is.
{sparagraph}
{itemize}
{frame}
{smodule}
{document}

{sfragment}
{document}

File: [courses/FAU/IWGS/course]{webapps/sec/serverside.en}]
{document}
{sfragment}[id=sec.serverside]{Generating HTML on the Server}
File: [courses/Jacobs/GenCS/course]{www/snip/serverside-intro.en}]
{document}
{sparagraph}
\usemodule{www/slides?webbrowser}
\usemodule{www/slides?webserver}
\usemodule[smglom/www]{mod?webapp}

As the \sn{WWW} is based on a \sn{client-server?client server architecture},
computation in \sns{web application} can be executed either on the
\sn{client-server?client} (the \sn{webbrowser?web browser}) or the
\sn{client-server?server} (the \sn{webserver?web server}). For both we have a
special technology; we start with computation on the \sn{webserver?web server}.
{sparagraph}
{document}

File: [courses/Jacobs/GenCS/course]{www/slides/serverside-scripting.en}]
{document}
{smodule}{serverside-scripting}
\usemodule[smglom/computing]{mod?programming}

{frame}[label=slide.serverside-scripting]
{Server-Side Scripting: \Sn{programming} \Sns{web-page?web page}}
{itemize}

{sparagraph}[title=Idea,id=sssf-idea]
Why write \sn{html?HTML} pages if we can also program them!\lec{easy to do}
{sparagraph}

```
{sdefinition}[id=sssf.def]
A \define{server-side scripting framework} is a \sn{webserver?web server}
extension that generates \sns{web-page?web page} upon
\sn{http-protocol?HTTP} requests.
{sdefinition}
```

```
{sexample}[for=server-side scripting framework]
\usemodule[courses/Jacobs/TDM/course]{doccomp/slides?perl}
$\PERLLanguage$ is a scripting language with good string manipulation
facilities. \inlinedef{\define{PERL CGI} is an early \sn{server-side scripting
framework} based on this.}
{sexample}
```

```
{sexample}[for=server-side scripting framework]
\usemodule[courses/FAU/IWGS/course]{webapps/slides?bottle-stpl}
\python is a scripting language with good string manipulation facilities. And
\sn{bottle-stpl?bottle WSGI} is a simple but powerful \sn{server-side scripting
framework} based on this.
{sexample}
```

```
{sparagraph}[title=Observation]
\usemodule[smglom/computing]{mod?database}
\Sns{server-side scripting framework} allow to make use of external
resources (e.g. \sns{database} or data feeds) and computational services
during \sn{web-page?web page} generation.
{sparagraph}
```

```
{sparagraph}[title=Observation]
\usemodule[smglom/computing]{mod?template-engine}
A \sn{server-side scripting framework} solves two problems:
{enumerate}
    making the development of functionality that generates \sn{html?HTML} pages
    convenient and \sn{efficient}, usually via a \sn{template-engine?template engine},
    and
    \inlinedef{binding such functionality to \sns{URL?URL} the
\define{post=s}{route}, we call this \define{routing}.}
{enumerate}
{sparagraph}
{itemize}
{frame}
{smodule}
{document}
```

```
File: [courses/FAU/IWGS/course]{webapps/snippet/bottle-trans.en}
{document}
{sparagraph}
\usemodule[courses/Jacobs/GenCS/course]{www/slides?serverside-scripting}
\usemodule[courses/Jacobs/GenICT/course]{python/slides/libraries?python-libraries}
We will look at the second problem: \sn{serverside-scripting?routing} first. There
is a dedicated \python \sn{python-libraries?library} for that.
{sparagraph}
{document}
```

File: [courses/FAU/IWGS/course]{webapps/sec/bottle.en}]
 {document}
 {sfragment}[id=sec.bottle]{Routing and Argument Passing in Bottle}
 File: [courses/FAU/IWGS/course]{webapps/snip/bottle-intro.en}]
 {document}
 {sparagraph}
 \usemodule[courses/Jacobs/GenICT/course]{python/slides/libraries?python-libraries}
 \usemodule[courses/Jacobs/GenCS/course]{www/slides?serverside-scripting}
 We wil now introduce the \lstinline|bottle| \sn{python-libraries?library}, which
 supplies a lightweight \sn{webserver?web server} and
 \sn{serverside-scripting?server-side scripting framework} \sn[post=ed]{implement} in
 \python. It is already \sn[post=ed]{install} on the JupyterLab cloud IDE at
 \url{http://jupyter.kwarc.info}. To \sn{install} it on your laptop, just type
 \lstinline|pip install bottle| in a \sn{shell}.
 {sparagraph}
 {document}

File: [courses/FAU/IWGS/course]{webapps/slides/bottle-routing.en}]
 {document}
 {smodule}{bottle-routing}
 \lstset{language=python,mathescape}

{frame}[label=slide.bottle-routing,fragile]
 {The Web Server and Routing in Bottle WSGI}
 {itemize}

{sdefinition}
 \definiendum[root=serverside routing]{routing}{Serverside routing} (or simply
 \define{routing}) is the process by which a \sn{webserver?web server}
 connects a \sn{http-protocol?HTTP} request to a function (called the
 \define{route function}) that provides a \sn{URI?web resource}. A single
 \sn{URI?URI} \sn{URI?path}/\sn{route function} pair is called a
 \define{route}.

{sdefinition}
 The \sn{bottle-stpl?bottle WSGI} \sn{python-libraries?library} supplies
 a simple \python \sn{webserver?web server} and \sn{routing}.
 {itemize}

The \lstinline[mathescape]|run(\$\pmetavar{keys}\$)| function starts the
 \sn{webserver?web server} with the configuration given in \$\pmetavar{keys}\$.

The \lstinline|@route| decorator connects \sn{URI?path}
 \sns{URI?component} to \python
 \sn{subroutine?function} that return \sr{words?word}{strings}.
 {itemize}

{sexample}[title=A Hello World route,id=bottle-hello.ex,for=routing]
 \ldots for \sn{localhost} on \sn{ports?port} 8080
 \lstinputmhlisting{webapps/code/hello.py}

This \sn{webserver?web server} answers to \sn{http-protocol?HTTP}
 \sn{http-protocol?GET} requests for the \sn{URL?URL}
 \url{http://localhost:8080/hello}
 {sexample}
 {itemize}

{frame}

{nparagraph}

Let us understand \sref{bottle-hello.ex} \sn{file-type?line} by \sn{file-type?line}: The first line imports the \sn{python-libraries?library}. The second establishes a \sn{route} with the name \linline|hello| and binds it to the \python function \linline|hello| in \sn{file-type?line} 3 and 4. The last \sn{file-type?line} configures the \linline|bottle| \sn{webserver?web server}: it serves content via the \sn{http-protocol?HTTP} protocol for \sn{localhost} on \sn{ports?port} 8080.

So, if we run the program from \sref{bottle-hello.ex}, then we obtain a \sn{webserver?web server} that will answer \sn{http-protocol?HTTP} \linline|GET| requests to the \sn{URL?URL} \url{http://localhost:8080/hello} with a \sn{http-protocol?HTTP} answer with the content \linline|Hello IWGS!|.

To keep the example simple, we have only returned a text string; A realistic application would have generated a full \sn{html?HTML} page (see below).

{nparagraph}

{nparagraph}

In the last \sn{file-type?line} of \sref{bottle-hello.ex}, we have also configured the \linline|bottle| \sn{webserver?web server} to use “\sr{debugger}{debug mode}”, which is very helpful during early development.

In this mode, the \linline|bottle| \sn{webserver?web server} is much more verbose and provides helpful \sn[post=ging]{debug} information whenever an \sr{bug}{error occurs}. It also disables some optimisations that might get in your way and adds some checks that warn you about possible misconfiguration.

Note that \sr{debugger}{debug mode} should be disabled in a production server for \sn{efficiency}.

{nparagraph}

{smodule}

{document}

File: [courses/FAU/IWGS/course]{webapps/slides/bottle-dynamic-routes.en}]

{document}

{smodule}{bottle-dynamic-routes}

\lstset{language=python,mathescape}

{nparagraph}

But we can do more with routes!

{nparagraph}

{frame}

{Dynamic Routes in Bottle}

{itemize}

{sdefinition}

A \define{dynamic route} is a route annotation that contains \define[post=s]{named wildcard}, which can be picked up in the

```
\sn{route function}.
{sdefinition}
```

```
{sexample}[id=bottle-dynamic-route.ex,for=dynamic route]
Multiple \stinline|@route| annotations per \sn{bottle-routing?route function}
$$ are allowed \ergo the \sn{web application} uses $$ to answer
multiple \sns{URL?URL}.
\stinputmhlisting[firstline=3,lastline=6]{webapps/code/hello-stranger.py} With the
\sr{named wildcard}{wildcard} \stinline|<name>| we can bind the
\sn{bottle-routing?route function} \stinline|greet| to all
\sns{URI?path} and via its argument \stinline|name| and customize
the greeting.
```

```
\titleemph{Concretely}: A \sn{http-protocol?HTTP} GET request to
{itemize}
\url{http://localhost} is answered with \stinline|Hello Stranger, how are you?|.
\url{http://localhost/hello/MiKo} is answered with \stinline|Hello MiKo, how are you?|.
{itemize}
```

```
Requests to e.g \url{http://localhost/hello} or
\url{http://localhost/hello/prof/kohlhase} lead to errors. \lec{404: not found}
{sexample}
{itemize}
{frame}
{smodule}
{document}
```

```
File: [courses/FAU/IWGS/course]{webapps/slides/bottle-route-filters.en}}
{document}
{smodule}{bottle-route-filters}
\lstset{language=python}
```

```
{nparagraph}
Often we want to have more control over the routes. We can get that by filters, which
can involve data types and/or \sr{regex}{regular expressions}.
{nparagraph}
```

```
{frame}[fragile]
{Restricting Dynamic Routes}
{itemize}
```

```
{sdefinition}
A \sn{bottle-dynamic-routes?dynamic route} can be restricted by a
\define{route filter} to make it more selective.
{sdefinition}
```

```
{sexample}[title=Concrete Filters,for=route filter]
We use \stinline|int| for integers and
\stinline[mathescape]|:re:$\pmetavar{regex}$| for
\sr{regex}{regular expressions}
\stinputmhlisting{webapps/code/route-filter.py}
```

```
Different route filters allow to classify paths and treat them differently.
{sexample}
```

{sparagraph}[title=Note,name=multi-wildcards.note]
Multiple \sns{bottle-dynamic-routes?named wildcard} are also possible,
in a \sn{dynamic route}; with and without \sr{route filter}{filters}
{sparagraph}

{sexample}[for=multi-wildcards.note,title=A route with two wildcards,for=named wildcard]
\stininputmhlisting{webapps/code/multi-wildcard.py}
{sexample}
{itemize}
{frame}
{smodule}
{document}

File: [courses/FAU/IWGS/course]{webapps/slides/bottle-request.en}]
{document}
{smodule}{bottle-request}
\lstset{language=python,mathescape}

{nparagraph}
We have already seen above that we want to use \sn{http-protocol?HTTP}
\sn{http-protocol?GET} and \sn{http-protocol?POST} request for different
facets of transmitting \sn{html?HTML} \sn{html-form-data?form data} to the
\sn{webserver?web server}. This is supported by \sn{bottle-stpl?bottle WSGI}
in two ways: we can specify the \sn{http-protocol?HTTP}
\sn{http-protocol?method} of a \sn{bottle-routing?route} and we have access to
the \sn{html-form-data?form data} (and other aspects of the request).
{nparagraph}

{frame}[fragile]
{Method-Specific Routes: \sn{http-protocol?HTTP}
\sn{http-protocol?GET} and \sn{http-protocol?POST}}
{itemize}

{sdefinition}
The \stinline|@route| decorator takes a \stinline|method| keyword to specify the
\sn{http-protocol?HTTP} request \sn{http-protocol?method} to be
answered. \lec{\sn{http-protocol?HTTP} \sn{http-protocol?GET} is the
default}
{itemize}
\stinline|@get(\$\pmetavar{path}\$)| abbreviates
\stinline|@route(\$\pmetavar{path}\$,method="GET")|
\stinline|@post(\$\pmetavar{path}\$)| abbreviates
\stinline|@route(\$\pmetavar{path}\$,method="POST")|
{itemize}
{sdefinition}

{sexample}[title=Login 1,id=login1.ex,for=login]
Managing \sns{login} with \sn{http-protocol?HTTP}
\sn{http-protocol?GET} and \sn{http-protocol?POST}.
\stininputmhlisting[linrange=1-11]{webapps/code/login.py}
{sexample}

{sparagraph}[title=Note]
We can also have a \sn{http-protocol?POST} request to the same

\sn{URI?path}; we use that for handling the \sn{html-form-data?form data} transmitted by the \sn{http-protocol?POST} action on submit.\lec{up next}

{sparagraph}

{itemize}

{frame}

{nparagraph}

Recall that we have already seen most of this in slide \ref{slide.forms-recap1}. The only new thing is that we return the \sn{html?HTML} as a string in the route function as a request to a \sn{http-protocol?HTTP} \sn{http-protocol?GET} request. Now comes the interesting part: the form uses the \sn{http-protocol?POST} \sn{http-protocol?method} in the form action and we have to specify a route for that. Recall from \sref[file=webapps/slides/getvspost.en]{getvspost.obs} that this allows for encrypted transmission, so we are less naive than our solution from slide \ref{slide.forms-recap1}.

{nparagraph}

{frame}

{Bottle Request: Dealing with \sn{http-protocol?POST} Data}

{itemize}

{sparagraph}[title=Recall]

from a \sn{html?HTML} form we get a \sn{http-protocol?GET} or \sn{http-protocol?POST} request with \sn{html-form-data?form data}

\stinline[mathescape]{\$n_1\$=\$v_1\$&&\cdots&&\$n_k\$=\$v_k\$} \lec{here}

\stinline|user=mkohlhase&login=noneofyourbusiness|}

{sparagraph}

\Sn{bottle-stpl?bottle WSGI} provides the \stinline|request| object for dealing with \sn{http-protocol?HTTP} request data.

{sexample}[title=Login 2,id=login2.ex,for={login}]

\usemodule[smglom/grammar]{mod?parser}

Continuing from \sref{login1.ex}: we \sn{parse} the request transmitted request and check password information:

\stinputmhlisting[linrange=13-20]{webapps/code/login.py}

We assume a \python function \stinline|check_login| that checks \sn{authentication} \sn{credential} and \sn{authenticator}, and keeps a list of \sr{log in}{logged in} users.

{sexample}

{itemize}

{frame}

{nparagraph}

The main new thing in~\sref{login2.ex} is that we use the \stinline|request.forms.get| method to \sn{query} the request object that comes with the \sn{http-protocol?HTTP} request triggering the route for the \sn{html-form-data?form data}.

{nparagraph}

{smodule}

{document}

{sfragment}

{document}

File: [courses/FAU/IWGS/course]{webapps/sec/stpl.en}]

{document}

{sfragment}[id=sec.stpl]{Templating in Python via STPL}

File: [courses/Jacobs/GenICT/course]{python/slides/html-in-python.en}]

{document}

{smodule}{html-python}

\lstset{language=python,aboveskip=2pt,belowskip=0pt}

{nparagraph}

In \useSGvar{courseacronym}, we use \python for \sn{programming}, so let us see how we would generate \sn{html?HTML} pages in \python.

{nparagraph}

{frame}

{What would we do in \python}

{itemize}

{sexample}[title=HTML Hello World in \python]

\lstinputmhlisting{python/code/hello.py}

{sexample}

{sparagraph}[title=Problem 1]

Most \sn{web-page?web page} content is static (page head, text blocks, etc.)

{sparagraph}

{sexample}[title=Python Solution]

\ldots use \python \sns{subroutine?function}:

\lstinputmhlisting[basicstyle=\small\sf]{python/code/hellofun.py}

{sexample}

{sparagraph}[title=Problem 2]

If \sn{html?HTML} markup dominates, want to use a \sn{html?HTML} editor (mode),

{itemize}

e.g. for \sn{html?HTML} syntax highlighting/indentation/completion/checking

{itemize}

{sparagraph}

{sparagraph}[title=Idea]

Embed \sn{program?program} snippets into \sn{html?HTML}.\lec{only execute these, copy rest}

{sparagraph}

{itemize}

{frame}

{smodule}

{document}

File: [courses/Jacobs/GenCS/course]{www/slides/template-engine.en}]

{document}

{smodule}{rc*template-engine}

\usemodule{www/slides?php}


```
{nparagraph}
We will now formalize and toolify the idea of “embedding code into
\sn{html?HTML}”. What comes out of this idea is called “templating”. It exists
in many forms, and in most \sns{programming language}.
{nparagraph}
```

```
{frame}
  {Template Processing for \sn{html?HTML}}
{itemize}
  \inputref[smglom/computing]{mod/template-engine.en}
```

```
{sparagraph}[title=Note]
No program code is left in the resulting \sn{web-page?web page} after
generation.\lec{important security concern}
{sparagraph}
```

```
{sparagraph}[title=Remark]
We will be most interested in \sn{html?HTML} \sns{template engine}.
{sparagraph}
```

```
{sparagraph}[title=Observation,id=sssf-routes-files]
We can turn a \sn{template engine} into a \sn{serverside-scripting?server-side
scripting framework} by employing the \sns{URI?URI} of
\sns{template-engine?template file} on a \sn{server} as
\sns{serverside-scripting?route} and extending the \sn{webserver?web server} by
\sn{template processing}.
{sparagraph}
```

```
{sexample}[id=PHP-sssf,for=server-side scripting framework]
\sn{php?PHP} (originally “Programmable Home Page Tools”) is a very successful
\sn{serverside-scripting?server-side scripting framework} following this model.
{sexample}
{itemize}
{frame}
{smodule}
{document}
```

```
File: [courses/FAU/IWGS/course]{webapps/slides/bottle-stpl.en}]
{document}
{smodule}{bottle-stpl}
\symdef{stplengine}{\comp{\mathrm{stpl}}}
\lstset{language=python}
```

```
{nparagraph}
\usemodule[smglom/www]{mod?webapp}
Naturally, \python comes with a \sn{template-engine?template engine}
in fact multiple ones. We will use the one from the bottle \sn{web application}
framework for \useSGvar{courseacronym}.
{nparagraph}
```

```
{frame}[fragile]% fragile needed.
  {${stplengine$: the “Simple Template Engine” from Bottle}
{itemize}
```

```
{sdefinition}[for=stplengine]
\Definame{bottle WSGI} supplies the \sn{template engine}
$\stplengine$ (Simple Template Engine).\lec{documentation at \cite{stpl:on}}
{sdefinition}
```

```
{sdefinition}[id=template-engine.def]
A \definame{template-engine?template engine} for a \sn{document-format?document
format} $F$ is a program that transforms
\definame[post=s]{template-engine?template}, i.e. \sr{words?word}{strings} or
\sns{file?file} through a mixture of program constructs and $F$ markup, into
$F$-strings or $F$-documents by executing the program constructs in the
\sn{template} (\definame{template-engine?template processing}).
{sdefinition}
```

```
{sparagraph}
$\stplengine$ uses the \linline|template| function for
\sn{template-engine?template processing} and
\linline[mathescape]{{{ $\text{\ldots} $ }}} to embed program objects into a
\sn{template-engine?template}; it returns a formatted
\sn{unicode} string. \linputmhlisting{webapps/code/stpl.py}
{sparagraph}
{itemize}
{frame}
{smodule}
{document}
```

```
File: [courses/FAU/IWGS/course]{webapps/slides/stpl-files.en}
{document}
{smodule}{stpl-files}
```

```
{nparagraph}
The $\stplengine$ \linline|template| function is a powerful enabling basic functionality in
\python, but it does not satisfy our goal of writing “\sn{html?HTML} with embedded
\python”. Fortunately, that can easily be built on top of the
\linline|template| functionality:
{nparagraph}
```

```
{frame}[fragile]% fragile needed.
{ $\stplengine$ Syntax and Template Files}
{itemize}
```

```
{sparagraph}[title=But what about\ldots]
\sn{html?HTML} files with embedded \python?
{sparagraph}
$ \stplengine$ uses \sns{template file} (extension \linline|.tpl|) for that.
```

```
{sdefinition}
A $ \stplengine$ \definame{template file} mixes \sn{html?HTML} with \definame{stpl python}:
{itemize}
\sn{stpl python} is exactly like \python but ignores indentation
and closes bodies with \linline|end| instead.
\sn{stpl python} can be embedded into the \sn{html?HTML} as
```

```
{itemize}
  a \define[post=s]{code line} starting with a \texttt{\char37},
  a \define[post=s]{code block} surrounded with \texttt{<\char37} and
\texttt{\char37 >}, and
  an \define[post=s]{stpl-files?expression} \linline[mathescape]|{\$ \pmetavar{exp}$}| as long
as \$\pmetavar{exp}$ evaluates to a string.
{itemize}
{itemize}
{sdefinition}
```

```
{sexample}[for=template file]
Two \sns{template file}\\
\parbox[c]{6.4cm}{\linputmhlisting[linewidth=6.4cm]{webapps/code/embedding.tpl}}\qqquad
\parbox[c]{3.8cm}{\linputmhlisting[linewidth=3.8cm]{webapps/code/for.tpl}}
{sexample}
{itemize}
{frame}
{smodule}
{document}
```

File: [courses/FAU/IWGS/course]{webapps/slides/stpl-functions.en}]

```
{document}
{smodule}{stpl-functions}
```

```
{nparagraph}
So now, we have template files. But experience shows that \sns{template file} can be
quite redundant; in fact, the better designed the \sn{web site} we want to create,
the more fragments of the \sns{template file} we want to reuse in multiple places --
with and without adaptations to the particular use case.
{nparagraph}
```

```
{frame}[fragile]
{Template Functions}
{itemize}
<1->
{sdefinition}
\sn{stpl-files?stpl python} supplies the \define[post=s]{template function}
{enumerate}
  \linline[mathescape]|include(\$ \pmetavar{tpl}$, \$ \pmetavar{vars}$)|, where
  \$ \pmetavar{tpl}$ is another \sn{template file} and \$ \pmetavar{vars}$ a set of
  variable declarations (for \$ \pmetavar{tpl}$).
  \linline[mathescape]|defined(\$ \pmetavar{var}$)| for checking definedness
  \$ \pmetavar{var}$
  \linline[mathescape]|get(\$ \pmetavar{var}$, \$ \pmetavar{default}$)|: return the
  value of \$ \pmetavar{var}$, or \$ \pmetavar{default}$
  \linline[mathescape]|setdefault(\$ \pmetavar{name}$, \$ \pmetavar{val}$)|
  {enumerate}
  {sdefinition}
<2->
{sexample}[title=Including Header and Footer in a template,for=template function]
In a coherent \sn{web-site?web site}, the
\sns{web-page?web page} often share common header and footer parts. Realize
this via the following page template:
```

```

\lstinputmhlisting{webapps/code/headerfooter.tpl}
{sexample}
<3->
{sexample}[title=Dealing with Variables and Defaults,for=template function]
\lstinputmhlisting{webapps/code/defaults.tpl}
{sexample}
{itemize}
{frame}
{smodule}
{document}

```

```

{sfragment}
{document}

```

```

File: [courses/Jacobs/GenCS/course]{www/slides/cookies.en}}
{document}
{smodule}{cookies}

```

```

{npargraph}
\usemodule[smglom/cs]{mod?loginout}

```

There is one problem however with \sns{web application} that is difficult to solve with the technologies so far. We want \sns{web application} to give the user a consistent user experience even though they are made up of multiple \sns{web-page?web page}. In a regular application we we only want to \sn{log in} once and expect the application to remember e.g. our username and password over the course of the various \sn[post=ions]{interact} with the system. For \sns{web application} this poses a technical problem which we now discuss.

```

{npargraph}

```

```

{frame}[label=slide.cookies]
{State in Web Applications and Cookies}
{itemize}
<1->
{sparagraph}[title=Recall]
\Sns{web application} contain multiple \sr{web page}{pages},
\sn{http-protocol?HTTP} is a stateless protocol.

```

```

{sparagraph}
<1->
{sparagraph}[title=Problem,id=webapp-state.problem]
How do we pass state between pages?\lec{e.g. username, password}

```

```

{sparagraph}
<2->
{sparagraph}[title=Simple Solution]
Pass information along in \sn{query} part of page \sns{URL?URL}.

```

```

{sparagraph}
<2->
{sexample}[title=HTTP GET for Single Login,for=HTTP]
Since we are generating pages we can generated augmented links
\only<2>\lstinputmhlisting[language=html,basicstyle=\small\tt]{www/code/cookies.html}}
{sexample}

```

```

<2->
{sparagraph}[title=Problem]
Only works for limited amounts of information and for a single session.
{sparagraph}
<3->
{sparagraph}[title=Other Solution]
Store \sn{state} \sn[post=ly]{persistent} on the \sn{client} hard
disk.
{sparagraph}
<3->
{sdefinition}[id=cookie.def]
A \define{cookie} is a text file stored on the client hard disk by the
\sn{web browser}. \Sns{webserver?web server} can request the
\sr{web browser}{browser} to store and send \sns{cookie}.
{sdefinition}
<4->
{sparagraph}[title=Note]
\Sns{cookie} are data, not programs, they do not generate pop ups or
behave like viruses, but they can include your log-in name and \sr{web browser}{browser} preferences.
{sparagraph}
<4->
{sparagraph}[title=Note]
\Sns{cookie} can be convenient, but they can be used to gather
information about you and your browsing habits.
{sparagraph}
<4->
{sdefinition}[id=third-party-cookie.def]
\Define[post=s]{third-party cookie} are used by advertising companies to track
users across multiple sites.\lec{but you can turn off, and even delete
\sns{cookie}}
{sdefinition}
{itemize}
{frame}

{npagraph}
Note that both solutions to the state problem are not ideal, for usernames and passwords
the \sn{URL?URL}-based solution is particularly problematic, since
\sn{http-protocol?HTTP} transmits \sns{URL?URL} in
\sn{http-protocol?GET} requests without encryption, and in our example passwords
would be visible to anybody with a packet sniffer. Here \sns{cookie} are
little better, since they can be requested by any website you visit.
{npagraph}
{smodule}
{document}

```

```

File: [courses/FAU/IWGS/course]{webapps/sec/contact-form.en}]
{document}
{sfragment}[id=sec.contact-form]{Completing the Contact Form}
File: [courses/FAU/IWGS/course]{webapps/slides/contact-form-recap.en}]
{document}
{smodule}{contact-form-recap}

```

```

{npagraph}
We are now equipped to finish the contact form example

```

We now come back to our worked \sn{html?HTML} example: the contact form from above. Here is the current state:

```
{nparagraph}

{frame}
{Back to our Contact Form (Current State)}
{itemize}
<1-> A contact form and message receipt \lec{communicate via
\sns{http-protocol?HTTP request}} \lstset{basicstyle=\footnotesize\sf}
{center}
\parbox[c]{5.5cm}{
\lstinline|contact4.html|
\lstinputmhlisting{digdocs/code/contact4.html}
\only<2>{\lstinputmhlisting{digdocs/code/contact-get.url}}}\qqquad
\parbox[c]{5cm}{
\lstinline|contact-after.html|
\lstinputmhlisting{digdocs/code/contact-after.html}
\only<2>{\lstinputmhlisting{digdocs/code/contact-after-get.url}}}
{center}
{onlyenv}<3>\centering
\parbox[c]{2cm}{\mhgraphics[width=2cm]{digdocs/PIC/browser4}}\hspace*{2cm}
\parbox[c]{5cm}{\mhgraphics[width=6cm]{digdocs/PIC/browser4-after}}
{onlyenv}
<4->
{sparagraph}[title=Problem]
The answer is a static \sn{html?HTML} document independent of
\sns{html-form-data?form data}.
{sparagraph}
<4->
{sparagraph}[title=Solution]
Generate the answer programmatically using the
\sns{html-form-data?form data}.\lec{up next}
{sparagraph}
{itemize}
{frame}
{smodule}
{document}
```

File: [courses/FAU/IWGS/course]{webapps/snip/contact-form-trans.en}]

```
{document}
{sparagraph}
\usemodule{webapps/slides?bottle-request}
\usemodule{smglom/computing}{mod?database}
There are two great flaws in the current state of the contact form:
{enumerate}
```

The “receipt page” \lstinline|contact-after.html| is static and does not take the data it receives from the contact form into account. It would be polite to give some record on what happened. We can fix this using \sn{bottle-stpl?bottle WSGI} using the methods we just learned.

Nothing actually happens with the message. It should be either entered into an internal message queue in a \sn{database} or ticketing system, or fed into an e-mail to a sales person. As we do not have access to the first, we will just use a

\python library to send an e-mail programmatically.

```
{enumerate}
{sparagraph}
{document}
```

File: [courses/FAU/IWGS/course][webapps/slides/contact-form-data.en]]

```
{document}
```

```
{smodule}{contact-form-data}
```

```
{frame}[label=slide.contact-form-data1]
```

```
{Completing the Contact Form}
```

```
{itemize}
```

\sn{bottle-stpl?bottle WSGI} has functionality (\stinline|request.GET| and
\stinline|request.POST|) to decode the \sn{html-form-data?form data} from a
\sn{http-protocol?HTTP request}. \lec{so we do not have to worry about the details}

```
{sexample}[title=Submitting a Contact Form,for=HTML form]
```

```
\stset{basicstyle=\small\sf}
```

We use a new route for \stinline|contact-form-after.html| with a corresponding
template file:

```
{center}
```

```
{tabular}{cc}
```

```
\stinline|contact.py| & \stinline|contact-after.tpl|\
```

```
\parbox[c]{6.5cm}{\stinputmhlisting{webapps/code/contact.py}}&
```

```
\parbox[c]{4.5cm}{\stinputmhlisting[linerange=4-14]{webapps/code/contact-after.tpl}}\
```

```
{tabular}
```

```
{center}
```

```
{sexample}
```

```
{itemize}
```

```
{frame}
```

```
{nparagraph}
```

Fortunately, the only remaining part: actually sending off an e-mail to the specified
mailbox is very easy: using the \stinline|smtplib| library we just create an e-mail
message object, and then specify all the components.

```
{nparagraph}
```

```
{frame}[label=slide.contact-form-data2]
```

```
{Sending off the e-mail}
```

```
{itemize}
```

We still need to \sn{implement} the \stinline|send-contact-email| function,
\ldots

Fortunately, there is a \python package for that: \stinline|smtplib|,
which makes this relatively easy.\lec{\sn{smtp-telnet?SMTP} \hateq
\sr{smtp-telnet?SMTP}{Simple Mail Transfer Protocol}}}

```
{sexample}[title=Continuing,for=HTML form]
```

```
\usemodule[smglom/cs]{mod?authentication}
```

```
\usemodule[smglom/cs]{mod?encryption}
```

```
\stinputmhlisting{webapps/code/send-contact-email.py}
```

Actually, this does not quite work yet as google requires \sn{authentication} and
\sn{encryption}, \ldots; \lec{google for “python smtplib gmail”}

```
{sexample}
```

```
{itemize}
{frame}
```

```
{nparagraph}
```

Once we have the e-mail message object `\stinline|msg|`, we open a “`\sn{smtp-telnet?SMTP}` connection” `\stinline|s|` send the message via its `\stinline|send_message|` method and close the connection by `\stinline|s.quit()|`. Again, the `\python` library hides all the gory details of the `\sn{smtp-telnet?SMTP}` protocol.

```
{nparagraph}
{smodule}
{document}
```

```
{sfragment}
{document}
```

```
{sfragment}
{document}
```

File: `[courses/FAU/IWGS/course]{webapps/sec/exercises.en}`
{document}

{nfragment}[id=sec.webapps-exercises]{Exercises}

File: `[courses/FAU/IWGS/course]{webapps/snip/exercises.en}`
{document}

{sparagraph}

`\usemodule[courses/Jacobs/GenCS/course]{www/slides?webapps}`

In the exercises in this `\currentsectionlevel`, we will take a closer look at `\sns{web application}`, templating and `\sn{html?HTML}` routing. Concretely, we will be using the Bottle framework`\footnote{See the documentation of bottle for reference:`

`\url{https://bottlepy.org/docs/dev/tutorial.html}`}, as demonstrated in the lecture.

```
{sparagraph}
{document}
```

`\includeproblem{webapps/prob/bottle_simple.en}`

`\includeproblem{webapps/prob/bottle_form.en}`

`\includeproblem{webapps/prob/bottle_get.en}`

`\includeproblem{webapps/prob/bottle_database.en}`

`\includeproblem{digdocs/prob/simple-css.en}`

```
{nfragment}
{document}
```

```
{sfragment}
{document}
```

File: `[courses/FAU/IWGS/course]{webapps/sec/frontend.en}`
{document}

{sfragment}[id=sec.frontend]{Frontend Technologies}

File: `[courses/FAU/IWGS/course]{webapps/snip/frontend.en}`


```
{document}
{sparagraph}
\usemodule[smglom/www]{mod?webapp}
\usemodule[courses/Jacobs/GenCS/course]{www/slides?CSS}
\usemodule[courses/Jacobs/GenCS/course]{www/slides?jquery}
We introduce three important concepts for building modern web front ends for
\sns{web application}:
{enumerate}
  Client-side computation: manipulating the \sr{web browser}{browser} \sn{DOM} via \sn{JavaScript}.
  \sr{CSS}{Cascading Stylesheets} (\sn{CSS}) for styling the layout of \sn{HTML} (and \sn{XML}).
  The \sn{jQuery} library: a symbiosis of \sn{JavaScript} and \sn{CSS} ideas to make
\sn{JavaScript} coding easier and more \sn{efficient}.
{enumerate}
{sparagraph}
{document}
```

```
File: [courses/FAU/IWGS/course]{webapps/sec/clientside.en}}
{document}
{sfragment}[id=sec.clientside]{Dynamic HTML: Client-side Manipulation of HTML Documents}
File: [courses/Jacobs/GenCS/course]{www/snippet/clientside-intro.en}}
{document}
{sparagraph}
We now turn to client-side computation:
{sparagraph}

{document}
```

```
File: [courses/Jacobs/GenCS/course]{www/snippet/clientside-interaction.en}}
{document}
{sparagraph}
\usemodule[smglom/math]{mod?mathematics}
\usemodule{www/slides?www-concepts}
\usemodule[smglom/computing]{mod?interpreter}
One of the main advantages of moving documents from their traditional ink-on-paper form
into an electronic form is that we can \sn{interact} with them more directly. But there
are many more \sn{post=ions}{interact} than just browsing \sns{hyperlink} we can think
of: adding margin notes, looking up definitions or translations of particular words, or
copy-and-pasting \sn{mathematical} formulae into a computer algebra system. All of them
(and many more) can be made, if we make documents programmable. For that we need three
ingredients:
{enumerate}[\em i)]
  a machine-accessible representation of the document structure, and
  a program \sn{interpreter} in the \sn{web browser}, and
  a way to send \sns{program} to the \sr{web browser}{browser}
together with the document.
{enumerate}
We will sketch the \sn{WWW} solution to this in the following.
{sparagraph}
{document}
```

```
File: [courses/Jacobs/GenCS/course]{www/snippet/dom-intro.en}}
{document}
```

```
{document}
{sparagraph}
\usemodule{www/slides?html}
\usemodule{www/slides?webbrowser}
```

To understand client-side computation, we first need to understand the way \sr{web browser}{browsers} render \sn{html?HTML} pages.

```
{sparagraph}
{document}
```

File: [courses/Jacobs/GenCS/course]{www/slides/browser-rendering-pipeline.en}]

```
{document}
{smodule}{browser-rendering-pipeline}
```

```
{frame}
{Background: Rendering Pipeline in \sr{web browser}{browsers}}
{itemize}
```

```
{sparagraph}[title=Observation]
The nested markup codes turn \sn{html?HTML} documents into trees.
{sparagraph}
```

```
{sdefinition}[id=DOM.def]
The \definiendum{DOM}{document object model} (\definame{DOM}) is a
\sn{data structure} for the \sn{html?HTML} document tree together with
a standardized set of access methods.
{sdefinition}
```

```
{sparagraph}[title=Rendering Pipeline]
Rendering a \sn{web-page?web page} proceeds in three steps
{enumerate}
the \sr{web browser}{browser} receives a \sn{html?HTML} document,
\sns{parse} it into an internal \sn{data structure}, the
\sn{DOM},
which is then painted to the screen. \lec{repaint whenever \sn{DOM} changes}
{enumerate}
```

```
\cmhtikzinput{www/tikz/dom-arch}
The \sn{DOM} is notified of any user events.\lec{resizing, clicks, hover,\ldots}
{sparagraph}
{itemize}
{frame}
```

```
{nparagraph}
The most important concept to grasp here is the tight synchronization between the
\sn{DOM} and the screen. The \sn{DOM} is first established by \sr{parse}{parsing}
(i.e. interpreting) the input, and is synchronized with the \sr{web browser}{browser} UI
and document viewport. As the \sn{DOM} is \sn{persistent} and synchronized, any change
in the \sn{DOM} is directly mirrored in the \sr{web browser}{browser} viewpoint, as a
consequence we only need to change the \sn{DOM} to change its presentation in the
\sr{web browser}{browser}. This exactly is the purpose of the client side scripting
language, which we will go into next.
```

```
{nparagraph}
{smodule}
{document}
```

File: [courses/Jacobs/GenCS/course]{www/sec/javascript.en}]

{document}

{sfragment}{JavaScript in HTML}

File: [courses/Jacobs/GenCS/course]{www/slides/javascript.en}]

{document}

{smodule}{javascript}

{frame}[label=slide.script]

{Dynamic \sn{html?HTML}}

{itemize}

{sdefinition}

We call a \sn{web page} \definame{dynamic}, if its presentation can change without the \sn{web browser} loading new content.

{sdefinition}

{sparagraph}[title=Idea]

Generate parts of the \sn{web-page?web page} \sn[post=ally]{dynamic} by manipulating the \sn{DOM}.

{sparagraph}

\inputref[smglom/www]{mod/javascript.en}

\sn{javascript?JavaScript} is standardized by ECMA in~\cite{ECMAScript09}.

{sexample}[id=script-simple.ex,for={HTML,DOM}]

We write the some text into \sn{html?HTML} \sn{DOM}.

\lstinputmhlisting[language=HTML,basicstyle=\footnotesize\ttfamily,

moreemph={[[2]script},moreemph={[[3]document.write}]{www/code/docwrite.html}

{sexample}

{sparagraph}[title=Application]

Write “gmail” or “google docs” as \sn{javascript?JavaScript} enhanced web applications.\lec{client-side computation for immediate reaction}

{sparagraph}

{sparagraph}[title=Current Megatrend]

\Sn{computation} in the “cloud”, \sr{web browser}{browsers} (or “apps”) as \sr{UI}{user interfaces}.

{sparagraph}

{itemize}

{frame}

{nparagraph}

The example above already shows a \sn{javascript?JavaScript} command:

\stinline|document.write|, which replaces the content of the \stinline|<body>| element with its argument -- this is only useful for testing and debugging purposes.

{nparagraph}

{nparagraph}

\usemodule[smglom/www]{mod?webapp}

Current \sns{web application} include simple office software (word processors, online spreadsheets, and presentation tools), but can also include more advanced applications such as project management, computer-aided design, video editing and point-of-sale. These are only possible if we carefully balance the effects of

server-side and client-side computation. The former is needed for computational resources and data persistence (data can be stored on the server) and the latter to keep personal information near the user and react to local context (e.g. screen size).

```
{nparagraph}
{smodule}
{document}
```

File: [courses/Jacobs/GenCS/course]{www/slides/js-browserlevel.en}]

```
{document}
{smodule}{js-browserlevel}
```

```
{nparagraph}
```

Here are three \sr{web browser}{browser} level functions that can be used for \sr{interact}{user interaction} (and finer debugging as they do not change the \sn{DOM}).

```
{nparagraph}
```

```
{frame}
```

```
{Browser-level \sn{javascript?JavaScript} functions: 1}
```

```
{itemize}
```

```
{sexample}[title=Logging to the \sr{web browser}{browser} console,
name=console-log.ex,for=subroutine?function]
```

```
\lstinputmhlisting[linenrange=3-3]{www/code/alert.html}
```

```
\cmhgraphics[width=4cm]{www/PIC/console.png}
```

```
{sexample}
```

```
{itemize}
```

```
{frame}
```

```
{note}
```

```
{sparagraph}[style=recap,for=console.log]
```

The \sn{subroutine?function} \sn{console.log} writes its \sn{subroutine?argument} into the \sn{console} of the \sn{web browser}.

```
{sparagraph}
```

```
{sparagraph}[style=elaboration,for=console.log,name=console.log-appl]
```

```
\usemodule[smglom/computing]{mod?bug}
```

\sr{console.log}{It} is primarily used for \sn[post=ging]{debug} the \sn{source code} of a \sn{web page}.

```
{sparagraph}
```

```
{sexample}[for=console.log-appl]
```

```
\usemodule[smglom/computing]{mod?bug}
```

If we want to know whether a \sn{subroutine?function} \lstinline|square| has been executed we add calls to \sn{console.log} like this:

```
{lstlisting}[gobble=6,language=JavaScript]
```

```
function square (n) {
```

```
  console.log ("entered function square with argument " + n);
```

```
  return (n * n);
```

```
  console.log ("exited function square with result " + n * n);
```

```
}
```

```
{lstlisting}
```

In the `\sn{console}` we can check whether the content contains
e.g. `\stinline|entered function square|` and
moreover whether `\sn{subroutine?argument}` and `\sn{value?value}` are as expected.

`{sexample}`
`{note}`

`{frame}`
`{Browser-level \sn{javascript?JavaScript} functions: 2}`
`{itemize}`

`{sexample}[title=Raising a \Sn{popup},for=JavaScript]`
`\stinputmhlisting[linrange=4-4]{www/code/alert.html}`
`\cmhgraphics[width=4cm]{www/PIC/alert.png}`

`{sexample}`
`{itemize}`
`{frame}`

`{note}`

`{sparagraph}[style=recap]`

The `\sn{subroutine?function}` `\sn{alert}` creates a `\sn{popup}` that contains the
`\sn{subroutine?argument}`.

`{sparagraph}`
`{note}`

`{frame}`
`{Browser-level \sn{javascript?JavaScript} functions: 3}`
`{itemize}`

`{sexample}[title=Asking for Confirmation,for=JavaScript]`
`\stinputmhlisting[linrange=5-5]{www/code/alert.html}`
`\cmhgraphics[width=4cm]{www/PIC/confirm.png}`

`{sexample}`
`{itemize}`
`{frame}`

`{note}`

`{sparagraph}[style=recap,name=popup.recap]`

The `\sn{subroutine?function}` `\sn{confirm}` creates a `\sn{popup}` that contains the
`\sn{subroutine?argument}` and a confirmation/cancel button pair and returns the corresponding
`\sr{Boolean data type}{Boolean}` `\sn{value?value}`.

`{sparagraph}`

`{sparagraph}[style=elaboration,for=popup.recap]`

If the user clicks on the confirmation button, the returned `\sn{value?value}` will be
`\sn{false}` and `\sn{true}` for the cancel button.

`{sparagraph}`

`{sexample}[for=JavaScript]`

You can play with this in the following frizzle:

`\stinputmhlisting[language=JavaScript]{www/code/confirm.html}`

`{sexample}`
`{note}`
`{smodule}`
`{document}`

File: [courses/Jacobs/GenCS/course]{www/slides/js-html.en}]

{document}

{smodule}{js-html}

{nparagraph}

\sn{javascript?JavaScript} is a client side \sn{programming language}, that means that the \sns{program} are delivered to the \sr{web browser}{browser} with the \sn{html?HTML} documents and is executed in the \sr{web browser}{browser}. There are essentially three ways of embedding \sn{javascript?JavaScript} into \sn{html?HTML} documents:

{nparagraph}

{frame}[label=slide.js-html1]

{Embedding \sn{javascript?JavaScript} into \sn{html?HTML}}

{itemize}

In a \linline|<script>| element in \sn{html?HTML}, e.g.

\linputmhlisting[language=html,linerange=3-5]{www/code/js-script.html}

External \sn{javascript?JavaScript} file via a \linline|<script>| element with \linline|src| \sn{xml-markup?attribute}:

\linputmhlisting[language=html,linerange=6-6]{www/code/js-script.html}

\ttitleemph{Advantage}: \sn{html?HTML} and \sn{javascript?JavaScript} code are clearly separated.

In \sns{event handler attribute} of various \sn{html?HTML} elements, e.g.

\linputmhlisting[language=html,linerange=9-9]{www/code/js-script.html}

{itemize}

{frame}

{nparagraph}

A related -- and equally important -- question is, \emph{when} the various embedded \sn{javascript?JavaScript} fragments are executed. Here, the situation is more varied

{nparagraph}

{frame}[label=slide.js-html2]

{Execution of \sn{javascript?JavaScript} Code}

{itemize}

{sparagraph}[title=Question]

When and how is \sn{javascript?JavaScript} code executed?

{sparagraph}

{sparagraph}[title=Answer]

While loading the \sn{html?HTML} page or afterwards triggered by \sns{event}.

{sparagraph}

\sn{javascript?JavaScript} in a \linline|script| element: during page load:\lec{not in a function}

\linputmhlisting[language=html,linerange=10-10]{www/code/js-script.html}

{sparagraph}

\sn{javascript?JavaScript} in \inlinedef{an \definame{event handler attribute} \linline|onclick|, \linline|ondblclick|, \linline|onmouseover|, \ldots} whenever the corresponding \sn{event} occurs}.

{sparagraph}

\sn{javascript?JavaScript} in a “special link”: when the anchor is clicked:

\linputmhlisting[language=html,linerange=11-11]{www/code/js-script.html}

```
{itemize}
{frame}
```

```
{nparagraph}
```

```
\usemodule[smglom/computing]{mod?stylesheet}
```

The first key concept we need to understand here is that the \sr{web browser}{browser} essentially acts as an user interface: it presents the \sn{html?HTML} pages to the user, waits for actions by the user -- \inlinedef{usually mouse clicks, drags, or gestures; we call them \definame[post=s]{event}} -- and reacts to them.

The second is that all \sns{event} can be associated to an element node in the \sn{DOM}: consider an \sn{html?HTML} anchor node, as we have seen above, this corresponds to a rectangular area in the \sr{web browser}{browser} \sn{window}. Conversely, for any point \$p\$ in the \sr{web browser}{browser} \sn{window}, there is a minimal \sn{DOM} element \$e(p)\$ that contains \$p\$ recall that the \sn{DOM} is a tree. So, if the user clicks while the mouse is at point \$p\$, then the \sr{web browser}{browser} triggers a \inline|click| \sn{event} in \$e(p)\$, determines how \$e(p)\$ handles a \inline|click| \sn{event}, and if \$e(p)\$ does not, bubbles the \inline|click| event up to the parent of \$e(p)\$ in the \sn{DOM} tree.

There are multiple ways a \sn{DOM} element can \sn{handle} an \sn{event}: some elements have default \sns{event handler}, e.g. an \sn{html?HTML} anchor \inline[mathescape]|| will handle a \inline|click| \sn{event} by issuing a \sn{http-protocol?HTTP} GET request for \$\pmetavar{URI}\$. Other \sn{html?HTML} elements can carry \sns{event handler attribute} whose \sn{javascript?JavaScript} content is executed when the corresponding \sn{event} is triggered on this element.

Actually there are more \sns{event} than one might think at first, they include:

```
{enumerate}
```

Mouse \sns{event}; \inline|click| when the mouse clicks on an element (touchscreen devices generate it on a tap); \inline|contextmenu|: when the mouse right-clicks on an element; \inline|mouseover| / \inline|mouseout|: when the mouse cursor comes over / leaves an element; \inline|mousedown| / \inline|mouseup|: when the mouse button is pressed / released over an element; \inline|mousemove|: when the mouse is moved.

Form element \sns{event}; \inline|submit|: when the visitor submits a \inline|<form>; \inline|focus|: when the visitor focuses on an element, e.g. on an \inline|<input>|.

Keyboard \sns{event}; \inline|keydown| and \inline|keyup|: when the visitor presses and then releases the button.

Document \sns{event}; \inline|DOMContentLoaded|: when the \sn{html?HTML} is loaded and processed, \sn{DOM} is fully built, but external resources like pictures \inline|| and \sns{stylesheet} may be not yet loaded. \inline|load|: the \sr{web browser}{browser} loaded all resources (images, styles etc); \inline|beforeunload| / \inline|unload|: when the user is leaving the page.

resource loading \sns{event}; \inline|onload|: successful load, \inline|onerror|: an error occurred.

```
{enumerate}
{nparagraph}
{smodule}
{document}
```

File: [courses/Jacobs/GenCS/course]{www/slides/js-html-ex.en}}

{document}

{smodule}{js-html-ex}

\usestructure{tree}

{nparagraph}

Let us now use all we have learned in an example to fortify our intuition about using

\sn{javascript?JavaScript} to change the \sn{DOM}.

{nparagraph}

{frame}

{Example: Changing Web Pages Programmatically}

{itemize}

{sexample}{title=Stupid but Fun,id=ex.pyramid,for=JavaScript}\strut\\

{minipage}[c]{8cm}

\lstinputmhlisting[linrange=2-18,linewidth=8cm,basicstyle=\small\sf]{www/code/triangle.html}

{minipage}\qqquad

{minipage}[c]{2.5cm}

{minipage}

{sexample}

{itemize}

{frame}

{nparagraph}

The \sn{html?HTML} document in \sref{ex.pyramid} contains an empty

\stinline|<div>| element whose \stinline|id| attribute has the value

\stinline|pyramid|. The subsequent \stinline|script| element contains some code that

builds a \sn{DOM} node-set of 10 text and \stinline|
| nodes in the

\stinline|triangle| variable. Then it assigns the \sn{DOM} node for the

\stinline|<div>| to the variable \stinline|elem| and deposits the \stinline|triangle|

node-set as \sn[post=ren]{child} into it via the \sn{javascript?JavaScript}

\stinline|innerHTML| method.

We see the result on the right of \sref{ex.pyramid}. It is the same as if the

\stinline|#|-strings and \stinline|
| sequence had been written in the

\sn{html?HTML} which at least for pyramids of greater depth would have

been quite tedious for the author.

{nparagraph}

{smodule}

{document}

{sfragment}

{document}

{sfragment}

{document}

File: [courses/FAU/IWGS/course]{digdocs/sec/css.en}}

{document}


```
{sfragment}[id=sec.css]{Cascading Stylesheets}
File: [courses/FAU/IWGS/course]{digdocs/snippet/css-intro.en}
{document}
{sparagraph}
\usemodule[courses/Jacobs/GenCS/course]{www/slides?CSS}
In this \currentsectionlevel we introduce a technology of digital documents which
naturally belongs into \sref[fallback=the discussion of digital
documents,file=digdocs/sec/digdocs.en]{sec.digdocs}: the specification of presentation
(layout, colors, and fonts) for marked-up documents.
{sparagraph}
{document}
```

```
File: [courses/FAU/IWGS/course]{digdocs/sec/css-intro.en}
{document}
{sfragment}[id=sec.css-intro]{Separating Content from Layout}
File: [courses/Jacobs/GenCS/course]{www/snippet/css-intro.en}
{document}
{sparagraph}
```

```
\usemodule{www/slides?CSS}
As the \sn{WWW} evolved from a hypertext system purely aimed at human readers to a
\sr{WWW}{Web of multimedia documents}, where machines perform added-value services
like searching or aggregating, it became more important that machines could understand
critical aspects \sns{web-page?web page}. One way to facilitate this is to
separate markup that specifies the content and functionality from markup that specifies
human-oriented layout and presentation (together called “styling”). This is what
“cascading style sheets” set out to do.
```

```
Another motivation for \sn{CSS?CSS} is that we often want the styling of a
\sn{web-page?web page} to be customizable (e.g. for vision impaired readers).
{sparagraph}
{document}
```

```
File: [courses/Jacobs/GenCS/course]{www/slides/CSS.en}
{document}
{smodule}{CSS}
```

```
{frame}[label=slide.css]
{\sn{CSS?CSS}: Cascading Style Sheets}
{itemize}
```

```
{sparagraph}[title=Idea]
Separate structure/function from appearance.
{sparagraph}
```

```
{sdefinition}[id=css.def]
\definiendum{CSS}{Cascading Style Sheets} (\sn{CSS?CSS}) is a \sn{style
language} that allows authors and users to attach \definame{style} (e.g., fonts,
colors, and spacing) to \sn{html?HTML} and \sn{xml?XML} documents.
{sdefinition}
```

```
{sexample}[id=css-simple.ex,for={HTML,CSS}]
Our \sn{text file} from \sref[file=www/slides/html.en]{html-simple.ex} with
embedded \sn{CSS?CSS}:
```

```
{columns}
{column}{5.6cm}
\lstinputmhlisting[language=HTML,basicstyle=\footnotesize\sf,belowskip=0pt]
{www/code/csshello.html}
{column}\quad
{column}{6cm}
{column}
{columns}
{sexample}
{itemize}
{frame}
```

{nparagraph}
Now that we have seen the example, let us fix the basic terminology of \sn{CSS?CSS}.

```
{frame}
{\sn{CSS?CSS}: Rules, Selectors, and Declarations}
{itemize}
```

```
{sdefinition}
A \sn{CSS?CSS} \definame{style sheet} consists of a sequence of
\definame[post=s]{rule} that in turn consist of a set of \definame[post=s]{selector}
that determine which \sn{xml?XML} \sr{XML element}{elements} the
\sn{rule} applies to and a \definame{declaration block} that specifies intended
presentation.
{sdefinition}
```

```
{sdefinition}
A \sn{CSS?CSS} \sn{declaration block} consists of a semicolon separated list of
\sns{declaration} in curly braces. Each \definame{declaration} itself consists of a
\definame{property}, a colon, and a \definame{CSS?value}.
{sdefinition}
```

```
{sexample}[id=css-simple-css.ex,for={CSS,rule}]
In \sref{css-simple.ex} we have three \sns{rule}, they address color and font
\sr{property}{properties}:
\lstinputmhlisting[language=HTML,linerange={4-7},gobble=7]{www/code/csshello.html}
{sexample}
```

```
{sparagraph}[title=Observation]
In modern \sns{web-site?web site}, \sn{CSS?CSS}
contributes as much -- if not more -- to the appearance as the choice of \sn{html?HTML}
elements.
{sparagraph}
{itemize}
{frame}
```

```
{nparagraph}
In \sref{css-simple-css.ex} the \sns{selector} are just
\sr{XML element}{element} names, they specify that the respective
\sns{declaration block} apply to all elements of this name.
{nparagraph}
{smodule}
{document}
```

File: [courses/Jacobs/GenCS/course]{www/slides/css-ex.en}}

{document}

{smodule}{css-ex}

\lstset{language=HTML,basicstyle=\footnotesize\sf}

{nparagraph}

We explore this new technology by way of an example. We rework the title box from the \sn{html?HTML} example above -- after all treating author/affiliation information as headers is not very semantic. Here we use \stinline|div| and \stinline|span| elements, which are generic block-level (i.e. paragraph-like) and inline containers, which can be styled via \sn{CSS?CSS} classes. The class \stinline|titlebox| is represented by the \sn{CSS?CSS} \sn{CSS?selector} \stinline|.titlebox|.

{nparagraph}

{frame}

{A Styled \sn{html?HTML} Title Box (Source)}

{itemize}

{sexample}[title=A style Title Box,for=HTML]

The \sn{html?HTML} source:

\stinputmhlising[firstline=2,lastline=14]{www/code/styled-html.html}

And the \sn{CSS?CSS} file referenced in the \stinline|<link>| element in

\sn{file-type?line} 3:

\stinputmhlising{www/code/style.css}

{sexample}

{itemize}

{frame}

{nparagraph}

And here is the result in the \sr{web browser}{browser}:

{nparagraph}

{frame}

{A Styled \sn{html?HTML} Title Box (Result)}

\cmhgraphics[archive=courses/Jacobs/GenCS/course,width=9cm]{www/PIC/styled-html}

{frame}

{smodule}

{document}

{sfragment}

{document}

File: [courses/FAU/IWGS/course]{digdocs/sec/css-fragment.en}}

{document}

{sfragment}[id=sec.css-fragment]{A small but useful Fragment of CSS}

File: [courses/FAU/IWGS/course]{digdocs/snip/css-fragment.en}}

{document}

{sparagraph}

\usemodule[courses/Jacobs/GenCS/course]{www/slides?CSS}

\sn{CSS?CSS} is a huge ecosystem of technologies, which is spread out over about 100 particular specifications -- see~\cite{w3c:css-allspecs:on} for an overview.

We will now go over a small fragment of \sn{CSS?CSS} that is already very useful for web applications in more detail and introduce it by example. For a more complete introduction, see e.g.~\cite{W3Schools:CSS-tutorial}.

{sparagraph}
{document}

File: [courses/FAU/IWGS/course]{digdocs/slides/css-selectors.en}]

{document}
{smodule}{CSS-selectors}
\lstset{language=HTML,mathescape}

{nparagraph}
Recall that \sns{CSS?selector} are the part of \sn{CSS?CSS} \sns{CSS?rule} that determine what \sr{XML element}{elements} a \sn{CSS?rule} affects. We now give the most important cases for our applications.

{nparagraph}

{frame}{fragile}
{\sn{CSS?CSS} \Sns{CSS?selector}}
{itemize}

{sparagraph}[title=Question]
Which \sr{XML element}{elements} are affected by a \sn{CSS?CSS} \sn{CSS?rule}?

{sparagraph}
\sr{XML element}{Elements} of a given name (optionally with given
\sns{xml-markup?attribute})
{itemize}
\Sns{CSS?selector}: name \hateq \lstinline|\$\pmetavar{elname}\$|,
\sns{xml-markup?attribute} \hateq
\lstinline|[\$\pmetavar{attname}]\$=\$\pmetavar{attval}]\$|
{itemize}

{sexample}[for=CSS?selector]
\lstinline|p[xml:lang='de']| applies to \lstinline|<p xml:lang="de">\$\ldots\$</p>|
{sexample}
Any \sr{XML element}{element} with a given \lstinline|class| \sns{xml-markup?attribute}
{itemize}
\Sn{CSS?selector}: \lstinline|. \$\pmetavar{classname}\$|
{itemize}

{sexample}[for=CSS?selector]
\lstinline|.important| applies to
\lstinline|<\$\pmetavar{el}\$ class='important'>\$\ldots\$</\$\pmetavar{el}\$>|
{sexample}
The \sr{XML element}{element} with a given \lstinline|id| \sn{xml-markup?attribute}
{itemize}
\Sn{CSS?selector}: \textsf{\#}\$\pmetavar{id}\$
{itemize}

{sexample}[for=CSS?selector]
\textsf{\#myRoot} applies to
\lstinline|<\$\pmetavar{el}\$ id='myRoot'>\$\ldots\$</\$\pmetavar{el}\$>|
{sexample}

{sparagraph}[title=Note]

Multiple \sns{CSS?selector} can be combined in a comma separated list.

{sparagraph}

For a full list see \url{https://www.w3schools.com/cssref/css_selectors.asp}.

{itemize}

{frame}

{smodule}

{document}

File: [courses/FAU/IWGS/course]{digdocs/slides/css-boxmodel.en}]

{document}

{smodule}{css-boxmodel}

{nparagraph}

We now come to one of the most important conceptual parts of \sn{CSS?CSS}: the \sr{CSS box model}{box

model}. Understanding it is essential for dealing with \sn{CSS?CSS} based layouts.

{nparagraph}

{frame}

{The \sr{CSS box model}{CSS Box Model}}

{itemize}

{sdefinition}

For layout, \sn{CSS?CSS} considers all \sn{html?HTML} \sr{XML element}{elements} as \definame[post=es]{box}, i.e. document areas with a given \definame{width} and \definame{height}. A \sn{CSS?CSS} \sn{box} has four parts:

{itemize}

\definame{content}: the content of the \sn{box}, where text and \sr{digital image}{images} appear.

\definame{padding}: clears an area around the \sn{content}. The \sn{padding} is transparent.

\definame{border} a border that goes around the \sn{padding} and \sn{content}.

\definame{margin} clears an area outside the \sn{border}. The \sn{margin} is transparent.

{itemize}

The latter three wrap around the \sn{content} and add to its size.

{sdefinition}

All parts of a \sn{box} can be customized with suitable \sn{CSS?CSS}

\sr{CSS?property}{properties}:

{center}

\parbox[c]{5cm}{\lstinputmhlisting[linerange=4-10,linewidth=5cm]{digdocs/code/css-boxmodel.html}}

\quad\parbox[c]{4cm}{\mhgraphics[width=5cm]{digdocs/PIC/css-boxmodel}}

{center}

Note that the overall \sn{width} of the \sn{CSS?CSS} \sn{box} is

$300 + 2 \cdot 3 \cdot 25 = 450$ pixels.

{itemize}

{frame}

{nparagraph}

As a summary of the above, we can visualize the \sn{CSS box model} in a diagram:

{nparagraph}

{frame}

{The \sr{CSS box model}{CSS Box Model}: Diagram}

{itemize}

{sdefinition}

The following diagram summarizes the \definame{CSS box model}

\cmhtikzinput{digdocs/tikz/css-boxmodel}

{sdefinition}

{itemize}

{frame}

{smodule}

{document}

File: [courses/FAU/IWGS/course]{digdocs/slides/css-cascading.en}}

{document}

{smodule}{css-cascading}

\usemodule{digdocs/slides?css-inheritance}

\usemodule{digdocs/slides?inspector}

{nparagraph}

We now come to a topic that is quite mind-boggling at first: The “cascading” aspect of \sn{CSS?CSS} \sns{stylesheet}. Technically, the story is quite simple, there are two independent mechanisms at work:

{itemize}

\inlinedef{\definame{inheritance}: if an \sr{XML element}{element} is fully contained in another, the inner (usually) \sns{inherit} all properties of the outer}.

\inlinedef{\sn{CSS?rule} \definame{priorization}: if more than one selector applies to an \sr{XML element}{element} (e.g. one by \sr{XML element}{element} name and one by \linline{id} \sn{xml-markup?attribute}), then we have to determine what \sn{CSS?rule} applies}.

{itemize}

Technically, \sn{priorization} takes care of them in an integrated fashion.

{nparagraph}

{frame}

{Cascading of \sns{CSS?selector} in \sn{CSS?CSS}: \Sn{priorization}}

{itemize}

Multiple \sn{CSS?CSS} \sns{CSS?selector} apply with the following \sns{priorization}:

{enumerate}

\label{li:important} important (i.e. marked with \linline{!important}) before unimportant

inline (specified via the \linline{style} \sn{xml-markup?attribute})

media-specific \sns{CSS?rule} before general ones

user-defined \sn{CSS?CSS} \sn{stylesheet} (e.g. in the \$firefoxbrowser\$ profile)

specialized before general \sns{CSS?selector}\lec{complicated; see e.g.~\cite{Wikipedia:CSS-specificity}}

\sn{CSS?rule} order: later before earlier \sns{CSS?selector}

\label{li:inheritance} parent inheritance: unspecified properties are \sn{post=ed}{inherit} from the \sn{parent}.

\Sn{style sheet} included or referenced in the \sn{html?HTML} document.
browser default
{enumerate}
{itemize}
{frame}

{nparagraph}

But do not despair with this technical specification, you do not have to remember it to be \sn{effective} with \sn{CSS?CSS} practically, because the \sns{CSS?rule} just encode very natural “behavior”. And if you need to understand what the browser -- which \sns{implement} these \sns{CSS?rule} -- really sees, use the integrated \sn{page inspector} tool (see slide~\ref{slide.inspector} for details).

{nparagraph}

{nparagraph}

We now look at an example to fortify our intuition.

{nparagraph}

{frame}[fragile]

{Cascading of \sns{CSS?selector} in \sn{CSS?CSS}: \Sn{priorization} Example}
{itemize}

{sexample}[for=CSS]

Can you explain the colors in the \sns{webbrowser?web browser} below?

\lstinputmhlisting[linewidth=10cm,linerange=11-14]{digdocs/code/css-cascading.html}

\parbox[c]{8.5cm}{\lstinputmhlisting[linewidth=8.1cm,linerange=4-7]{digdocs/code/css-cascading.html}}\quad

\parbox[c]{2.5cm}{\mhgraphics[width=2.5cm]{digdocs/PIC/css-cascading}}

{sexample}

{itemize}

{frame}

{nparagraph}

For instance, the words \nlex{very important} get a red background, as the class \lstineline|markedimportant| is marked as important by the \sn{CSS?CSS} \sn{keyword} \lstineline|!important|, which makes (cf. \sn{CSS?rule} \ref{li:important} above) the color red win against the color yellow inherited from the parent \lstineline|<div>|\sr{XML element}{element} (\sn{CSS?rule} \ref{li:inheritance} above).

{nparagraph}

{smodule}

{document}

File: [courses/FAU/IWGS/course]{digdocs/slides/css-inheritance.en}]

{document}

{smodule}{css-inheritance}

{nparagraph}

Let us now look at \sn{CSS?CSS} \sn{inheritance} in a little more detail.

{nparagraph}

{frame}[label=slide.cssi]

{Cascading in \sn{CSS?CSS}: Inheritance}

{itemize}

```

{sdefinition}
\Sn{child} \sr{XML element}{elements} can \definame[post=s]{inherit} some
\sr{CSS?property}{properties} (called \definame{inheritable}) from their
\sns{parent}. In a nutshell:
{itemize}
  text-related \sr{CSS?property}{properties} are \sn{inheritable};
e.g. \stinline|color|, \stinline|font|, \stinline|letter-spacing|,
\stinline|line-height|, \stinline|list-style|, and \stinline|text-align|
  \sn{box}-related \sr{CSS?property}{properties} are not;
e.g. \stinline|background|, \stinline|border|, \stinline|display|,
\stinline|float|, \stinline|clear|, \stinline|height|, \stinline|width|,
\stinline|margin|, \stinline|padding|, \stinline|position|, and
\stinline|text-align|.
{itemize}
{sdefinition}

```

```

{sparagraph}[title=Note]
\Sn{inheritance} is integrated into \sn{priorization}. \lec{recall case 7. above}
{sparagraph}
\Sn{inheritance} makes for consistent text \sr{CSS?property}{properties} and
smaller \sn{CSS?CSS} \sns{stylesheet}.
{itemize}
{frame}
{smodule}
{document}

```

```

File: [courses/FAU/IWGS/course]{digdocs/snippet/css-layout-trans.en}
{document}
{sparagraph}
\usemodule[courses/Jacobs/GenCS/course]{www/slides?CSS} So far, we have looked at the
mechanics of \sn{CSS?CSS} from a very general perspective. We will now come to a set of
\sn{CSS?CSS}
behaviors that are useful for specifying layouts of pages and texts.
{sparagraph}

{document}

```

```

File: [courses/FAU/IWGS/course]{digdocs/slides/cssflow.en}
{document}
{smodule}{cssflow}
\lstset{language=HTML}

{nparagraph}
Recall that \sn{CSS?CSS} is based on the \sn{css-boxmodel?box} model, which understands
\sn{html?HTML} \sr{XML element}{elements} as \sn[post=es]{box}, and layouts as
\sr{property}{properties} of \sr{css-boxmodel?box}{boxes} nested in
\sr{css-boxmodel?box}{boxes} (as the corresponding \sn{html?HTML} \sr{XML element}{elements} are).

```

If we can specify how inner \sn[post=es]{box} \sn{float} inside outer \sn[post=es]{box} -- via the \sn{CSS?CSS} \stinline|float| \sns{CSS?rule}, we can already do quite a lot, as the following examples show.

{nparagraph}

{frame}[t,fragile,label=slide.cssflow]

{\sn{CSS Flow}: How \sn[post=es]{box} \sn{float} to their Place}

{itemize}

{sdefinition}

\define{CSS Flow} describes how different \sr{XML element}{elements} are distributed in the visible area. \lec{how they flow; hence the name}\

The \define{float} property allows to influence that.

{sdefinition}

<1->

{sexample}[for={CSS,float}]

Block-level \sn[post=es]{box} (here div \sn{graph?node}) \sn{float} to the left:\

{onlyenv}<1>

\parbox[c]{4.7cm}{\lstinputmhlising[linewidth=4.7cm,linrange=13-16]{digdocs/code/cssflow-boxeffect.html}}+

\parbox[c]{5cm}{\lstinputmhlising[linewidth=5.5cm,linrange=4-9]{digdocs/code/cssflow-boxeffect.html}}=

\parbox[c]{.7cm}{\mhgraphics[width=.7cm]{digdocs/PIC/cssflow-boxeffect}}

{onlyenv}

{sexample}

<2->

{sexample}[for={CSS,float}]

\stinline[mathescape]|float:left| \sns{float} \sn[post=es]{box} as far as they will

go:\lec{without overlap}

{onlyenv}<2>

\parbox[c]{4.7cm}{\lstinputmhlising[linewidth=4.7cm,linrange=14-17]{digdocs/code/cssflow-float.html}}+

\parbox[c]{5cm}{\lstinputmhlising[linewidth=5.5cm,linrange=4-10]{digdocs/code/cssflow-float.html}}=\

\parbox{5cm}{\mhgraphics[width=5cm]{digdocs/PIC/cssflow-float}}

{onlyenv}

{sexample}

<3->

{sexample}[for={CSS,float}]

\stinline[mathescape]|float:right| in a \stinline|div| will \sn{float} inside the corresponding \sn{box}.\

{onlyenv}<3>

\parbox[c]{5.5cm}{\lstinputmhlising[linewidth=5.5cm,linrange=19-24]{digdocs/code/cssflow-position.html}}+

\parbox[c]{4.3cm}{\lstinputmhlising[linewidth=4.3cm,linrange=10-15]{digdocs/code/cssflow-position.html}}=\

\parbox{5cm}{\mhgraphics[width=5cm]{digdocs/PIC/cssflow-position}}

{onlyenv}

{sexample}

<4->

{sexample}[for={CSS,float}]

\stinline[mathescape]|float:left| will let contents flow around an obstacle\

{onlyenv}<4>

\parbox[c]{5.5cm}{\lstinputmhlising[linewidth=5.5cm,linrange=19-24]{digdocs/code/cssflow-position2.html}}+

\parbox[c]{4.3cm}{\lstinputmhlising[linewidth=4.3cm,linrange=10-15]{digdocs/code/cssflow-position2.html}}=\

\parbox{5cm}{\mhgraphics[width=5cm]{digdocs/PIC/cssflow-position2}}\

The large space (\$>\$2px) is caused because there is no linebreaking.

{onlyenv}

{sexample}

```
{itemize}
{frame}
{smodule}
{document}
```

File: [courses/Jacobs/GenCS/course]{www/slides/responsive-design.en}]

```
{document}
{smodule}{responsive-design}
{nparagraph}
```

One of the important applications of the content/form separation made possible by \sn{CSS?CSS} is to tailor \sn{web-page?web page} layout to the screen size and resolution of the device it is viewed on. Of course, it would be possible to maintain multiple layouts for a \sn{web-page?web page} one per screensize/resolution class, but a better way is to have one layout that changes according to the device context. This is what we will briefly look at now.

```
{nparagraph}
```

```
{frame}
{\sn{CSS?CSS} Application: Responsive Design}
{itemize}
```

```
{sparagraph}[title=Problem]
```

What is the screen size/resolution of my device?

```
{sparagraph}
```

```
{sdefinition}
\definiendum[root=responsive web design]{RWD}{Responsive web design}
(\definame{RWD}) designs web documents so that they can be viewed with a minimum of
resizing, panning, and scrolling -- across a wide range of devices (from desktop
monitors to mobile phones).
```

```
{sdefinition}
```

```
{sexample}[for=web page]
A \sn{web-page?web page} with content blocks
```

```
{center}
```

```
{tabular}{ccc}
```

Desktop & Tablet & Phone\\

```
{tabular}
```

```
{center}
```

```
{sexample}
```

```
{sparagraph}[title=Implementation]
```

\sn{CSS?CSS} based layout with relative sizes and

\inlinedef{\definiendum{media query}{media queries}\xspace -- \sn{CSS?CSS} conditionals based on client screen size/resolution\ldots}

```
{sparagraph}
```

```
{itemize}
```

```
{frame}
```

```
{smodule}
```

```
{document}
```

```
{sfragment}
```

{document}

File: [courses/FAU/IWGS/course]{digdocs/sec/css-tools.en}]

{document}

{sfragment}[id=sec.css-tools]{CSS Tools}

File: [courses/FAU/IWGS/course]{digdocs/snippet/css-tools.en}]

{document}

{sparagraph}

\usemodule[courses/Jacobs/GenCS/course]{www/slides?CSS}

\usemodule[courses/Jacobs/GenCS/course]{www/slides?browser-rendering-pipeline}

In this \currentsectionlevel we introduce a technology of digital documents which

naturally As \sn{CSS?CSS} has grown to be very complex and moreover, the

\sr{webbrowser?web browser}{browser} \sn{DOM} of which \sn{CSS?CSS} is

part can even be modified after loading the \sn{html?HTML} (see \sref[fallback=next

chapter,file=webapps/sec/clientside.en]{sec.clientside}), we need tools to help us

develop \sn{effective} and maintainable \sn{CSS?CSS}.

{sparagraph}

{document}

File: [courses/FAU/IWGS/course]{digdocs/slides/inspector.en}]

{document}

{smodule}{inspector}

{frame}[label=slide.inspector]

{But how to find out what the \sn{web browser} really sees?}

{itemize}

\sn{CSS?CSS} has many interesting \sn{inheritance} \sns{CSS?rule}.

{sdefinition}

The \definame{page inspector} tool gives you an overview over the internal state of the \sn{web browser} and its \sn{DOM}.

{sdefinition}

{sexample}[for=page inspector]

\cmhgraphics[width=11cm]{digdocs/PIC/inspector}

{sexample}

{itemize}

{frame}

{smodule}

{document}

File: [courses/FAU/IWGS/course]{digdocs/slides/colorpicker.en}]

{document}

{smodule}{colorpicker}

{nparagraph}

In \sn{CSS?CSS} we can specify colors by various names, but the full range of possible colors can only be specified by numeric (usually \sn{hexadecimal})

numbers. For instance in \sref[fallback=the initial example for

CSS,archive=courses/Jacobs/GenCS/course,file=/www/slides/CSS.en]{css-simple.ex}, we specified the background color of the page as \stinline|#d0e4fe|, which is a pain for the author.

Fortunately, there are tools that can help.

{nparagraph}

{frame}[fragile]

{Picking \sn{CSS?CSS} Colors}

{itemize}

{sparagraph}[title=Problem]

Colors in \sn{CSS?CSS} are specified by funny names (e.g. \lstinline|CornflowerBlue|) or \sn{hexadecimal} numbers, (e.g. \lstinline|#6495ED|).

{sparagraph}

{sparagraph}[title=Solution]

Use an online color picker,

e.g. \url{https://www.w3schools.com/colors/colors_picker.asp}

\cmhgraphics[width=11cm]{digdocs/PIC/colorpicker}

{sparagraph}

{itemize}

{frame}

{smodule}

{document}

{sfragment}

{document}

File: [courses/FAU/IWGS/course]{digdocs/sec/css-contact.en}]

{document}

{sfragment}[id=sec.css-contact]{Worked Example: The Contact Form}

File: [courses/FAU/IWGS/course]{digdocs/slides/css-worked-example.en}]

{document}

{smodule}{css-worked-example}

\usemodule{digdocs/slides?css-boxmodel}

\lstset{language=HTML}

{nparagraph}

To fortify our intuition on \sn{CSS?CSS}, we take up the “contact form” example from above and improve the layout in a step-by-step process concentrating on one aspect at a time.

{nparagraph}

{frame}[fragile,t,label=slide.css-worked-example]

{CSS in Practice: The Contact Form Example (Continued)}

{itemize}

<1-> Recap: The unstyled contact form -- \only<2->{Dream vs. Reality}

{onlyenv}<1,2>

{center}

\only<1>{\parbox[c]{6.5cm}{\lstinputmhlisting{digdocs/code/contact4.html}}}

\only<2>{\parbox[c]{5cm}{\cmhgraphics[width=5cm]{digdocs/PIC/design}}}

\qqquad

\parbox[c]{3cm}{\mhgraphics[width=3cm]{digdocs/PIC/browser4}}

{center}

{onlyenv}

<3-> Add a \sn{CSS?CSS} file with font information

```

{onlyenv}<3>
{center}
\parbox[c]{6.8cm}{
\lstinputmhlisting[linerange={2-3,11-12}]{digdocs/code/csscontact1.html}
\clstinputmhlisting{digdocs/code/csscontact1.css}}\qqquad
\parbox[c]{3cm}{\mhgraphics[width=3cm]{digdocs/PIC/cssbrowser1}}
{center}
{onlyenv}
<4-> Add lots of color \lec{oops, what about the size}
{onlyenv}<4>
{center}
\parbox[c]{5.5cm}{
\lstinputmhlisting[linerange={5-5,8-8,9-11},basicstyle=\footnotesize\sf]
{digdocs/code/csscontact2.html}
\clstinputmhlisting[linerange={7-10}]{digdocs/code/csscontact2.css}}\qqquad
\parbox[c]{5cm}{\mhgraphics[width=5cm]{digdocs/PIC/cssbrowser2}}
{center}
{onlyenv}
<5-> Add size information and a dotted frame
{onlyenv}<5>
{center}
\parbox[c]{7cm}{
\lstinputmhlisting[linerange={4-15},basicstyle=\footnotesize\sf]
{digdocs/code/csscontact3.html}}\quad
\parbox[c]{3cm}{\mhgraphics[width=3.5cm]{digdocs/PIC/cssbrowser3}}
{center}
{onlyenv}
<6-> Add a cat that plays with the submit button\lec{because we can}
{onlyenv}<6>
{center}
\parbox[c]{5cm}{
\lstinputmhlisting[linerange={8-11},basicstyle=\footnotesize\sf]
{digdocs/code/csscontact4.html}}\quad
\parbox[c]{5.5cm}{\mhgraphics[width=6cm]{digdocs/PIC/cssbrowser4}}
{center}
{onlyenv}
{itemize}
{frame}

```

{nparagraph}

This worked example should be enough to cover most layout needs in practice. Note that in most use cases, these generally layout primitives will have to be combined in different and may be even new ways.

{nparagraph}

{nparagraph}

Actually, the last “improvement” may have gone a bit overboard; but we used it to show how absolute positioning of `\sr{digital image}{images}` (or actually any `\sn{CSS?CSS}` `\sn[post=es]{box}` for that matter) works in practice.

{nparagraph}

{smodule}

{document}

{sfragment}

{document}

{sfragment}
{document}

File: [courses/FAU/IWGS/course]{webapps/sec/jquery.en}}

{document}

{sfragment}{jQuery: Write Less, Do More}

File: [courses/Jacobs/GenCS/course]{www/snip/jquery-intro.en}}

{document}

{sparagraph}

\usemodule{www/slides?jquery-layers} While \sn{javascript?JavaScript} is fully sufficient to manipulate the \sn{html?HTML} DOM, it is quite verbose and tedious to write. To remedy this, the web developer community has developed libraries that extend the \sn{javascript?JavaScript} language by new functionalities that more concise programs and are often used Instead of pure \sn{javascript?JavaScript}.

{sparagraph}

{document}

File: [courses/Jacobs/GenCS/course]{www/slides/jquery.en}}

{document}

{smodule}{jquery}

{frame}

{\sn{jQuery}: Write Less, Do More}

{itemize}

{sdefinition}

\Definame{jQuery} is a feature-rich \sn{javascript?JavaScript} \sn{library} that simplifies tasks like \sn{html?HTML} document traversal and manipulation, \sn{event}{handle}{handling}, animation, and \sn{ajax?Ajax}.

{sdefinition}

{sparagraph}[title=Using]

{itemize}

Download from \url{https://jquery.com/download/}, save on your system\lec{remember where}

integrate into your \sn{html?HTML} (usually in the \lstinline|<head>|)

\lstinputmhlisting[language=html,linerange=1-1]{www/code/jquery-script.html} or

from the \sn{internet} directly \lec{only works if you are online}

\lstinputmhlisting[language=html,linerange=2-2,basicstyle=\small\sf]{www/code/jquery-script.html}

{itemize}

{sparagraph}

{itemize}

{frame}

{smodule}

{document}

File: [courses/Jacobs/GenCS/course]{www/slides/jquery-layers.en}}

{document}

{smodule}{jquery-layers}

{nparagraph}

The key feature of \sn{jquery?jQuery} is that it borrows the notion of “\sns{selector}” to describe \sn{html?HTML} \sn{graph?node} \sns{set?set} from \sn{CSS?CSS} actually, \sn{jquery?jQuery} uses the \sn{CSS?CSS} \sns{CSS?selector} directly and then uses \sn{javascript?JavaScript}-like methods to act on them. In fact, the name \sn{jquery?jQuery} comes from the fact that \sns{selector} “query” for \sns{graph?node} in the \sn{DOM}.

{nparagraph}

{frame}{fragile}% needed somehow

{\sn{jquery?jQuery} Philosophy and Layers}

{itemize}

{sparagraph}[title=jQuery Philosophy]

Select a \sn[pre=sub]{tree} from the \sn{DOM}, and operate on it.

{sparagraph}

{sparagraph}[title=Syntax Convention]

\sn{jQuery} \sns{instruction} start with a

\linline|\$| to distinguish it from \sn{javascript?JavaScript}.\$

{sparagraph}

{sexample}[for=jQuery]

The following \sn{jQuery} command achieves a lot in four steps:

\linputmhlisting{www/code/jquery-layers.js}

{enumerate}

Find elements in the \sn{DOM} by \sn{CSS?CSS} selectors,
e.g.

\linline|\$("#myId")|)%\$\lec{\hateq query for nodes; hence the name \sn{jQuery}}

do something to them, here \linline|show()|\lec{chaining of methods}

change their layout by changing \sn{CSS?CSS} attributes, e.g. \linline|css("color","green")|

change their behavior, e.g. \linline|slideDown()|

{enumerate}

{sexample}

{sparagraph}[title=Good News]

\sn{jQuery} \sns{selector} \hateq \sn{CSS?CSS} \sn{selector}.

{sparagraph}

{itemize}

{frame}

{smodule}

{document}

File: [courses/Jacobs/GenCS/course]{www/slides/jquery-inserting.en}}

{document}

{smodule}{jquery-inserting}

\usestructure{tree}

{nparagraph}

We will now show a couple of \sn{jQuery} methods for inserting material into \sn{html?HTML} elements and discuss their behavior in examples

```

{npargraph}

{frame}[label=slide.jquery-inserting]
  {Inserting Material into the \sn{DOM}}
{itemize}

{sparagraph}[title=Inserting before the first \sn{child}]
\stinputmhlisting[linerange=1-1]{www/code/jquery-inserting.js}
  {sparagraph}

{sparagraph}[title=Inserting after the last \sn{child}]
\stinputmhlisting[linerange=2-3]{www/code/jquery-inserting.js}
  {sparagraph}

{sparagraph}[title=Inserting before/after an \sn{element}]
\stinputmhlisting[linerange=4-5]{www/code/jquery-inserting.js}
  {sparagraph}
  {itemize}
  {frame}
  {smodule}
  {document}

```

```

File: [courses/Jacobs/GenCS/course]{www/slides/dhtml-applications.en}
{document}
{smodule}{dhtml-applications}

```

```

{npargraph}
Let us fortify our intuition about \sn{dynamic} \sn{html?HTML} by going into a more
involved example. We use the \stinline|toggle| method from the \sn{jQuery}
layout layer to change visibility of a \sn{DOM} element. This method adds and removes a
\stinline|style="display:none"| attribute to an \sn{html?HTML} element and thus toggles
the visibility in the \sr{web browser}{browser} \sn{window}.
  {npargraph}

```

```

{frame}
  {Applications and useful tricks in Dynamic \sn{html?HTML}}
{itemize}

```

```

{sparagraph}[title=Observation]
\sn{jQuery} is not limited to adding material to the \sn{DOM}.
  {sparagraph}

```

```

{sparagraph}[title=Idea]
Use \sn{jQuery} to change \sn{CSS} \sr{property}{properties} in the \sn{DOM} as
well.
  {sparagraph}

```

```

{sexample}[title=Visibility,for={jQuery,CSS}]
Hide document parts by setting \sn{CSS?CSS} \stinline|style| \sns{xml-markup?attribute} to
\stinline|display:none|.
\stset{moreemph={[[2]script,onClick],moreemph={[[3]document.getElementById,style,display,toggleDiv]]}
\stinputmhlisting[language=HTML,basicstyle=\footnotesize\sf,belowskip=0pt]
{www/code/dropper.html}

```



```
{sexample}  
{itemize}  
{frame}  
{smodule}  
{document}
```

File: [courses/Jacobs/GenCS/course]{www/slides/buttons-ex.en}]

```
{document}  
{smodule}{buttons-ex}
```

```
{frame}  
  {Fun with Buttons (Three easy Interactions)}  
{itemize}
```

```
{sexample}[id=ex.hover,title=A Button that Changes Color on Hover,  
for={jQuery,HTML}]  
\lstinputmhlisting[language=HTML,linerange={15-21},gobble=2,  
basicstyle=\footnotesize\sf]{www/code/buttons.html}  
{itemize}
```

The \sn{HTML} has a button with text “hover”.

The \sn{jQuery} code selects it via its \lstinline{id} and catches its hover \sn{event} via the \lstinline{hover()} method.

This takes two functions as arguments:

```
{itemize}
```

The first is called when the mouse moves into the button, the second when it leaves.

The first changes changes the button color to red, the second reverts this.

```
{itemize}  
{itemize}  
{sexample}  
{itemize}  
{frame}
```

```
{frame}  
  {Fun with Buttons (Three easy Interactions)}  
{itemize}
```

```
{sexample}[id=ex.readmore,title=A Button that Uncovers Text,  
for={jQuery,HTML}]  
\lstinputmhlisting[language=HTML,linerange={23-34},gobble=2,  
basicstyle=\footnotesize\sf]{www/code/buttons.html}  
{itemize}
```

The \sn{HTML} has two buttons (one of them visible) and a text.

The \sn{jQuery} code selects both buttons via their \lstinline{read} class.

A click \sn{event} activates the \lstinline{click()} method taking an event handler function:

```
{itemize}
```

This selects the text via its \lstinline{id} attribute \lstinline{rTeX} and uses the \lstinline{toggle()} method which changes the \lstinline{display} between \lstinline{none} and \lstinline{block}.

The first \sn{parameter} of \lstinline{toggle()} is a duration for the animation.

The second is a completion function to be run after animation finishes.

Here completion function makes the respective other button visible (read

more/less).

{itemize}

{itemize}

{sexample}

{itemize}

{frame}

{frame}

{Fun with Buttons (Three easy Interactions)}

{itemize}

{sexample}[id=ex.click,title=A Button that Plays a Sound,
for={jQuery,HTML}]

\stinputmhlisting[language=HTML,linrange={36-45},gobble=2,
basicstyle=\footnotesize\sf]{www/code/buttons.html}

{itemize}

The \sn{HTML} has a button with text “sound” and an \stinline|onclick|
attribute.

That activates the \stinline|playSound| function on a URL:

The \stinline|playSound| function is defined in the \stinline|script|
element: it

{itemize}

logs the action and \sn{URL} in the \sr{web browser}{browser} console,
makes a new audio object \stinline|a|, which
plays it via the \stinline|play()| method.

{itemize}

{itemize}

{sexample}

{itemize}

{frame}

{nparagraph}

For reference, here is the full code of the examples in one file:

\stinputmhlisting[language=HTML,basicstyle=\footnotesize\sf]{www/code/buttons.html} It
has a bit more general \sn{CSS} and includes \sn{jQuery} in the beginning.

{nparagraph}

{smodule}

{document}

{sfragment}

{document}

File: [courses/FAU/IWGS/course]{webapps/sec/recap.en}]

{document}

{sfragment}[id=sec.webapps-recap]{Web Applications: Recap}

File: [courses/FAU/IWGS/course]{webapps/slides/recap.en}]

{document}

{frame}

\usemodule{webapps/slides?bottle-stpl}

\usemodule[srnglom/www]{mod?webapp}

\usemodule[courses/Jacobs/GenCS/course]{www/slides?jquery-layers}

\usemodule{webapps/slides?bottle-routing}

\usemodule[srnglom/computing]{mod?Turing-complete}

{What Tools have we seen so far?}

{itemize}

<1-> HTML (Hypertext Markup Language)

{itemize}

Text-based \sr{markup format}{markup language} for the web.

Tree structure (realized as the DOM in the browser)

{itemize}

easy search\&find \ogre Selection

\sn{DOM} changes easy by clear dependencies.

{itemize}

{itemize}

<2-> \sn{CSS} (\sr{CSS}{Cascading Stylesheets})

{itemize}

Language for specifying layout of \sn{HTML}/\sn{DOM}

\sn{CSS} selection ties layout specifications into \sn{HTML}/\sn{DOM}

{itemize}

<3-> \Sn{bottle WSGI} (Server-Side \sn{web page} generation via \python)

{itemize}

full \sn{programming language} for comprehensive functionality

\sns{route} for complex but coherent \sns{web site}

\sn{template engine} for \sn{HTML}-centered \sn{web page} design

{itemize}

<4-> \sn{JavaScript} (client-side scripting)

{itemize}

full \sn{programming language} \lec{\sn{Turing complete}}

\sn{post=matic}{program} changes to the \sn{DOM} \ergo \sn{dynamic} \sn{HTML}

{itemize}

navigating the \sn{DOM} via JS-selection\lec{relatively clumsy, but sufficient}

\sn{jQuery} navigates the \sn{DOM} via \sn{CSS} \sns{CSS?selector}\lec{reuses

successful concepts}

{itemize}

{itemize}

{itemize}

{frame}

{document}

File: [courses/FAU/IWGS/course]{webapps/slides/frontend-recap.en}]

{document}

{frame}

\usemodule[smsglom/www]{mod?html}

\usemodule[smsglom/www]{mod?web-page}

\usemodule[courses/Jacobs/GenCS/course]{www/slides?jquery-layers}

{Recap: Web Application Frontend}

{itemize}

{sparagraph}[title=Recap: Web Application Frontend]

{center}

{slideshow}

\nextslide{\Sns{web page} are just \sn{HTML} \sns{file}.\\\hfill

\nextslide{Layout is specified by \sn{CSS} instructions and \sns{CSS?selector}\\\hfill

\nextslide{\sn{JavaScript} specifies behavior\\hfill

\nextslide{for \sn{post=ing}{interact} with the user\\hfill

\nextslide{\sn{jQuery} \hateq more succinct \sn{JavaScript}\\hfill

\lastslide{\sn{jQuery} attaches behaviors to \sn{DOM} elements via \sn{CSS}

\sns{CSS?selector}\\hfill

{slideshow}

{center}

{sparagraph}

{itemize}

{frame}

{document}

{sfragment}

{document}

{sfragment}

{document}

File: [courses/FAU/IWGS/course]{course/sec/concl1.en}]

{document}

{sfragment}{What did we learn in IWGS-1?}

File: [courses/FAU/IWGS/course]{course/slides/outline.en}]

{document}

{frame}[label=slide.iwgs-outline]

\usemodule[courses/Jacobs/GenICT/course]{python/slides/nutshell?python-nutshell}

\usemodule[courses/Jacobs/GenCS/course]{www/slides?CSS}

\usemodule[courses/Jacobs/GenCS/course]{www/slides?javascript}

\usemodule[smglom/www]{mod?webapp}

{Outline of \useSGvar{courseacronym} 1:}

{itemize}

\Sn{programming} in \python:\lec{main tool in
\useSGvar{courseacronym}}

{itemize}

Systematics and culture of \sn{programming}

Program and control structures

Basic data strutures like numbers and strings, character encodings, unicode, and
regular expressions

{itemize}

Digital documents and document processing:

{itemize}

text files

markup systems, \sn{html?HTML}, and \sn{CSS?CSS}

\sn{xml?XML}: Documents are trees.

{itemize}

Web technologies for \sn{interactive} documents and
\sns{web application}

{itemize}

\sn{internet} infrastructure: web browsers and servers

serverside computing: bottle routing and

client-side \sn[post=ion]{interact}: dynamic \sn{html?HTML},

\sn{javascript?JavaScript}, \sn{html?HTML} forms

{itemize}

\Sn{web application} project\lec{fill in the blanks to obtain a working web app}

{itemize}

{frame}

{document}

File: [courses/FAU/IWGS/course]{course/slides/outline2.en}]

{document}

{frame}[label=slide.iwgs-outline2]

\usemodule[courses/Jacobs/GenICT/course]{python/slides/nutshell?python-nutshell}

\usemodule[courses/Jacobs/GenCS/course]{xml/slides?xml-nutshell}

\usemodule{databases/slides?json}

\usemodule[smglom/computing]{mod?database}

\usemodule[smglom/www]{mod?semantic-web}

\usemodule[smglom/computing]{mod?image-processing}

{Outline of \useSGvar{courseacronym}-II:}

{itemize}

<1-> \Sns{database}

{itemize}

CRUD operations, \sn[post=ing]{query}, and python embedding

\sn{xml?XML} and \sn{json?JSON} for file based data storage

{itemize}

<2-> \textrm{BooksApp}: a Books Application with \sn{persistent} storage

<3-> \Sn{image processing}

{itemize}

Basics

Image transformations, Image Understanding

{itemize}

<4-> Ontologies, \sn{semantic web}, and WissKI

{itemize}

Ontologies\lec{inference \ergo get out more than you put in}

\sn{semantic web} Technologies\lec{standardize ontology formats and

inference}

Using \sn{semantic web} Tech for cultural heritage research data \ergo the

WissKI System

{itemize}

<5-> Legal Foundations of Information Systems

{itemize}

Copyright \& Licensing

Data Protection (GDPR)

{itemize}

{itemize}

{frame}

{document}

{sfragment}

{document}

{document}