

Using Python and Latexslides to Make Slides

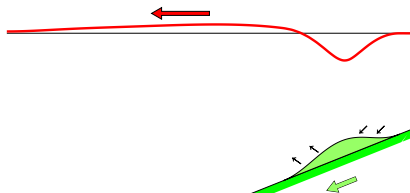
Hans Petter Langtangen^{1,2} Ilmar M. Wilbers^{1,3}

Simula Research Laboratory¹

Dept. of Informatics, University of Oslo²

Dept. of Mathematics, University of Oslo³

June 2010



List of Topics

- 1 Intro to Latexslides
 - Plain Text Slides
 - Figures
 - Computer Code
- 2 More information



Do you use \LaTeX for writing slides?

Continue studying these slides if your answer to at least one of the following questions is 'yes':

- 1 Are you using prosper for writing slides?
- 2 Have you not yet discovered latex-beamer?
- 3 Would you like your slide collection to be independent of what is the currently most popular \LaTeX slide package?
- 4 Would you like to write less \LaTeX source code when you create presentations?
- 5 Would like to get more flexibility than what plain ASCII files with \LaTeX source provide?

1 Intro to Latexslides

- Plain Text Slides
- Figures
- Computer Code

2 More information

What is Latexslides?

- A Python module
- You write slides as Python code, i.e., as function calls
- The function calls are translated to \LaTeX
- Changes are easier to perform in the Python code than in the corresponding \LaTeX code – that is the main purpose of Latexslides
- From the Python code you can automatically generate prosper or beamer \LaTeX code and HTML

1 Intro to Latexslides

- Plain Text Slides
- Figures
- Computer Code

2 More information

First example: simple bullet lists

Here is how we wrote the previous slide:

```
BulletSlide('What is Latexslides?',  
['A Python module',  
 'You write slides as Python code, i.e., as function calls',  
 r'The function calls are translated to \LaTeX',  
 'Changes are easier to perform in the Python code than '  
 r'in the corresponding \LaTeX~code -- that is the main '  
 'purpose of Latexslides',  
 'From the Python code you can automatically generate '  
 r'prosper or beamer \LaTeX~code and HTML'],)
```

Explanations:

- The first argument is the title of the slide
- Bullet lists are simply Python lists of (raw) strings

A general slide is defined by using Slide

```
Slide(title='title', content=[
    Text(r'some plain text'),
    BulletList([r'item1',
                r'item2',
                r'item3',
                ],),
    Code(r'some code'),],)
```

- Use raw strings if the text has \LaTeX commands with backslash (always using raw strings is a good habit!)
- The `title=` and `content=` keywords can be omitted if they are the first two arguments given to `Slide` or one of its subclasses.

The available objects on a slide are `Text`, `Code` and `BulletList`

About the appearance of the three slide elements

Text, code and bullets can be typeset as shadowed blocks by using `TextBlock`, `CodeBlock` and `BulletBlock` instead of `Text`, `Code` and `BulletList`

- Note that this text is not a block since we used
`BulletList`
instead of `BulletBlock`

Each block may have a title!

The title is enabled by the argument

```
heading='Each block may ...'
```

Note that

```
Code("...")
```

can be used within the other objects

Blocks and slides

If only a single block is used on a slide, subclasses of `Slide` can be used, providing a simpler syntax:

- `BulletSlide`
- `TextSlide`
- `RawSlide`

These only have the following keywords:

- `title`
- `text/bullets`
- `block_heading`

`RawSlide` simply takes the raw \LaTeX code for a slide such that old code can be reused (see the code for the first slide)

Blocks and slides

If only a single block is used on a slide, subclasses of `Slide` can be used, providing a simpler syntax:

- `BulletSlide`
- `TextSlide`
- `RawSlide`

These only have the following keywords:

- `title`
- `text/bullets`
- `block_heading`

`RawSlide` simply takes the raw \LaTeX code for a slide such that old code can be reused (see the code for the first slide)

Blocks and slides

If only a single block is used on a slide, subclasses of `Slide` can be used, providing a simpler syntax:

- `BulletSlide`
- `TextSlide`
- `RawSlide`

These only have the following keywords:

- `title`
- `text/bullets`
- `block_heading`

`RawSlide` simply takes the raw \LaTeX code for a slide such that old code can be reused (see the code for the first slide)

Blocks and slides

If only a single block is used on a slide, subclasses of `Slide` can be used, providing a simpler syntax:

- `BulletSlide`
- `TextSlide`
- `RawSlide`

These only have the following keywords:

- `title`
- `text/bullets`
- `block_heading`

`RawSlide` simply takes the raw \LaTeX code for a slide such that old code can be reused (see the code for the first slide)

Blocks and slides

If only a single block is used on a slide, subclasses of `Slide` can be used, providing a simpler syntax:

- `BulletSlide`
- `TextSlide`
- `RawSlide`

These only have the following keywords:

- `title`
- `text/bullets`
- `block_heading`

`RawSlide` simply takes the raw \LaTeX code for a slide such that old code can be reused (see the code for the first slide)

Blocks and slides

If only a single block is used on a slide, subclasses of `Slide` can be used, providing a simpler syntax:

- `BulletSlide`
- `TextSlide`
- `RawSlide`

These only have the following keywords:

- `title`
- `text/bullets`
- `block_heading`

`RawSlide` simply takes the raw \LaTeX code for a slide such that old code can be reused (see the code for the first slide)

How to dim blocks and bullet points

- Want to dim the blocks (as in the previous slide)? Just add an argument

```
dim='blocks'
```

- Want bullet items to pop up one by one? Just add an argument

```
dim='progressive'
```

- Want one bullet visible and the other dimmed? Just add

```
dim='single' #dim=False (default) turns off dimming
```

Note that subbullets appear at the same time:

- Want the previous effect but with all bullets appearing at the end? Just add

```
dim='single_then_all'
```

- This is much easier than editing the underlying \LaTeX code!

How to dim blocks and bullet points

- Want to dim the blocks (as in the previous slide)? Just add an argument

```
dim='blocks'
```

- Want bullet items to pop up one by one? Just add an argument

```
dim='progressive'
```

- Want one bullet visible and the other dimmed? Just add

```
dim='single' #dim=False (default) turns off dimming
```

Note that subbullets appear at the same time:

```
* Subbullet 1
```

- Want the previous effect but with all bullets appearing at the end? Just add

```
dim='single_then_all'
```

- This is much easier than editing the underlying \LaTeX code!

How to dim blocks and bullet points

- Want to dim the blocks (as in the previous slide)? Just add an argument

```
dim='blocks'
```

- Want bullet items to pop up one by one? Just add an argument

```
dim='progressive'
```

- Want one bullet visible and the other dimmed? Just add

```
dim='single' #dim=False (default) turns off dimming
```

Note that subbullets appear at the same time:

- Subbullet 1

- Want the previous effect but with all bullets appearing at the end? Just add

```
dim='single_then_all'
```

- This is much easier than editing the underlying \LaTeX code!

How to dim blocks and bullet points

- Want to dim the blocks (as in the previous slide)? Just add an argument

```
dim='blocks'
```

- Want bullet items to pop up one by one? Just add an argument

```
dim='progressive'
```

- Want one bullet visible and the other dimmed? Just add

```
dim='single' #dim=False (default) turns off dimming
```

Note that subbullets appear at the same time:

- Subbullet 1
- Want the previous effect but with all bullets appearing at the end? Just add

```
dim='single_then_all'
```

- This is much easier than editing the underlying \LaTeX code!

How to dim blocks and bullet points

- Want to dim the blocks (as in the previous slide)? Just add an argument

```
dim='blocks'
```

- Want bullet items to pop up one by one? Just add an argument

```
dim='progressive'
```

- Want one bullet visible and the other dimmed? Just add

```
dim='single' #dim=False (default) turns off dimming
```

Note that subbullets appear at the same time:

- Subbullet 1
- Want the previous effect but with all bullets appearing at the end? Just add

```
dim='single_then_all'
```

- This is much easier than editing the underlying \LaTeX code!

How to dim blocks and bullet points

- Want to dim the blocks (as in the previous slide)? Just add an argument

```
dim='blocks'
```

- Want bullet items to pop up one by one? Just add an argument

```
dim='progressive'
```

- Want one bullet visible and the other dimmed? Just add

```
dim='single' #dim=False (default) turns off dimming
```

Note that subbullets appear at the same time:

- Subbullet 1
- Want the previous effect but with all bullets appearing at the end? Just add

```
dim='single_then_all'
```

- This is much easier than editing the underlying L^AT_EX code!

Why? plain \LaTeX is so easy...

- Latexslides turns the talk into living data structures
- With the data structures, you can easily generate Prosper, Beamer, HTML or write your own format output
- Some of us experimented with the idea for fun, now we're regularly using it – it's simply more convenient
- Latexslides talks can make use of future fancy \LaTeX slide packages
- You can tweak the resulting \LaTeX file if you want
- Talks are composed as lists of slide objects – you can import slide objects from previous talks and compose new collections
- Figures are definitely easier with Latexslides

Why? plain \LaTeX is so easy...

- Latexslides turns the talk into living data structures
- With the data structures, you can easily generate Prosper, Beamer, HTML or write your own format output
- Some of us experimented with the idea for fun, now we're regularly using it – it's simply more convenient
- Latexslides talks can make use of future fancy \LaTeX slide packages
- You can tweak the resulting \LaTeX file if you want
- Talks are composed as lists of slide objects – you can import slide objects from previous talks and compose new collections
- Figures are definitely easier with Latexslides

Why? plain \LaTeX is so easy...

- Latexslides turns the talk into living data structures
- With the data structures, you can easily generate Prosper, Beamer, HTML or write your own format output
- Some of us experimented with the idea for fun, now we're regularly using it – it's simply more convenient
- Latexslides talks can make use of future fancy \LaTeX slide packages
- You can tweak the resulting \LaTeX file if you want
- Talks are composed as lists of slide objects – you can import slide objects from previous talks and compose new collections
- Figures are definitely easier with Latexslides

Why? plain \LaTeX is so easy...

- Latexslides turns the talk into living data structures
- With the data structures, you can easily generate Prosper, Beamer, HTML or write your own format output
- Some of us experimented with the idea for fun, now we're regularly using it – it's simply more convenient
- Latexslides talks can make use of future fancy \LaTeX slide packages
- You can tweak the resulting \LaTeX file if you want
- Talks are composed as lists of slide objects – you can import slide objects from previous talks and compose new collections
- Figures are definitely easier with Latexslides

Why? plain \LaTeX is so easy...

- Latexslides turns the talk into living data structures
- With the data structures, you can easily generate Prosper, Beamer, HTML or write your own format output
- Some of us experimented with the idea for fun, now we're regularly using it – it's simply more convenient
- Latexslides talks can make use of future fancy \LaTeX slide packages
- You can tweak the resulting \LaTeX file if you want
- Talks are composed as lists of slide objects – you can import slide objects from previous talks and compose new collections
- Figures are definitely easier with Latexslides

Why? plain \LaTeX is so easy...

- Latexslides turns the talk into living data structures
- With the data structures, you can easily generate Prosper, Beamer, HTML or write your own format output
- Some of us experimented with the idea for fun, now we're regularly using it – it's simply more convenient
- Latexslides talks can make use of future fancy \LaTeX slide packages
- You can tweak the resulting \LaTeX file if you want
- Talks are composed as lists of slide objects – you can import slide objects from previous talks and compose new collections
- Figures are definitely easier with Latexslides

Why? plain \LaTeX is so easy...

- Latexslides turns the talk into living data structures
- With the data structures, you can easily generate Prosper, Beamer, HTML or write your own format output
- Some of us experimented with the idea for fun, now we're regularly using it – it's simply more convenient
- Latexslides talks can make use of future fancy \LaTeX slide packages
- You can tweak the resulting \LaTeX file if you want
- Talks are composed as lists of slide objects – you can import slide objects from previous talks and compose new collections
- Figures are definitely easier with Latexslides

Why? plain \LaTeX is so easy...

- Latexslides turns the talk into living data structures
- With the data structures, you can easily generate Prosper, Beamer, HTML or write your own format output
- Some of us experimented with the idea for fun, now we're regularly using it – it's simply more convenient
- Latexslides talks can make use of future fancy \LaTeX slide packages
- You can tweak the resulting \LaTeX file if you want
- Talks are composed as lists of slide objects – you can import slide objects from previous talks and compose new collections
- **Figures are definitely easier with Latexslides**

Why? plain \LaTeX is so easy...

- Latexslides turns the talk into living data structures
- With the data structures, you can easily generate Prosper, Beamer, HTML or write your own format output
- Some of us experimented with the idea for fun, now we're regularly using it – it's simply more convenient
- Latexslides talks can make use of future fancy \LaTeX slide packages
- You can tweak the resulting \LaTeX file if you want
- Talks are composed as lists of slide objects – you can import slide objects from previous talks and compose new collections
- Figures are definitely easier with Latexslides

List of Topics

1 Intro to Latexslides

- Plain Text Slides
- **Figures**
- Computer Code

2 More information

Handling figures is really easy

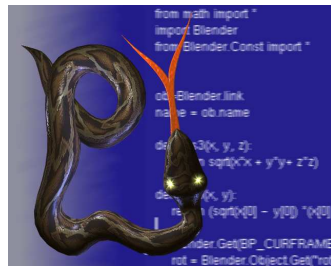
- Putting figures in \LaTeX slides is not that easy, especially not if you want them to the left or right of the figure and move them around later
- Here is what you do with Latexslides, just add

```
figure='figs/python1.ps',    # filename
figure_pos='e',              # east ('e'), west ('w'),
                             # north ('n'), south ('s')
figure_fraction_width=0.5,
left_column_width=0.6      # => 0.4 fraction width for figure
```

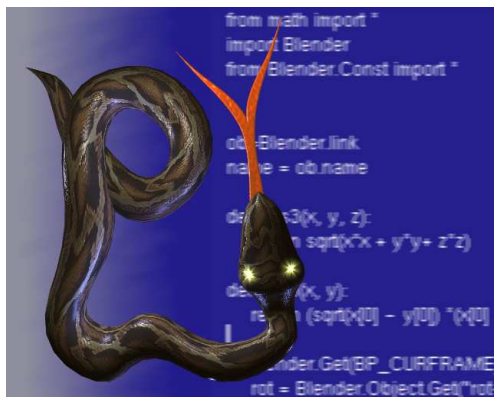
See next slides for an example

Slide with a figure

- Bullets to the west
- Figure to the east
- Easy to change: 'e' \rightarrow 'w'
- ...and then text is to the right

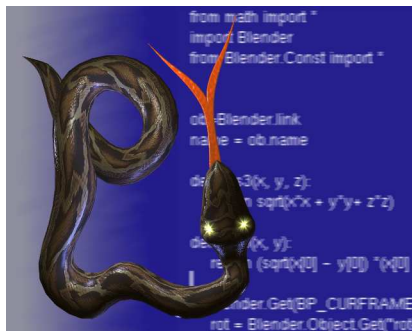


Slide with a figure



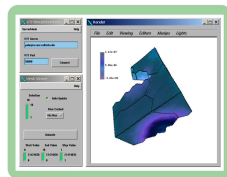
- Bullets to the east
- Figure to the west
- Easy to change: 'w'
→ 'n'
- ...and then text below
the figure

Slide with a figure



- Bullets to the south
- Figure to the north
- Easy to change: 'n' → 's'
- ...and then text is above the figure

Several figures in one slide



- You simply provide a tuple (or list) of figure file names and a tuple of fraction widths
- Example:

```
figure=('figs/python2.ps','figs/python3.ps'),  
figure_fraction_width=(0.45,0.55),  
figure_pos='n',
```

1 Intro to Latexslides

- Plain Text Slides
- Figures
- Computer Code

2 More information

Code objects take care of verbatim text

- Want to include computer code or some other verbatim text?

```
bullets=[r'Here is an example:' +  
Code("""  
def mypyfunc(somearg):  
    for i in somearg:  
        p = process(i)  
        if p in mylist:  
            return p  
        else:  
            return None  
""")
```

- Code objects are wrapped in fancyvrb "Verbatim" environments

Result of using Code objects

Here is the result of the constructions on the previous slide:

- Here is an example:

```
def mypyfunc(somearg):  
    for i in somearg:  
        p = process(i)  
        if p in mylist:  
            return p  
    else:  
        return None
```

Code objects can also use ptex2tex environments

- Can set `Code.ptex2tex_envir = "pycod"` (for instance), which then applies to all Code objects
- Can set `Code.latex_envir = "minted"` (or "Verbatim", which then applies these \LaTeX environments to all Code objects
- Can alternatively provide `ptex2tex_envir` argument to Code constructor:

```
bullets=[r'Here is an example:' +
Code("""
def mypyfunc(somearg):
    for i in somearg:
        p = process(i)
        if p in mylist:
            return p
        else:
            return None
""", ptex2tex_envir='pycod')
```


Result of using Code objects

Here is the result of the constructions on the previous slide:

- Here is an example:

```
def mpyfunc(somearg):  
    for i in somearg:  
        p = process(i)  
        if p in mylist:  
            return p  
    else:  
        return None
```

- Here pycod corresponds to ANS_Python in .ptex2tex.cfg
- Note that the slides should be written to a file with extension .p.tex
- Note that ptex2tex must be installed and used

Code objects take care of verbatim text

- Can also just insert ptex2tex environment delimiters in the code:

```
# Recall to use raw strings because of \b...!!
bullets=[r'Here is an example:' + Code(r"""
\bpypcod
def mypyfunc(somearg):
    for i in somearg:
        p = process(i)
        if p in mylist:
            return p
        else:
            return None
\epypcod
""")]
```

- Any ptex2tex_envir argument will overrule pypcod here

Result of using Code objects

Here is the result of the constructions on the previous slide:

- Here is an example:

```
def mypyfunc(somearg):  
    for i in somearg:  
        p = process(i)  
        if p in mylist:  
            return p  
    else:  
        return None
```

Code objects can also use ptex2tex environments

- Set `Code.ptex2tex_envir = "cod"`

```
Code.ptex2tex_envir = "cod"
...
Slide(...)
bullets=[r'Here is an example:' +
Code("""
def mypyfunc(somearg):
    for i in somearg:
        p = process(i)
        if p in mylist:
            return p
        else:
            return None
""")
```

Result of using Code objects

Here is the result of the constructions on the previous slide:

- Here is an example:

Code

```
def mypyfunc(somearg):  
    for i in somearg:  
        p = process(i)  
        if p in mylist:  
            return p  
    else:  
        return None
```

- Here cod corresponds to CodeRule in .ptex2tex.cfg

Code objects can also avoid using `ptex2tex` environments and instead a hardcoded `Verbatim` or `minted` environments

- Set `Code.latex_envir = "minted"` (or `"Verbatim"`)

```
Code.latex_envir = "minted"
...
Slide(...)
bullets=[r'Here is an example:' +
Code("""
def mypyfunc(somearg):
    for i in somearg:
        p = process(i)
        if p in mylist:
            return p
        else:
            return None
""", fontsize='tiny', leftmargin='15mm')
```

- Note that `fontsize` and `leftmargin` can only be set with effect when `ptex2tex` is not used

Result of using Code objects

Here is the result of the constructions on the previous slide:

- Here is an example:

```
def mypyfunc(somearg):  
    for i in somearg:  
        p = process(i)  
        if p in mylist:  
            return p  
    else:  
        return None
```

- This code style corresponds to `MintedPython`, but the \LaTeX environment it is hardcoded in the `.tex` file (no use of `ptex2tex`)
- This minted code style requires `pygmentize` to be installed and \LaTeX to be invoked by `latex -shell-escape`

List of Topics

1 Intro to Latexslides

- Plain Text Slides
- Figures
- Computer Code

2 More information

Adding sections and subsections

- Adding a section is just like adding a slide:

```
sec = Section('Long title', 'Short title')
```

The short title is optional, and will be used if there is not enough room for the long title

- SubSection works the same way, but a Section needs to be defined prior to a SubSection
- Slide objects are automatically a part of the current section or subsection
- If no sections are defined, all slides will be part of the main talk

You can turn off the header and footer

- Want to navigate in your talk? Click in the header!
- Sometimes the navigation header and the author/title in the footer is disturbing
- Turn header/footer decoration off for all slides:

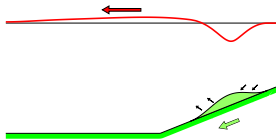
```
header_footer = False
```

There is support for mapping slides

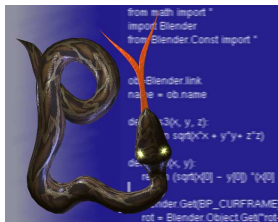
Michael Alley-style mapping slides can be created, using `MappingSlide` and a list of heading-figure pairs (optionally heading-figure-width triples, where width denotes the relative width of the figure). Here is an example:

```
mapping_slide2 = \  
MappingSlide([('Wave motion', 'figs/wave-dueto-slide.ps'),  
              ('Some Python', 'figs/python1.ps'),  
              ])
```

See the next slide for the result (the headings are placed to the right of the figures, and the figure-heading pairs appear diagonally on the slide).



Wave motion



Some Python

The look of the file header

```
from latexslides import *

# First set some module variables:
package = BeamerSlides
theme = 'blue2'
header_footer = True

# Add newcommands:
newcommands = r"""
\newcommand{\OBS}[1]{\marginpar{\scriptsize##1}}
"""
```

Can I change from Beamer to Prosper or HTML?

- Of course, this is trivial:

```
#package = BeamerSlides  
package = ProsperSlides  
package = HTMLSlides
```

- Prosper is fine (best?) for handouts
- Handouts for Beamer are made setting the keyword

```
handout=True
```

for colour prints and

```
colour=False
```

for b/w handouts

The titlepage

```
ifi = "Dept.~of Informatics, University of Oslo"
math = "Dept.~of Mathematics, University of Oslo"
simula = "Simula Research Laboratory"

hpl = 'Hans Petter Langtangen'
ilmarw = 'Ilmar M. Wilbers'

slides = BeamerSlides(
    title='Using Python and Latexslides to Make Slides',
    author_and_inst=[(hpl, simula, ifi),
                     (ilmarw, simula, math)],
    date='March 2008',
    titlepage_figure='figs/wave-dueto-slide.ps',
    # Figure to the south of the title:
    titlepage_figure_pos='s',
    titlepage_figure_fraction_width=0.5,
    # Used if titlepage_figure_pos is 'e':
    #titlepage_left_column_width=1.0,
    toc_heading='List of Topics',
    toc_figure='figs/python1.ps',
```

Emacs commands

- The authors have found the following Emacs shortcuts very helpful:

- Alt + up-arrow:

```
(global-set-key [ (meta up)] " = Slide('',
content=[BulletBlock(bullets=[
    '',")
```

- Alt + down-arrow:

```
(global-set-key [ (meta down)] "
)], # end bullets and BulletBlock
], # end contents
)")
```

- These should be included in the .emacs file in your home directory
- This example is for the opening and closing of a BulletBlock, but illustrate how Emacs shortcuts can be used

Slide Objects 1

- You may save each slide in a Slide object (recommended!!)

```
slides = BeamerSlides(...)
motivation2 = \
    Slide('Motivation Cont.',
          [BulletBlock([...], ...), ...])
```

- A list of all slide objects in a file can be generated with the following executable:

```
extract_slidenames mytalk.py
```

- The generated list can be included at the bottom of your file

Slide Objects 2

- Talks can be composed of lists of slide objects

```
slides = BeamerSlides(...)
collection = [header, title, sec1, intro1, test,
              sec2, plainloop]
# Can make some slides invisible:
for s in intro1, plainloop: s.hidden = True
# Or perhaps more elegant:
collection = [header, title, sec1, intro1.hide,
              test, sec2, plainloop.hide]

slides.add_slides(collection)
# Write slides to file:
f = open('exampletalk.tex', 'w')
f.write(slides.get_latex())
# Or the simplest, which will output the
# necessary latex commands as well:
slides.write(filename)
```

- In this way you can reuse old slides in new contexts without cut and paste, i.e., you can have a single source for each slide

How to Compile the Talk

- You write your talk as Python code in a plain text file, say `mytalk.py`
- The next step is to generate \LaTeX code:
`unix> python mytalk.py`
- The latex commands you need to run will be the output if the `write` function is used

How to write mathematics

Use triple-quoted raw strings and just write the plain \LaTeX code

Latexslide source:

```
TextBlock(heading='Latexslide source:',
            text=r"""Here is an equation
\[\text{ax}^2 + \text{bx} + \text{c} = 0\]
that is easy to solve.
""")
```

Result:

Here is an equation

$$ax^2 + bx + c = 0$$

that is easy to solve.

Learning Latexslides

- Have a look at the source code for this presentation, it can be found in the file `exampletalk.py`. Going through the presentation and the source code simultaneously should get you started.
- When running

```
unix> latexslides mytalk.py
```

the file `mytalk.py` is created. This file contains the basics and will help you get started on a new talk.